UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA DE TELEINFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE TELEINFORMÁTICA

DOUTORADO EM ENGENHARIA DE TELEINFORMÁTICA

DAVID NASCIMENTO COELHO

SPARSE KERNEL PROTOTYPE-BASED CLASSIFIERS: NEW CONTRIBUTIONS

FORTALEZA

2025

DAVID NASCIMENTO COELHO

SPARSE KERNEL PROTOTYPE-BASED CLASSIFIERS: NEW CONTRIBUTIONS

Tese apresentada ao Programa de Pós-Graduação em Engenharia de Teleinformática do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do título de doutor em Engenharia de Teleinformática. Área de Concentração: Sinais e Sistemas

Orientador: Prof. Dr. Guilherme de Alencar Barreto

FORTALEZA

2025

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

C615s Coelho, David Nascimento.
Sparse Kernel Prototype-based Classifiers: New Contributions / David Nascimento Coelho. – 2025.
139 f. : il. color.

Tese (doutorado) – Universidade Federal do Ceará, Centro de Tecnologia, Programa de Pós-Graduação em Engenharia de Teleinformática, Fortaleza, 2025.
Orientação: Prof. Dr. Guilherme de Alencar Barreto.

1. Classificação. 2. Modelos baseados em protótipos. 3. Métodos de Kernel. 4. Métodos de esparsificação. 5. Fluxo contínuo de dados. I. Título.

CDD 621.38

DAVID NASCIMENTO COELHO

SPARSE KERNEL PROTOTYPE-BASED CLASSIFIERS: NEW CONTRIBUTIONS

Tese apresentada ao Programa de Pós-Graduação em Engenharia de Teleinformática do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do título de doutor em Engenharia de Teleinformática. Área de Concentração: Sinais e Sistemas

Aprovada em: 14 de Novembro de 2025

BANCA EXAMINADORA

Prof. Dr. Guilherme de Alencar Barreto (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. João Baptista de Oliveira e Souza Filho
Universidade Federal do Rio de Janeiro (UFRJ)

Prof. Dr. Marcos Eduardo Ribeiro Do Valle Mesquita
Universidade Estadual de Campinas (UNICAMP)

Prof. Dr. César Lincoln Cavalcante Mattos
Universidade Federal do Ceará (UFC)

Prof. Dr. Ajalmar Rêgo da Rocha Neto
Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)

Aos meus filhos, Isabela e José Pedro, que me deram a motivação necessária para finalização deste trabalho.

ACKNOWLEDGEMENTS

A Deus, por todas as pessoas e oportunidades que colocou em meu caminho.

À Nice Aguiar, que me acompanha desde a graduação, dividindo todas as derrotas, conquistas, tristezas e alegrias.

Aos meus pais, irmãos e sobrinhos, que nos momentos de minha ausência dedicados ao estudo, sempre fizeram entender que o futuro é feito a partir da constante dedicação no presente.

Ao Prof. Dr. Guilherme Barreto por toda paciência, dedicação e orientação durante o período do mestrado e do doutorado.

Aos amigos do laboratório SPIRAL pelas discussões, ajuda e momentos de descontração. Um agradecimento especial ao Renan Bessa e ao Igor Sousa pelas contribuições na reta final deste trabalho.

Ao colegiado do curso de Engenharia da Computação da UFC Campus Sobral, que permitiu que eu fizesse parte do doutorado em dedicação exclusiva.

Aos professores do Programa de Pós-graduação em Engenharia de Teleinformática e todos que, direta ou indiretamente me ajudaram e são exemplos que levo para minha atuação profissional como docente.

À Isabela Aguiar Coelho e ao José Pedro Aguiar Coelho, minhas maiores motivações na reta final deste trabalho.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), os quais financiaram esta pesquisa: CAPES (Código Financeiro 001) e CNPq (processo no. 309379 / 2019-9).

"A maior recompensa para o trabalho do homem não é o que ele ganha com isso, mas o que ele se torna com isso." (John Ruskin)

RESUMO

Esta tese tem por objetivo contribuir para o desenvolvimento de classificadores baseados em protótipos utilizando métodos de *kernel*. Inicialmente, investiga-se a aplicação do truque de *kernel* à rede neural *self-organizing map* (SOM) na classificação de bancos de dados do tipo *batch*, avaliando-se a influência de diferentes funções de *kernel* e métodos de rotulação. Os resultados indicaram que o uso de funções de *kernel* além das tradicionais (Gaussiana e Linear) pode gerar desempenhos superiores. Verificou-se também que, nos bancos de dados analisados, há vantagens em computar tanto a medida de distância quanto a definição do protótipo vencedor no espaço de atributos. Em seguida, considerando a importância da definição do número de protótipos para a acurácia de classificadores baseados em protótipos, propõe-se a utilização de métodos de esparsificação, tais como *approximate linear dependence*, *novelty*, *surprise* e *coherence*, para seleção automática dessa quantidade, utilizando, como classificador, o algoritmo K-vizinhos mais próximos ponderados. Esses métodos são comparados quanto à influência na acurácia e na quantidade de protótipos selecionados para o modelo. Nos testes com o modelo proposto, denominado *sparse kernel* (SPARK), nenhum método de esparsificação ou função *kernel* apresentou desempenho consistentemente superior em todos os conjuntos de dados, destacando que a eficácia do modelo depende das características do problema e requer seleção criteriosa de métodos, funções *kernel* e hiperparâmetros. Na maioria dos casos, o desempenho superou o do kernel-SOM, no qual a definição da quantidade de protótipos apresenta maior custo computacional. Por fim, considerando aplicações em que os dados chegam em fluxo contínuo, propõe-se um novo arcabouço teórico, denominado *sparse online kernel* (SPOK), para a construção de modelos adaptativos baseados em protótipos utilizando métodos de *kernel*. O modelo foi avaliado na classificação de dados sintéticos e reais em fluxo contínuo. Os resultados evidenciaram que, mesmo utilizando apenas protótipos e estratégias de K-vizinhos mais próximos, o modelo atinge, com poucos protótipos, desempenho comparável ao estado da arte em classificadores de fluxo contínuo, os quais são, geralmente, baseados em janelamento, comitês ou árvores de decisão.

Palavras-chave: Classificação; Modelos baseados em protótipos; Métodos de *kernel*; Métodos de esparsificação; Fluxo contínuo de dados; Modelos adaptativos

ABSTRACT

The main purpose of this thesis is to contribute to the development of prototype-based classifiers using *kernel* methods. Initially, the application of the *kernel trick* to the artificial neural network self-organizing map (SOM) is investigated for the classification of batch datasets (in which all training data are simultaneously available), also verifying the influence of different kernel functions and labeling methods. The results of these initial experiments showed that using *kernel* functions other than the commonly employed ones (Gaussian and Linear) can lead to superior performance. Furthermore, it was verified that, in the analyzed datasets, there are advantages in computing both the distance measure and the definition of the winning prototype in the feature space. Then, considering that defining the number of prototypes is very important for the accuracy of prototype-based classifiers, this work proposes the use of sparsification methods, such as *approximate linear dependence* (ALD), novelty, surprise and coherence, for the automatic selection of this quantity, employing the weighted K-nearest neighbors algorithm as the classifier. These methods are compared with respect to their influence on classifier accuracy and on the number of prototypes selected. In the experiments carried out with the proposed model, here referred to as *sparse kernel* (SPARK), no sparsification method or kernel function consistently outperformed the others across all datasets. These findings highlight that the effectiveness of the model depends on the dataset characteristics, reinforcing the need for a careful selection of sparsification methods, *kernel* functions, and hyperparameters adapted to each specific task. Moreover, in most cases, the performance was superior to that of the kernel-SOM algorithm, in which the definition of the number of prototypes implies a higher computational cost. Finally, in many recent applications, data may be provided as a continuous stream. In this context, a new theoretical framework, here referred to as *sparse online kernel* (SPOK), is proposed for building adaptive prototype-based models using kernel methods. The proposed model was evaluated in the classification of synthetic and real data streams. The results showed that, even relying solely on prototypes and K-nearest neighbor strategies, the model achieved, with only a few prototypes, performance comparable to the state-of-the-art in data stream classifiers, which are often based on data windowing, ensembles or decision trees.

Keywords: Classification; Prototype-based Models; Kernel Methods; Sparsification Methods; Data Streaming; Adaptive Models.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AD	average distance
ALD	aproximate linear dependency
ARF	adaptive random forest
ART	adaptive resonance theory
DACC	dynamic adaption to concept changes
DWKNN	distance weighted K-nearest neighbors
EF-KSOM	energy function based kernel SOM
GD-KSOM	gradient descent based kernel SOM
GNG	growing neural gas
GPR	Gaussian process regression
KAF	kernel adaptive filtering
KKM	kernel K-means
KLVQ	kernel learning vector quantization
KM	K-means
KNG	kernel neural gas
KNN	K-nearest neighbors
KNNC	kernel nearest neighbor cassifier
KNNC-ALD	kernel nearest neighbor classifier via ALD criterion
KNN_s	KNN with a sliding window
KQD	kernel quadratic discriminant
KSOM	kernel self-organizing map
KWKNN	kernel weighted K-nearest neighbors
LSSVM	least squares support vector machine
LVGB	leveraging bagging
LVQ	learning vector quantization
MD	minimum distance

MV	majority voting
NG	neural gas
NLL	negative log likeklihood
NNC	nearest neighbor classifier
NPC	nearest prototype classifier
NSE	nonstationary environments
OLS	ordinary least squares
OSVM	online support vector machine
PAW-KNN	probabilistic adaptive windowing K-nearest neighbors
PBC	prototype-based classifiers
RAN	resource-allocating network
RKHS	reproducing kernel hilbert space
ROC	receiver operating characteristic
SAM-KNN	self adjusting memory K-nearest neighbors
SOM	self-organizing map
SPARK	sparse kernel
SPOK	sparse online kernel
WKNN	weighted K-nearest neighbors
WTA	winner takes all

LIST OF SYMBOLS

$\mathbb{R}$	Real numbers
$\mathbb{C}$	Complex numbers
$\mathscr{A}$	Discrete Input Space
$\mathscr{X}$	Continuous Input Space
$\mathscr{F}$	High-dimensional Reproducing Kernel Hilbert Space
$\mathscr{O}(.)$	Computational complexity
T	Transpose
$\dagger$	Pseudo-inverse
$'$	Scalar derivate
∇	Vectorial derivate
∂	Partial derivate
N	Total number of samples
P	Number of attributes
C	Number of classes
Q	Number of prototypes / Dictionary Size
K	Number of neighbors (for classification)
i, j	Iteration variables
n	Sample subscript
p	Attribute subscript
q	Prototype subscript
k	Neighbor subscript
t	Discrete time step
t_{max}	Maximum Number of Discrete Time Steps
$\mathbf{W}$	Matrix of prototypes' weights
$\mathbf{w}_q$	Prototype's weights vector
$\tilde{x}_q$	Model's input vector

$\tilde{c}_q$	Model's class label (of samples or prototypes)
$\mathscr{D}$	Dictionary
$\mathbf{S}$	Covariance Matrix
$\mathbf{A}$	Symmetric and Positive-definite Matrix
$\mathbf{I}$	Identity matrix
$\mathbf{B}$	Squared matrix
$\mathbf{u}, \mathbf{m}$	Column Vectors
M	Minkowski norm degree
$d(\cdot,\cdot)$	Distance measure
d^+	Distance, to the sample, from the closest prototype of class c
d^-	Distance, to the sample, from the closest prototype of all classes but c
$\|.\|_2$	Euclidean norm (distance)
$\|.\|_M$	Minkowski norm (distance)
$f(\cdot)$	Arbitrary function
$\phi(\cdot)$	Nonlinear mapping function
$\kappa(\cdot,\cdot)$	Kernel Function
$q^*(\cdot)$	Winner prototype argument
$\Theta(\cdot,\cdot)$	Binary loss function
$I(\cdot,\cdot)$	Identity binary function
$\rho(\cdot)$	Probability
$\mathbf{x}_n$	Sample's input vector
c_n	Sample's class label/output
$\hat{c}_n$	Classifier's class label/output predciton
$\mathbf{y}_n$	Sample's encoded class label/output
$\hat{\mathbf{y}}_n$	Classifier's encoded class label/output predciton
$c(\mathbf{w}_{q^*}(t))$	Winning prototype class label
$\mathbf{y}(\mathbf{w}_q)$	Encoded prototype class label
N_{tr}	Number of training samples

N_{ts}	Number of test samples
N_{ep}	Number of epochs
N_r	Number of realizations of an experiment
N_t	Number of trials of random search
N_p	Percentage of the data used for training
N_f	Number of folds for cross-validation
H	System Model
E_b	Batch classification error measure
E_s	Sequential classification error measure
$h(\cdot,\cdot,\cdot)$	Neighborhood function
$\eta(.)$	Learning Rate
k_{nn}	Kernel of a sample and itself
$\mathbf{k}_{n-1}$	Kernel vector
$\mathbf{K}$	Kernel matrix
$\tilde{\mathbf{K}}$	Kernel matrix of dictionary
$\dot{\mathbf{K}}$	Regularized kernel matrix
$\mathbf{a}_t$	ALD coefficients
J_q	Cost function of prototype q
J_{PB}	Cost function of a prototype-based classifier
λ	Weighting factor (for hyperparameter optimization)
μ_z	Empirical Mean (for z-score)
σ_z	Empirical Standard Deviation (for z-score)
σ_r	Regularization parameter
σ_h	Neighborhood function influence
$\mathbf{r}_q(t)$	Prototype's geometrical position
$\mathbf{r}_{q^*}(t)$	Winning prototype's geometrical position
N_h	Neighborhood layer size
v_0	Initial value of σ_h

v_f	Final value of σ_h
σ_κ	Kernel function's hyperparameter
l	Kernel function's hyperparameter
θ	Kernel function's hyperparameter
α	Kernel function's hyperparameter
δ_1, δ_2	Sparsification values
ν_1, ν_2	Sparsification levels
ψ	Coherence parameter
S_t	Surprise Measure
σ_t	Prediction variance
σ_n	Variation of the noise contained in an observation
$\mathbf{X}$	Set of input variables
$\mathbf{c}$	Set of target variables
$\rho(\mathbf{X}, \mathbf{c})$	Joint distribution between inputs and outputs
$\rho(\mathbf{X})$	Distribution of the input data
$\rho(\mathbf{c}\|\mathbf{X})$	Conditional probability of the output
β	Maximum number of prototypes
$\mathbf{s}$	score vector
s_i	score
ε	minimum score
z_i	last prediction of prototype i was correct
$\mathbf{z}$	vector of z_i variables
$\mathscr{W}_s$	Sliding Window sizes

CONTENTS

1 INTRODUCTION

Prototype-based models in machine learning make use of a number of appealing concepts, such as the explicit representation of observations in terms of memorized exemplars and the comparison of observations with this reference set of exemplars in terms of similarity (BIEHL *et al.*, 2016). Some examples of prototype-based training algorithms are the K-means (KM) (MACQUEEN, 1967), the neural gas (NG) (MARTINETZ *et al.*, 1993), the learning vector quantization (LVQ) (KOHONEN, 1990a) and the Kohonen's self-organizing map (SOM) (KOHONEN, 1990b). They have been applied to supervised and unsupervised machine learning problems, such as condition monitoring of induction motors (SOUSA *et al.*, 2019), financial time series forecasting (BAN *et al.*, 2013), image compression (KRISHNA *et al.*, 1997) and detection of geochemical anomaly patterns (BIGDELI *et al.*, 2022).

Of great interest to this work, tasks in industry, business, medicine and science can be modeled as classification problems. Examples include bankruptcy prediction, credit scoring, medical diagnosis, quality control, handwritten character recognition, and speech recognition (ZHANG, 2000). It is important to mention that some nonlinear features may be embedded in these problems. To model them, kernel-based methods have been introduced (JäKEL *et al.*, 2007). The underlying idea of these methods is to apply a kernel function $\kappa(\cdot,\cdot) : \mathscr{X} \times \mathscr{X} \to \mathbb{R}$ to any pair of training vectors so that the result can be interpreted as an inner product between two vectors $\phi(\mathbf{x}_i)$ and $\phi(\mathbf{x}_j)$, where $\phi : \mathscr{X} \to \mathscr{F}$, and $\mathscr{F}$ is a high-dimensional (possibly infinite dimensional) feature space (YIN, 2006): $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$. Since $\phi(\cdot)$ is usually unknown, inner products in the feature space are computed through the kernel function without using the feature vectors $\phi(\mathbf{x}_i)$ and $\phi(\mathbf{x}_j)$ directly. This property of kernel methods is known as the *kernel trick* (VALYON, 2006). The process of *kernelization* has been applied to prototype-based algorithms, producing their kernelized versions, such as the kernel K-means (KKM) (ZHANG; RUDNICKY, 2002), the kernel self-organizing map (KSOM) (LAU *et al.*, 2006), the kernel neural gas (KNG) (QIN; SUGANTHAN, 2004a) and the kernel learning vector quantization (KLVQ) (HOFMANN; HAMMER, 2013).

Finally, in many modern applications, the data are not completely available for offline processing, but rather it comes in the form of streams. As examples, one can mention health care (SPANGENBERG *et al.*, 2017), behavior recognition (CHUA *et al.*, 2011), structural health monitoring (LI; YU, 2015) and Internet of Things (ALIYU *et al.*, 2018). Other examples, such as internet business and social media, lead to vast amounts of data created every second in tweets,

click-streams, or log files (AUGENSTEIN *et al.*, 2017). These kinds of applications bring some issues that need to be tackled. Firstly, keeping large volumes of streaming data in a hardware memory is often infeasible. Hence, online processing is required so that models can immediately retain as much information as possible from the incoming input-output sample and then discard it. Furthermore, data independence and stationarity assumptions seem unrealistic because changes in underlying data distribution can occur, for example, due to changes in populations or changing personal interests (ŽLIOBAITė *et al.*, 2015).

These restrictions have motivated the development of many methods for online processing. Some of them, such as the online support vector machine (OSVM) (LI; YU, 2015) and the adaptive random forest (ARF) (GOMES *et al.*, 2017b), are adaptations of batch algorithms, while some others are based on adaptive windows (BIFET; GAVALDA, 2007) (BIFET *et al.*, 2013), short-term and long-term memories (LOSING *et al.*, 2016), or ensembles (GOMES *et al.*, 2017a). Prototype-based classifiers (PBC) have also been used for streaming data processing, achieving competitive results compared to state-of-the-art stream classifiers in terms of time and memory complexity, as well as accuracy (LOSING *et al.*, 2015) (HEUSINGER *et al.*, 2020). Nevertheless, performances of PBC are highly dependent on the number of labeled prototypes. In scenarios where all the data are available for offline processing, a set of prototypes can be either optimally determined using evolutionary algorithms, such as in Soares Filho e Barreto (2014), or added/removed adaptively (ALBUQUERQUE *et al.*, 2018). However, in most applications, that number is set by trial and error or exhaustive grid search. For streaming processing, a PBC model must be capable of updating continuously the number of prototypes, on the fly, according to the demands of the task. An alternative approach to address this problem comes from the field of kernel adaptive filtering (KAF), where sparsification methods have been developed to control model complexity by verifying whether a given sample is important or redundant to the existing model (LIU *et al.*, 2009).

All the previous topics led to the following research question: Is it possible to solve complex classification problems through sparse prototype-based models that can be applied to batch and online data processing?

1.1 Goals

The main goal of this thesis is to develop sparse kernel prototype-based models to efficiently handle classification tasks in batch and online learning scenarios.

1.1.1 Specific Goals

To achieve the overall aim of this research, the following specific goals were established:

- Investigate the use of kernel prototype-based models for batch learning in classification problems;
- Evaluate the impact of the chosen kernel functions in the performance of prototype-based classification;
- Analyze different sparsification procedures in the performance of prototype-based classification models;
- Develop new and/or extend current kernel prototype-based classifiers for online learning tasks, such as streaming data classification.

1.2 Scientific production

The partial results of this doctorate thesis generated the following publications.

- Coelho, D. N.; Barreto, G. A.; Medeiros, C. M. Detection of Short Circuit Faults in 3-Phase Converter-Fed Induction Motors Using Kernel SOMs. In: *2017 12th International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM)*. IEEE, 2017. p. 1–7. Available at: https://ieeexplore.ieee.org/abstract/document/8020016. DOI: 10.1109/WSOM.2017.8020016.

- Coelho, D. N.; Barreto, G. A. Approximate Linear Dependence as a Design Method for Kernel Prototype-Based Classifiers. In: *International Workshop on Self-Organizing Maps*. Springer, Cham, 2019. p. 241–250. Available at: https://link.springer.com/chapter/10.1007/978-3-030-19642-4_24. DOI: 10.1007/978-3-030-19642-4_24.

- Coelho, D. N.; Barreto, G. A. A Sparse Online Approach for Streaming Data Classification via Prototype-Based Kernel Models. *Neural Processing Letters*, 2022, p. 1–28. Available at: https://link.springer.com/article/10.1007/s11063-021-10701-9. DOI: 10.1007/s11063-021-10701-9.

In addition to these articles, the author of this thesis has also co-authored the following related articles:

- Freitas, D. C. C.; Braga, A. P. S.; Coelho, D. N.; Cavalcanti Neto, E.; Silva, H. S. Análise de Técnicas para Aproximação da Curva de Descarga de Baterias. In: Congresso Brasileiro de

Automática, 2018, João Pessoa. Anais do Congresso Brasileiro de Automática. Sociedade Brasileira de Automática (SBA), 2020. Available at: https://www.sba.org.br/open_journal_systems/index.php/cba/article/view/379. Accessed on: 24 jun. 2025.

– Bessa, R.; Barreto, G. A.; Coelho, D. N.; De Moura, E. P.; Murta, R. H. F. On Least Squares Support Vector Regression for Predicting Mechanical Properties of Steel Rebars. *Metals*, v. 14, p. 695, 2024. Available at: https://www.mdpi.com/2075-4701/14/6/695. DOI: 10.3390/met14060695.

1.3 Thesis Organization

The remaining text is divided according to the contributions achieved throughout this doctorate research. Each chapter highlights either theory, results, and/or discussions on a specific topic.

In Chapter 2, the theory of prototype and *kernel* based models are presented, especially in solving classification problems.

In Chapter 3, the results of applying kernel functions to the SOM algorithm in the task of classification problems is evaluated. The batch learning mode and the corresponding experiments are detailed.

In Chapter 4, the aproximate linear dependency (ALD) is used as a design method for kernel prototype-based classifiers. The algorithm's characteristics are discussed and the 1-nearest neighbor is used as a strategy for classification. Motivated by these initial results, the author evaluates other sparsification methods for the active selection of prototypes. As a consequence of the studies carried out in this chapter, a new framework, called SPARK, is proposed. Thus, the quantity of prototypes and its classification accuracy are analysed, taking into account different kernel functions and sparsification methods. Also, the distance weighted kernel K-nearest neighbor is used as an strategy for classification.

In Chapter 5, the online processing of streaming data is discussed, and a new algorithm, called SPOK-NN is proposed. Its formalization and application to simulated and real world online classification tasks are presented, comparing its results to those of other state-of-the-art algorithms. As in Chapter 4, the use of different kernel functions and sparsification methods are analyzed.

Finally, in Chapter 6, final considerations and perspectives for future work are made.

2 PROTOTYPE-BASED CLASSIFIERS: FROM LINEAR TO KERNEL-BASED APPROACHES

In this chapter, the theory of kernel prototype-based classifiers is introduced. Initially, it is important to define some terms that will be used from now on, such as:

- Sample: an input-output pair describing an object to be classified.
- Class: a set of similar objects.
- Attribute: a characteristic (feature) of an object that serves as an input to the classifier.
- Metaparameter: an experiment variable that must be defined before hyperparameters optimization step.
- Hyperparameter: a variable of the classifier which must be defined before training and should be optimized.
- Parameter: a variable that is adjusted during training so the classifier can adapt to the problem it is intended to solve.
- Training: stage of the model building in which the classifiers' parameters are adjusted.
- Training Iteration: a single presentation of a training sample used to update the classifier's parameters.
- Training Epoch: one complete pass through the entire training data set, with each sample presented once to update the classifier's parameters.
- Test: stage of the model building in which the trained classifier is applied to previously unseen data, and performance measures are computed.
- Training/testing realization: a full cycle in which training is completed and the classifier is evaluated on test data.
- Generatilization: it refers to the classifier's ability to perform well on new, unseen data. A classifier with high generalization capacity is able to correctly classify data points that were not part of the training set, thus ensuring its effectiveness and robustness when applied to real-world scenarios.

The classification problem will be defined from the previous terms.

2.1 The classification Problem

Formally, in the context of classification, which is a supervised learning task, a dataset is given as a group of tuples (samples) $\{(\mathbf{x}_n, c_n) \in \mathbb{R}^p \times \{1,\ldots,C\}\}_{n=1}^{N}$, where $\mathbf{x}_n$ is the n-th input feature vector, c_n is its respective n-th output class label, and C is the finite and

predefined number of classes ($C \ll N$).

Depending on the problem, there are several ways of encoding class labels. In this thesis, a column vector $\mathbf{y}_n \in \mathbb{R}^C$ is represented by means of the one-hot encoding scheme. As such, the component of the output vector referring to the class to which the input data belongs has the value $+1$, while the other components have the value -1. As an example, if the problem has three classes, and the current sample belongs to class number two, its label is $c_n = 2$ and its encoded label will be $\mathbf{y}_n = [-1, +1, -1]^T$. In this sense, a dataset is given as a collection of input-output pairs $\{(\mathbf{x}_n, \mathbf{y}_n) \in \mathbb{R}^P \times \mathbb{R}^C\}_{n=1}^N$

In batch learning, the dataset of N samples is previously stored and can be divided into two disjoint subsets: the training set (with N_{tr} samples) and the test set (with N_{ts} samples). The classifier model H is built from the training set, and the test set is used to validate it (LOSING *et al.*, 2018). Several training-testing strategies can be used to evaluate the models, such as the k-fold cross-validation, in which the dataset is divided in k folds and all but one fold are used for training, while the remaining one is used for testing. The process is repeated k times until each one of the k-folds is used for testing.

To evaluate the generalization ability of the trained models to unseen samples, one can use the classification batch error measure, which is given by

$$E_b = \frac{1}{N_{ts}} \sum_{n=1}^{N_{ts}} \Theta(c_n, \hat{c}_n), \tag{2.1}$$

where $\hat{c}_n = H(\mathbf{x}_n)$ is the classifier's output prediction, and the function $\Theta(.,.)$ is either equal to 0, if $c_n = \hat{c}_n$, or 1, otherwise. Like c_n, the label prediction $\hat{c}_n$ can also be encoded. This topic will be further discussed in Section 2.6.

2.2 Competitive Learning Algorithms

Within the field of artificial neural networks, prototype-based algorithms are also called competitive learning algorithms (BIEHL *et al.*, 2016; KOHONEN, 2013). Models and algorithms based on the principle of competitive learning include the NG and its variants (MARTINETZ *et al.*, 1993), the family of LVQ algorithms (KOHONEN, 1990a) and the Kohonen's SOM (KOHONEN, 1990b).

Competitive learning algorithms learn from data a mapping from a continuous input space $\mathscr{X}$ onto a discrete set $\mathscr{A}$ of Q prototypes. The map $q^*(\cdot) : \mathscr{X} \to \mathscr{A}$, defined by the set of weight vectors $\mathbf{W} \in \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_Q\}, \mathbf{w}_q \in \mathbb{R}^P \subset \mathscr{X}$, assigns to each input vector

$\mathbf{x}(t) \in \mathbb{R}^P \subset \mathscr{X}$ a winning prototype $q^*(t) \in \mathscr{A}$, determined by a measure of (dis)similarity such as

$$q^*(t) = \arg\min_{\forall q} \|\mathbf{x}(t) - \mathbf{w}_q(t)\|_2^2, \tag{2.2}$$

where $\|\cdot\|_2$ denotes the Euclidean distance and t symbolizes a discrete time step associated with the iterations of the algorithm. The Euclidean distance and other measures of (dis)similarity are detailed in Appendix A.

Competitive learning algorithms are distinguished, in part, by the updating rules of their prototypes. By means of the winner takes all (WTA) approach, at each iteration of the algorithm, only the winning prototype is updated as

$$\mathbf{w}_{q^*}(t+1) = \mathbf{w}_{q^*}(t) + \eta(t)[\mathbf{x}(t) - \mathbf{w}_{q^*}(t)], \tag{2.3}$$

where $0 < \eta(t) < 1$ is the learning rate. For the SOM network (KOHONEN, 1990b), another unsupervised algorithm, all the prototypes are updated by the rule

$$\mathbf{w}_q(t+1) = \mathbf{w}_q(t) + \eta(t)h(q^*,q,t)[\mathbf{x}(t) - \mathbf{w}_q(t)], \tag{2.4}$$

where $h(q^*,q,t)$ is a neighborhood function which defines the region of influence around the winning prototype. With the LVQ1 classifier (KOHONEN, 1990a), a supervised algorithm, only the winning prototype is updated according to the rule

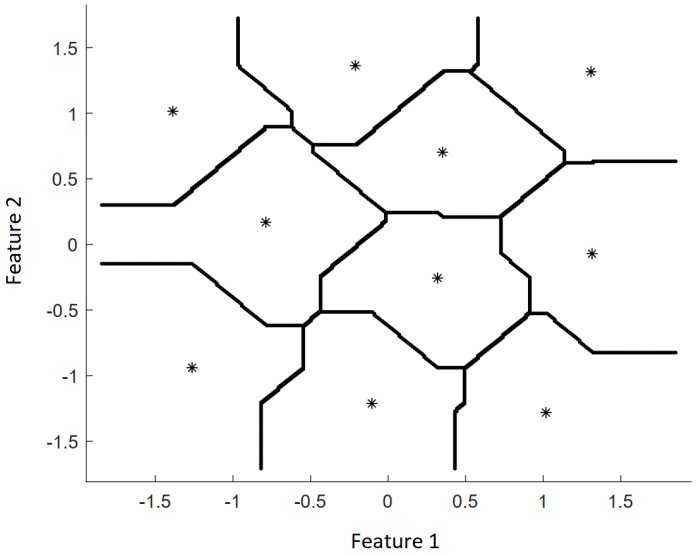
$$\mathbf{w}_{q^*}(t+1) = \begin{cases} \mathbf{w}_{q^*}(t) + \eta(t)\left[\mathbf{x}(t) - \mathbf{w}_{q^*}(t)\right], \text{ if } c_t = c(\mathbf{w}_{q^*}(t)) \\ \mathbf{w}_{q^*}(t) - \eta(t)\left[\mathbf{x}(t) - \mathbf{w}_{q^*}(t)\right], \text{ if } c_t \neq c(\mathbf{w}_{q^*}(t)), \end{cases} \tag{2.5}$$

where c_t is the sample's class label and $c(\mathbf{w}_{q^*}(t))$ is the winning prototype class label.

After the training phase of a prototype-based algorithm, there is an associated convex region for each prototype, called Voronoi cell, so that any sample within this region is closer to that prototype than to the other prototypes. The set of all Voronoi cells forms a Voronoi diagram (which is related to the Dirichlet tessellation) (AURENHAMMER, 1991). In Figures 1 and 2, two dimensional Voronoi diagrams are shown. In these diagrams, the symbol "$*$" represents a prototype, and the Voronoi cells boundaries represent the points that are equally distant from two or more prototypes. It is important to mention that, in both figures, the prototypes have the same position at the plane, but the use of different distance measures leads to distinct Voronoi cell boundaries (LEE, 1980).

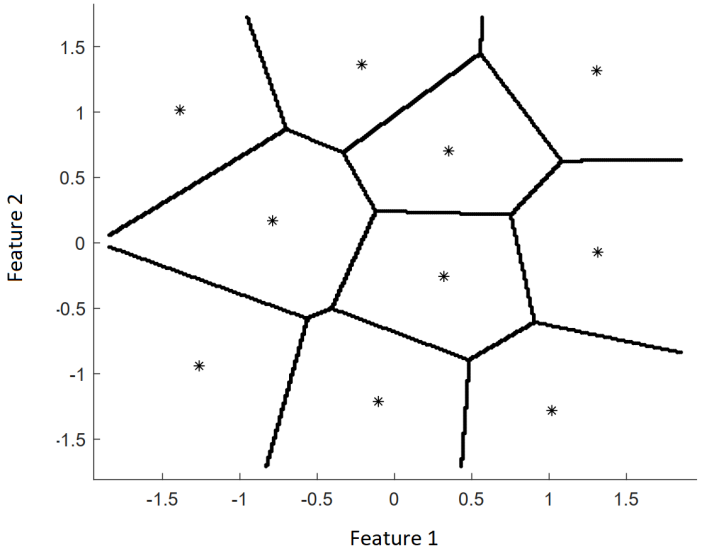
Unsupervised prototype-based models can be directly applied to clustering and vector quantization problems. In contrast, supervised prototype-based models are suitable for

Figure 1 – Voronoi Diagram using the Manhattan Distance.



Source: Author

Figure 2 – Voronoi Diagram using the Euclidean Distance.



Source: Author

classification tasks, although the labels must be assigned to each prototype. When unsupervised prototype-based models are employed in classification, a post-training strategy must be used to assign a class label to each prototype. Three of these strategies are shown in the next section.

2.3 Prototype Labeling Strategies

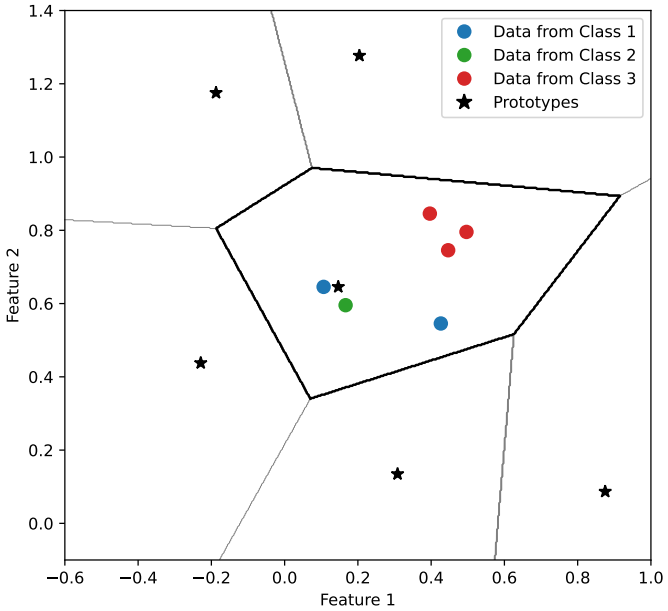
As previously mentioned, the unsupervised learning algorithms require post-training prototype labeling strategies in order to be applied to pattern classification tasks. During the

prototype labeling phase, training data samples are presented to the algorithm once more, but the prototypes' weights are not updated. Once this phase is completed, the algorithm becomes a prototype-based classifier (BIEHL *et al.*, 2016; MATTOS; BARRETO, 2011). Basically, all prototype labeling strategies aim at associating to each prototype a class label, so that some rule, such as the nearest prototype rule in Eq. (2.2), is used for pattern classification purposes.

Three labeling methods are used in this thesis, namely: the minimum distance (MD), the average distance (AD) and the majority voting (MV) method. The MD method, which is the simplest, assigns to each prototype the label of its nearest training sample. In the AD method, one first needs to compute the distances from a given prototype to the samples of all classes which are mapped to this prototype, then compute the average distance per class. Finally, the prototype receives the label of the class whose associated average distance is the smallest one. In the MV method, each prototype inherits the most frequent class label among the labels of the samples mapped to it.

Figure 3 shows a Voronoi diagram after the training phase. In the highlighted Voronoi cell, there are six training samples: two from class 1 (blue), one from class 2 (green), and three from class 3 (red). If the minimum distance method is used, the class label assigned to the prototype would be class 1. Using the average distance criterion, class 2 would be assigned. Finally, with the majority voting method, the prototype would receive the label of class 3.

Figure 3 – Illustration of the labeling methods used in this work.



Source: Author.

An advantage of using unsupervised trained models (such as the SOM) with post-training labeling strategies, is that one just needs to define the total number of prototypes for the model. On the downside, there can be classes without prototypes. In supervised models, in which the prototypes are already labeled before the training phase, one needs to define the number of prototypes for each class.

Finally, it should be noted that competitive learning classifiers, such as SOM- or LVQ-based, generate piecewise linear decision boundaries. Although each local separation corresponds to a linear segment between Voronoi regions, the overall decision surface is generally nonlinear and may exhibit a complex geometry. Kernelized versions of these prototype-based classifiers provide an additional level of flexibility by replacing the standard Euclidean metric with other similarity functions defined in high-dimensional feature spaces.

2.4 Kernelizing Prototype-based Models

Using the kernel trick, the search for the winning prototype, as originally shown in Eq. (2.2), becomes:

$$\begin{aligned} q^*(t) &= \arg\min_{\forall q} \left\| \phi\left(\mathbf{x}(t)\right) - \phi\left(\mathbf{w}_q(t)\right) \right\|_2^2, \\ &= \arg\min_{\forall q} J_q(\mathbf{x}(t)), \end{aligned} \tag{2.6}$$

where $J_q(\mathbf{x}(t))$ is a cost function. Using some linear algebra and the kernel trick, this expression can be expanded as

$$\begin{aligned} J_q(\mathbf{x}(t)) &= \left\| \phi(\mathbf{x}(t)) - \phi(\mathbf{w}_q(t)) \right\|_2^2, \\ &= (\phi(\mathbf{x}(t)) - \phi(\mathbf{w}_q(t)))^T (\phi(\mathbf{x}(t)) - \phi(\mathbf{w}_q(t))), \\ &= \phi(\mathbf{x}(t))^T \phi(\mathbf{x}(t)) + \phi(\mathbf{w}_q(t))^T \phi(\mathbf{w}_q(t)) - 2\phi(\mathbf{x}(t))^T \phi(\mathbf{w}_q(t)), \\ &= k(\mathbf{x}(t), \mathbf{x}(t)) + k(\mathbf{w}_q(t), \mathbf{w}_q(t)) - 2k(\mathbf{x}(t), \mathbf{w}_q(t)). \end{aligned} \tag{2.7}$$

By the same token, prototype updating rules can also be kernelized. For example, the gradient descent based kernel SOM (GD-KSOM) (ANDRAS, 2002) and the energy function based kernel SOM (EF-KSOM) (LAU *et al.*, 2006) have the following prototype updating rule:

$$\mathbf{w}_q(t+1) = \mathbf{w}_q(t) - \eta(t) h(q^*, q, t) \nabla J_q(\mathbf{x}(t)), \tag{2.8}$$

where the general expression for the gradient vector $\nabla J_q(\mathbf{x}(t))$ is given by

$$\nabla J_q(\mathbf{x}(t)) = \frac{\partial J_q(\mathbf{x}(t))}{\partial \mathbf{w}_q(t)} = \frac{\partial \kappa(\mathbf{w}_q(t), \mathbf{w}_q(t))}{\partial \mathbf{w}_q(t)} - 2\frac{\partial \kappa(\mathbf{w}_q(t), \mathbf{x}(t))}{\partial \mathbf{w}_q(t)}. \tag{2.9}$$

Its important to mention that, the only difference between GD-KSOM and EF-KSOM is in the way these algorithms select the winning prototype. The former executes this operation in the input space (Eq. (2.2)), while the latter executes it in the feature space (Eq. (2.6)).

Finally, different kernel functions lead to different variants of the prototype updating rule in Eq. (2.8) and to different kernelized selection of winning prototypes in Eq. (2.6). Some of these kernel functions are briefly described in the next subsection.

2.4.1 Kernel Functions

The linear kernel is the simplest one, where the output of the kernel function is equal to the dot product of two input vectors plus a hyperparameter $\theta \in \mathbb{R}$. This kernel, for two given vectors, $\mathbf{x}_i \in \mathbb{R}^p$ and $\mathbf{x}_j \in \mathbb{R}^p$, can be formally defined as

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j + \theta, \tag{2.10}$$

where θ is a constant bias parameter. If θ is set to 0, the kernelized version of an algorithm reduces to its linear form.

The Gaussian kernel function has the general form

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|_2^2}{2\sigma_\kappa^2}\right), \tag{2.11}$$

where $\sigma_\kappa \in \mathbb{R}^+$ is a scale parameter (a.k.a. the width parameter). A suitable value for the hyperparameter σ_κ should be carefully tuned to the problem at hand (HARKAT *et al.*, 2020). If overestimated, the exponential behaves almost linearly and the projection to the high-dimensional feature space loses its nonlinear nature. If underestimated, the function lacks regularization and the decision boundaries tend to become highly sensitive to noise in the training data.

The exponential kernel is very similar to the Gaussian kernel. The main difference is that neither the scale parameter σ_κ nor the Euclidean norm is squared:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|_2}{\sigma_\kappa}\right). \tag{2.12}$$

Another widely used kernel function is the polynomial one. In its simplest form $k(\mathbf{x}_i, \mathbf{x}_j) = \left(\mathbf{x}_i^T \mathbf{x}_j + 1\right)^l$, the hyperparameter l is an integer degree. However, this function can also have a fractional degree (ROSSIUS *et al.*, 1998), and two additional hyperparameters can be added:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \left(\alpha \mathbf{x}_i^T \mathbf{x}_j + \theta\right)^l, \tag{2.13}$$

resulting in a function with more degrees of freedom to build a nonlinear mapping.

The Cauchy kernel function has the following general form:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \left(1 + \frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|_2^2}{\sigma_\kappa^2}\right)^{-1}, \tag{2.14}$$

where $\sigma_\kappa \in \mathbb{R}^+$ is a scale parameter. This kernel function is a long-tailed kernel, a term borrowed from probability to denote distributions in which extremely small or large values have a relatively high probability of occurring, in contrast to the Gaussian distribution, where values far from the mean are rare. For this reason, the Cauchy kernel can be used to provide long-range influence and sensitivity over the high-dimensional feature space (DE SOUZA, 2010).

The Log kernel function was introduced in Boughorbel *et al.* (2005), and its expression is given by

$$k(\mathbf{x}_i, \mathbf{x}_j) = -\log\left(1 + \frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|_2^l}{\sigma_\kappa^2}\right), \tag{2.15}$$

where log denotes the natural logarithm. The Log kernel function belongs to a class of kernel functions that are not strictly positive definite, known as *conditionally positive definite* kernel functions[1], which have been shown to perform very well in practical applications (PONTE, 2020).

The sigmoid function is a class of functions that is widely used as activation functions in artificial neural networks, which has also been used as a kernel function (BISHOP, 2006; CARRINGTON *et al.*, 2014). This kernel has its equation

$$k(\mathbf{x}_i, \mathbf{x}_j) = \tanh\left(\alpha \mathbf{x}_i^T \mathbf{x}_j + \theta\right) \tag{2.16}$$

based on the hyperbolic tangent, where $\alpha \in \mathbb{R}^+$ is a horizontal scaling parameter and $\theta \in \mathbb{R}$ is a central vertical bias (CARRINGTON *et al.*, 2014).

Finally, the KMOD (Kernel with MOderate Decreasing) (AYAT *et al.*, 2002) is characterized by a fast decay of the image of the original points near the origin and a moderate decay towards infinity. These characteristics allow quite distant input vectors to still be considered, while maintaining proximity information (ROCHA NETO, 2011). The general equation for the KMOD kernel is

$$k(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{\exp\left(\theta/\sigma_\kappa^2\right) - 1}\left[\exp\left(\frac{\theta}{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2 + \sigma_\kappa^2}\right) - 1\right]. \tag{2.17}$$

[1] Let $\mathscr{X}$ be a nonempty set. A kernel $\kappa(\cdot,\cdot)$ is called *conditionally positive definite* if and only if it is symmetric and $\sum_{i,j}^{N} a_i a_j \kappa(\mathbf{x}_i, \mathbf{x}_j) \geq 0$, for $N \geq 1$, $a_1, \ldots, a_n \in \mathbb{R}$ with $\sum_{i=1}^{N} a_i = 0$, and $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathscr{X}$.

Table 1 – Common kernel functions and their hyperparameters.

Name	$\kappa(\mathbf{x},\mathbf{y})$	**Hyperparameters**
Linear Kernel	$\langle \mathbf{x}_i, \mathbf{x}_j \rangle + \theta = \mathbf{x}_i^T \mathbf{x}_j + \theta$	θ
Gaussian Kernel	$\exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma_\kappa^2}\right)$	σ_κ
Exponential Kernel	$\exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2}{\sigma_\kappa}\right)$	σ_κ
Polynomial Kernel	$\left(\alpha.\langle \mathbf{x}_i, \mathbf{x}_j \rangle + \theta\right)^l = \left(\alpha \mathbf{x}_i^T \mathbf{x}_j + \theta\right)^l$	α, θ, l
Cauchy Kernel	$\left(1 + \frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{\sigma_\kappa^2}\right)^{-1}$	σ_κ
Log Kernel	$-\log\left(1 + \frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^l}{\sigma_\kappa^2}\right)$	σ_κ, l
Sigmoid Kernel	$\tanh\left(\alpha \langle \mathbf{x}_i, \mathbf{x}_j \rangle + \theta\right) = \tanh\left(\alpha \mathbf{x}_i^T \mathbf{x}_j + \theta\right)$	α, θ
Kmod Kernel	$\frac{1}{\exp(\theta/\sigma_\kappa^2)-1}\left[\exp\left(\frac{\theta}{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 + \sigma_\kappa^2}\right) - 1\right]$	θ, σ_κ

All these kernel functions and their corresponding hyperparameters are summarized in Table 1. The equations for $J_q(\mathbf{x}(t))$ and $\nabla J_q(\mathbf{x}(t))$, for each kernel, are developed in Appendix B.

The next subsection provides a detailed description of the K-nearest neighbors framework.

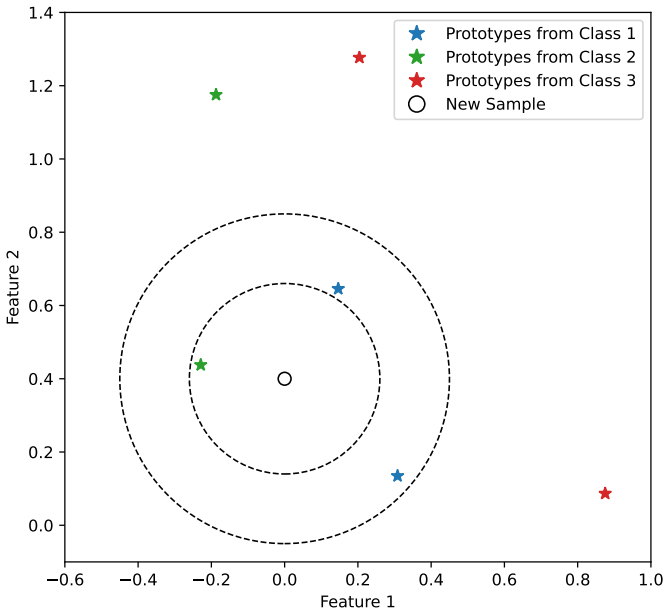
2.5 Classification using Prototype-based Models

K-nearest neighbors (KNN) methods represent one of the simplest and most intuitive nonparametric techniques in the field of statistical discrimination (FIX; HODGES, 1989; COVER; HART, 1967; ZHANG, 2022). These methods can be equally used for classification (DUDA *et al.*, 2001; SYRIOPOULOS *et al.*, 2023) and regression tasks (NADARAYA, 1964; WATSON, 1964; FRÍAS; MARTÍNEZ, 2025).

In its simplest version, when used for classification, considering just one neighbor, a new observation inherits the same label as the closest sample from a dictionary composed either of training samples or prototypes. This method is also known as the nearest neighbor classifier (NNC) or nearest prototype classifier (NPC) (BIEHL *et al.*, 2016). A first extension of this idea is to use more than one neighbor, resulting in the *K*-NN variant. In this case, not only the closest observation within the learning set is used for the sake of classification, but also the *K* nearest

ones (HECHENBICHLER; SCHLIEP, 2004). The decision is in favor of the class label, most of these neighbors belong to. The drawback of this extension is that K is a hyperparameter that must be selected (or optimized) before training, and different values of K for the same sample can lead to different classification outputs. Figure 4 illustrates a new sample to be classified. The dashed circles represent points that are equally distant from this new sample, and the colored asterisks represent prototypes from different classes. In this figure, for $K = 1$, the predicted label 'Class 2' will be chosen as the model's output, otherwise, if $K = 3$, the label 'Class 1' will be the classifier's prediction.

Figure 4 – Classification for different values of the hyperparameter K.



Source: Author.

More sophisticated methods can be derived from this majority voting KNN strategy. The weighted K-nearest neighbors (WKNN) (DUDANI, 1976; HECHENBICHLER; SCHLIEP, 2004) is based on the idea that prototypes closer to a new observation should get a higher weight in the class prediction of this observation.

The first step of this method is to calculate the distances $d(\cdot,\cdot) \in \mathbb{R}$ from the new sample to the model's prototypes. Then, one must hold the $K+1$ nearest neighbors and their distances from this sample. Next, these distances have to be transformed, according to an arbitrary function $f(.)$, into similarity measures which can be used as weights. This arbitrary function must satisfy the following properties:

– $f(d(\cdot,\cdot)) \geq 0$ for all $d(\cdot,\cdot)$;

- $f(d(\cdot,\cdot))$ gets its maximum for $d(\cdot,\cdot)=0$;
- $f(d(\cdot,\cdot))$ decreases monotonously for $d(\cdot,\cdot)\to\infty$.

The rectangular function (in which all distances have the same weight, turning the KNN into a special case of WKNN) and the triangular function, which can be defined as

$$f(d(\cdot,\cdot)) = \max\{0, 1-|d(\cdot,\cdot)|\}, \tag{2.18}$$

are examples of functions that can be used for the transformation[2]. An important step before using the triangular function is to standardize the prototypes distances based on the distance to the first neighbor $(K+1)$ that is not considered when calculating the final prediction. This standardization is performed for each of the K-nearest neighbors as follows:

$$d(\mathbf{x},\mathbf{w}_k) \leftarrow \frac{d(\mathbf{x},\mathbf{w}_k)}{d(\mathbf{x},\mathbf{w}_{K+1})}, \tag{2.19}$$

so that standardized distances always take values within the interval [0,1]. The WKNN algorithm can be summarized in the following steps:

1. Consider $H = \left\{(\mathbf{w}_q, \tilde{c}_q), q = 1,...,Q\right\}$ a model composed by a set of prototypes $\mathbf{w}_q$ and their labels $\tilde{c}_q$, and $\mathbf{x}(t)$ a new observation whose class label has to be predicted.
2. Find the $K+1$ nearest neighbors $\{\mathbf{w}_k\}_{k=1}^{K+1}$ to $\mathbf{x}(t)$ according to a distance measure $d(\mathbf{w}_q,\mathbf{x}(t))$.
3. Use the $(K+1)$-th neighbor to standardize the K smallest distances via Equation (2.19).
4. Use an arbitrary function $f(d(\cdot,\cdot))$ to transform the standardized distances in similarity measures.
5. As a prediction $\hat{c}_n$ for the observation's class, choose the class which shows the biggest similarity measure sum

$$\hat{c}_n = \max_{\forall c}\left(\sum_{k=1}^{K} f(d(\mathbf{w}_k,\mathbf{x}_n)).I(\tilde{c}_k,c)\right), \tag{2.20}$$

where the function $I(\tilde{c}_k,c)$ is either equal to 1, if $\tilde{c}_k = c$, or 0, otherwise.

Kernel methods can be also applied to the WKNN, building the kernel weighted K-nearest neighbors (KWKNN) (RUBIO *et al.*, 2010). In this strategy, the kernelized squared Euclidean distance, defined in Equation (2.7), can be used as a distance, such that

$$\begin{aligned} d(\mathbf{w}_q,\mathbf{x}(t)) &= \left\|\phi(\mathbf{x}(t)) - \phi(\mathbf{w}_q)\right\|_2^2, \\ &= \kappa(\mathbf{x}(t),\mathbf{x}(t)) + \kappa(\mathbf{w}_q,\mathbf{w}_q) - 2\kappa(\mathbf{x}(t),\mathbf{w}_q). \end{aligned} \tag{2.21}$$

[2] For other examples, see Hechenbichler e Schliep (2004).

After defining the class label prediction, additional information can be encoded in the model's output. Some techniques for achieving this are discussed in the next section.

2.6 Class Label Prediction Encoding

After predicting the class to which a sample belongs, one could simply defines the models' output $\hat{\mathbf{y}}(t)$ using the one-hot encoding approach. However, PBC models can also provide additional information to the user by using alternative enconding approaches, depending on the classification strategy.

Firstly, using the nearest neighbor strategy, one can hold the distances, to the sample, from the closest prototypes of each class. Then, each component $\hat{y}_c(t)$ of the class prediction vector $\hat{\mathbf{y}}(t)$ can be calculated as

$$\hat{y}_c(t) = \frac{d_c^-(t) - d_c^+(t)}{d_c^-(t) + d_c^+(t)}, \tag{2.22}$$

where $d_c^+(t)$ is the distance from the sample to the closest prototype of class c, and $d_c^-(t)$ is the distance from the sample to the closest prototype of all the classes but c. A similar equation appears in the prototypes' weights update function of the GLVQ algorithm (SATO; YAMADA, 1995). The use of this encoding method confers the subsequent advantages:

- $\hat{y}_c(t)$ always take values within the interval $[-1, +1]$
- If the sample $\mathbf{x}(t)$ exactly matches a prototype, the distance between them is 0, and the models' output will be in the one-hot enconding format.
- In the prediction vector $\hat{\mathbf{y}}(t)$, only the position corresponding to the predicted class will be positive.

Moreover, using the KNN strategy, one can compute the average of the nearest protoypes' labels, such as

$$\hat{\mathbf{y}}(t) = \frac{\sum_{k=1}^{K} \mathbf{y}(\mathbf{w}_k)}{K}, \tag{2.23}$$

where $\mathbf{y}(\mathbf{w}_k)$ is the one-hot encoded prototype's label. This kind of enconding has the following properties:

- In the prediction vector $\hat{\mathbf{y}}(t)$, the component corresponding to the predicted class will have the highest value.
- The value $\hat{y}_c(t)$, at each prediction vector's position, will be as high as the number of nearest neighbors, from class c, to the sample $\mathbf{x}(t)$.

Furthermore, with the WKNN and KWKNN strategies, one must follow steps 1 to 4 of the WKNN algorithm. Then, assuming that each neighboring prototype has its label $\mathbf{y}(\mathbf{w}_i)$ in one-hot encoding form, the model's prediction can be computed as

$$\hat{\mathbf{y}}(t) = \frac{\sum_{k=1}^{K} \mathbf{y}(\mathbf{w}_k).f(d(\mathbf{x}(t),\mathbf{w}_k))}{\sum_{k=1}^{K} f(d(\mathbf{x}(t),\mathbf{w}_k))}. \tag{2.24}$$

It is important to note that the KNN encoding is just a particular case of the WKNN, where $f(d(\mathbf{x},\mathbf{w}_k)) = 1$ (using the retangular transformation function).

Similar to one-hot encoding, when using these alternative encoding strategies for classifier prediction, one can simply select the highest value in the output prediction vector to define the estimated class of the sample. However, other applications and analyses may also arise from these encoding strategies, such as the reject option (CHOW, 1970) and receiver operating characteristic (ROC) curves (FAWCETT, 2006).

2.7 Chapter Summary

In this chapter, the theory behind kernel prototype-based classifiers was introduced. Concepts such as the classification problem, prototype labeling strategies, and kernel functions form the foundation for understanding the main contributions of this thesis, each of which is addressed in a dedicated chapter. In the next chapter, the impact of the chosen kernel functions and prototype labeling strategies on the performance of KSOM classifier variants is evaluated.

3 CLASSIFICATION USING KERNEL SELF-ORGANIZING MAPS

Prototype-based models have been used for diverse classification tasks such as motor failure detection (COELHO *et al.*, 2017), face recognition (RUSTAM; RIKA, 2018), identification of human phosphorylated proteins (CUI; DING, 2020), Fake News Detection (NGUYEN *et al.*, 2023) and Identification of salinity sources in groundwater (JAFARI *et al.*, 2025). In this chapter, the GD-KSOM and EF-KSOM algorithms (LAU *et al.*, 2006) are used as examples of kernel prototype-based models applied to such tasks.

First, the aforementioned algorithms are briefly described, pointing out their main hyperparameters. Some equations have already been presented in Chapter 2, but they are repeated here to make this chapter self-contained. Next, the experimental setup and the metaparameters (a term defined at the beginning of Chapter 2) are shown. After that, the classification datasets are detailed. Finally, the influence of kernel functions and labeling methods on classification accuracy is examined.

The eight kernel functions (all listed in Table 1) are used to generate different nonlinear mappings of the data. In addition, the three labeling methods described in Section 2.3 are used to convert the KSOM variants into pattern classifiers. These methods are identified by the following abbreviations: minimum distance (MD), average distance (AD) and majority voting (MV). Moreover, the NNC is used for comparing the generalization performance of each model.

Finally, all prototype-based models were implemented from scratch in MATLAB (R2023b), running on Windows 10 Home, on an HP notebook with an Intel Core i7-7500U processor (2.70 GHz) and 16 GB of RAM.

The motivation for focusing on GD-KSOM and EF-KSOM is twofold. First, these two algorithms are widely used kernel extensions of SOM, yet their comparative behavior under different kernel functions, prototype-labeling strategies and datasets has not been systematically examined in the literature. This chapter therefore fills this gap by providing a broad and controlled evaluation of these variants, extending the preliminary results of (LAU *et al.*, 2006) with additional datasets and a richer set of kernel functions. Second, the results obtained here establish a quantitative baseline against which the new method proposed in Chapter 4 can be compared, thus clarifying the relative contributions and performance gains obtained by the approach developed in this thesis.

3.1 Kernel SOM Models

The SOM model's prototypes have a topological organization in space that can be, for example, one-dimensional or two-dimensional. The set $\mathscr{A}$ of Q prototypes must be defined before the training step. For instance, 30 prototypes can be arranged in a two-dimensional grid with 5 rows and 6 columns.

Once the topological distribution of the prototypes is determined, it is necessary to initialize their weight vectors. One possible strategy is to place prototypes near the mean vector of the training dataset. Alternatively, Q feature vectors can be randomly sampled from the training set.

Then, in general, the SOM's training algorithm consists of a predefined number of epochs N_{ep}. In each epoch ep, the training samples are presented, once each, to the model, in order to update the prototypes' weights. The total number of discrete time steps (iterations) t_{max} can be calculated by multiplying the number of epochs N_{ep} by the number of training samples N_{tr}.

An important variable to be defined is the learning rate η $(0 < \eta \ll 1)$. It can be set as a constant η_0 or, to increase the probability of convergence, it is common to make the learning rate decrease over time, such as

$$\eta(t) = \eta_0 \left(1 - \frac{t}{t_{max}}\right), \tag{3.1}$$

where t is the current iteration (time step), and η_0 is the initial learning rate (a hyperparameter).

In each iteration, the winning prototype q^* must first be identified. In the standard formulation of the algorithm, the similarity between samples and prototypes can be computed using the squared Euclidean distance

$$q^*(t) = \arg\min_{\forall q} \|\mathbf{x}(t) - \mathbf{w}_q(t)\|_2^2. \tag{3.2}$$

Alternatively, one can use the kernelized version of Equation 3.2 to find q^*, such as

$$\begin{aligned} q^*(t) &= \arg\min_{\forall q} \left\| \phi\left(\mathbf{x}(t)\right) - \phi\left(\mathbf{w}_q(t)\right) \right\|_2^2, \\ &= \arg\min_{\forall q} J_q(\mathbf{x}(t)), \end{aligned} \tag{3.3}$$

As previously described in Section 2.4, this expression can be expanded as

$$\begin{aligned} J_q(\mathbf{x}(t)) &= \left\| \phi(\mathbf{x}(t)) - \phi(\mathbf{w}_q(t)) \right\|_2^2, \\ &= k(\mathbf{x}(t), \mathbf{x}(t)) + k(\mathbf{w}_q(t), \mathbf{w}_q(t)) - 2k(\mathbf{x}(t), \mathbf{w}_q(t)). \end{aligned} \tag{3.4}$$

Its important to mention that, the only difference between GD-KSOM and EF-KSOM lies in the way these algorithms select the winning prototype. The former executes this operation in the input space (Eq. (3.2)), while the latter executes it in the feature space (Eq. (3.4)).

In the original SOM model, after identifying the winning prototype, all the prototypes are updated according to the following rule:

$$\mathbf{w}_q(t+1) = \mathbf{w}_q(t) + \eta(t)h(q^*,q,t)[\mathbf{x}(t) - \mathbf{w}_q(t)]. \tag{3.5}$$

The kernel SOM models (GD-KSOM and EF-KSOM), by their turn, have the following prototype updating rule:

$$\mathbf{w}_q(t+1) = \mathbf{w}_q(t) - \eta(t)h(q^*,q,t)\nabla J_q(\mathbf{x}(t)), \tag{3.6}$$

where the general expression of the gradient vector $\nabla J_q(\mathbf{x}_t)$ is given by

$$\nabla J_q(\mathbf{x}(t)) = \frac{\partial J_q(\mathbf{x}(t))}{\partial \mathbf{w}_q(t)} = \frac{\partial \kappa(\mathbf{w}_q(t),\mathbf{w}_q(t))}{\partial \mathbf{w}_q(t)} - 2\frac{\partial \kappa(\mathbf{w}_q(t),\mathbf{x}(t))}{\partial \mathbf{w}_q(t)}. \tag{3.7}$$

The neighborhood function $h(q^*,q,t)$ depends on the geometric positions $\mathbf{r}_q(t)$ and $\mathbf{r}_{q^*}(t)$ of the current prototype and the winning prototype, respectively. In its simplest form, its value is either 1 if the current prototype is a neighbor of the winning prototype, or 0 otherwise. This corresponds to the rectangular neighborhood proposed by Kohonen, where the number of neighbors is defined by the layer size N_h. Different layer sizes lead to different numbers of neighbors. In Figure 5, the neighborhood layers are illustrated in a two-dimensional network. The dashed squares represent layer sizes $N_h = 1$, $N_h = 2$, and $N_h = 3$ centered on the prototype with geometric position $\mathbf{r}_{q^*}(t) = [6,5]$.

Alternatively, the Gaussian function is often used as a neighborhood function as it makes the training algorithm converge faster. This function can be defined as

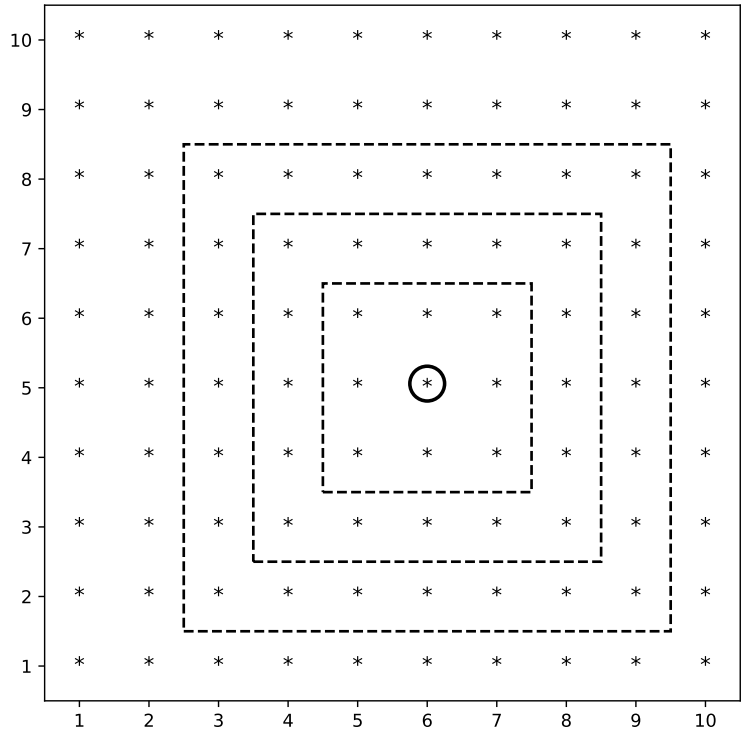
$$h(q^*,q,t) = exp\left(-\frac{\left\|\mathbf{r}_q(t) - \mathbf{r}_{q^*}(t)\right\|_2^2}{2\sigma_h^2(t)}\right), \tag{3.8}$$

where $\sigma_h(t)$ defines how much influence the winning prototype has on its neighbors. This parameter requires a decay scheme, such as

$$\sigma_h(t) = v_0\left(\frac{v_f}{v_0}\right)^{\frac{t}{t_{max}}}, \tag{3.9}$$

where v_0 and v_f are hyperparameters that define the initial and final values of σ_h respectively. By choosing these values properly, the model can be made to start with a high neighborhood influence and end with a low one.

Figure 5 – SOM Neighborhood in a 2-dimensional space.



Source: (COELHO, 2012) (adapted)

Once the network training is completed, a labeling method must be applied (see Section 2.3) to assign a label to each prototype. The final model $H = \{(\mathbf{w}_q, \tilde{c}_q), q = 1, ..., Q\}$ is thus composed of a set of prototypes $\mathbf{w}_q$ and their corresponding labels $\tilde{c}_q$. Finally, to classify new incoming samples, the hyperparameter K of the KNN prediction strategy must be specified.

The hyperparameters values of the kernel SOM models are summarized in Table 2. One important detail is that the set of 30 prototypes is organized in a 6×5 rectangular grid. The same topological arrangement was used in (COELHO *et al.*, 2017) for a comparative study. Additionally, this structure was kept fixed to allow a focused comparison of the kernel functions and labeling strategies. Finally, some of the hyperparameters are fixed (preassigned), while others have their search space also specified in Table 2. The heuristics used for hyperparameter optimization are described in the next section.

Finally, Algorithm 1 summarizes the training process of the EF-KSOM model. It is important to note that, disregarding the data vector dimension, the computational complexity of this algorithm is $\mathcal{O}(N_{ep} \cdot N_{tr} \cdot Q)$.

Table 2 – The hyperparameters of the Kernel SOM Algorithm.

Symbol	Definition	Values
Q	Number of prototypes	30
N_{ep}	Number of Epochs	50
v_0	Initial value of neighborhood influence	0.8
v_f	Final value of neighborhood influence	0.3
η_0	Initial Learning Rate	0.7
α	Kernels' Hyperparameter 1	$[2^{-10},...,2^{10}]$
θ	Kernels' Hyperparameter 2	$[0,\pm 2^{-10},...,\pm 2^{10}]$
σ_κ	Kernels' Hyperparameter 3	$[2^{-10},...,2^{10}]$
l	Kernels' Hyperparameter 4	$[0.2,0.4,...,2.8,3]$
K	Number of neighbors (for classification)	1

Algoritmo 1: Pseudocode of the EF-KSOM training and prototype labeling.

Input: $\{(\mathbf{x}_n, c_n)\}_{n=1}^{N_{tr}}$, Q, N_{ep}, v_0, v_f, η_0, $\kappa(\cdot,\cdot)$, α, θ, σ_κ, l, K
Output: $H = \{(\mathbf{w}_q, \tilde{c}_q)\}_{q=1}^{Q}$
begin
  Initialize prototypes' weights $\{(\mathbf{w}_q)\}_{q=1}^{Q}$
  Initialize prototypes' grid positions $\{(\mathbf{r}_q)\}_{q=1}^{Q}$
  Calculate maximum number of time steps $t_{max} = N_{ep} \times N_{tr}$
  Initialize descrete time step $t = 0$
  for $epochs = 1 : N_{ep}$ **do**
    Shuffle input-output pairs: $\{(\mathbf{x}_n, c_n)\}_{n=1}^{N_{tr}}$
    for $n = 1 : N_{tr}$ **do**
      Update discrete time step: $t = t + 1$
      Get sample $\mathbf{x}(t)$ from input-output pairs
      Calculate the learning step η using Eq. 3.1
      Find the winning prototype q^* using Eq. 3.4
      for $q = 1 : Q$ **do**
        Calculate the neighborhood function $h(q, q^*, t)$ using Eq. 3.8
        Update the weight vector $\mathbf{w}_q$ using Eq. 3.6
      end
    end
  end
  Use a labeling strategy to generate prototypes' labels $\{(\tilde{c}_q)\}_{q=1}^{Q}$
end

3.2 Hyperparameters Optimization

In a machine learning system, hyperparameters are parameters that define the model's architecture. They must be determined before the training and testing processes and have a direct influence on the performance of a given model (YANG; SHAMI, 2020). Hyperparameter optimization involves finding the set of hyperparameters that minimizes a cost function applied

to a machine learning model using the training dataset. Using cross-validation, the optimal set of hyperparameters minimizes the mean error of the model's prediction on the validation set (BERGSTRA; BENGIO, 2012).

There are several strategies to find the best combination of hyperparameters' values. The first one, is the *trial and error*, where the user will test values based on experience. Besides this, there are some systematic ways for optimizing the hyperparameters.

The *grid search*, for example, is an algorithm that evaluates all possible combinations of hyperparameter values within a predefined grid. Specifically, it computes the Cartesian product of the given values for each hyperparameter, which makes this approach extremely costly. The computational cost increases exponentially with the dimensionality of the search space (HUTTER *et al.*, 2019).

Alternatively, *random search* is an optimization algorithm designed to enhance the performance of a function by exploring a set of randomly generated numbers within the specified search domain. It iteratively samples combinations until a predefined stopping criterion is met. In the context of hyperparameter optimization, this method provides a compelling alternative to grid search. Unlike grid search, which exhaustively evaluates all possible hyperparameter combinations, random search introduces a more efficient approach by randomly selecting combinations within a defined range. Notably, this method often outperforms grid search, particularly in scenarios where only a subset of hyperparameters has a significant impact on the learning algorithm's performance (LIASHCHYNSKYI; LIASHCHYNSKYI, 2019).

Last but not least, stochastic optimization techniques such as particle swarm optimization (PSO) and genetic algorithms (GA) can be employed to determine the model's hyperparameters. However, these strategies were not used in this work.

In the next section, the experimental setup for batch datasets is described. It is important to note that this setup is also used in the next chapter to evaluate another model.

3.3 Experimental Setup for Batch Datasets and KSOM

To evaluate the KSOM models on each dataset for batch learning, two sets of experiments are carried out. In the first, 10 independent realizations are executed. For each realization, the following steps are performed.

- (*i*) Shuffle the dataset;
- (*ii*) Holdout procedure: partition the data into training and test sets;

- (*iii*) Z-score normalization: compute the sample mean vector μ_z and the standard deviation vector σ_z of the training samples. Then, normalize the feature vectors so that all attributes have zero mean and unit variance;
- (*iv*) Hyperparameter optimization: apply random search with 100 trials on the training dataset to find the best combination of hyperparameters;
- (*v*) Training: update the model's parameters using the training dataset;
- (*vi*) Performance testing: use the test dataset to validate the trained model.

For the holdout procedure, the data is randomly divided as follows: 70% for training and 30% for testing. At the end of the testing phase, several statistical performance measures are computed for each classifier, such as accuracy, error rate and F1-score (macro-averaged in the case of multiclass problems).

To search for the optimal hyperparameter values, a 5-fold cross-validation strategy is performed using the entire training portion (i.e., the 70% split from the holdout procedure). Each combination of hyperparameters is evaluated over 100 trials of random search, and the batch classification error $0 \leq E_b \leq 1$ (already defined in Eq. 2.1) is used as the loss function for assessing the performance of prototype-based algorithms during hyperparameter selection. Once the optimal hyperparameters are determined, the full training dataset is used again to update/define the model parameters before the final evaluation on the test set.

To summarize, the metaparameters used in this setup are listed next.

- Number of realizations: $N_r = 10$;
- Percentage of training data: $N_p = 70\%$;
- Normalization procedure: Z-score;
- Number of folds for cross-validation: $N_f = 5$;
- Number of random search trials: $N_t = 100$;
- Loss function for random search: minimum batch classification error E_b.

In the second set of experiments, the optimal hyperparameters obtained in the first set are used, and 100 independent realizations are executed. In these realizations, the same sequence of steps from the first set is performed, except for step (iv) (hyperparameter optimization). These experiments are conducted to verify the models' sensitivity to data shuffling.

In the next section, the datasets for batch learning used throughout this thesis are described in detail.

3.4 Datasets for batch learning

The *motor failure* dataset (COELHO *et al.*, 2014) consists of 294 feature vectors, each containing six harmonics of the fast Fourier transform of a line current measurement from a three-phase induction motor. The samples are organized into seven class labels, where 1 is for normal operation condition (42 samples) and the other 6 labels correspond to short-circuit faulty conditions (252 samples). In Coelho *et al.* (2014), the best results were obtained when the problem was treated as binary and the classes were balanced by adding 210 artificial samples of normal condition to the dataset. The same methodology is adopted in this thesis.

The *pap-smear* (cervical cancer) dataset (JANTZEN *et al.*, 2005) consists of 917 images of Pap-smear cells classified by cytotechnicians and doctors[1]. Each cell is described by 20 numerical features, and the dataset is divided into seven classes. Samples from 3 classes originate from normal cells (totaling 242 samples) while samples from the other four classes correspond to abnormal cells (totaling 675 samples). So, the classification problem for this dataset can be treated as binary.

In the *vertebral column* dataset[2] proposed by ROCHA NETO (2011), six biomechanical attributes are derived from the shape and orientation of the pelvis and lumbar spine of 310 patients. Each patient's condition is classified as *normal* (100 samples), *disk hernia* (60 samples), or *spondylolisthesis* (150 samples). This dataset can also be treated as binary, with the patient's condition classified as *normal* (100 samples) or *abnormal* (210 samples). In this thesis, the binary labels are used.

Finally, the last dataset used for the batch learning experiments, proposed by Freire *et al.* (2009), concerns a wall-following robot that chooses among four actions (move forward, slight right turn, sharp right turn, and slight left turn) as it navigates a room using 24 ultrasound sensors arranged circularly around its body[3]. The output of each sensor can be used as an individual feature, or the outputs can be merged into either four or two features. In this thesis, the two-feature configuration is used.

The characteristics of the datasets are summarized in Table 3, where # denotes the cardinality of a set. In the next section, the KSOM model variants are evaluated on all these datasets.

It is important to mention that the decision regarding the number of classes and

[1] https://mde-lab.aegean.gr/index.php/downloads/
[2] https://doi.org/10.24432/C5K89B
[3] https://doi.org/10.24432/C57C8W

Table 3 – Summary table for the datasets used for batch learning.

Dataset	#Samples	#Attributes	#Classes
Motor Failure	504	6	2
Cervical Cancer	917	20	2
Vertebral Column	310	6	2
Wall Following	5456	2	4

features in each dataset was driven by the need to compare the KSOM models with existing results from other works in the literature, as well as with the model proposed in Chapter 4. These comparisons are presented in Section 4.4.

3.5 Evaluation of Kernel SOM-based Classifiers on Batch Learning

Initially, to establish a baseline for the results of the KSOM-based classifiers, two types of linear classifiers were used.

The first classifier applies a linear mapping between the input (feature vectors) and the output (encoded label vectors). Considering that both the feature vectors and the encoded label vectors are column vectors, this mapping is defined by the equation

$$\hat{\mathbf{y}}(t) = \mathbf{W}\mathbf{x}(t), \tag{3.10}$$

where $\mathbf{W} \in \mathbb{R}^{(P+1)\times C}$ is the weight matrix. It is important to note that the dimension $P+1$ arises from the addition of a bias term $b = 1$ to each feature vector. Moreover, the weights of this matrix are not known *a priori*. However, using the training data, there are techniques to estimate these weights in order to minimize the error in mapping the input vectors to the encoded label vectors. A widespread technique for this is the Moore-Penrose pseudoinverse (MOORE, 1920; PENROSE, 1955), also known as ordinary least squares (OLS).

The second classifier is multiclass logistic regression (COX, 1958; BISHOP, 2006), using the one-versus-rest multiclass strategy and the L-BFGS solver algorithm.

For these experiments, each dataset was shuffled 100 times, and the data was divided into training and testing sets. For the first linear classifier, the OLS algorithm was applied to compute the weight matrix $\mathbf{W}$, and the accuracy obtained by applying this linear mapping on the testing set was then calculated. For the multiclass logistic regression, the *LogisticRegression*

Table 4 – Summary of accuracies of the OLS and logistic regression models.

Dataset	OLS	Logistic Regression
Motor Failure	67.2± 3.2	68.6 ± 3.6
Cervical Cancer	93.1± 1.1	93.8 ± 1.3
Vertebral Column	76.4± 3.1	85.4 ± 3.3
Wall Following	53.9± 0.6	87.5 ± 0.7

Table 5 – Performances of the evaluated KSOM models for the Motor Failure dataset (1st set of experiments).

Algorithm	Labeling	linear	gaussian	polynomial	exponential	cauchy	log	sigmoidal	kmod
EF-KSOM	Majority Voting	73.6± 4.0	73.2± 3.4	73.4± 3.0	**75.6 ± 2.6**	73.7± 2.9	**74.1 ± 4.5**	73.2± 4.8	**74.6 ± 2.3**
	Average Distance	**69.7 ± 4.0**	**69.9 ± 3.7**	**73.0 ± 1.9**	64.3± 3.9	67.6± 3.3	68.2± 4.4	62.1± 15.4	65.7± 3.9
	Minimum Distance	69.4± 4.6	**72.3 ± 4.9**	**71.8 ± 5.7**	**73.4 ± 1.9**	70.4± 3.3	**71.8 ± 3.5**	70.2± 4.1	69.9± 4.1
GD-KSOM	Majority Voting	**73.5 ± 3.9**	73.2± 3.4	72.2± 4.5	**74.5 ± 3.2**	72.6± 4.6	72.0± 4.0	**73.6 ± 7.2**	72.9± 3.4
	Average Distance	70.7± 3.4	69.7± 5.0	**71.1 ± 4.1**	**72.2 ± 4.7**	70.5± 4.7	70.1± 5.2	68.6± 5.0	**70.9 ± 5.4**
	Minimum Distance	69.4± 4.6	69.5± 6.0	**71.3 ± 2.6**	70.2± 2.6	68.2± 4.0	69.5± 5.8	**72.6 ± 5.1**	**73.4 ± 2.6**

function from the *scikit-learn* Python package was used to train the model and obtain the accuracy measure.

The performance achieved by both strategies across all datasets is summarized in Table 4.

In the following subsections, the evaluation of kernel SOM-based classifiers on each dataset for batch learning is described in detail.

3.5.1 Results for the Motor Failure Dataset

The performance of the KSOM model applied to the Motor Failure dataset is presented in Table 5. The top three mean accuracies in each row (corresponding to one KSOM training algorithm and one labeling method) are highlighted in boldface. The same presentation format is used for the other datasets.

The best performance (75.6 ± 2.6) was obtained with the EF-KSOM model combined with the majority voting labeling method and the exponential kernel function. When comparing labeling methods, the best performances were generally achieved with majority voting. Regarding the kernel functions, the exponential kernel reached the highest performance and was selected four times (out of six) as one of the best-performing kernels.

Table 6 – Performances of the evaluated KSOM models for the Motor Failure dataset (2nd set of experiments).

Algorithm	Labeling	linear	gaussian	polynomial	exponential	cauchy	log	sigmoidal	kmod
EF-KSOM	Majority Voting	73.6± 2.9	**77.9 ± 3.5**	74.4± 3.5	73.1± 3.7	75.6± 3.0	**76.8 ± 3.2**	75.3± 3.6	**76.1 ± 3.7**
	Average Distance	**69.9 ± 3.8**	**73.9 ± 3.2**	**71.7 ± 3.7**	66.5± 3.0	64.5± 4.4	65.9± 5.5	67.7± 3.5	65.9± 4.2
	Minimum Distance	69.6± 4.0	**75.9 ± 2.8**	**77.9 ± 3.6**	66.4± 4.0	70.6± 3.2	73.1± 3.3	**77.4 ± 3.2**	69.8± 4.7
GD-KSOM	Majority Voting	**73.5 ± 3.9**	**73.5 ± 3.7**	72.6± 3.5	**74.1 ± 3.1**	73.1± 3.4	73.2± 3.2	72.7± 3.6	73.1± 3.7
	Average Distance	70.3± 3.8	**71.3 ± 4.0**	**71.2 ± 3.4**	62.2± 3.4	70.8± 3.5	70.5± 3.9	70.6± 3.4	**71.5 ± 3.8**
	Minimum Distance	69.9± 3.9	70.0± 4.0	69.6± 3.7	68.6± 5.7	**70.1 ± 4.3**	**70.2 ± 3.6**	**70.2 ± 3.9**	**70.1 ± 4.3**

The best hyperparameters obtained from the first set of experiments with the Motor Failure dataset are reported in Table 52, in Appendix C. Using these values, 100 new realizations were carried out. The results from this experiment are shown in Table 6. The best performance (77.9 ± 3.5) was achieved with the EF-KSOM model combined with the majority voting labeling method and the Gaussian kernel function. When comparing labeling methods, the highest mean values were generally obtained with the majority voting strategy. Regarding the kernel functions, the Gaussian kernel reached the highest mean values and was selected five times (out of six) as one of the best-performing kernels.

As can be seen, the best result from the second set of experiments (77.9 ± 3.5) was higher than that of the first set (75.6 ± 2.6), indicating that once the best hyperparameters are found, shuffling again the data and dividing it into different training and test sets does not negatively affect classifier performance. Moreover, this best result is 10.7% higher than the result of (67.2 ± 3.2) obtained with the OLS classifier, and 9.3% higher than the result of (68.6 ± 3.6) obtained with the logistic regression classifier, both of which are reported in Table 4.

A summary of the results from these two sets of experiments is reported in Table 7. In this table, the *comparisons* column defines the following: the *best result* refers to the combination of model, labeling strategy, and kernel function that achieved the highest mean accuracy. The *best model* is the model that, when fixing the same kernel function and labeling strategy, most frequently produced the highest mean accuracy. The *best labeling strategy* is the labeling method that, when fixing the model and the kernel function, most often led to the highest accuracy. Finally, the *best kernels* are the kernels that, when fixing the model and labeling method, most frequently appeared among the top three mean accuracies. Similar summary tables

Table 7 – Summary of the results for the Motor Failure dataset.

Dataset	Experiment	Comparisons	
Motor Failure	Optimizing Hyperparameters	Best Result	EF-KSOM Majority Voting Exponential
		Best Model	EF-KSOM
		Best Labeling Strategy	Majority Voting
		Best Kernels	Exponential
	Using the Best Hyperparameters	Best Result	EF-KSOM Majority Voting Gaussian
		Best Model	EF-KSOM
		Best Labeling Strategy	Majority Voting
		Best Kernels	Gaussian

Table 8 – Performances of the evaluated KSOM models for the Cervical Cancer dataset (1st set of experiments).

Algorithm	Labeling	linear	gaussian	polynomial	exponential	cauchy	log	sigmoidal	kmod
EF-KSOM	Majority Voting	**88.9** ± **1.6**	87.1± 5.2	**89.0** ± **1.6**	86.0± 5.4	88.8± 1.4	87.8± 0.9	**89.4** ± **0.7**	87.6± 1.8
	Average Distance	88.3± 1.6	**89.1** ± **2.2**	87.6± 5.3	87.6± 5.6	**88.9** ± **1.8**	88.2± 1.7	**89.1** ± **1.4**	88.8± 1.9
	Minimum Distance	86.6± 4.9	**88.3** ± **2.7**	87.4± 2.5	79.5± 19.8	85.9± 8.4	**88.6** ± **2.6**	**87.9** ± **4.6**	85.7± 6.3
GD-KSOM	Majority Voting	**89.4** ± **1.8**	88.4± 2.0	88.5± 3.4	**90.1** ± **1.8**	88.1± 1.3	88.5± 1.7	**89.7** ± **1.3**	88.7± 1.2
	Average Distance	**89.4** ± **1.9**	88.2± 2.0	88.7± 1.7	88.4± 2.1	87.8± 1.4	**89.8** ± **1.0**	**88.9** ± **1.1**	88.8± 1.5
	Minimum Distance	85.6± 6.6	**88.6** ± **3.6**	87.7± 2.2	**89.2** ± **2.1**	**88.7** ± **1.8**	87.9± 3.0	85.2± 6.7	86.7± 3.5

are provided for the remaining datasets.

3.5.2 Results for the Cervical Cancer Dataset

The performance of the KSOM model applied to the Cervical Cancer dataset is presented in Table 8. The highest performance was achieved with the GD-KSOM model combined with the majority voting labeling method and the exponential kernel function. Regarding the kernel functions, the sigmoidal kernel reached the highest mean values and was selected five times (out of six) as one of the best-performing kernels. When comparing the labeling methods and KSOM models, the differences among the results of each strategy are minimal.

The best hyperparameters obtained from the first set of experiments with the cervical cancer dataset are reported in Table 53, in Appendix C. Using these values, 100 new realizations were carried out. The results from this experiment are presented in Table 9. Overall, for this

Table 9 – Performances of the evaluated KSOM models for the Cervical Cancer dataset (2nd set of experiments).

Algorithm	Labeling	linear	gaussian	polynomial	exponential	cauchy	log	sigmoidal	kmod
EF-KSOM	Majority Voting	88.8± 1.5	**89.1 ± 1.5**	**89.0 ± 1.3**	88.8± 1.5	**88.9 ± 1.7**	88.7± 1.5	88.8± 1.6	**88.9 ± 1.6**
	Average Distance	87.8± 4.2	88.6± 1.8	88.5± 3.0	**88.9 ± 1.5**	88.2± 1.7	83.9± 8.2	**88.8 ± 1.8**	**88.7 ± 1.4**
	Minimum Distance	**87.4 ± 3.7**	**89.0 ± 1.7**	86.9± 3.7	86.6± 3.5	86.5± 4.1	**88.0 ± 2.9**	87.3± 3.8	**87.4 ± 3.6**
GD-KSOM	Majority Voting	88.6± 1.6	**88.9 ± 1.7**	**88.9 ± 1.5**	88.2± 1.8	88.7± 1.7	88.7± 1.6	**88.9 ± 1.5**	88.4± 1.6
	Average Distance	**88.4 ± 2.2**	87.6± 3.7	88.1± 3.7	88.0± 3.5	**88.7 ± 1.6**	**88.6 ± 2.5**	88.5± 1.5	88.1± 3.5
	Minimum Distance	87.3± 4.2	87.4± 4.0	**88.1 ± 3.1**	**88.2 ± 3.5**	87.6± 3.7	87.0± 5.0	87.3± 4.0	**88.1 ± 2.8**

Table 10 – Summary of the results for the Cervical Cancer dataset.

Dataset	Experiment	Comparisons	
Cervical Cancer	Optimizing Hyperparameters	Best Result	GD-KSOM Majority Voting Exponential
		Best Model	GD-KSOM
		Best Labeling Strategy	Average Distance
		Best Kernels	Sigmoidal
	Using the Best Hyperparameters	Best Result	EF-KSOM Majority Voting Gaussian
		Best Model	EF-KSOM
		Best Labeling Strategies	Majority Voting
		Best Kernels	Kmod

dataset, there were no significant differences when using different kernels, labeling methods, or algorithms. The average accuracies range from 86.5 to 89.1. The best accuracy, 89.1%, is 4% and 4.7% lower than the ones reported in Table 4.

A summary of the results from these two sets of experiments is reported in Table 10. It is important to note that, when fixing the hyperparameters, as with the motor failure dataset, the best result is achieved with the EF-KSOM model and the majority voting labeling method.

3.5.3 Results for the Vertebral Column Dataset

The performance of the KSOM model applied to the Vertebral Column dataset is presented in Table 11. The best performance (78.2 ± 3.8) was achieved with the GD-KSOM model combined with the majority voting labeling strategy and the Cauchy kernel function. Regarding the kernel functions, the exponential kernel was selected four times (out of six) as

Table 11 – Performances of the evaluated KSOM models for the Vertebral Column dataset (1st set of experiments).

Algorithm	Labeling	linear	gaussian	polynomial	exponential	cauchy	log	sigmoidal	kmod
EF-KSOM	Majority Voting	71.5± 5.1	73.5± 5.8	73.8± 7.6	72.6± 14.0	74.1± 5.6	**75.7 ± 4.1**	**77.4 ± 4.8**	**76.9 ± 4.4**
	Average Distance	56.7± 9.8	**69.5 ± 5.8**	60.9± 19.4	56.6± 13.6	**66.5 ± 8.9**	62.8± 10.0	**66.0 ± 13.1**	61.6± 10.8
	Minimum Distance	71.6± 6.0	**73.0 ± 8.5**	69.0± 4.0	**73.0 ± 6.7**	70.1± 11.6	**74.7 ± 4.2**	72.0± 11.5	69.4± 10.5
GD-KSOM	Majority Voting	74.4± 7.4	75.6± 4.7	75.4± 6.0	**77.7 ± 5.1**	**78.2 ± 3.8**	**76.6 ± 4.1**	75.4± 4.4	73.1± 5.7
	Average Distance	**62.3 ± 10.4**	53.2± 14.8	49.2± 13.8	**64.8 ± 9.9**	**60.2 ± 15.7**	54.0± 15.9	53.1± 14.1	57.5± 12.8
	Minimum Distance	68.4± 6.2	**70.8 ± 5.1**	68.6± 7.4	**73.1 ± 4.8**	70.6± 4.4	68.6± 3.9	69.9± 6.6	**70.4 ± 9.0**

Table 12 – Performances of the evaluated KSOM models for the Vertebral Column dataset (2nd set of experiments).

Algorithm	Labeling	linear	gaussian	polynomial	exponential	cauchy	log	sigmoidal	kmod
EF-KSOM	Majority Voting	69.9± 6.0	75.6± 4.7	**81.9 ± 3.4**	**76.9 ± 4.6**	75.2± 5.6	74.9± 4.9	**76.3 ± 5.3**	75.3± 5.1
	Average Distance	59.0± 10.5	**64.1 ± 8.6**	59.3± 10.3	59.9± 14.9	63.3± 11.1	63.6± 11.0	**72.2 ± 5.9**	**65.3 ± 8.8**
	Minimum Distance	69.9± 5.6	72.6± 6.2	68.1± 6.1	**74.1 ± 5.2**	73.0± 5.6	70.1± 6.7	**76.2 ± 6.1**	**74.6 ± 5.4**
GD-KSOM	Majority Voting	70.3± 5.5	71.3± 5.5	71.5± 5.2	**72.3 ± 5.3**	**72.0 ± 5.4**	**72.3 ± 5.7**	71.7± 5.8	70.9± 5.2
	Average Distance	59.5± 11.7	58.9± 11.5	57.1± 11.6	**60.0 ± 11.9**	58.9± 11.2	58.4± 11.7	**62.3 ± 6.5**	**61.6 ± 10.9**
	Minimum Distance	69.1± 7.4	68.4± 6.2	**70.7 ± 6.3**	68.2± 5.9	**71.1 ± 6.1**	69.6± 6.4	**70.0 ± 6.5**	**70.0 ± 6.0**

one of the best-performing kernels. When comparing the labeling methods and algorithms, no significant differences were observed among the results of each strategy.

The best hyperparameters obtained from the first set of experiments with the Vertebral Column dataset are reported in Table 54, in Appendix C. Using these values, 100 new realizations were carried out. The results from this experiment are presented in Table 12. The highest performance (81.9 ± 3.4) was achieved with the EF-KSOM model combined with the majority voting labeling strategy and the polynomial kernel function. Regarding the kernel functions, the sigmoidal kernel was selected five times (out of six) as one of the best-performing kernels. The best accuracy (81.9%) is 5.5% higher than the result obtained with the OLS classifier and 3.5% lower than the result obtained by the logistic regression classifier, both reported in Table 4.

A summary of the results from these two sets of experiments is reported in Table 13. It is important to note that, when fixing the hyperparameters, as with the motor failure and

Table 13 – Summary of the results for the Vertebral Column Dataset.

Dataset	Experiment	Comparisons	
Vertebral Column	Optimizing Hyperparameters	Best Result	GD-KSOM Majority Voting Cauchy
		Best Model	EF-KSOM
		Best Labeling Strategy	Majority Voting
		Best Kernels	Exponential or Cauchy
	Using the Best Hyperparameters	Best Result	EF-KSOM Majority Voting Polynomial
		Best Model	EF-KSOM
		Best Labeling Strategies	Majority Voting
		Best Kernels	Sigmoidal

Table 14 – Performances of the evaluated KSOM models for the wall-following dataset (1st set of experiments).

Algorithm	Labeling	linear	gaussian	polynomial	exponential	cauchy	log	sigmoidal	kmod
EF-KSOM	Majority Voting	68.8± 6.0	75.7± 4.0	**76.0 ± 2.9**	**78.9 ± 1.9**	75.7± 2.7	74.7± 2.7	**77.2 ± 3.3**	75.8± 4.5
	Average Distance	87.4± 4.3	**89.5 ± 0.9**	**90.2 ± 1.1**	89.0± 1.6	**90.1 ± 1.0**	74.1± 3.0	89.3± 1.6	89.0± 2.3
	Minimum Distance	58.0± 13.8	75.2± 2.4	74.3± 2.4	**79.5 ± 1.3**	76.3± 2.8	**76.1 ± 1.8**	73.8± 1.8	**77.1 ± 2.1**
GD-KSOM	Majority Voting	69.6± 3.5	76.4± 5.5	77.0± 3.7	**90.8 ± 1.9**	**90.1 ± 2.7**	89.5± 2.7	90.0± 3.3	**90.4 ± 4.5**
	Average Distance	50.4± 13.1	74.8± 2.9	73.3± 2.8	**77.9 ± 4.6**	73.3± 7.9	**75.1 ± 2.7**	70.9± 4.5	**77.4 ± 1.6**
	Minimum Distance	67.7± 8.0	**75.4 ± 2.7**	74.3± 3.6	**79.4 ± 2.5**	**74.7 ± 4.5**	73.4± 6.1	74.2± 2.8	73.8± 3.3

cervical cancer datasets, the best result is achieved with the EF-KSOM model and the majority voting labeling method.

3.5.4 Results for the Wall Following Dataset

Finally, the performance of the KSOM model applied to the Wall-Following dataset is presented in Table 14. The best performance (90.8 ± 1.9) was achieved with the GD-KSOM model combined with the majority voting labeling strategy and the exponential kernel function. Regarding the kernel functions, the exponential kernel was selected five times (out of six) as one of the best-performing kernels. When comparing the labeling methods and algorithms, no significant differences were observed among the results of each strategy.

The best hyperparameters obtained from the first set of experiments with the Wall

Table 15 – Performances of the evaluated KSOM models for the wall-following dataset (2nd set of experiments).

Algorithm	Labeling	linear	gaussian	polynomial	exponential	cauchy	log	sigmoidal	kmod
EF-KSOM	Majority Voting	69.5± 4.0	78.1± 2.7	**78.2 ± 2.4**	**79.2 ± 1.7**	76.6± 2.2	76.6± 2.2	**79.1 ± 2.2**	76.2± 2.5
	Average Distance	86.0± 4.1	**89.9 ± 1.1**	**89.7 ± 1.2**	89.7± 1.6	**90.0 ± 1.1**	76.1± 3.1	88.1± 2.5	**89.9 ± 1.1**
	Minimum Distance	63.6± 11.1	**76.4 ± 3.2**	**77.5 ± 3.2**	**78.1 ± 2.3**	72.1± 8.1	**76.3 ± 2.9**	72.4± 2.4	**76.3 ± 2.6**
GD-KSOM	Majority Voting	69.5± 3.8	69.7± 3.7	69.1± 4.2	**89.4 ± 1.2**	**89.6 ± 1.2**	**89.4 ± 1.3**	**89.4 ± 1.3**	**89.4 ± 1.3**
	Average Distance	**49.0 ± 10.8**	46.9± 9.8	47.1± 8.8	**49.7 ± 11.3**	47.7± 9.3	**49.5 ± 10.5**	47.2± 10.7	**48.1 ± 10.8**
	Minimum Distance	**65.7 ± 9.8**	**63.0 ± 11.7**	**65.2 ± 10.7**	**63.6 ± 11.2**	**64.3 ± 11.1**	**65.6 ± 9.9**	63.2± 11.3	64.1± 10.9

Table 16 – Summary of the Results for the Wall Following Dataset.

Dataset	Experiment	Comparisons	
Wall Following	Optimizing Hyperparameters	Best Result	GD-KSOM Majority Voting Exponential
		Best Model	GD-KSOM
		Best Labeling Strategy	Majority Voting or Average Distance
		Best Kernels	Exponential
	Using the Best Hyperparameters	Best Result	EF-KSOM Average Distance Cauchy
		Best Model	EF-KSOM
		Best Labeling Strategies	Majority Voting or Average Distance
		Best Kernels	Exponential

Following dataset are reported in Table 55, in Appendix C. Using these values, 100 new realizations were carried out. The results from this experiment are presented in Table 15. The highest performance (90.0 ± 1.1) was achieved with the EF-KSOM model combined with the average distance labeling strategy and the Cauchy kernel function. Regarding the kernel functions, the exponential kernel was selected five times (out of six) as one of the best-performing kernels. When comparing the KSOM models, the EF-KSOM was the best overall.

A summary of the results from these two sets of experiments is reported in Table 16. It is important to note that, when fixing the hyperparameters, as with the other datasets, the best result is achieved with the EF-KSOM model.

3.5.5 A summary of the Results

The results of this section can be summarized as follows:

- In most tests, using the EF-KSOM model proved to be a better choice than the GD-KSOM classifier. In other words, calculating the distance and selecting the winning prototype in the feature space leads to better results.
- The *majority voting* labeling strategy was the most effective. Since only the nearest neighbor classifier was tested, a prototype that is close to many samples from the same class during training can serve as a good predictor for its region of influence (Voronoi cell).
- For both KSOM models, no single kernel function consistently outperformed the others. However, using a kernel function other than the linear one led to better results. Throughout the tests, only the *linear* and *log* kernels were never considered, even once, as the best-performing ones. In most cases, the Gaussian kernel (the most common in the literature) was not the best. This highlights the importance of testing alternative kernel functions when dealing with different applications. This finding also reinforces the results of other works that evaluate different kernel functions and applications. For example, In Ngu e Yeo (2022), several different kernel functions were evaluated in their proposed model for predictive control of a wastewater treatment process, and the polynomial kernel performed best.

In Table 17, a summary of the best performance obtained by the evaluated KSOM models compared with those obtained by the baseline classifiers is presented. Considering the OLS classifier, for three datasets, improvements of 10.7, 5.5, and 46.1 in mean accuracy were observed. In the Cervical Cancer dataset, however, the result was 3.0 lower. For the logistic regression classifier, improvements of 9.3 and 2.5 in mean accuracy were observed for two datasets, while the results were 3.7 and 3.5 lower for the other two.

3.6 Final Considerations

In this chapter, two kernelized variants of the KSOM training algorithm were applied to classification problems using the KNNC. In addition, different labeling methods and kernel functions were compared to evaluate their influence on this task. Regarding the prototype labeling strategy, majority voting often produced the best performance. Considering the eight investigated kernel functions, five were frequently selected as the best: Gaussian, exponential,

Table 17 – Summary of the best performance of the evaluated KSOM models compared with the OLS and logistic regression classifiers.

Dataset	KSOM Models	OLS	Logistic Regression
Motor Failure	77.9 ± 3.5	67.2 ± 3.2	68.6 ± 3.6
Cervical Cancer	90.1 ± 1.8	93.1 ± 1.1	93.8 ± 1.3
Vertebral Column	81.9 ± 3.4	76.4 ± 3.1	85.4 ± 3.3
Wall Following	90.8 ± 1.9	53.9 ± 0.6	87.5 ± 0.7

kmod, sigmoidal, and polynomial. Finally, the best training algorithm was EF-KSOM, in which both the distance measure and the winning prototype calculations are performed in the feature space.

It is important to note that, in all the tests carried out so far, the number of prototypes (30) was kept constant. In the next chapter, a framework to automatically select this hyperparameter is proposed.

4 ON BUILDING SPARSE KERNEL PROTOTYPE-BASED CLASSIFIERS

In this chapter, a simple design scheme for building kernelized prototype-based classifiers is presented. First, this is achieved through the approximate linear dependence (ALD) method. The ALD is a sparsification procedure widely used in kernel adaptive filtering (ENGEL *et al.*, 2004)), and the kernel nearest neighbor classifier (KNNC) is used to predict the labels of new samples. Motivated by the initial results, the sparsification procedures Novelty, Coherence, and Surprise are also tested and compared with ALD, and the KWKNN scheme is additionally applied to classify new samples.

4.1 The Kernel Nearest Neighbor Classifier via ALD Criterion

The proposed training method automatically selects a subset of the training samples to construct a dictionary[1] $\mathscr{D}_{t-1} = \left\{(\tilde{\mathbf{x}}_j, \tilde{c}_j)\right\}_{j=1}^{Q_{t-1}}$, whose size is determined by a single scalar parameter. The samples' inputs $\tilde{\mathbf{x}}_j$ in $\mathscr{D}_{t-1}$ are *approximately* linearly independent feature vectors. The goal of the proposed approach is to use the samples of the dictionary as prototype vectors in the feature space, so that they can be employed in a kernelized nearest-neighbor classification scheme.

Initially, the first training sample $(\mathbf{x}_1, c_1)$ is added to the dictionary. Then, at training time step t $(2 \leq t \leq N_{tr})$, with N_{tr} denoting the number of training samples, after having observed $t-1$ training samples, the dictionary $\mathscr{D}_{t-1}$ consists of a relevant subset $\left\{(\tilde{\mathbf{x}}_j, \tilde{c}_j)\right\}_{j=1}^{Q_{t-1}}$ of these samples. When a new incoming training sample $(\mathbf{x}_t, c_t)$ is available, one must test if it should be added or not to the dictionary. To this end, it is necessary to estimate a vector of coefficients $\mathbf{a} = \left(a_1, \ldots, a_{Q_{t-1}}\right)^T$ satisfying the ALD criterion

$$\delta_1(t) \leq \nu_1, \tag{4.1}$$

where

$$\delta_1(t) \stackrel{def}{=} \min_{\mathbf{a}} \left\| \sum_{j=1}^{Q_{t-1}} a_j \phi\left(\tilde{\mathbf{x}}_j\right) - \phi\left(\mathbf{x}_t\right) \right\|^2, \tag{4.2}$$

and ν_1 is the sparsity level parameter (a hyperparameter that should be optimized). By expanding the minimization problem in Eq. (4.2) and applying the kernel trick, $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \left\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \right\rangle$,

[1] also known as codebook (GARCIA; FORSTER, 2012).

one can write

$$\delta_1(t) \stackrel{def}{=} \min_{\mathbf{a}} \left\{ \sum_{i,j=1}^{Q_{t-1}} a_i a_j \kappa\left(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j\right) - 2 \sum_{j=1}^{Q_{t-1}} a_j \kappa\left(\tilde{\mathbf{x}}_j, \mathbf{x}_t\right) + \kappa\left(\mathbf{x}_t, \mathbf{x}_t\right) \right\}, \tag{4.3}$$

or, using the matrix notation,

$$\delta_1(t) = \min_{a} \left\{ \mathbf{a}^T \tilde{\mathbf{K}}_{t-1} \mathbf{a} - 2\mathbf{a}^T \tilde{\mathbf{k}}_{t-1}\left(\mathbf{x}_t\right) + k_{tt} \right\}, \tag{4.4}$$

where $\tilde{\mathbf{K}}_{t-1}$ is a $Q_{t-1} \times Q_{t-1}$ kernel matrix built using the current dictionary. The (i,j)-th entry of this matrix is given by $\left[\tilde{\mathbf{K}}_{t-1}\right]_{i,j} = \kappa\left(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j\right)$, with $i, j = 1, \ldots, Q_{t-1}$. The Q_{t-1}-dimensional vector $\tilde{\mathbf{k}}_{t-1}$ is defined as

$$\tilde{\mathbf{k}}_{t-1}\left(\mathbf{x}_t\right) = \left[\kappa(\mathbf{x}_t, \tilde{\mathbf{x}}_1) \;\; \cdots \;\; \kappa(\mathbf{x}_t, \tilde{\mathbf{x}}_i) \;\; \cdots \;\; \kappa(\mathbf{x}_t, \tilde{\mathbf{x}}_{Q_{t-1}})\right]^T, \tag{4.5}$$

while $k_{tt} = \kappa\left(\mathbf{x}_t, \mathbf{x}_t\right)$. Solving Eq. (4.4) leads to the optimal $\mathbf{a}_t$, which is given by

$$\mathbf{a}_t = \tilde{\mathbf{K}}_{t-1}^{-1} \tilde{\mathbf{k}}_{t-1}\left(\mathbf{x}_t\right), \tag{4.6}$$

so that Eq. (4.2) can be rewritten as

$$\delta_1(t) = k_{tt} - \tilde{\mathbf{k}}_{t-1}\left(\mathbf{x}_t\right)^T \mathbf{a}_t \tag{4.7}$$

or

$$\delta_1(t) = k_{tt} - \tilde{\mathbf{k}}_{t-1}\left(\mathbf{x}_t\right)^T \tilde{\mathbf{K}}_{t-1}^{-1} \tilde{\mathbf{k}}_{t-1}\left(\mathbf{x}_t\right). \tag{4.8}$$

If $\delta_1(t) > \nu_1$, then the sample $(\mathbf{x}_t, c_t)$ must be added to the dictionary; that is, $\mathscr{D}_t = \mathscr{D}_{t-1} \cup \{(\mathbf{x}_t, c_t)\}$ and $Q_t = Q_{t-1} + 1$. However, if $\delta_1(t) \leq \nu_1$, the sample is approximate linear dependent and must not be added to the dictionary; that is, $\mathscr{D}_t = \mathscr{D}_{t-1}$ and $Q_t = Q_{t-1}$.

For classification purposes, the ALD-based selection of prototype vectors for the dictionary can be carried out in two straightforward ways, which are described next.

Design Method 1 - Randomly select an initial data sample. This sample becomes the first element of the dictionary. Then, take the remaining samples of the training dataset one by one and apply the ALD criterion according to (4.6) and (4.7). Note that each prototype vector in $\mathscr{D}_t$ carries its class label for the sake of classification. The classifier designed using this method will hereafter be referred to as KNNC-ALD-1 (*kernel nearest neighbor classifier via ALD criterion 1*).

Design Method 2 - According to this method, one dictionary must be built for each class. For a problem with C classes, it is required C dictionaries $\mathscr{D}_t^{(c)}$, $c = 1, 2, \ldots, C$. To construct

them, apply Design Method 1 to the data samples of the c-th class, $c = 1, 2, \ldots, C$. Repeat this procedure for all classes individually. Then, merge the class-conditional dictionaries into a single larger dictionary: $\mathscr{D}_t = \mathscr{D}_t^{(1)} \cup \mathscr{D}_t^{(2)} \cup \cdots \cup \mathscr{D}_t^{(C)}$. The classifier designed using this method will hereafter be referred to as KNNC-ALD-2 (*kernel nearest neighbor classifier via ALD criterion 2*).

For the classification of an incoming sample, use the kernelized distance in Eq. (3.4) in order to find the closest prototype. The search is executed over all the samples in the model's dictionary. Assign to that sample, the same class of the nearest prototype.

It should be noted that the only hyperparameters of the proposed approach are ν_1 (the sparsity level) and those associated with the chosen kernel function (such as the scale parameter σ_κ of the Gaussian kernel). However, since the kernel parameters are common to all kernel-based methods, the only tunable parameter specific to the proposed approach is the sparsity level ν_1.

Also, it is important to mention that there are alternative ways to achieve sparse prototype-based models, such as those described in Hofmann *et al.* (2014), Albuquerque *et al.* (2018), and Soares Filho e Barreto (2014). More on the issue of sparse dictionary learning can also be found in Mairal *et al.* (2010). The ALD method was initially chosen because it can be regarded as an approximation of PCA in the feature space (ENGEL *et al.*, 2004), since it implies that eigenvectors with eigenvalues that are significantly larger than ν_1 are projected almost entirely onto the span of the dictionary vectors. Consequently, the dictionary contains vectors that effectively represent the directions of the largest variances of the data in the feature space.

4.2 Evaluation of the KNNC-ALD on Batch Learning

The preliminary simulation results of the kernel nearest neighbor classifier via ALD criterion (KNNC-ALD) are reported in this section, evaluating the classification performance of the model with four different kernel functions (linear, Gaussian, Cauchy, and log) when applied to real-world datasets. It is important to note that the hyperparameter θ of the linear kernel was set to 0, and the hyperparameter l of the log kernel was set to 2 to simplify the hyperparameter analysis. As in the experiments of Chapter 3, all models were implemented from scratch in MATLAB (version R2023b), running on Windows 10 Home on an HP notebook with a Core i7-7500U processor, 2.70 GHz, and 16 GB of RAM.

Table 18 – Preliminary evaluation of the proposed KNNC-ALD classifiers with Iris dataset and linear kernel.

Method	ν_1	**Kernel**	σ_κ	acc_tr	acc_ts	#Prot	#c1	#c2	#c3
KNNC-ALD-1	0.001	linear	.	95.6	90.5	49	15	13	21
KNNC-ALD-1	**0.01**	**linear**	.	**86.7**	**84.9**	**12**	**3**	**3**	**6**
KNNC-ALD-1	0.1	linear	.	75.9	73.9	5	1	2	2
KNNC-ALD-2	0.001	linear	.	99.5	93.6	89	30	30	29
KNNC-ALD-2	0.01	linear	.	94.2	91.2	30	10	10	10
KNNC-ALD-2	**0.1**	**linear**	.	**90.7**	**89.0**	**13**	**4**	**4**	**5**

4.2.1 Initial Tests

The tests were started with the *Iris dataset*[2], which is one of the benchmarking datasets widely used in the literature on classification methods and contains three classes of 50 instances each, with each class representing a different species of iris plant. This dataset has been used in this section as a sanity check to verify the correctness of the implemented algorithm. It was also used to analyze how the number of prototypes and the classifier accuracy vary with changes in the hyperparameter ν_1, the kernel functions and their hyperparameters, and the dictionary building method.

The results of the proposed approaches using the linear kernel are presented in Table 18. Since this kernel function has no hyperparameters ($\kappa(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$), only the dictionary building method and the sparsity level ν_1 are analyzed here. It should be noted that larger values of ν_1 lead to dictionaries with fewer prototypes, as it becomes harder for a new sample not to be considered approximate linear dependent (see Eq. (4.7)). For small enough values of ν_1, all samples from the training dataset will be used as prototypes. Comparing the results highlighted in boldface in Table 18, it can be observed that high accuracy rates are achieved by the KNNC-ALD-2 classifier with basically the same number of prototypes as the KNNC-ALD-1, in both the training and test sets. It is also worth noting that this quantity of prototypes corresponds to approximately 12% of the entire training set.

When using the Gaussian kernel, for example, increasing the value of the scale parameter σ_κ decreases the number of selected prototypes. To achieve good classification performance with a reduced number of prototypes, both ν_1 and σ_κ should be optimized. Furthermore, with this kernel function, the KNNC-ALD-2 classifier achieved the best results. These results are presented in Table 19.

[2] https://doi.org/10.24432/C56C76

Table 19 – Preliminary evaluation of the proposed KNNC-ALD classifiers with Iris dataset and Gaussian kernel.

Method	ν_1	**Kernel**	σ_κ	acc_tr	acc_ts	#Prot	#c1	#c2	#c3
KNNC-ALD-1	**0.001**	**Gaussian**	**20**	**98.7**	**93.1**	**88**	**30**	**28**	**30**
KNNC-ALD-1	0.01	Gaussian	20	91.4	89.9	21	7	5	9
KNNC-ALD-1	0.1	Gaussian	2	94.9	91.5	24	7	7	10
KNNC-ALD-2	**0.001**	**Gaussian**	**20**	**100**	**93.2**	**99**	**33**	**32**	**34**
KNNC-ALD-2	0.01	Gaussian	20	95.4	90.9	32	9	10	13
KNNC-ALD-2	0.1	Gaussian	2	95.7	91.7	30	8	10	12
KNNC-ALD-2	0.1	Gaussian	5	93.6	91.1	18	5	5	8
KNNC-ALD-2	0.1	Gaussian	10	91.1	88.2	12	3	4	5

4.2.2 More General Tests

In the following experiments, for each evaluated dataset, eight variants of the proposed algorithm are tested, consisting of four different kernel functions (linear, Gaussian, Cauchy, and log), and two design methods (KNNC-ALD-1 and KNNC-ALD-2).

Also, similar to the first set of experiments for the KSOM algorithms (Section 3.3), ten independent runs are performed. For each run, the following steps are carried out:

- (*i*) Holdout procedure: partition the data into training and test sets;
- (*ii*) Z-score normalization: compute the empirical mean μ_z and standard deviation σ_z vectors of the training samples, and normalize the feature vectors so that all attributes have zero empirical mean and unit variance;
- (*iii*) Hyperparameter optimization: use the training dataset and 100 trials of random search to identify the best combination of hyperparameters;
- (*iv*) Training: update the model parameters using the training dataset;
- (*v*) Performance evaluation: use the test data to assess the trained model.

For the holdout step, the data are randomly divided into 70% for training and 30% for testing. At the end of the testing phase, several statistical performance metrics can be computed for each classifier, including accuracy, error rate, and F1-score (macro-averaged in the case of multiclass problems). In this chapter, only accuracy is used to compare the classifiers' performances.

Finally, a 5-fold cross-validation strategy is performed to search for the optimal values of the hyperparameter ν_1 (from the ALD criterion) and those associated with each kernel function. The figure of merit used for evaluating the performances of the algorithms while

choosing the optimal hyperparameters is given by

$$J_{PB}(\mathscr{D}, E_b) = E_b + \lambda D_s, \tag{4.9}$$

where E_b is the classifier's batch error ($0 \leq E_b \leq 1$), D_s ($0 \leq D_s \leq 1$) is the ratio between the number of samples Q selected for the dictionary $\mathscr{D}$ and the total number of training samples N_{tr}, and λ is a weighting term between these two factors ($\lambda \geq 0$). If $\lambda = 0$, the number of prototypes is not considered. By increasing λ, the J_{PB} index penalizes hyperparameters that lead the algorithm to build dictionaries with a large number of prototypes.

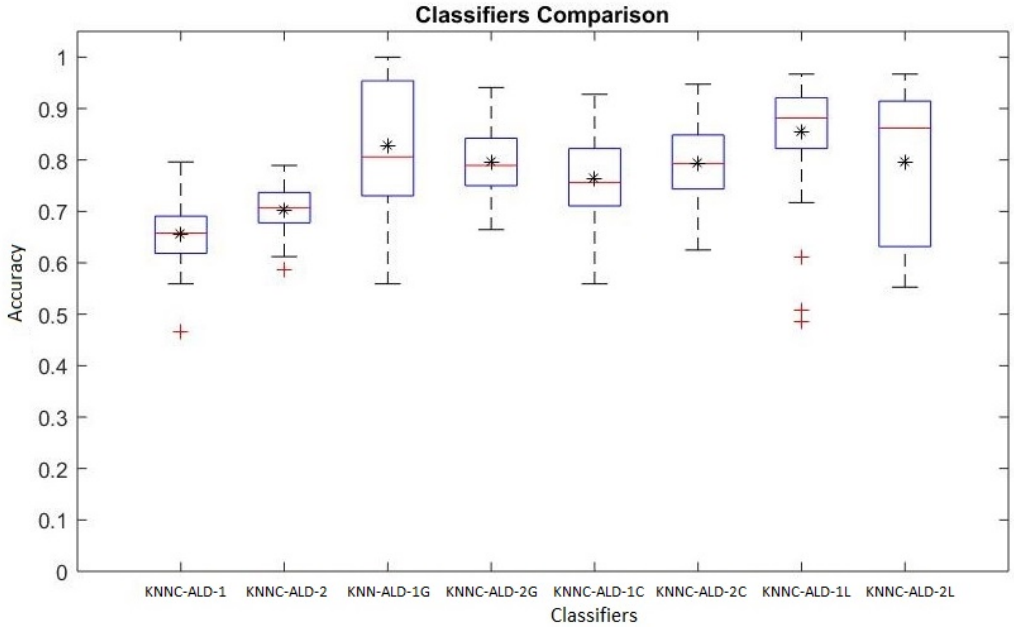
It is important to mention that there are two restrictions on the choice of hyperparameters. First, the minimum number of samples selected for the dictionary must be greater than the number of classes in the problem. This restriction prevents hyperparameter values that do not allow the addition of prototypes to the model, as the model will add at least one prototype per class using design method 2. Second, in the hyperparameter optimization procedure, the maximum number of samples selected for the dictionary must be less than 50% of the total number of training samples. This prevents the selection of hyperparameter values that would lead to excessive and, hence, unnecessary insertions of prototypes into the dictionary.

The motor failure dataset (see Section 3.4) was the first to be investigated. The accuracy for the test set obtained with the dataset of 504 samples (252 per class) and $\lambda = 0.5$ (see Eq. (4.9)) is shown in Figure 6. The last letter ('G', 'C', or 'L') in the acronyms of this figure stands for the Gaussian, Cauchy, or Log kernel functions, respectively. If there is no letter after the numbers '1' or '2' in the acronym, the result refers to the linear kernel function. The same approach is used in Figure 7. First, it can be inferred that classification performance improves when nonlinear kernel functions are used. Considering the optimal performance achieved, with the Gaussian kernel and the KNNC-ALD-1 classifier, the following results were obtained: this classifier achieved 100% accuracy but required 62.5% of the training samples (220 samples out of 352) as prototypes

To evaluate the proposed classifiers on datasets with more attributes and samples, the Pap-smear (Cervical Cancer) dataset was selected (see Section 3.4). The classification problem for this dataset was also treated as binary. The results achieved by the proposed classifiers, using $\lambda = 1$,[3] are shown in Figure 7. The worst mean and maximum accuracy rates were achieved by using the log kernel. Finally, considering the best result of the proposed approach, obtained with

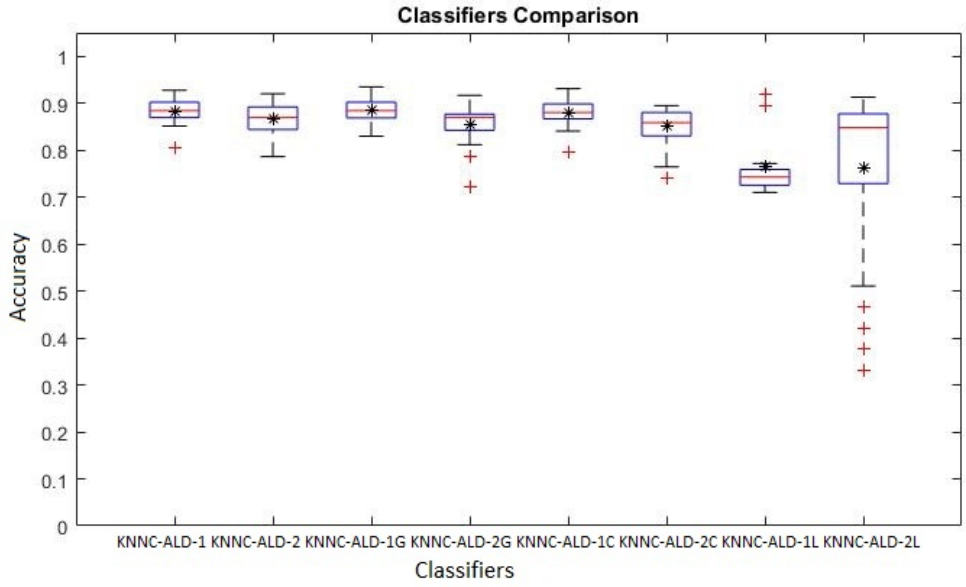
[3] This weighting term (λ) differs from the one used with the motor failure dataset, as this value resulted in better overall performance for the Pap-smear dataset.

Figure 6 – Results using the KNNC-ALD and the motor failure dataset.



Source: Author

Figure 7 – Results using the KNNC-ALD and the Cervical Cancer dataset.



Source: Author

the Gaussian kernel and the KNNC-ALD-1 classifier, the following results were achieved: this classifier reached 92% of accuracy while using only 3.27% of the training samples (21 out of 641).

4.3 Sparsification Procedures

Motivated by the results obtained by the KNNC-ALD model, other sparsification procedures were researched in order to build sparse kernel prototype based classifiers. Here these procedures are briefly described and the algorithm for the general framework sparse kernel (SPARK) is shown. It is important to mention that, for all sparsification procedures, both design methods can be used (one dictionary per class or just one dictionary for the entire dataset).

4.3.1 Coherence Criterion

Richard *et al.* (2009) proposed a sparsification strategy that employs a coherence parameter in order to control the model order increase. They applied this strategy for nonlinear filtering algorithms building in order to solve nonlinear dynamical systems identification.

They define the coherence parameter ψ as

$$\psi = \max_{i \neq j} \left| \kappa(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j) \right| \forall i, j \in \mathscr{D} \tag{4.10}$$

where κ is a unit-norm kernel[4]. This parameter reflects the most extreme correlations in a dictionary. Then, in time step t, they suggest inserting a new sample $(\mathbf{x}_t, c_t)$ to a dictionary $\mathscr{D}_{t-1}$ if the coherence of the increased dictionary remains below a given threshold ν_1, namely

$$\delta_1(t) = \max_{\forall \tilde{\mathbf{x}}_i \in \mathscr{D}_{t-1}} |\kappa(\tilde{\mathbf{x}}_i, \mathbf{x}_t)| \leq \nu_1 \tag{4.11}$$

where $\nu_1 \in [0, 1[$ determines both the sparsity level and the coherence of the dictionary. In summary, if $\delta_1(t) \leq \nu_1$, then the sample $(\mathbf{x}_t, c_t)$ must be added to the dictionary; that is, $\mathscr{D}_t = \mathscr{D}_{t-1} \cup \{(\mathbf{x}_t, c_t)\}$ and $Q_t = Q_{t-1} + 1$. However, if $\delta_t > \nu_1$, the sample is extremely correlated to at least one of the current samples and must not be added to the dictionary, so that $\mathscr{D}_t = \mathscr{D}_{t-1}$ and $Q_t = Q_{t-1}$.

It is important to mention that, to calculate the coherence measure, one does not need to compute the kernel matrix. Also, the computational complexity of this criterion is only linear in the dictionary size. Finally, similarly to the ALD criterion, it uses only the inputs in order to decide if a sample must be added or not to the dictionary. The next two criteria use both input and output of a sample in order to do this.

[4] This means that $\kappa(\mathbf{x}_t, \mathbf{x}_t) = 1$ for every $\mathbf{x}_t \in \mathbb{R}^P \subset \mathscr{X}$. Otherwise, one must substitute $\kappa(\tilde{\mathbf{x}}_i, \mathbf{x}_t) / \sqrt{\kappa(\mathbf{x}_t, \mathbf{x}_t) * \kappa(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_i)}$ for $\kappa(\tilde{\mathbf{x}}_i, \mathbf{x}_t)$ in Equation 4.11.

4.3.2 Novelty Criterion

Platt (1991) created a model named resource-allocating network (RAN), that allocates a new computational unit whenever an unusual pattern is presented to the network. This algorithm is used to predict complex time series, such as the chaotic time series created by the Mackey-Glass delay-difference equation. The RAN consists of a two-layer network, and uses a double-check novelty condition. An input-output pair $(\mathbf{x}_t, \mathbf{y}_t)$ is considered novel if the input is far away from existing centers (of the first layer), and if the difference between the desired output and the network output (the second layer output) is large. Here, these conditions were adapted, in order to build sparse kernel prototype-based classifiers, in the following manner.

Similarly to the other sparsification procedures, at time step t, a new sample $(\mathbf{x}_t, c_t)$ is presented to the model, which is composed by a dictionary $\mathscr{D}_{t-1} = \left\{(\tilde{\mathbf{x}}_j, \tilde{c}_j)\right\}_{j=1}^{Q_{t-1}}$. However, in order to be added to the dictionary, the new sample must fulfill two conditions. Firstly, the input of this sample must be far from the closest input of a prototype in the dictionary, considering some (dis)similarity measure such as

$$\delta_1(t) = \left\|\tilde{\mathbf{x}}_{j^*} - \mathbf{x}_t\right\|_2^2 > \nu_1, \tag{4.12}$$

where ν_1 as a hyperparameter and $\tilde{\mathbf{x}}_{j^*}$ is the closest prototype in the dictionary to the input sample. Here, the kernelized squared Euclidean measure

$$\begin{aligned} \delta_1(t) &= \left\|\phi(\tilde{\mathbf{x}}_{j^*}) - \phi(\mathbf{x}_t)\right\|_2^2 > \nu_1, \\ &= \kappa(\tilde{\mathbf{x}}_{j^*}, \tilde{\mathbf{x}}_{j^*}) + \kappa(\mathbf{x}_t, \mathbf{x}_t) - 2\kappa(\tilde{\mathbf{x}}_{j^*}, \mathbf{x}_t) > \nu_1 \end{aligned} \tag{4.13}$$

is used. The second condition is based on the difference between the desired output and the predicted one.

Firstly, one must calculate the predicted output $\hat{\mathbf{y}}_t$ using any KNN strategy. Then, if the difference between the real output $\mathbf{y}_t$ and the predicted output $\hat{\mathbf{y}}_t$ is greater than a hyperparameter ν_2, such as

$$\delta_2(t) = \|\mathbf{y}_t - \hat{\mathbf{y}}_t\|_2^2 > \nu_2, \tag{4.14}$$

the sample must be added to the dictionary.

This second condition can also be extended to classification problems as follows: if the current model H_t makes an error:

$$\hat{c}_t \neq c_t, \tag{4.15}$$

in other words, if the predicted class $\hat{c}_t = \arg\max \hat{\mathbf{y}}_t$ differs from the actual class $c_t = \arg\max \mathbf{y}_t$, the second condition is satisfied and the sample must be added to the dictionary.

When comparing these two ways of applying the second condition, it was observed that the error-based approach as shown in Eq. (4.15) yields the best results. Therefore, this approach is utilized for comparing the novelty with the other sparsification criteria.

As a final remark, an important feature of the novelty criterion is that, similarly to the coherence, one does not need to compute a kernel matrix as in the ALD criterion.

4.3.3 Surprise Criterion

Liu *et al.* (2009) introduced an information measure approach, called *Surprise*, in order to determine useful data to be learned and remove redundant ones. This criterion captures the amount of information a datum contains which is transferable to a learning system. The authors claim that methods like ALD, Coherence and Novelty are effective in practical applications, but are heuristic in nature. So, the surprise information criterion provides a unifying view on existing sparsification methods.

They define Surprise as the negative log likeklihood (NLL) of an observation, given the learning machine's hypothesis. Also, this measure is computed based on the theory of Gaussian process regression (GPR).

More formally, Surprise is a subjective information measure of a sample $(\mathbf{x}_t, c_t)$ with respect to a learning system H. Denoted by $S_H(\mathbf{x}_t, c_t)$, it is defined as the NLL of the exemplar given the learning system's hypothesis on the data distribution

$$S_H(\mathbf{x}_t, c_t) = -\ln p(\mathbf{x}_t, c_t|H), \tag{4.16}$$

where $p(\mathbf{x}_t, c_t|H)$ is the subjective probability of $(\mathbf{x}_t, c_t)$ hypothesized by H. Applying this definition directly to the active online learning problem, we have the surprise of $(\mathbf{x}_t, c_t)$ to the current learning system H_{t-1}, such as

$$S_{H_{t-1}}(\mathbf{x}_t, c_t) = -\ln p(\mathbf{x}_t, c_t|H_{t-1}), \tag{4.17}$$

where $p(\mathbf{x}_t, c_t|H_{t-1})$ is the posterior distribution of $(\mathbf{x}_t, c_t)$ hypothesized by H_{t-1}. For simplicity, they define $S_t = S_{H_{t-1}}(\mathbf{x}_t, c_t)$.

If $p(\mathbf{x}_t, c_t|H_{t-1})$ is very large, the new datum $(\mathbf{x}_t, c_t)$ is well expected by the learning system H_{t-1}. If $p(\mathbf{x}_t, c_t|H_{t-1})$ is small, the new datum "surprises" the learning system, which

means either the data contains something new for the system to discover or it is suspicious. According to this measure, we can classify the new exemplar into three categories:

- Redundant: $S_t < \nu_1$;
- Learnable: $\nu_1 \leq S_t \leq \nu_2$;
- Abnormal: $S_t > \nu_2$,

where ν_1 and ν_2 are hyperparameters. The choice of these thresholds are problem-dependent.

When developing this criterion with the GPR model, firstly, Liu *et al.* (2009) calculated the output prediction as

$$\hat{y}_t = \tilde{\mathbf{k}}_{t-1}^T(\mathbf{x}_t)\left[\sigma_n^2\mathbf{I} + \tilde{\mathbf{K}}_{t-1}\right]^{-1} y_{t-1}, \tag{4.18}$$

where $\tilde{\mathbf{k}}_{t-1}$ and $\tilde{\mathbf{K}}_{t-1}$ were already defined for the ALD criterion, σ_n^2 is the variance of the noise contained in an observation (here, we consider it a constant $\sigma_n^2 = 0.0001$), $\mathbf{I}$ is the identity matrix, and $\mathbf{y}_{t-1}$ is a vector containing all the prototypes outputs. Then, the authors calculate the prediction variance as

$$\sigma_t^2 = \sigma_n^2 + k_{tt} - \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^T\left[\sigma_n^2\mathbf{I} + \tilde{\mathbf{K}}_{t-1}\right]^{-1}\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t). \tag{4.19}$$

It is important to mention that, if the noise variance σ_n^2 is considered 0, Eq. (4.19) reduces to Eq. (4.8) of the ALD method.

Then, discarding constant terms, considering just one output, and applying the definition of Surprise in Eq. (4.16), this measure can be written as

$$S_t = \ln\sigma_t + \frac{(y_t - \hat{y}_t)^2}{2\sigma_t^2}. \tag{4.20}$$

Some observations were made. First, little surprise is expected when there is redundancy in the data. Second, the proposed measure offers a general framework for redundancy removal, anomaly detection, and knowledge discovery. Third, the measure is proportional to both the magnitude of the prediction error and the prediction variance (particularly when the error is very small). Finally, the surprise measure becomes large when the prediction error is high and the variance is low.

Here, for prototype-based models, this measure was adapted to

$$S_t = \ln\sigma_t + \frac{\|\mathbf{y}_t - \hat{\mathbf{y}}_t\|_2^2}{2\sigma_t^2}, \tag{4.21}$$

where $\mathbf{y}_t$ and $\hat{\mathbf{y}}_t$ are, respectively, the sample's encoded output and the prototype-based model's encoded prediction, and σ_t is calculated using Eq. (4.19).

4.3.4 The Sparse Kernel Prototype-Based classifiers Framework

Algorithms 2 and 3 represent the training functions of the SPARK framework using, respectively, design methods 1 and 2. After the model's training (construction of a dictionary of samples), a classification strategy such as KNNC or KWKNN can be used for pattern recognition.

An important feature of this framework is that, after the hyperparameters optimization step, in order to build a classification model, the training algorithm just need one epoch, meaning that a sample is just used once, improving the training step speed.

Algoritmo 2: spark_training_design_method_1()

Input: $(\mathbf{x}_n, c_n)\}_{n=1}^{N}$, $\kappa(\cdot,\cdot)$, ν_1, ν_2
Output: $\mathscr{D}$
begin
  # *Add first sample to the dictionary:*
  $\mathscr{D} \leftarrow (\mathbf{x}_1, c_1)$; $Q \leftarrow 1$
  for $n = 2 : N$ **do**
    Get new input-output pair: $(\mathbf{x}_n, c_n)$
    Apply a sparsification criterion considering $\mathscr{D}$ and $(\mathbf{x}_n, c_n)$
    (e.g. ALD, Coherence, Novelty, Surprise)
    if *sparsification criterion was met* **then**
      # *Add Prototype to Dictionary*
      $\mathscr{D} \leftarrow \mathscr{D} \cup \{(\mathbf{x}_n, y_n)\}$
      $Q \leftarrow Q + 1$;
    end
  end
end

In the next section, a comparison of the sparsification procedures in the task of sparse kernel prototype-based classifiers building is made.

4.4 Classification using the SPARK Framework

Here, the experimental setup used for the KSOM model in Section 3.3 was also applied, and the same datasets described in Section 3.4 were explored to evaluate the SPARK model. The only differences are that the KWKNN is also tested for pattern classification, and, in this framework, there is no need for a labeling strategy. Considering all possibilities, there were 64 tests for each dataset (4 sparsification strategies, 2 design methods, 8 kernel functions). It is important to mention that, as the KWKNN classifier is used, the hyperparameter K is also optimized. Finally, in the tables presented in the following subsections, SM and DM stand for,

Algoritmo 3: spark_training_design_method_2()

Input: $(\mathbf{x}_n, c_n)\}_{n=1}^{N}$, $\kappa(\cdot,\cdot)$, ν_1, ν_2
Output: $\mathscr{D}$
begin
  # Add first sample to a class-conditional dictionary:
  $\mathscr{D}^{(c_1)} \leftarrow (\mathbf{x}_1, c_1)$; $Q^{(c_1)} \leftarrow 1$;
  for $n = 2 : N$ **do**
    Get new input-output pair: $(\mathbf{x}_n, c_n)$
    if $\mathscr{D}^{(c_n)} = \varnothing$ **then**
      # Add first sample to a class-conditional dictionary:
      $\mathscr{D}^{(c_n)} \leftarrow \mathscr{D}^{(c_n)} \cup \{(\mathbf{x}_n, y_n)\}$
      $Q^{(c_n)} \leftarrow Q^{(c_n)} + 1$;
    else
      Apply a sparsification criterion considering $\mathscr{D}^{(c_n)}$ and $(\mathbf{x}_n, c_n)$
      (e.g. ALD, Coherence, Novelty, Surprise)
      if *sparsification criterion was met* **then**
        # Add Prototype to class-conditional Dictionary
        $\mathscr{D}^{(c_n)} \leftarrow \mathscr{D}^{(c_n)} \cup \{(\mathbf{x}_n, y_n)\}$
        $Q^{(c_n)} \leftarrow Q^{(c_n)} + 1$;
      end
    end
  end
  $\mathscr{D} = \mathscr{D}^{(1)} \cup \mathscr{D}^{(2)} \cup \cdots \cup \mathscr{D}^{(C)}$; $Q = Q^{(1)} \cup Q^{(2)} \cup \cdots \cup Q^{(C)}$;
end

respectively, *sparsification method* and *design method*. Additionally, the three best results, in each row (for each model's configuration), are highlighted in boldface. This last feature of the tables is used to highlight the best kernel functions for each dataset.

To establish a baseline for the results of the SPARK model (along with those already established in the previous chapter), the mean accuracy from other works that used the same datasets is displayed in Table 20. For the motor failure dataset, an LSSVM achieved 99.5% performance, but it required all the training samples as support vectors (in this case, 80% of the entire dataset). For the cervical cancer dataset, the authors achieved an accuracy of 93.4%, similar to the ones reported in Table 4, as they also used a linear classifier. For the vertebral column dataset, an accuracy of 85.9% was reported using an SVM with the KMOD kernel function. Finally, for the wall-following dataset, a mean accuracy of 96.8% was obtained using a single-hidden-layer MLP.

In the next subsections, the results with each data set are described in detail.

Table 20 – Summary of performances of other works.

Dataset	Mean Accuracy	Model	Reference
Motor Failure	99.5	LSSVM with Gaussian kernel	(COELHO *et al.*, 2014)
Cervical Cancer	93.4	Linear Classifier	(JANTZEN *et al.*, 2005)
Vertebral Column	85.9	SVM with KMOD kernel	(ROCHA NETO, 2011)
Wall Following	96.8	MLP with one hidden layer	(FREIRE *et al.*, 2009)

Table 21 – Performances of the SPARK models for the Motor Failure dataset (1st set of experiments).

SM	DM	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	1	68.4± 3.0	83.8± 11.7	77.8± 6.6	**89.0± 10.2**	87.8± 8.6	**89.1± 3.9**	66.7± 6.4	**89.2± 12.8**
	2	71.0± 3.4	82.6± 12.3	80.2± 4.0	**84.2± 9.5**	**84.3± 13.3**	80.7± 16.1	69.4± 4.9	**88.6± 13.2**
COH	1	77.5± 9.3	83.2± 13.2	83.9± 7.1	87.4± 13.0	**96.6± 8.6**	**92.2± 2.7**	80.1± 9.4	**98.4± 1.8**
	2	82.0± 6.4	85.7± 14.6	85.0± 5.6	83.9± 12.5	**93.3± 9.4**	**92.2± 2.4**	86.3± 7.9	**90.9± 11.0**
NOV	1	89.5± 3.6	**99.7± 0.8**	89.3± 2.3	**99.9± 0.3**	89.8± 3.3	90.4± 3.6	85.0± 4.0	**95.5± 3.6**
	2	87.8± 3.4	**99.9± 0.4**	88.4± 2.8	**99.3± 1.2**	91.1± 2.2	89.5± 3.5	84.5± 3.7	**96.1± 5.1**
SUR	1	70.5± 4.9	**93.4± 5.3**	76.6± 3.8	92.6± 3.4	**93.6± 3.5**	81.8± 11.7	68.6± 3.5	**92.9± 3.3**
	2	71.4± 4.7	89.4± 7.5	78.4± 7.7	**90.1± 3.5**	**93.3± 3.2**	88.8± 6.8	70.3± 3.5	**90.8± 8.0**

4.4.1 SPARK and the Motor Failure Dataset

In Table 21, the performance of each SPARK model configuration applied to the motor failure dataset, for the first set of experiments (where the hyperparameter optimization step is done for each experiment), is shown. The best performance (99.9 ± 0.4) was obtained using the *novelty* sparsification strategy combined with the *Gaussian* kernel function and design method 2. As can be seen in Table 22, this performance was achieved using only 12.5% of the training dataset samples as prototypes (44 samples out of 352). Comparing the kernel functions, *kmod*, *exponential*, and *Cauchy* also led the algorithm to reach high performance. Moreover, comparing the design methods, both performed better half of the time. However, *design method 2*, being the best method, caused a greater increase in accuracy.

In Table 23, the performance of each SPARK model configuration applied to the motor failure dataset, for the second set of experiments (using the best hyperparameters achieved

Table 22 – Number of Prototypes of the SPARK models and Motor Failure Dataset (1st set of experiments).

SM	DM	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	1	6	229	91	212	216	281	21	228
	2	9	228	100	221	229	281	10	220
COH	1	169	219	139	224	224	281	174	214
	2	190	221	104	222	215	281	141	230
NOV	1	71	46	58	45	61	75	75	48
	2	69	44	67	45	72	63	61	50
SUR	1	7	210	87	170	110	242	31	173
	2	15	217	137	104	137	240	40	140

Table 23 – Mean Accuracy of SPARK and Motor Failure Dataset (2nd set of experiments).

SM	DM	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	1	62.5± 6.5	**93.4± 1.8**	79.6± 3.7	**93.4± 1.9**	**97.3± 1.5**	86.6± 14.5	65.7± 9.4	93.1± 2.0
	2	70.3± 4.7	93.5± 2.0	86.4± 3.1	**94.6± 1.9**	**97.4± 1.4**	91.4± 2.4	63.8± 7.1	**99.5± 0.8**
COH	1	90.9± 2.4	**93.7± 1.9**	88.7± 3.1	89.6± 2.5	**98.8± 1.3**	**92.1± 2.2**	90.1± 2.1	78.6± 3.5
	2	**93.2± 1.9**	80.7± 24.5	92.4± 2.2	**96.3± 1.9**	**99.7± 0.7**	92.0± 2.4	91.4± 2.3	86.4± 3.0
NOV	1	89.3± 3.0	**99.9± 0.7**	89.6± 2.6	**99.8± 0.7**	90.7± 2.8	90.3± 2.7	87.1± 2.9	**99.8± 0.5**
	2	89.7± 2.5	**99.9± 0.4**	89.7± 2.7	**99.6± 0.8**	91.4± 3.1	91.3± 2.6	89.4± 3.0	**99.2± 0.8**
SUR	1	61.8± 6.7	92.7± 2.0	80.2± 4.0	**93.5± 2.1**	**96.2± 1.7**	93.4± 1.6	66.4± 4.8	**97.8± 1.1**
	2	66.5± 5.6	88.3± 3.1	88.9± 2.8	**94.1± 2.5**	**95.8± 1.9**	93.2± 2.0	73.8± 6.5	**96.4± 1.6**

in the first set of experiments) is shown. The best performance was also reached with the *novelty* sparsification strategy combined with the *Gaussian* kernel function and *design method 2*. Comparing the kernel functions, the *kmod*, *exponential* and *Cauchy* also led the algorithm to reach high performance. Moreover, comparing the design methods, the *design Method 2* was, most of the time, the best one. As can be seen, the best result from the second set of experiments (99.9 ± 0.4) was similar to that of the first set, indicating that once the best hyperparameters are found, shuffling the data again and dividing it into different training and test sets does not negatively affect classifier performance. Moreover, this best result is 32.7% higher than the result obtained with the linear classifier reported in Table 4, and 22% higher than the result obtained with the KSOM model (with 30 prototypes) reported in Chapter 3. Finally, this result was similar to the one reported in (COELHO *et al.*, 2014), as shown in Table 20, but the SPARK model only required 44 prototypes, instead of the entire dataset.

Table 24 – Number of Prototypes of SPARK and Motor Failure Dataset (2nd set of experiments).

SM	DM	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	1	7	223	102	212	226	281	26	225
	2	21	221	98	218	220	281	9	224
COH	1	192	225	166	187	225	281	163	110
	2	188	220	162	228	214	281	150	129
NOV	1	65	46	72	46	79	77	63	47
	2	80	44	69	47	79	71	67	50
SUR	1	7	225	91	230	113	226	14	180
	2	14	190	140	135	146	242	38	146

In Table 25, the summary of results for the motor failure dataset is shown. In this table, as in the summary tables that will be presented for each dataset, in the *comparisons column*, the *best result* is the combination of sparsification strategy, design method, kernel function, and number of nearest neighbors that achieved the best performance. Moreover, the *best design method* is the model that, fixing the same kernel function and sparsification strategy, most often led to the highest performance. Furthermore, the *best sparsification* is the sparsification strategy that, fixing the same kernel function and design method, most often led to the highest performance. Finally, the *best kernels* are the kernels that, fixing the design method and the sparsification strategy, most often led to the best three mean accuracies.

An important thing to note is that, for this dataset, in both sets of experiments, the same configuration achieved the best result: design method 2, novelty sparsification strategy, and Gaussian kernel.

4.4.2 SPARK and the Cervical Dataset

In table 26, the performance of each SPARK model configuration applied to the cervical cancer dataset, for the first set of experiments (where the hyperparameters optimization step is done for each experiment) is shown. The best performance (92.9 ± 2.1, with 109 prototypes) was achieved when using the *ALD* sparsification strategy combined with the *log* kernel and the *design method 2*. Also, a more balanced result between accuracy and number of prototypes (91.2 ± 2.5, with 22 prototypes) was achieved with the *surprise sparsification strategy* combined with the *sigmoidal* kernel and the *design method 2*. Comparing the kernel functions, the *log*, *linear* and *sigmoidal* were the ones the lead the algorithm to reach the best results. Moreover, comparing the design methods, the *Design Method 2* led the algorithm to achieve the highest performances.

Table 25 – Summary of Results SPARK and Motor Failure Dataset.

Dataset	Experiment	Comparisons	
Motor Failure	Optimizing Hyperparameters	Best Result	novelty design method 2 Gaussian 3NN
		Best Design Method	-
		Best Sparsification	novelty
		Best Kernels	kmod or exponential or Cauchy
	Using Best Hyperparameters	Best Result	novelty design method 2 Gaussian 3NN
		Best Design Method	design method 2
		Best Sparsification	novelty or coherence or surprise
		Best Kernels	kmod exponential Cauchy

Table 26 – Mean Accuracy of SPARK and Cervical Cancer Dataset (1st set of experiments).

SM	DM	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	1	**89.1± 4.3**	86.9± 3.5	89.4± 3.3	88.6± 3.6	**90.4± 3.4**	**92.2± 1.2**	87.8± 7.6	86.3± 7.1
	2	**90.6± 3.4**	89.2± 3.0	89.4± 3.2	85.6± 5.9	88.0± 4.8	**92.9± 2.1**	**90.2± 3.0**	89.1± 3.6
COH	1	**87.5± 3.6**	86.8± 8.8	**87.9± 2.3**	86.2± 4.7	**87.5± 4.3**	50.0± 39.4	86.2± 3.8	86.2± 7.7
	2	**90.2± 4.6**	86.4± 3.6	**88.5± 2.3**	86.7± 4.4	84.3± 6.3	83.8± 16.0	**89.9± 3.5**	86.0± 7.3
NOV	1	**87.9± 3.5**	86.7± 3.8	87.2± 2.9	**88.5± 4.0**	**88.5± 4.5**	84.2± 3.5	86.0± 7.7	86.8± 4.5
	2	**77.4± 26.5**	61.1± 36.6	**79.7± 13.3**	51.3± 36.5	**77.4± 27.7**	**79.1± 16.1**	71.8± 25.4	56.0± 39.3
SUR	1	79.3± 19.6	87.8± 7.2	**89.9± 3.8**	87.8± 3.0	**88.2± 5.1**	**88.8± 6.8**	**88.3± 6.1**	86.6± 4.9
	2	**88.3± 4.6**	**88.0± 4.7**	85.5± 2.3	87.3± 6.2	87.4± 7.6	**90.2± 6.2**	**91.2± 2.5**	86.9± 5.3

Table 27 – Number of Prototypes of SPARK and Cervical Cancer Dataset (1st set of experiments).

SM	DM	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	1	24	53	39	3	4	270	7	9
	2	20	23	19	16	13	109	16	38
COH	1	18	9	13	49	48	1	3	27
	2	18	31	15	6	4	2	11	23
NOV	1	11	17	47	9	9	11	14	59
	2	2	2	11	2	2	2	2	2
SUR	1	260	139	26	106	120	67	11	137
	2	20	15	42	3	4	6	22	26

Table 28 – Mean Accuracy of SPARK and Cervical Cancer Dataset (2nd set of experiments).

SM	DM	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	1	87.6± 5.3	**89.4± 6.1**	**89.2± 5.5**	82.8± 8.3	83.1± 7.0	**90.3± 12.1**	88.2± 8.7	80.4± 10.2
	2	**89.4± 6.4**	85.6± 6.9	85.7± 6.2	81.3± 15.2	80.3± 15.4	**93.9± 2.0**	**90.0± 7.2**	88.9± 6.6
COH	1	90.4± 5.8	76.6± 20.8	**90.5± 3.8**	87.5± 6.0	87.9± 6.0	**92.4± 1.1**	**92.0± 2.0**	82.7± 9.6
	2	**90.8± 5.0**	87.1± 4.6	78.3± 10.9	79.8± 6.3	76.8± 8.7	**90.3± 1.2**	**90.4± 1.3**	83.5± 6.5
NOV	1	81.2± 8.1	86.2± 5.5	**87.9± 3.1**	81.6± 12.4	84.9± 2.6	**87.1± 4.3**	85.5± 5.8	**88.1± 2.9**
	2	**87.3± 2.9**	84.7± 2.6	**84.8± 2.6**	**84.8± 2.4**	84.4± 2.9	81.2± 18.3	78.9± 16.9	**86.2± 2.6**
SUR	1	**92.2± 3.2**	51.5± 37.6	89.1± 2.3	**91.8± 1.3**	**91.9± 1.2**	62.0± 35.6	81.9± 19.9	90.5± 1.9
	2	**92.3± 2.6**	**90.4± 1.2**	89.1± 4.4	**90.0± 1.2**	89.2± 4.4	76.1± 20.6	84.4± 15.3	**91.3± 1.4**

In Table 28, the mean accuracy of each SPARK model configuration applied to the cervical cancer dataset, for the second set of experiments (using the best hyperparameters achieved in the first set of experiments) is shown. The best performance (93.9 ± 2.0, with 89 prototypes) was achieved when using the *ALD* sparsification strategy combined with the *log* kernel and the *design method 2*. Comparing the kernel functions, the *log*, *linear* and *sigmoidal* were also the ones that lead the algorithm to reach the best results. Moreover, Comparing the design methods, both were better half of the time, however, being the best method, *design method 2* causes a greater increase in accuracy. As can be seen, the best result from the second set of experiments (93.9 ± 2.0, with 89 prototypes) is 3.8% higher than the result obtained with the KSOM model (with 30 prototypes) reported in Chapter 3. Also, it was similar to the result obtained by the linear classifier reported in Table 20.

In Table 30, the summary of results for the cervical cancer dataset is shown. An

Table 29 – Number of Prototypes of SPARK and Cervical Cancer Dataset (2nd set of experiments).

SM	DM	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	1	25	53	41	38	42	270	10	19
	2	19	29	21	17	14	89	15	36
COH	1	19	6	88	52	45	270	53	23
	2	19	33	9	20	8	270	270	19
NOV	1	9	17	52	9	31	30	15	56
	2	61	48	68	60	69	2	2	60
SUR	1	61	1	132	126	144	1	10	113
	2	65	270	35	270	23	2	33	143

Table 30 – Summary of Results SPARK and Cervical Cancer Dataset.

Dataset	Experiment	Comparisons	
Motor Failure	Optimizing Hyperparameters	Best Result	ALD design method 2 log 5NN
		Best Design Method	Design Method 2
		Best Sparsification	ALD
		Best Kernels	log linear sigmoidal
	Using Best Hyperparameters	Best Result	ALD design method 2 log 5NN
		Best Design Method	Design Method 2
		Best Sparsification	ALD
		Best Kernels	log linear sigmoidal

important thing to note is that, for this dataset, in both sets of experiments, the same configuration achieved the best result: design method 2, ALD sparsification strategy, and log kernel.

4.4.3 SPARK and the Vertebral Column Dataset

In Table 31, the performance of each SPARK model configuration applied to the vertebral column dataset, for the first set of experiments (where the hyperparameters optimization step is done for each experiment) is shown. The best performance (81.7 ± 4.0, with 173 prototypes) was achieved when using the *coherence* sparsification strategy combined with the *log* kernel and the *design method 2*. Also, a more balanced result between accuracy and number

Table 31 – Mean Accuracy of SPARK and Vertebral Column Dataset (1st set of experiments).

SM	DM	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	1	71.8± 7.1	72.4± 7.8	**77.0± 5.7**	72.0± 6.9	74.6± 6.8	**79.8± 4.9**	**76.2± 4.9**	75.6± 9.0
	2	73.1± 6.4	74.6± 8.0	**74.1± 5.2**	**75.6± 7.2**	74.0± 5.4	**75.1± 7.4**	73.9± 5.3	73.5± 7.0
COH	1	77.4± 6.9	76.5± 5.1	76.2± 5.1	**79.6± 4.2**	**77.5± 2.7**	**79.8± 2.9**	76.1± 4.5	76.9± 7.4
	2	76.5± 5.9	**79.2± 4.1**	**78.0± 3.7**	76.7± 4.1	77.6± 4.9	**81.7± 4.0**	77.6± 4.6	75.4± 4.3
NOV	1	74.7± 4.5	75.6± 4.8	75.5± 6.3	**77.4± 5.3**	76.9± 4.2	76.1± 5.4	**77.0± 2.0**	**78.6± 4.4**
	2	75.6± 4.6	74.6± 5.3	76.6± 6.1	76.2± 4.0	**77.5± 3.6**	**77.8± 6.1**	75.5± 7.6	**78.1± 5.2**
SUR	1	**76.8± 7.6**	**77.0± 4.0**	**76.6± 5.9**	73.8± 6.3	75.5± 5.3	71.9± 6.1	74.4± 7.4	75.6± 5.3
	2	73.9± 6.1	**76.5± 5.2**	**75.7± 6.1**	73.3± 8.2	73.3± 7.3	73.3± 7.5	73.5± 6.9	**75.9± 6.5**

Table 32 – Number of Prototypes of SPARK and Vertebral Column Dataset (1st set of experiments).

SM	DM	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	1	7	23	18	32	30	173	4	60
	2	14	23	9	28	21	173	5	24
COH	1	37	18	41	24	13	173	4	38
	2	7	15	28	33	7	173	7	52
NOV	1	17	14	20	32	15	18	32	12
	2	9	26	32	14	13	12	9	36
SUR	1	6	34	33	92	10	30	17	106
	2	11	30	21	8	22	57	20	39

of prototypes (79.2 ± 4.1, with 15 prototypes) was achieved with the *coherence* sparsification strategy combined with the *gaussian* kernel and the *design method 2*. Comparing the kernel functions, the *exponential*, *Gaussian* and *log* were, most of the time, the best ones. Moreover, comparing the design methods, the *Design Method 2* led the algorithm to reach the best results.

In Table 33, the performance of each SPARK model configuration applied to the vertebral column dataset, for the second set of experiments (using the best hyperparameters achieved in the first set of experiments) is shown. The best performance (80.9 ± 4.2, with 50 prototypes) was achieved when using the *coherence* sparsification strategy combined with the *kmod* kernel and the *design method 2*. Comparing the kernel functions, the *kmod* and *log* were the ones that lead the algorithm to reach the best results. Moreover, comparing the design methods, the *Design Method 1* was, most of the time, the best one. As can be seen, the best result from the second set of experiments (80.9 ± 4.2, with 50 prototypes) is 1% lower than the

Table 33 – Mean Accuracy of SPARK and Vertebral Column Dataset (2nd set of experiments).

SM	DM	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	1	68.5± 7.7	73.2± 5.9	74.3± 5.2	75.1± 4.7	**75.8± 5.5**	**79.8± 4.4**	66.1± 9.2	**77.9± 4.6**
	2	67.5± 7.5	76.5± 5.9	70.4± 6.1	**76.8± 5.6**	74.6± 5.9	**79.7± 6.6**	64.6± 10.1	**78.1± 5.6**
COH	1	**77.9± 4.8**	74.7± 4.7	77.2± 5.2	75.9± 5.9	73.3± 6.2	**80.4± 3.2**	71.9± 6.2	**77.4± 4.9**
	2	65.5± 9.5	71.8± 5.4	74.8± 5.3	**79.2± 4.8**	69.5± 5.2	**80.3± 3.7**	68.1± 8.4	**80.9± 4.2**
NOV	1	74.9± 5.1	**77.2± 5.5**	76.0± 4.4	**77.1± 5.2**	**76.5± 6.8**	**76.6± 5.2**	75.4± 5.4	72.2± 6.3
	2	70.6± 5.4	**76.3± 6.2**	**77.1± 4.9**	72.9± 7.4	71.9± 6.6	**76.1± 5.6**	69.8± 7.5	**78.1± 4.6**
SUR	1	67.6± 9.2	72.0± 5.9	75.0± 6.2	**80.0± 4.8**	**79.8± 3.5**	76.4± 5.3	74.6± 5.0	**78.4± 6.1**
	2	68.4± 7.3	71.1± 6.2	74.0± 5.6	64.4± 12.0	70.9± 9.5	**77.5± 4.4**	**76.0± 6.6**	**76.8± 5.1**

Table 34 – Number of Prototypes of SPARK and Vertebral Column Dataset (2nd set of experiments).

SM	DM	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	1	10	22	32	30	30	173	6	64
	2	14	28	20	28	21	173	6	27
COH	1	39	45	35	30	20	173	13	36
	2	7	41	35	45	6	173	9	50
NOV	1	19	15	20	29	18	17	20	12
	2	12	23	37	16	16	16	7	33
SUR	1	8	20	37	112	141	29	16	91
	2	12	33	35	9	17	52	25	38

result obtained with the KSOM model (with 30 prototypes) reported in Chapter 3. This shows that, for some datasets, updating the positions of prototypes is a better solution than just adding more prototypes to the model. Finally, this result was 5% lower than the one reported in Table 20, where a SVM with kmod kernel was used.

In Table 35, the summary of results for the vertebral column dataset is shown. An important thing to note is that, for this dataset, in both sets of experiments, the best result was achieved by the: design method 2 and the coherence sparsification strategy.

4.4.4 SPARK and the Wall Following Dataset

In Table 36, the performance of each SPARK model configuration applied to the Wall Following dataset, for the first set of experiments (where the hyperparameters optimization step

Table 35 – Summary of Results SPARK and Vertebral Column Dataset.

Dataset	Experiment	Comparisons	
Motor Failure	Optimizing Hyperparameters	Best Result	coherence design method 2 log 6NN
		Best Design Method	design method 2
		Best Sparsification	coherence
		Best Kernels	exponential Gaussian log
	Using Best Hyperparameters	Best Result	coherence design method 2 kmod 9NN
		Best Design Method	design method 1
		Best Sparsification	coherence
		Best Kernels	log, kmod

Table 36 – Mean Accuracy of SPARK and Wall Following Dataset (1st set of experiments).

SM	DM	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	1	49.8± 23.9	**91.6± 1.1**	78.4± 7.5	91.4± 2.7	**92.1± 3.2**	**92.1± 6.1**	54.7± 17.6	91.1± 3.0
	2	79.8± 6.0	**94.2± 2.3**	88.1± 2.5	**93.0± 3.1**	**92.5± 3.0**	82.1± 17.7	74.0± 9.3	90.3± 3.0
COH	1	90.6± 3.1	91.3± 1.9	90.9± 2.8	**92.5± 1.9**	**92.3± 1.2**	18.7± 17.3	86.0± 8.4	**92.6± 1.2**
	2	92.0± 1.4	93.6± 2.1	90.4± 4.9	**94.3± 1.7**	**94.0± 1.9**	**96.4± 0.3**	87.8± 5.3	93.9± 1.4
NOV	1	94.7± 1.4	94.8± 1.7	**95.3± 1.1**	94.5± 1.9	**95.4± 1.2**	95.0± 1.8	88.1± 4.9	**95.1± 0.9**
	2	**63.7± 8.1**	59.4± 8.9	57.1± 15.6	**66.5± 5.8**	60.8± 11.2	65.3± 11.0	54.6± 16.6	**66.3± 11.1**
SUR	1	90.3± 2.3	**96.6± 0.4**	89.3± 1.7	**95.6± 1.0**	**96.2± 1.3**	92.0± 5.1	64.6± 9.8	**95.5± 1.7**
	2	84.0± 2.5	**94.4± 1.7**	85.7± 4.4	**95.2± 1.6**	**96.0± 0.9**	92.8± 4.3	80.4± 5.0	**94.2± 3.1**

is done for each experiment) is shown. The best performance (96.6 ± 0.4, with 443 prototypes) was achieved when using the *surprise* sparsification strategy combined with whe *Gaussian* kernel and the *design method 1*. Also, a more balanced result between accuracy and number of prototypes (96.2 ± 1.3, with 127 prototypes) was achieved with the *surprise* sparsification strategy combined with the *cauchy* kernel and the *design method 1*. Comparing the kernel functions, the *exponential* and *cauchy* were the ones the lead the algorithm to reach the best results. Moreover, comparing the design methods, the *design method 1* was, most of the time, the best one.

Table 37 – Number of Prototypes of SPARK and Wall Following Dataset (1st set of experiments).

SM	DM	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	1	5	169	22	101	173	732	6	191
	2	9	159	20	167	138	732	10	104
COH	1	112	156	183	183	170	1	172	156
	2	153	129	133	116	140	732	225	195
NOV	1	65	74	74	67	70	66	55	65
	2	4	4	5	4	4	4	4	4
SUR	1	153	443	128	156	127	240	7	132
	2	40	137	121	134	133	198	41	244

In Table 38, the performance of each SPARK model configuration applied to the Wall Following dataset, for the second set of experiments (using the best hyperparameters achieved in the first set of experiments) is shown. The best performance (96.7 ± 0.4 with 724 prototypes) was reached with the *surprise* sparsification strategy combined with the *kmod* kernel function and *design method 2*. Also, a more balanced result between accuracy and number of prototypes (96.3 ± 0.5, with 124 prototypes) was achieved with the *surprise* sparsification strategy combined with the *cauchy* kernel and the *design method 1*. Comparing the kernel functions, the *log*, *gaussian* and *cauchy* were also the ones that lead the algorithm to reach the best results. Moreover, comparing the design methods, the *Design Method 2* was, most of the time, the best one. As can be seen, the best result from the second set of experiments (96.3 ± 0.5, with 124 prototypes) is 5.5% higher than the result obtained with the KSOM model (with 30 prototypes) reported in Chapter 3. Also, it was just 0.5% lower than the result obtained by a single-hidden-layer MLP reported in Table 20.

In Table 40, the summary of results for the Wall Following dataset is shown. An important thing to note is that, for this dataset, in both sets of experiments, using design method 1 and the surprise sparsification strategy led to a better performance.

4.4.5 A summary of the Results

In Table 41, the summary of the best mean accuracies obtained using SPARK models is shown. It is important to mention that the number of prototypes is determined during training, based on the optimized hyperparameters. As can be seen from this table, no single sparsification method or kernel function consistently outperformed the others across all datasets. However, in most cases, design method 2 led the algorithm to achieve the best performance.

In Table 42, a comparison of the best accuracies obtained by the KSOM and SPARK

Table 38 – Mean Accuracy of SPARK and Wall Following Dataset (2nd set of experiments).

SM	DM	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	1	45.3± 19.8	**93.3± 1.1**	76.0± 10.1	92.9± 1.3	**93.9± 1.1**	90.7± 19.8	57.4± 19.4	**93.7± 1.1**
	2	73.5± 5.8	**95.3± 0.9**	87.3± 3.2	**95.3± 0.9**	**95.2± 1.0**	**96.0± 1.7**	72.5± 8.0	94.9± 0.8
COH	1	93.5± 1.1	**94.4± 0.9**	93.1± 1.2	**94.8± 0.8**	93.8± 1.0	**96.3± 0.5**	94.0± 1.0	93.8± 1.0
	2	94.4± 0.8	**95.2± 0.9**	95.7± 0.5	94.9± 1.0	95.1± 1.0	**96.3± 0.5**	**95.5± 0.6**	**96.0± 0.6**
NOV	1	**95.0± 0.9**	**95.1± 0.8**	**95.2± 0.9**	**95.1± 1.0**	**95.2± 0.9**	**95.3± 1.1**	91.5± 2.6	**95.1± 0.9**
	2	**75.3± 7.9**	63.0± 9.2	74.1± 7.6	**95.0± 0.9**	64.2± 9.5	**95.3± 0.9**	64.6± 9.8	63.5± 10.1
SUR	1	50.1± 17.0	**96.3± 0.5**	92.1± 1.4	**96.5± 0.5**	**96.3± 0.5**	**96.4± 0.5**	61.0± 16.0	**96.2± 0.4**
	2	79.7± 6.2	**96.1± 0.7**	93.8± 1.8	95.9± 0.6	**96.5± 0.4**	**96.4± 0.6**	68.6± 10.5	**96.7± 0.4**

Table 39 – Number of Prototypes of SPARK and Wall Following Dataset (2nd set of experiments).

SM	DM	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	1	6	159	22	160	174	732	8	180
	2	12	162	51	165	137	732	9	162
COH	1	139	158	201	191	174	732	215	156
	2	151	125	262	122	152	732	214	203
NOV	1	64	74	71	77	81	75	59	69
	2	15	4	20	74	4	72	4	4
SUR	1	11	171	210	732	124	229	105	700
	2	26	132	354	137	634	190	10	724

Table 40 – Summary of Results SPARK and Wall Following Dataset.

Dataset	Experiment	Comparisons	
Motor Failure	Optimizing Hyperparameters	Best Result	surprise design method 1 Gaussian 1NN
		Best Design Method	design method 1
		Best Sparsification	surprise
		Best Kernels	exponentinal Cauchy
	Using Best Hyperparameters	Best Result	surprise design method 1 Gaussian 4NN
		Best Design Method	design method 2
		Best Sparsification	surprise
		Best Kernels	log Gaussian Cauchy

Table 41 – SPARK Models Best Mean Accuracies.

Dataset	SPARK Model	Mean Accuracy	#Prototypes
Motor Failure	Design Method 2 Novelty Gaussian kernel	99.9 ± 0.4	44
Cervical Cancer	Design Method 2 ALD log kernel	93.9 ± 2.0	89
Vertebral Column	Design Method 2 coherence kmod kernel	80.9 ± 4.2	50
Wall Following	Design Method 1 surprise Cauchy	96.3 ± 0.5	124

Table 42 – Comparison of performances between KSOM and SPARK models.

	KSOM		SPARK	
Dataset	Best Mean Accuracy	#Prototpyes	Best Mean Accuracy	#Prototypes
Motor Failure	77.9	30	99.9	44
Cervical Cancer	90.1	30	93.9	89
Vertebral Column	81.9	30	80.9	50
Wall Following	90.8	30	96.3	124

models is presented. As can be seen from this table, in most cases, automatically selecting the number of prototypes leads to higher classification rates. However, as observed in the vertebral column dataset, updating the prototype weights can allow the model to achieve higher accuracy with a low number of prototypes.

4.5 Final Considerations

In this chapter, a new framework called SPARK was proposed for designing sparse kernel prototype-based classifiers. The experimental results demonstrated that the algorithm can achieve high mean accuracy while maintaining sparse models with a low number of prototypes. An important observation is that increasing the number of prototypes does not necessarily lead to improved classification performance. In many cases, the best results were obtained with configurations that selected significantly fewer prototypes, emphasizing the importance of effective sparsification over model complexity.

Furthermore, no single sparsification method or kernel function consistently outperformed the others across all datasets. These findings highlight that the effectiveness of this model depends on the characteristics of the dataset, reinforcing the need for careful selection of sparsification criteria, kernel functions, and hyperparameters tailored to each specific task.

Until this chapter, the studied datasets are in batch format, meaning that all data is already made available a priori. Thus, the algorithms presented so far are designed to operate with this type of data. In the next chapter, a framework for handling a continuous stream of data is proposed.

5 A SPARSE ONLINE APPROACH FOR PROTOTYPE-BASED KERNEL MODELS

In this chapter, a novel method for designing sparse kernel prototype-based classifiers is introduced, which is capable of handling the challenges posed by the processing of streaming data. As done in Chapter 4, sparsity in the proposed model is initially pursued through the use of the ALD method (ENGEL *et al.*, 2004), which is a sequential procedure widely used in kernel adaptive filters to build a dictionary of relevant samples extracted from the incoming data flow. However, two major differences are introduced here. Firstly, items can also be removed from the dictionary in order to better tackling long-term changes in the data, such as concept drifting, as time passes by. Secondly, prototypes in the dictionary are updated in order to better fit to short-term changes in the dynamics of the data. In this regard, the proposed sparse kernel PBC model is a fully adaptive one, capable of adapting automatically to the problem complexity.

This chapter is organized as follows. Section 5.1 provides the fundamentals of sequential learning and concept drift. The proposed framework is developed in Section 5.2. The data sets and methods used are shown in Section 5.3. The simulation results are reported and discussed in Section 5.4 and the conclusion is made in Section 5.6.

5.1 Sequential Learning and Concept Drift

As previously seen, formally, in the context of classification, which is a supervised learning task, a data set is given as a group of tuples (samples) $\{(\mathbf{x}_n, c_n) \in \mathbb{R}^p \times \{1,\ldots,C\}\}_{n=1}^{N}$, where $\mathbf{x}_n$ is the attributes vector (input) and c_n is its respective class label. In batch learning, the data set is previously stored and can be divided into two subsets: training and test. The classifier model H is built from the training set, and the test set is used to validate it (LOSING *et al.*, 2018). Several training-testing strategies can be used to evaluate the models, such as the k-fold cross-validation, in which the data set is divided in k groups and all but one group (the test one) are used for training.

To analyse the built models in terms of the generalization ability to unseen samples, one can use the classification batch error measure, which is given by

$$E_b = \frac{1}{N_{ts}} \sum_{i=1}^{N_{ts}} \Theta(c_i, \hat{c}_i), \tag{5.1}$$

where $\hat{c}_i = H(\mathbf{x}_i)$ is the classifiers' prediction, N_{ts} is the number of test samples, and the function $\Theta(.,.)$ is either equal to 0, if $c_i = \hat{c}_i$, or 1, otherwise.

In data streaming problems, for which the data set is given as a sequence of potentially infinite length, only one sample $(\mathbf{x}_t, c_t)$ arrives and is processed at time step t. This application scenario often requires online learning, meaning that the classifier model needs to be updated immediately upon the arrival of the current input-output pair or after the collection of a mini-batch of samples. In order to evaluate the classifier model, it is common to resort to the interleaved test-then-train (a.k.a. prequential) evaluation method (GAMA *et al.*, 2014). In this case, for each new unseen data $\mathbf{x}_t$, the current model H_{t-1} is used to predict its label, i.e. $\hat{c}_t = H_{t-1}(\mathbf{x}_t)$. With this methodology, a single presentation of a given sample is carried out, all data is used to train and test the algorithm, and no holdout set is needed for testing.

To evaluate the models in terms of the generalization ability to unseen samples, one can use the classification sequential error measure, which is given by

$$E_s = \frac{1}{t}\sum_{i=1}^{t}\Theta(c_i, \hat{c}_i), \tag{5.2}$$

where t is the current iteration.

One of the challenges in the context of data streaming is that stationarity is an unrealistic assumption over a long period of time. Such changes in data statistics can occur, for example, due to adversary activities (spam filtering task) or it is a natural consequence of an environment with different operating points (ŽLIOBAITė *et al.*, 2015). Unexpected changes in underlying data distribution over time have received different names depending on the field. It is known as concept drift (in machine learning, data mining and predictive analysis) (GAMA *et al.*, 2014), nonstationarity (in signal processing) (HAYKIN; LI, 1995), covariate shift or data set shift (in pattern recognition) (MORENO-TORRES *et al.*, 2012). In the current thesis, this phenomenon is refered henceforth to as concept drift.

Formally, a concept drift between time point t_0 and time point t_1 can be defined as (GAMA *et al.*, 2014)

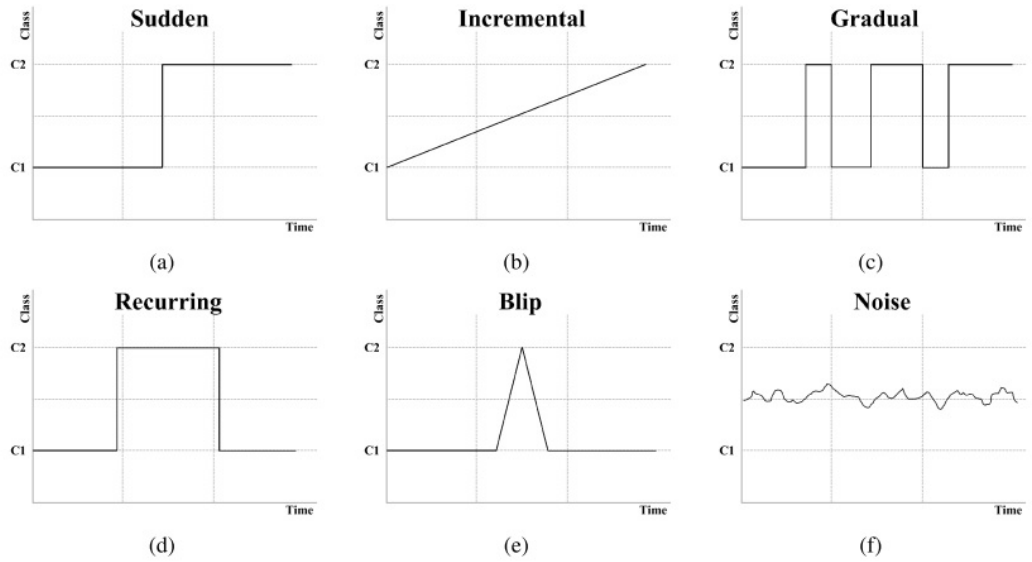
$$\exists \mathbf{X} : \rho(\mathbf{X}, \mathbf{c})_{t_0} \neq \rho(\mathbf{X}, \mathbf{c})_{t_1}, \tag{5.3}$$

where $\rho(\mathbf{X}, \mathbf{c})_{t_0}$ is the joint distribution at time t_0 between the set of input variables $\mathbf{X}$ and their target variables $\mathbf{c}$. Using the Bayes rule, Eq. (5.3) can be rewritten as

$$\exists \mathbf{X} : \rho(\mathbf{c}|\mathbf{X})_{t_0} p(\mathbf{X})_{t_0} \neq \rho(\mathbf{c}|\mathbf{X})_{t_1} \rho(\mathbf{X})_{t_1}, \tag{5.4}$$

where $\rho(\mathbf{X})$ is the distribution of the input data and $\rho(\mathbf{c}|\mathbf{X})$ is the conditional probability of the output. A virtual drift occurs if the distribution of the input data $\rho(\mathbf{X})$ changes. A real drift refers to a change in the conditional probability $\rho(\mathbf{c}|\mathbf{X})$.

Figure 8 – Types of concept drift.



Source: Wadewale e Desai (2015)

Furthermore, the concept drift can be distinguished as being abrupt (sudden, instantaneous), gradual, incremental or recurrent (LI *et al.*, 2020). In sudden drifts, the data changes instantly and without alternation (IWASHITA; PAPA, 2018). In the field of monitoring and control (of a chemical plant, for example), an abrupt drift can be represented by an abnormal condition of the system that happens in the order of seconds or minutes (ŽLIOBAITė *et al.*, 2015). In incremental and gradual drifts, the changes occur slowly. An incremental drift happens when the data values gradually change over time. This can be instantiated by a digital facial recognition system that must recognize individuals who are in a constant aging process. On the other hand, gradual drifts also include changing in class distribution (IWASHITA; PAPA, 2018). Moreover, a recurring concept drift may occur due to cyclic phenomena, such as seasons of the year or may be associated with irregular phenomena, such as inflation rates (TSYMBAL, 2004). Also, in real world problems, the concepts can change in more than one way, being heterogeneous. In Figure 8, these and other two types of drifts (blip and noise) are illustrated. See Wadewale e Desai (2015), Webb *et al.* (2016), Iwashita e Papa (2018) and Babüroğlu *et al.* (2024) for a broader view on concept drift characterizing.

Finally, different strategies for updating learning models have been developed to deal with concept drift, and they can be distinguished as passive (lazy) or active. In passive strategies, the models are constantly (or periodically) retrained. Alternatively, in active strategies, learning models may use trigger mechanisms to identify statistical changes and initiate a model update (ŽLIOBAITė *et al.*, 2015). See Ditzler *et al.* (2015) for a broader view on learning in

nonstationary environments.

5.2 The Proposed Model

In this section, a formal description of each constituent part of the resulting model and the pseudocodes for implementation are provided. Also, a discussion about the numerical methods that make it feasible for practical purposes is done. Codes are available at Github[1]. Furthermore, it is important to mention that the proposed algorithm updates the classification model using a passive strategy (see Section 5.1).

5.2.1 Dictionary growing procedure

In order to add samples to the dictionary, the same strategy as the one explained in Section 4.1 is used, but one hyperparameter is added. This method will be explained here again to make this chapter self-contained.

The proposed model starts with no prototype in the dictionary. The first sample entering the dictionary is the first one to appear in the streaming. Then, subsequent samples will be tested for their relevance in order to be included into the dictionary. The procedure to add prototypes to the model's dictionary was initially based on the ALD criterion (ENGEL *et al.*, 2004), which constructs a dictionary consisting of a subset of size Q_{t-1} of the input-output pairs presented so far. This dictionary is denoted $\mathscr{D}_{t-1}$.

More formally, using the ALD criterion, at time step t, after having observed $t-1$ samples, the dictionary $\mathscr{D}_{t-1}$ is comprised of a subset of Q_{t-1} relevant pairs $\left\{(\tilde{\mathbf{x}}_j, \tilde{c}_j)\right\}_{j=1}^{Q_{t-1}}$. The set of vectors $\{\tilde{\mathbf{x}}_j\}_{j=1}^{Q_{t-1}}$ in $\mathscr{D}_{t-1}$ comprises an approximately linearly independent set of vectors. The proposed approach takes the pairs in the dictionary as labeled prototype vectors in feature space, so that they can be used as a kernelized nearest neighbor classifier, since the labels $\left\{\tilde{c}_j\right\}_{j=1}^{Q_{t-1}}$ of each input vector are kept in the dictionary.

When a new incoming sample $\mathbf{x}_t$ is available, one must test if it should be added or not to the dictionary. In order to do this, it is necessary to estimate a vector of coefficients $\mathbf{a} = \begin{bmatrix} a_1 & \cdots & a_{Q_{t-1}} \end{bmatrix}^T$ satisfying the ALD criterion:

$$\delta_1(t) \leq \nu_1, \tag{5.5}$$

[1] https://github.com/davidcoelho89/spok-nn

where

$$\delta_1(t) \stackrel{def}{=} \min_{\mathbf{a}} \left\| \sum_{j=1}^{Q_{t-1}} a_j \phi\left(\tilde{\mathbf{x}}_j\right) - \phi\left(\mathbf{x}_t\right) \right\|^2, \tag{5.6}$$

and ν_1 is the sparsity level parameter (which is a hyperparameter and should be optimized). Developing the minimization problem in Eq. (5.6) and using the kernel trick, $\kappa(\mathbf{x},\mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$, one can write

$$\delta_1(t) \stackrel{def}{=} \min_{\mathbf{a}} \left\{ \sum_{i,j=1}^{Q_{t-1}} a_i a_j \kappa\left(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j\right) - 2 \sum_{j=1}^{Q_{t-1}} a_j \kappa\left(\tilde{\mathbf{x}}_j, \mathbf{x}_t\right) + \kappa\left(\mathbf{x}_t, \mathbf{x}_t\right) \right\}, \tag{5.7}$$

or, using matrix notation,

$$\delta_1(t) = \min_{\mathbf{a}} \left\{ \mathbf{a}^T \tilde{\mathbf{K}}_{t-1} \mathbf{a} - 2\mathbf{a}^T \tilde{\mathbf{k}}_{t-1}\left(\mathbf{x}_t\right) + k_{tt} \right\}, \tag{5.8}$$

where $\tilde{\mathbf{K}}_{t-1}$ is a $Q_{t-1} \times Q_{t-1}$ kernel matrix built using the current dictionary. The (i,j)-th entry of this matrix is given by $\left[\tilde{\mathbf{K}}_{t-1}\right]_{i,j} = \kappa\left(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j\right)$, with $i,j = 1,\ldots,Q_{t-1}$. The Q_{t-1}-dimensional vector $\tilde{\mathbf{k}}_{t-1}$ is defined as

$$\tilde{\mathbf{k}}_{t-1}\left(\mathbf{x}_t\right) = \left[\kappa(\mathbf{x}_t, \tilde{\mathbf{x}}_1) \;\; \cdots \;\; \kappa(\mathbf{x}_t, \tilde{\mathbf{x}}_i) \;\; \cdots \;\; \kappa(\mathbf{x}_t, \tilde{\mathbf{x}}_{Q_{t-1}})\right]^T, \tag{5.9}$$

while $k_{tt} = \kappa\left(\mathbf{x}_t, \mathbf{x}_t\right)$. Solving Eq. (5.8) leads to the optimal $\mathbf{a}_t$, which is given by

$$\mathbf{a}_t = \tilde{\mathbf{K}}_{t-1}^{-1} \tilde{\mathbf{k}}_{t-1}\left(\mathbf{x}_t\right), \tag{5.10}$$

so that Eq. (5.6) can be rewritten as

$$\delta_1(t) = k_{tt} - \tilde{\mathbf{k}}_{t-1}\left(\mathbf{x}_t\right)^T \mathbf{a}_t. \tag{5.11}$$

If $\delta_1(t) > \nu_1$, then the sample must be added to the dictionary; that is, $\mathscr{D}_t = \mathscr{D}_{t-1} \cup \{(\tilde{\mathbf{x}}_t, \tilde{y}_t)\}$ and $Q_t = Q_{t-1} + 1$. However, if $\delta_1(t) \leq \nu_1$, the sample is approximate linear dependent and must not be added to the dictionary; that is, $\mathscr{D}_t = \mathscr{D}_{t-1}$ and $Q_t = Q_{t-1}$.

Differently from the method introduced in Chapter 4, in the current method, a hyperparameter β limits the number of prototypes that can be added to the dictionary. Thus, once the dictionary size reaches this limit, the only way a new sample can be added is if it belongs to a class that has not yet been observed by the algorithm[2].

Remark 1: It is important to mention that β limits the number of prototypes for the entire dictionary, but each class can have a different number of prototypes. Also, this hyperparameter can be set to infinite, so that there is no limit for adding new prototypes.

[2] Therefore, the maximum number of prototypes that can belong to the dictionary at a given time step t is $\beta + C_{no}$, where C_{no} is the number of classes that had not been observed when the dictionary reached β prototypes.

Remark 2: There are other dictionary sparsification procedures in the field of kernel adaptive filtering such as Coherence (RICHARD *et al.*, 2009), Novelty (PLATT, 1991) and Surprise (LIU *et al.*, 2008), that were already used in Chapter 4 and can be used within the framework here developed. In this regard, results from these other sparsification procedures will be reported later in this chapter. Also, there are other ways to achieve sparse prototype-based models, such as the ones described in Hofmann *et al.* (2014), Albuquerque *et al.* (2018), and Soares Filho e Barreto (2014). More on the issue of sparse online dictionary learning can also be found in (MAIRAL *et al.*, 2010). The ALD method was initially chosen because it can be taken as an approximation of PCA in the feature space (ENGEL *et al.*, 2004), as it implies that eigenvectors with eigenvalues that are significantly larger than ν_1 are projected almost in their entirety onto the span of the dictionary vectors. Thus, the dictionary contains vectors that represent well the directions of the largest variances of the data in the feature space. Indeed, the ALD procedure can be seen as an approximate way of performing online sparse kernel PCA on sequential data.

5.2.2 ALD-based kernel nearest neighbor classifiers

The just described ALD-based selection of prototypes for the dictionary can be used as a standalone classifier by means of kernelized distances, such as the ones introduced in Section 2.5. For this purpose, the approaches suggested in Chapter 4 are followed, which are reproduced below for the sake of completeness.

Design Method 1: This approach involves the construction of a single dictionary. The first sample in the streaming will be the first item of the dictionary. Then, for each new sample, apply the ALD criterion according to Eq. (5.10) and Eq. (5.11). Note that each prototype vector in $\mathscr{D}_t$ carries its class label c_t for the sake of classification.

Design Method 2: This approach requires the construction of smaller class-conditional dictionaries which are then merged later. For a problem with C classes, it is required C dictionaries $\mathscr{D}_t^{(i)}$, $i = 1,2,\ldots,C$. For this purpose, whenever a data sample $(\mathbf{x}_t, c_t)$ from the i-th class emerges, apply the Design Method 1 to the dictionary $\mathscr{D}_{t-1}^{(i)}$. The model's dictionary is the union of all the C class-conditional dictionaries $\mathscr{D}_t = \mathscr{D}_t^{(1)} \cup \mathscr{D}_t^{(2)} \cup \cdots \cup \mathscr{D}_t^{(C)}$. In Remark 2, it was stated that, if a dictionary is built from the ALD procedure, it represents the data distribution. As Design Method 2 builds one dictionary per class, it allows a better modeling of the spatiotemporal dynamics of each class than the Design Method 1. For this reason, it is more suitable for data streaming processing than the Design Method 1.

For the classification of an incoming sample, use the kernelized distance in Eq. (2.7) in order to find the closest prototype. The search is executed over all the samples in the model's dictionary. Assign to that sample, the same class of the nearest prototype.

Remark 3: Other nearest neighbor strategies (such as the KNN and KWKNN) could be used for classification purposes. But the nearest neighbor approach was, initially, used for the sake of simplicity. Results from applying the KWKNN to this framework are discussed later in this chapter.

It should be noted that the only hyperparameters so far are ν_1 (the sparsity level), β (maximum number of prototypes), and those associated with the chosen kernel function (as the scale parameter σ_κ of the gaussian kernel function). However, since the kernel parameters are common to all kernel-based methods, the only tunable hyperparameters so far are the sparsity level ν_1 and the maximum number of prototypes β (which can be disconsidered if it is set to infinity).

Remark 4: In the SPARK framework, introduced in Chapter 4, the dictionary only grows as time passes and the prototypes in the dictionary are not updated. As a consequence, the SPARK-based models cannot handle nonstationary data, typical of streaming scenarios. In this chapter, a generalization of the SPARK framework is introduced, what makes the models also capable of removing obsolete items from the dictionary and updating the prototypes in order to better model the changes in the data dynamics. In fact, the SPARK framework is to be considered a special case of the framework being developed. Also, as will be detailed further, only the design method 2 will be used for streaming data processing tasks discussed in the current chapter.

5.2.3 Updating the Prototypes

As mentioned above, if there is no change in the underlying data distribution, just the ALD-based growing procedure and the design method 2 are sufficient to build a prototype-based kernel classifier. However, as also mentioned previously, one important issue in data streams is the concept drift. One contribution of this chapter is the verification that prototype updating helps the classifier to deal suitably with gradual or incremental concept drift. In the following, this simple but powerful idea is developed.

One-shot learning vs. Incremental learning - One important aspect of this proposal is to extract information not only from the samples *in* the dictionary, but also from the samples which are

not included in it. When the proposed classifier decides to include an incoming sample into the dictionary, this is a type of *one-shot learning* (CARPENTER *et al.*, 1991), i.e. all the information in that sample is instantaneously absorbed *as is* by the classifier. This type of learning strategy provides long-term stability to the model, since only relevant changes are incorporated into it.

However, even if a sample is not added to the dictionary, it can also carry important information that should not be discarded, such as a small change in the data distribution, a common scenario in streaming data processing. It was observed that these small changes were not captured by the learning of the previous framework, when it was applied to streaming data processing. To tackle this limitation, the idea of allowing the prototypes in the dictionary to be updated slightly by gradient descent rule, similar to the one used by the EF-KSOM in Eq. (2.8), but disregarding the neighborhood function, came out:

$$\mathbf{w}_{i^*}(t+1) = \mathbf{w}_{i^*}(t) - \eta(t)\nabla J_i(\mathbf{x}(t)), \tag{5.12}$$

where only the closest prototype $\mathbf{w}_i^*$ is updated. In this regard, Eq. (5.12) is a winner-take-all competitive learning rule.

It should be noted that the one-shot learning mechanism of the proposed approach is inspired by an equivalent procedure in the family of competitive learning algorithms known as adaptive resonance theory (ART) models (CARPENTER *et al.*, 1991). In ART models, if an input item is different enough from the ones stored, it will be added *as is* to the set of prototypes. This is known as vigilance mechanism, which is similar to the novelty mechanism (PLATT, 1991) used by many kernel adaptive filters for dictionary building. Then, as the learning process advances, the stored prototypes are updated incrementally in a winner-take-all basis.

A common learning rule for different kernels - Since the recursive rule in Eq. (5.12) depends on the computation of the gradient vector $\nabla J_i(\mathbf{x}(t))$, a different rule should emerge for each different choices of the kernel function. However, for certain types of kernel functions, such as must of those listed in Table 1, it was noticed that the gradient vector $\nabla J_i(\mathbf{x}(t))$ is proportional to the difference between the prototype vector and the current input sample, i.e. $\nabla J_i(\mathbf{x}(t)) \propto \mathbf{w}_{i^*}(t) - \mathbf{x}(t)$, with the final expression being very similar to each other[3]. The calculation of the corresponding gradient vectors, considering each kernel function separately, is developed in the Appendix B.

[3] Sometimes normalized by the quadratic norm of the difference vector, i.e. $\|\mathbf{w}_{i^*}(t) - \mathbf{x}(t)\|^2$.

Thus, for the sake of simplicity, a common learning rule for all the kernel functions is given by

$$\mathbf{w}_{i^*}(t+1) = \mathbf{w}_{i^*}(t) + \eta(t)[\mathbf{x}(t) - \mathbf{w}_{i^*}(t)], \tag{5.13}$$

where learning rate η is to be very small in order to allow the classifier to absorb small variations in the data while avoiding catastrophic forgetting phenomena (RICHARDSON; THOMAS, 2008). The learning rate can follow a decaying scheme or be kept constant in a small value. Here, the second option was chosen, and a recommended value is $\eta \leq 0.1$.

Remark 5: The proposed prototype updating strategy grants plasticity to the ALD-based classifier without compromising the stability of the learned dictionary. This is possible by combining ALD-based one-shot prototype selection with gradual updating of the prototype locations by Eq. (5.13). Thus, the resulting model is capable of gradual learning without catastrophic forgetting, dealing gracefully with the stability-plasticity dilemma common to self-organizing neural networks (GROSSBERG, 1987; RICHARDSON; THOMAS, 2008; MERMILLOD *et al.*, 2013; BRNA *et al.*, 2019).

Remark 6: The update rule in Eq. (5.13) was chosen for the sake of simplicity, but, as the proposed algorithms are considered instantiations of a more comprehensive framework developed with the aim of building sparse adaptive prototype-based kernel models, the interested reader can certainly use other similar learning rules, such as those based on the Kernel Generalized Learning Vector Quantization (KGLVQ) (HAMMER *et al.*, 2014; QIN; SUGANTHAN, 2004b).

5.2.4 Dictionary Pruning Procedure

In data streaming problems, keeping the model complexity bounded, i.e. maintaining the smallest number of items in the dictionary as time passes, while keeping the highest accuracy rate as possible are important issues. The ALD and other sparsification methods only include items in the dictionary. The user can, of course, control the inclusion rate either by manipulating the parameter ν_1, which gradually limits the acceptance of new items over time, or by defining a maximum value β for the size of the dictionary. The ideal situation is to make the dictionary fully adaptive; that is, it can not only include items in, but also delete items from the dictionary when they become irrelevant. Some algorithms use performance-based techniques to limit memory usage. In Li *et al.* (2020), for example, when the ensemble size exceeds a predefined threshold, the worst-performing classifiers are removed.

Algoritmo 4: score_update_procedure()

Input: s_{i^*}, z_{i^*}, $c(\mathbf{w}_{i^*})$, $c(\mathbf{x})$
Output: s_{i^*}
begin
  if $c(\boldsymbol{w}_{i^*}) = c(\boldsymbol{x})$ **then**
    if $\boldsymbol{w}_{i^*}$ *predicted correctly the class label the last time it was chosen, and* $s_{i^*} < 0$ **then**
      $s_{i^*} \leftarrow s_{i^*} + 1$;
    end
    $z_{i^*} \leftarrow 1$
  else
    if $\boldsymbol{w}_{i^*}$ *predicted incorrectly the class label the last time it was chosen* **then**
      $s_{i^*} \leftarrow s_{i^*} - 1$;
    end
    $z_{i^*} \leftarrow 0$
  end
end

Bearing this in mind, a new dictionary pruning strategy was developed, which is based on a binary variable $\{z_j\}_{j=1}^{Q_t}$ indicating whether a given prototype produced a correct prediction the last time it was selected to classify a sample (1 if correct, 0 otherwise), and on a score $\{s_j\}_{j=1}^{Q_t}$ assigned to each prototype in the dictionary. The score of a prototype that has just been inserted into the dictionary is set to 0, and the value of the variable z_i is set to 1. Then, when a new sample is classified and the algorithm observes its label, the score and the variable z_{i^*} of the closest prototype are updated as shown in Algorithm 4, henceforth called *score_update_procedure()*.

As one can notice from this pruning rule, a prototype that incorrectly classifies samples in sequence is penalized. If this prototype, twice in a row, correctly classifies samples in sequence, its score is improved. Also, the maximum score for a prototype is 0. After the score is updated, the prototype will be removed if the two conditions below are jointly fulfilled.

Condition 1 verifies if its score is less than the minimum score ε. This constant is set to -10 in this thesis.

Condition 2 verifies if there is at least another prototype, in the dictionary, representing the same class.

While the first condition is a performance-related verification, the second condition is necessary to avoid dictionaries without representatives of a particular class. Finally, this approach does not penalize prototypes that are not being selected in the last iterations. So, if there is some kind of recurring concept drift and "old" prototypes are important for these concepts, the

algorithm can still have good performance.

5.2.5 Rebuilding the Kernel Matrix

One main difference among standard prototype-based classifiers and their kernelized counterparts is that every time the content of the dictionary is altered by insertion/deletion/updating of prototypes, the kernel matrix has to be reshaped, which can be the most time-consuming part of the algorithm. For this purpose, many important linear algebra results will be used. In this section, it is described how to do that for the three situations, starting with the case of insertion of items into the dictionary.

After insertion of items: In order to apply the ALD test to an incoming sample, the inverse of the kernel matrix $\tilde{\mathbf{K}}_{t-1}$ has to be calculated, as shown in Eq. (5.10). As the size Q_t of the dictionary increases, it becomes unviable, in terms of the computational efforts required, to recompute the full inverse matrix each time the dictionary is modified. A feasible alternative is to calculate such inverses recursively using the following method described in (ENGEL *et al.*, 2004). This reduces the computational complexity of building a kernel matrix from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$.

Every time a new prototype is added to the dictionary, its corresponding class kernel matrix (for the design method 2), or the entire kernel matrix (for the design method 1), is updated as follows:

1. The current regularized kernel matrix (Van Vaerenbergh; SANTAMARÍA, 2014) is represented by

$$\dot{\mathbf{K}}_{t-1} = \tilde{\mathbf{K}}_{t-1} + \sigma_n \mathbf{I}_{Q_{t-1}}, \tag{5.14}$$

   where $\mathbf{I}_{Q_{t-1}}$ is the identity matrix of dimension Q_{t-1}, and σ_n is a small constant (here set to 0.001) used to handle matrix ill-conditioning which inevitably leads to numerical problems during inversion.

2. The kernel function between each dictionary prototype and the current input sample is calculated and kept in the vector

$$\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) = [\kappa(\mathbf{x}_t, \tilde{\mathbf{x}}_1) \ \cdots \ \kappa(\mathbf{x}_t, \tilde{\mathbf{x}}_i) \ \cdots \ \kappa(\mathbf{x}_t, \tilde{\mathbf{x}}_{Q_{t-1}})]^T. \tag{5.15}$$

3. Finally, one needs to calculate the kernel function between the sample and itself

$$k_{tt} = \kappa(\mathbf{x}_t, \mathbf{x}_t). \tag{5.16}$$

4. With the previous results, the new kernel matrix is given by

$$\dot{\mathbf{K}}_t = \begin{bmatrix} \dot{\mathbf{K}}_{t-1} & \tilde{\mathbf{k}}_{t-1} \\ \tilde{\mathbf{k}}_{t-1}^T & k_{tt} + \sigma_n \end{bmatrix}. \tag{5.17}$$

In order to update the inverse kernel matrix, the variables

$$\dot{\mathbf{a}}_t = \dot{\mathbf{K}}_{t-1}^{-1} \tilde{\mathbf{k}}_{t-1} \tag{5.18}$$

and

$$\dot{\delta}_1(t) = k_{tt} - \tilde{\mathbf{k}}_{t-1}^T \dot{\mathbf{a}}_t + \sigma_n \tag{5.19}$$

need to be calculated. Then, the inverse matrix is updated as

$$\dot{\mathbf{K}}_t^{-1} = \frac{1}{\dot{\delta}_1(t)} \begin{bmatrix} \dot{\delta}_1(t)\dot{\mathbf{K}}_{t-1}^{-1} + \dot{\mathbf{a}}_t \dot{\mathbf{a}}_t^T & -\dot{\mathbf{a}}_t \\ -\dot{\mathbf{a}}_t^T & 1 \end{bmatrix}. \tag{5.20}$$

As one can notice, Eq. (5.15) and Eq. (5.16) are already used for the ALD test. Also, Eq. (5.18) and Eq. (5.19) just differ from the ALD ones, namely, Eq. (5.10) and Eq. (5.11), by the regularization term $\sigma_n \mathbf{I}_{Q_{t-1}}$. Thus, they just need to be calculated once, either using σ_n in Eq. (5.10) and Eq. (5.11), or setting σ_n to 0 in the inversion procedure.

After removal of items: For this case, every time a prototype is removed from the dictionary, the dimensions of the kernel matrix and, hence, its inverse will be reduced. For this purpose, the k-th column and the k-th row corresponding to the removed prototype need to be withdrawn from the previous kernel matrix. In order to calculate the inverse of this modified kernel matrix, the Sherman-Morrison identity (SHERMAN; MORRISON, 1950) must be used:

$$(\mathbf{B} - \mathbf{u}\mathbf{m}^T)^{-1} = \mathbf{B}^{-1} + \frac{(\mathbf{B}^{-1}\mathbf{u})(\mathbf{m}^T\mathbf{B}^{-1})}{1 - \mathbf{m}^T\mathbf{B}^{-1}\mathbf{u}}. \tag{5.21}$$

In Juárez-Ruiz *et al.* (2016), one can find how to compute the inverse square matrix after removing, at the same time, any row and column of the original squared matrix. This reduces the computational complexity of building this kernel matrix from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$. Here, the removed row and columns have the same index i, and one can define

$$\mathbf{B} = \tilde{\mathbf{K}}_{t-1}, \tag{5.22}$$

$$\mathbf{u} = \mathbf{B}_i - \mathbf{e}_i, \tag{5.23}$$

and

$$\mathbf{m} = \mathbf{e}_i, \tag{5.24}$$

where $\mathbf{B}_i \in \mathbb{R}^{Q_{t-1}}$ is i-th column vector of $\tilde{\mathbf{K}}_{t-1}$ and $\mathbf{e}_i \in \mathbb{R}^{Q_{t-1}}$ is the i-th canonical column vector[4]. Then, one can use the matrix identity in Eq. (5.21) to compute the matrix $\tilde{\mathbf{K}}_t^{-1}$ after removing its i-th column and its i-th row.

After prototype updating: It is important to mention that, if a prototype is updated, this is equivalent to removing the *old* prototype, and adding a *new* (updated) one. So, in this case, one have to apply sequentially the just described methods to emulate the removal of a prototype and the insertion of a prototype in order to update the kernel matrix and compute its inverse.

The algorithm formed by the procedures of adding, online updating and pruning of prototypes is named as *SParse Online Kernel adaptive* Nearest-Neighbor (SPOK-NN). For a better understanding of the proposed approach, the main routines of the SPOK-NN classifier are presented in three smaller procedures, namely: (*i*) updating the error measure (Algorithm 5); (*ii*) adding or updating a prototype (Algorithm 6); and (*iii*) pruning the dictionary (Algorithm 7). These procedures are used in the main pseudocode of the SPOK-NN classifier (Algorithm 8) considering the design method 2.

Algoritmo 5: error_measure_update_procedure()

Input: E_s, t, $\hat{c}(\mathbf{x}_t)$, c_t

Output: E_s

begin

  if $\hat{c}(\mathbf{x}_t) = c_t$ **then**

    $E_s \leftarrow \frac{(t-1)*E_s}{t}$;

  else

    $E_s \leftarrow \frac{(t-1)*E_s+1}{t}$;

  end

end

[4] A binary vector of length Q_{t-1} with the i-th element set to 1. All the other elements are set to zero.

Algoritmo 6: add_or_update_prototype() (Design Method 2)

Input: ν_1, β, Q, $\mathbf{s}$, $\{(\mathbf{x}_t, c_t)\}$,$\mathscr{D}^{c_t}$, Q^{c_t}, $\tilde{\mathbf{K}}^{c_t}$, $(\tilde{\mathbf{K}}^{c_t})^{-1}$,

Output: $\mathbf{s}$, $\mathscr{D}^{c_t}$, Q, Q^{c_t}, $\tilde{\mathbf{K}}^{c_t}$, $(\tilde{\mathbf{K}}^{c_t})^{-1}$

begin

 # ALD Test:

Compute $\mathbf{a}_t$ and $\delta_1(t)$ using Eq. (5.10) and Eq. (5.11);

if $\delta_1(t) > \nu_1$ *and* $Q < \beta$ **then**

 # Add Prototype to Dictionary

$\mathscr{D}^{c_t} \leftarrow \mathscr{D}^{c_t} \cup \{(\mathbf{x}_t, c_t)\}$;

Update $\tilde{\mathbf{K}}^{c_t}$ and $(\tilde{\mathbf{K}}^{c_t})^{-1}$ using Eqs. (5.17) and (5.20);

$Q^{c_t} \leftarrow Q^{c_t} + 1$;

$\mathbf{s} \leftarrow \mathbf{s} \cup \{0\}$;

else

Update nearest prototype of $\mathscr{D}^{c_t}$ using Eq. (5.13);

Update $\tilde{\mathbf{K}}^{c_t}$ and $(\tilde{\mathbf{K}}^{c_t})^{-1}$ using Eqs. (5.17), (5.20) and (5.21);

end

end

Algoritmo 7: pruning_procedure() (Design Method 2)

Input: $\mathscr{D}$, Q, ε, $\mathbf{s}$, $\{(\mathscr{D}^{(i)})\}_{i=1}^{C}$, $\{(Q^{(i)})\}_{i=1}^{C}$, $\{(\tilde{\mathbf{K}}^{(i)})\}_{i=1}^{C}$, $\{(\tilde{\mathbf{K}}^{(i)})^{-1}\}_{i=1}^{C}$

Output: $\mathbf{s}$, $\{(\mathscr{D}^{(i)})\}_{i=1}^{C}$, $\{(Q^{(i)})\}_{i=1}^{C}$, $\{(\tilde{\mathbf{K}}^{(i)})\}_{i=1}^{C}$, $\{(\tilde{\mathbf{K}}^{(i)})^{-1}\}_{i=1}^{C}$

begin

for $q = 1 : Q$ **do**

Get q-th prototype $\left\{(\mathbf{x}_q, c_q)\right\}$ from dictionary $\mathscr{D}$;

if $s_q < \varepsilon$ *and* $Q^{(c_q)} > 1$ **then**

$\mathscr{D}^{(c_q)} \leftarrow \mathscr{D}^{(c_q)} - \left\{(\mathbf{x}_q, c_q)\right\}$;

$Q^{(c_q)} \leftarrow Q^{(c_q)} - 1$;

Update $(\tilde{\mathbf{K}}^{c_q})^{-1}$ using Eq. (5.21);

Update $\tilde{\mathbf{K}}^{c_q}$ removing its q-th column and q-th row;

$\mathbf{s} \leftarrow \mathbf{s} - \left\{(s_q)\right\}$

end

end

end

Algoritmo 8: SPOK-NN (Design Method 2)

Input: $\{(\mathbf{x}_t, c_t)\}_{t=1}^{T}$, $\kappa(\cdot,\cdot)$, γ, ν_1, β, η, ε, σ_n

Output: $\mathscr{D}$

begin

 # Add first sample to the dictionary:

$E_s \leftarrow 1$; $\mathscr{D} \leftarrow (\mathbf{x}_1, c_1)$; $Q \leftarrow 1$, $\mathbf{s} \leftarrow 0$;

$\mathscr{D}^{(c_1)} \leftarrow (\mathbf{x}_1, c_1)$; $Q^{(c_1)} \leftarrow 1$;

$\tilde{\mathbf{K}}^{c_1} \leftarrow \kappa(\mathbf{x}_1, \mathbf{x}_1) + \sigma_n$; $(\tilde{\mathbf{K}}^{c_1})^{-1} \leftarrow 1/\tilde{\mathbf{K}}^{c_1}$;

for $t = 2 : T$ **do**

Get new input-output pair: $(\mathbf{x}_t, c_t)$

 # Classify new data using nearest neighbor rule:

Find $q^*(\mathbf{x}_t)$ using Eq. (2.6) ;

Then $\hat{c}(\mathbf{x}_t) \leftarrow c(\mathbf{w}_{q^*})$;

Update the Error Measure using Algorithm 5;

 # Add first sample of a class conditional dictionary

if $\mathscr{D}_t^{(c_t)} = \varnothing$ **then**

$\mathscr{D}^{(c_t)} \leftarrow (\mathbf{x}_t, c_t)$; $Q^{(c_t)} \leftarrow 1$; $\mathbf{s} \leftarrow \mathbf{s} \cup \{0\}$;

$\tilde{\mathbf{K}}^{c_i} \leftarrow \kappa(\mathbf{x}_t, \mathbf{x}_t) + \sigma_n$; $(\tilde{\mathbf{K}}^{c_i})^{-1} \leftarrow 1/\tilde{\mathbf{K}}^{c_i}$;

else

Update or add a prototype using Algorithm 6;

end

Update the winner score using Algorithm 4;

Run the pruning procedure using Algorithm 7;

$\mathscr{D} = \mathscr{D}^{(1)} \cup \mathscr{D}^{(2)} \cup \cdots \cup \mathscr{D}^{(C)}$; $Q = Q^{(1)} \cup Q^{(2)} \cup \cdots \cup Q^{(C)}$;

end

end

In the next section, the main characteristics of the datasets used in this thesis, as well as the methodology adopted to assess the performance of the proposed algorithm on streaming data classification, are introduced.

5.3 Data Sets and Methods

For evaluating the SPOK-NN model in streaming data classification, its performance is tested using three artificial and six real-world data sets. The artificial drift data sets were

Table 43 – Summary of the characteristics of the artificial data sets.

Data set	#Samples	#Features	#Classes	Drift Type
Moving Squares	200K	2	4	Real Gradual
RBF Interchanging	200K	2	15	Real Abrupt
Chessboard	200K	2	8	Virtual Recurring

generated by the Massive Online Analysis (MOA) framework (BIFET *et al.*, 2010b). This framework provides learning algorithms, data generators, evaluation methods and statistics for data streaming analysis. The artificial data sets used in this thesis are described below.

The **Moving Squares** dataset is composed of four uniformly distributed square regions moving horizontally at a constant speed. The direction is inverted whenever the leading square reaches a predefined boundary. Each square represents a different class. This data set is used in order to test if the algorithm can handle gradual drifts. The **Interchanging RBF** is composed of 15 Gaussians with random covariance matrices replacing each other every 3000 samples. Thereby, the number of Gaussians switching their position increases each time by one until all are simultaneously changing their location. This allows to evaluate an algorithm in the context of abrupt drift with increasing strength. For the **Transient Chessboard** data set, virtual drift is generated by revealing successively parts of a chessboard. This is done square by square randomly chosen from the whole chessboard such that each square represents an own concept. Whenever four fields have been revealed, samples covering the whole chessboard are presented. This recurring alternation of patterns penalizes algorithms tending to discard former concepts. To reduce the impact of classification by chance, Losing *et al.* (2016) used eight classes instead of two. The same strategy was used here to asses the generalization ability of the proposed SPOK-NN classifier in handling recurring concept drifts. In Table 43, the main characteristics of the artificial data sets, such as number of samples, features, and classes, are summarized.

Additionally, six real-world data sets that also contain different kinds of drifts are used in order to assess the proposed classifier in more realistic nonstationary scenarios. The **Forest Cover Type** assigns cartographic variables of $30m \times 30m$ cells obtained from US Forest Service data. Only forests with minimal human-cause disturbances were used, so that resulting forest cover types are more a result of ecological processes (BIFET *et al.*, 2013). The **Electricity** data set is initially described in Harries (1999) and holds information of the Australian New South Wales electricity market, whose prices are affected by supply and demand of the market itself and are set every 5 min. A class label identifies the change of the price relative to a moving average of the last 24 hours (GOMES *et al.*, 2017b). The **Outdoor** set was obtained from images

Table 44 – Summary of the characteristics of the real-world data sets.

Data set	#Samples	#Features	#Classes
Cover Type	581012	54	7
Electricity	45312	5	2
Outdoor	4000	21	40
Poker Hand	829200	10	10
Rialto	82250	27	10
Weather	18159	8	2

recorded by a mobile in a garden environment (LOSING *et al.*, 2015). The task was to classify 40 different objects, each approached ten times under varying lighting conditions affecting the color based representation. Each approach consists of 10 images and is represented in temporal order. The objects are encoded in a normalized 21-dimensional RG-chromaticity histogram (LOSING *et al.*, 2016). The **Poker Hand** set is formed by one million instances representing all possible poker hands. Each card in a hand is described by two attributes: suit and rank. So, each hand is described by 10 attributes. The class indicates the value of a hand. The same data set as in Bifet *et al.* (2013) was used, where virtual drift is introduced via sorting the instances rank and suit, and duplicates are removed. The **Rialto** is composed of 10 colorful buildings next to the Rialto bridge in Venice encoded in a normalized 27-dimensional RGB histogram. The images were obtained from time-lapse videos captured by a webcam with fixed position. Continuously changing weather and lighting conditions affect the representation. The overnight recordings were removed and the labels were generated by manually masking the corresponding buildings (LOSING *et al.*, 2016). Finally, the **Weather** data set was introduced by Elwell e Polikar (2011). The task goal is to predict whether it is going to rain on a certain day or not. It is important to mention that there is a imbalance towards no rain (69%). A summary of the main characteristics of the real-world data sets is provided in Table 44.

Training and Evaluation Methodology: For each data set, the construction of the SPOK-NN classifier is carried out in the following manner. First of all, a kernel function is chosen[5]. Then, all the hyperparameters but ν_1 and σ_κ are kept constant. A grid search is carried out in order to optimize the two remaining hyperparameters. For this purpose, the first 1000 samples of the streaming are used with the aim of minimizing the following cost function:

$$J_{PB}(\mathscr{D}, E_s) = E_s + \lambda D_s, \tag{5.25}$$

[5] In this chapter, just three kernel functions are initially used in order to make the results analysis as simple as possible: linear, Gaussian and Cauchy.

Table 45 – Summary table with the hyperparameters' values and ranges used for the grid search aiming at minimizing the cost function in Eq. (5.25).

Hyperparameter	Symbol	Values
Sparsity level	ν_1	$\{2^{-4}, 2^{-3}, \ldots, 2^3\}$
Maximum number of prototypes	β	$\{600, 1000\}$
Learning rate	η	0.1
Minimum score	ε	-10
Kernel parameter	σ_κ	$\{2^{-10}, 2^{-9}, \ldots, 2^9\}$
Weighting factor	λ	0.5

where D_s is the ratio between the number of samples selected to the dictionary Q_t and the number of samples observed by the algorithm (in this case, 1000); E_s is the sequential error measure, already defined in Eq. (5.2); and $\lambda \geq 0$ is a positive weighting factor that controls the relative importance of D_s.

The rationale for introducing the cost function in Eq. (5.25) is to find a good trade-off between the minimum error (quantified in E_s) and the complexity of the model (quantified in D_s). For larger values of λ, the minimization process is prone to choose hyperparameters that leads to dictionaries with less prototypes and with a bigger classification error. Once the optimization procedure is finished, the dictionary built with the best hyperparameters is used as an initial dictionary for the evaluation of the $N - 1000$ remaining samples.

In Table 45, all the hyperparameters' values and ranges used for the grid search over ν_1 and σ_κ are reported. One important thing to be mentioned is that there are two restrictions on the hyperparameters choice: the minimum number of samples selected to the dictionary should be bigger than the number of classes of the problem. This restriction prevents hyperparameters' values that do not allow the addition of prototypes to the model. Also, the maximum number of samples selected to the dictionary, in the hyperparameter optimization procedure, should be less than 60% of the total of samples used (in this case, 600 prototypes). This prevents the choice of hyperparameters' values that allow excessive and, hence, unnecessary insertions of prototypes into the dictionary.

The remaining $N - 1000$ samples are used to evaluate the classifiers' performance. This is done through the *interleaved test-then-train* (a.k.a prequential) method. According to this approach, at each instant t, one uses the last model H_{t-1} to predict the sample's label. This prediction $\hat{c}_t$ is then compared with the sample's actual label c_t, and an error or a hit is computed. Thereafter, this sample is used to update the last model, creating the current model H_t. So, each sample is used to test the model before being used to train it and the error can be incrementally

updated.

The SPOK-NN classifier was implemented from scratch in MATLAB (R2023b), running on Windows 10 Home, on an HP notebook with an Intel Core i7-7500U processor (2.70 GHz) and 16 GB of RAM.

5.4 Results and Discussion

The results shown in this section are organized in order to emphasize the characteristics of the proposed SPOK-NN classifier and its ability to handle streaming data classification problems with concept drift.

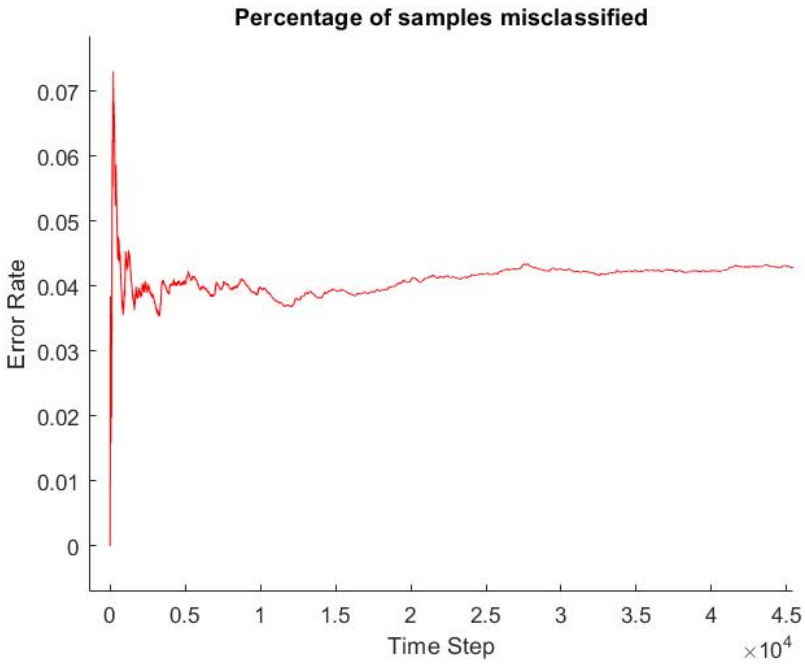
Proof of concept with artificial data sets: Firstly, with the Moving Squares data set, 8.64 (avg.) $\pm$ 1.91 (std.) prototypes were needed to represent the four classes. Even with the movement of each square, the test-then-train error was just 0.0429, showing that the algorithm can handle gradual drift suitably. The evolution of error rate over time, from the first time step until the 45000-th, can be observed in Figure 9. It is interesting to mention the short-term large oscillations observed in the first 10000 steps. This is largely due to a transient period that the classifier experiences while learns the dynamics of the problem. After that, learning tends to be smoother, with smaller variations of the error rate. From the 45000-th step onwards, the error rate remains approximately constant until the last sample (which is 200k for this data set).

With the RBF Interchanging data set, 19.60 (avg.) $\pm$ 2.33 (std.) prototypes were needed to represent the 15 classes. Even with each RBF replacing each other every 3000 samples, the test-then-train error was just 0.0171, showing that the algorithm can handle abrupt drifts nicely. The evolution of the number of prototypes over time is shown in Figure 10 for this data set. One can easily observe that this curve is very spiky. This occurs due to the very nature of the task, in which sudden changes occur every 3000 samples. The proposed SPOK-NN classifier is able to change the number of items in its dictionary by inserting and removing samples according to the demands of the task. The final location of the prototypes is shown in Figure 11.

Finally, with the Transient Chessboard data set, the 8 classes were represented by 74.86 ± 5.26 prototypes, and this number leads to a final error rate of 0.1487. The main conclusion is that the SPOK-NN classifier can suitably hold past concepts that do not conflict with current ones. The final locations of the prototypes are shown in Figure 12.

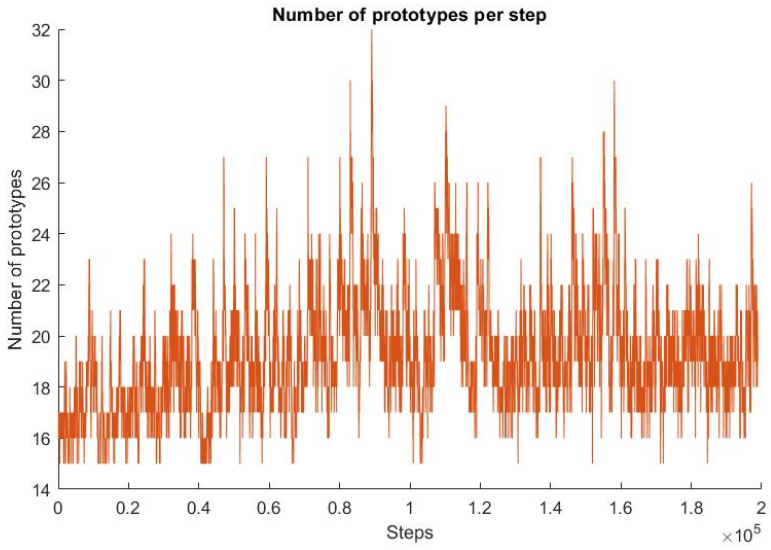
Performance comparisons with the real-world data sets. For the following numerical ex-

Figure 9 – Evolution of the rate of misclassified samples by the proposed SPOK-NN classifier (Moving Squares data set).



Source: Author

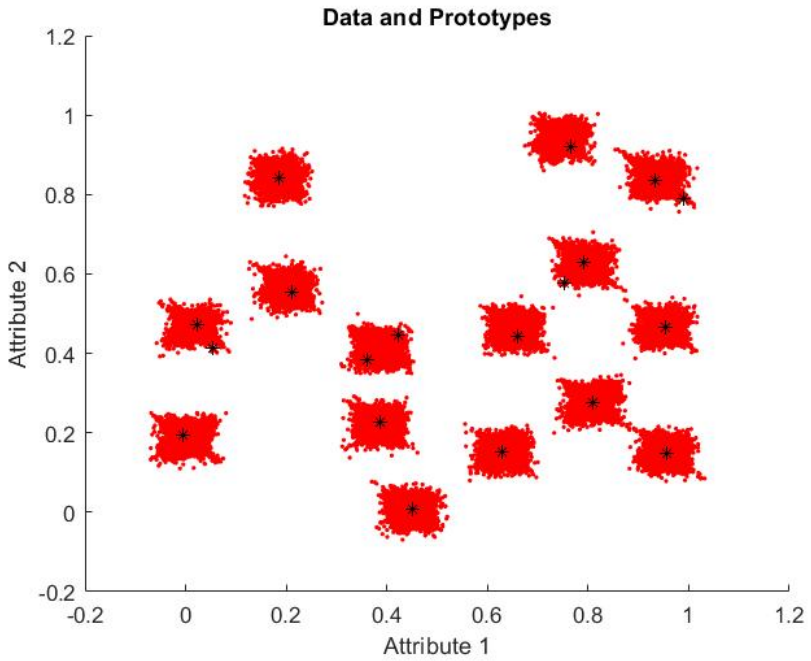
Figure 10 – Number of prototypes per iteration step given by the SPOK-NN classifier to handle the RBF Interchanging data set. The spiky nature of this curve reflects the ability of the SPOK-NN classifier to rapidly adapting its dictionary to the sudden changes imposed by the task.



Source: Author

periments, the same real world data sets used in (LOSING *et al.*, 2016) are considered for a comparative study among 6 algorithms, namely: Learn++.NSE (ELWELL; POLIKAR, 2011), dynamic adaption to concept changes (DACC) (JABER *et al.*, 2013), leveraging bagging (LVGB) (BIFET *et al.*, 2010a), probabilistic adaptive windowing *K*-nearest neighbors (PAW-KNN)

Figure 11 – Final prototypes locations given by the SPOK-NN classifier at the end of the processing of the RBF Interchanging data set.



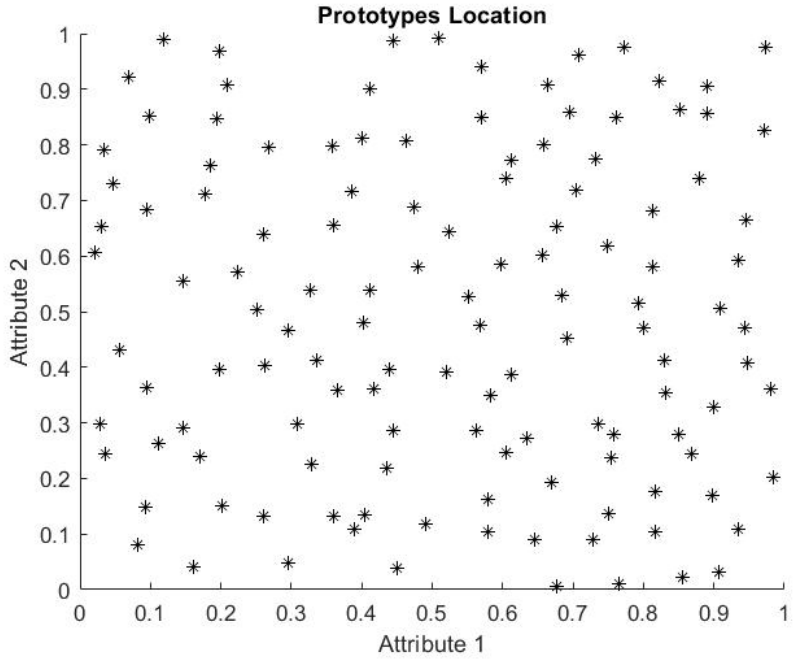
Source: Author

(BIFET *et al.*, 2013), self adjusting memory K-nearest neighbors (SAM-KNN) (LOSING *et al.*, 2016) and a KNN with a sliding window (KNN$_s$). The results in (LOSING *et al.*, 2016) are used as a baseline to assess if the proposed method is suitable for streaming data. It should be noted that a comprehensive comparison of model complexity involving all these algorithms are not carried out in this thesis, since they are built on different learning and data representation paradigms, such as ensembles, trees or data windows. However, a choice for a simplified view of model's complexity was indeed made and explored in the experiments to be reported soon, taking into account the number of prototypes and the classifier accuracy.

The ideas that led to the development of the algorithms previously listed differ in many ways from those behind the SPOK-NN algorithm. In simple words, the SPOK-NN classifier is an online algorithm that builds on-the-fly a fully adaptive dictionary (i.e., that grows, shrinks, and adapts) and then applies a kernelized nearest neighbor rule to classify new data. The L++.NSE, DACC and LVGB algorithms, in their turn, use ensembles[6] of decision trees as classifier models. The KNN$_s$ and PAW algorithms are based on sliding windows only, of fixed or adaptive length, and a standard (i.e. non-kernelized) nearest neighbor rule; that is, they do not actually learn the underlying dynamics of the stream data, since they process the temporally

[6] The Diversity for Dealing with Drifts (DDD) (MINKU; YAO, 2012) is another well known method of ensembles for dealing with streaming data.

Figure 12 – Final locations of the prototypes provided by the SPOK-NN classifier at the end of the processing of the Transient Chessboard data set.



Source: Author

Table 46 – Summary of the performance comparison between the SPOK-NN classifier and alternative methods in real-world data sets (adapted from (LOSING *et al.*, 2016)).

Data Set	**L++.NSE**	**DACC**	**LVGB**	**KNN$_s$**	**KNN$_{W_A}$**	**SAM**	**SPOK**
Covertype	0.1500	0.1005	0.0907	**0.0421**	0.0676	0.0480	0.1167
Electricity	0.2724	0.1687	**0.1678**	0.2861	0.2613	0.1752	0.2582
Outdoor	0.5780	0.3565	0.3997	0.1398	0.1630	**0.1125**	0.1909
Poker Hand	0.2214	0.2097	**0.1365**	0.1708	0.2794	0.1845	0.2689
Rialto	0.4036	0.2893	0.3964	0.2274	0.2496	**0.1858**	0.3817
Weather	0.2288	0.2678	0.2189	**0.2153**	0.2311	0.2174	0.2590

local subsequence of data items provided by the sliding window to make a decision. Finally, among the evaluated algorithms, the SAM-KNN algorithm is the only one that keeps an adaptive dictionary of relevant items as the SPOK-NN classifier. However, the way the dictionary is built is very different. Furthermore, the former uses the standard nearest neighbor, while the latter uses a kernelized version of this rule. Additional details on these differences will be provided later on this section.

The results of the performance comparison in terms of the error rate between the proposed SPOK-NN classifier and the aforementioned alternative algorithms are summarized in Table 46. It can be observed that the error rates of the SPOK-NN were always between the best and worst values of the alternative algorithms.

Table 47 – Performance comparison between the SPOK-NN classifier and the best and worst methods in real-world data sets.

	SPOK-NN		Best Result		Worst Result	
Data set	**#Prot.**	**Error**	**Algorithm**	**Error**	**Algorithm**	**Error**
CoverType	1000	0.1167	KNN_s	0.0421	L++.NSE	0.1500
Electricity	1000	0.2582	LVGB	0.1678	KNN_s	0.2861
Outdoor	192	0.1909	SAM-KNN	0.1125	L++.NSE	0.5780
Poker Hand	24	0.2689	LVGB	0.1365	PAW-KNN	0.2794
Rialto	89	0.3817	SAM-KNN	0.1858	L++.NSE	0.4036
Weather	20	0.2590	KNN_s	0.2153	DACC	0.2678

In Table 47, a closer look at the results shown in Table 46 is done, highlighting the performance comparison also in terms of the number of prototypes used by the SPOK-NN classifier and the best and the worst performing algorithms for each data set. From this perspective, it should be emphasized that the number of prototypes required by the SPOK-NN for achieving such intermediate error rates is much lower than the window/dictionary lengths used by the alternative algorithms. With the Outdoor, Poker Hand, Rialto and Weather data sets, just a few prototypes (192, 24, 89 and 20, respectively) were necessary for achieving acceptable error rates. This is a major feature of the SPOK-NN classifier; that is, the ability to achieve a reasonable balance between model compactness and accuracy, which is important in applications of stream data processing.

Alternatively, the insertion of many more items into the dictionary could have been allowed through the parameters ν_1 (used in the ALD test) and λ (used in the cost function). Also, it would be very simple to increase the maximum allowed number of prototypes in the dictionary (β). However, those options were not chosen since the larger the number of prototypes stored in the dictionary or in the sliding window, the larger the computational costs in memory usage and operations required. Last but not least, a larger number of prototypes does not necessarily lead to a lower error rate. To support this claim, the KNN_s was tested with different sliding window sizes ($\mathscr{W}_s$). These results are summarized in Table 48. As can be seen, for the datasets Moving Squares, RBF Interchanging, Electricity and Rialto, a wider window size in fact led to a larger error rate. For the SPOK-NN algorithm, an increase in the maximum allowed number of prototypes (β) from 600 to 1000 had no effect in the algorithm performance for these data sets because the final number of prototypes is much lower than the prespecified upper limit β. Different error rates occurred for the CoverType and Electricity data sets, because the SPOK-NN algorithm reached the upper limit in the number of prototypes for both. In these cases, an increase in β indeed improved the SPOK-NN performance.

Table 48 – Error rates of KNN_s and SPOK-NN with different window sizes.

	Error Rate			
	KNN_s		**SPOK-NN**	
Data Set	$\mathscr{W}_s = 600$	$\mathscr{W}_s = 1000$	$\beta = 600$	$\beta = 1000$
Moving Squares	0.5603	0.6016	0.0429	0.0429
RBF Interchanging	0.0682	0.1060	0.0171	0.0171
Chessboard	0.1682	0.1670	0.1487	0.1487
Covertype	0.0799	0.0641	0.1253	0.1167
Electricity	0.2298	0.2242	0.2667	0.2582
Outdoor	0.2230	0.2065	0.1909	0.1909
Poker Hand	0.1944	0.1818	0.2689	0.2689
Rialto	0.2653	0.2725	0.3817	0.3817
Weather	0.2240	0.2210	0.2590	0.2590

Table 49 – Best results achieved by the SPOK-NN classifier for each evaluated data sets and the corresponding values of the hyperparameters.

Data set	β	**Error**	**#Prot**	ν_1	**Kernel**	σ_κ
Moving Squares	1000	0.0429	8	0.5	Cauchy	0.0625
RBF Int.	1000	0.0171	19	0.125	Cauchy	0.25
Chessboard	1000	0.1487	78	0.25	Gaussian	0.25
Cover type	1000	0.1167	1000	1	Linear	-
Electricity	1000	0.2582	1000	0.0313	Linear	-
Outdoor	1000	0.1909	192	0.0625	Cauchy	0.5
Poker Hand	1000	0.2689	24	1	Cauchy	1
Rialto	1000	0.3817	89	0.125	Gaussian	0.125
Weather	1000	0.2590	20	0.25	Cauchy	64

Even reaching the upper limit in number of prototypes, the chosen numbers for β (600 and 1000) are much smaller than the maximum window length (i.e., 5000) used in Losing *et al.* (2016). Furthermore, the memories in the KNN_s algorithm (LOSING *et al.*, 2016) are just repositories of items, while the nearest prototype in SPOK-NN's dictionary is updated for each input pattern by means of the WTA learning shown in Eq. (5.13).

A summary table with the best results: A summary report of the SPOK-NN classifier, including the best results obtained for each dataset and the hyperparameter values selected by grid search, is shown in Table 49. In the artificial data sets, with just a few prototypes, the errors are 0.0171, 0.0429 and 0.1487. As mentioned before, for the real-world data sets, the error rates are competitive with state of the art algorithms (see (LOSING *et al.*, 2016)), an achievement reached by the combined use of a fully adaptive dictionary and a kernelized nearest neighbor classifier.

A closer look into Table 47 reveals that the algorithms achieving the best results in terms of error rates for the real world data sets are the following ones: KNN_s (CoverType and Weather), LVGB (Electricity and Poker Hand) and SAM-KNN (Rialto and Outdoor). However,

attention also should be paid to the number of prototypes required to reach those error rates. The KNN$_s$ algorithm, for example, has a fixed sliding window of 5000 samples.

For the sake of comparison, taking the results provided in Losing *et al.* (2016), the SAM-KNN algorithm required from 500 to 3500 prototypes to reach a low error rate for the interchanging RBF data set. The SPOK-NN, by its turn, using less than 40 prototypes, reached a low error rate. For the moving squares data set, the SPOK-NN demands just 8 prototypes to reach 0.0429 error rate, while the SAM-KNN algorithm needed from 50 (the minimum size of their short-term memory sliding window) to 200 prototypes.

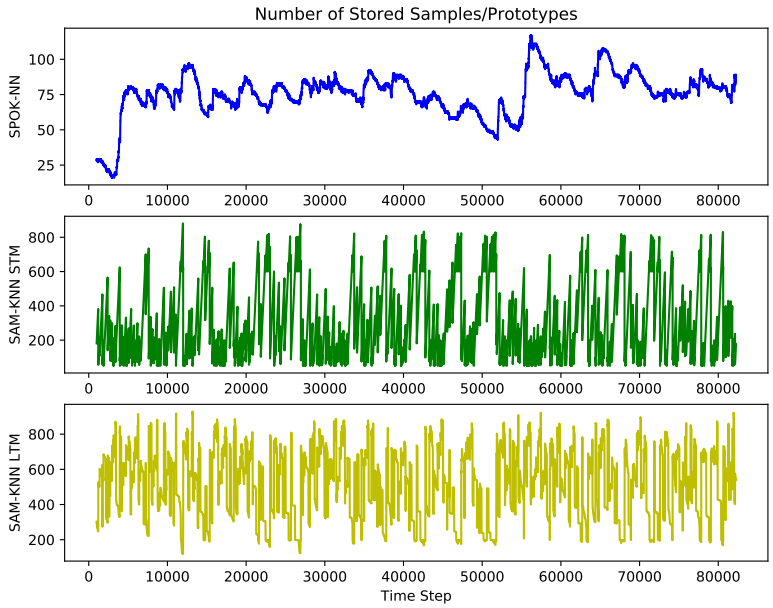
Since the authors in Losing *et al.* (2016) did not mention the number of prototypes used by their algorithms for the real world data sets, a implementation of the SAM-KNN algorithm provided by them[7] was used to observe the differences about the adaptation of SPOK-NN dictionary and the two types of memory in SAM-KNN (short-term and long-term). For these experiments, the values for β (maximum dictionary size of SPOK-NN) and L_{max} (maximum number of stored examples in SAM-KNN memories) were set to 1000.

The SAM-KNN relies on a window (short-term memory) of adaptive size, with the latest samples. Each time this window is shortened, some samples are stored in a dictionary (the long-term memory). Every time the sum of samples in these two memories reaches L_{max}, the samples in LTM are compressed in half, by using the kMeans++ algorithm. This behavior can be seen in Figures 13 and 14. With the Rialto data set (Figure 13), the adaptation of each SAM-KNN memory is shown in a different subfigure. Taking into account the sum of samples/prototypes in both memories, 824.25 ± 109.28 prototypes were necessary to reach an error rate of 0.1831. Moreover, with the Weather data set (Figure 14), these memories are shown simultaneously in a subfigure, and summed in the other one. Taking into account the sum of samples/prototypes in both memories, 799.24 ± 109.62 prototypes were necessary to reach an error rate of 0.2252.

From this perspective, an interesting feature of the SPOK-NN algorithm is its fully adaptive dictionary, in the sense that it chooses the adequate number of prototypes on the fly and updates their positions if necessary. It does not rely on either sliding windows (SAM-KNN and KNN$_s$) or ensembles (the LVGB relies on these two concepts, and needs to constantly verify the error on a window in order to update its models). In order to update the SPOK-NN model, just the current sample is needed. The evolution of the dictionary size for the SPOK-NN algorithm is also shown in Figures 13 and 14. With the Rialto data set, 75.97 ± 15.40 prototypes were

[7] https://github.com/vlosing/SAMkNN

Figure 13 – Comparison between number of stored samples/prototypes per iteration in SPOK-NN dictionary and SAM-KNN memories (STM and LTM) with Rialto dataset.



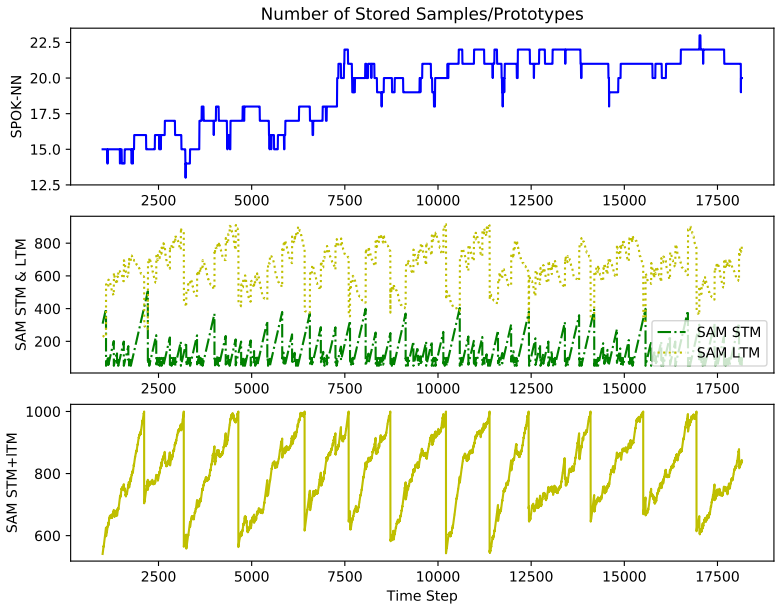
Source: Author

necessary to reach an error rate of 0.3817. With the Weather data set, 19.25 ± 2.38 prototypes were necessary to reach an error rate of 0.2590.

In summary, the final number of prototypes in the SPOK-NN dictionary is much smaller than the number of items jointly stored in SAM-KNN's memories (STM+LTM), with both algorithms achieving suitable error rates. More specifically, the final error rate for the Rialto data set was higher (though acceptable) for the SPOK-NN. However, for the Weather data set, the error rate achieved by the SPOK-NN algorithm was equivalent to that achieved by the SAM-KNN and this was possible using a much smaller number of prototypes. More sofisticated models could have been built over the prototypes in the dictionary, such as a distance weighted K-nearest neighbors (DWKNN) (RUBIO *et al.*, 2010), least squares support vector machine (LSSVM) (SUYKENS; VANDEWALLE, 1999), or kernel quadratic discriminant (KQD) (HAASDONK; PęKALSKA, 2009) in order to achieve even smaller error rates. But, in this preliminar tests, the methodology was kept as simple as possible in order to meet the requirements of stream data processing.

The role of prototype updating in SPOK-NN: In order to highlight the importance of updating the prototypes for the performance of the SPOK-NN algorithm, an evaluation was conducted just

Figure 14 – Comparison between number of stored samples/prototypes per iteration in SPOK-NN dictionary and SAM-KNN memories (STM and LTM) with Weather dataset.



Source: Author

Table 50 – Best results achieved by the SPOK-NN with or without the update procedure.

	With Update		**Without Update**	
Data Set	**Error**	**#Prot**	**Error**	**#Prot**
Moving Squares	0.0429	8	0.2239	19
RBF Int	0.0171	19	0.0517	18
Chessboard	0.1487	78	0.1923	164
Electricity	0.2582	1000	0.3443	860
Outdoor	0.1909	192	0.1741	615
Rialto	0.3817	89	0.4567	1000
Weather	0.2590	20	0.3670	11

with the adding and removing procedures. These results are summarized at Table 50. In most data sets, when not using the updating procedure, it was observed a deterioration in accuracy (even with the increase in the number of prototypes in the dictionary). With the Outdoor, there was a 1.68% decrease in error, but the dictionary was more than three times bigger comparing to the one built with the prototype updating procedure included.

As a final remark, it is important to mention that a fundamental difference of the proposed method with respect to the ones evaluated in this chapter, is that it is the only one that relies on kernel computations. As such, the choice of the kernel function and the associated hyperparameters directly affects the error rate. So, as the scale parameter σ_κ affects also the

Table 51 – Performance comparison between the SPOK-KNN classifier and the best and worst methods in real-world data sets.

	SPOK-KNN		SPOK-NN		Best Result		Worst Result	
Data set	**#Prot.**	**Error**	**#Prot.**	**Error**	**Algorithm**	**Error**	**Algorithm**	**Error**
CoverType	1000 (2-NN)	0.1085	1000	0.1167	KNN_s	0.0421	L++.NSE	0.1500
Electricity	1000 (7-NN)	0.2469	1000	0.2582	LVGB	0.1678	KNN_s	0.2861
Outdoor	435 (6-NN)	0.1141	192	0.1909	SAM-KNN	0.1125	L++.NSE	0.5780
Poker Hand	1000 (9-NN)	0.2108	24	0.2689	LVGB	0.1365	PAW-KNN	0.2794
Rialto	98 (2-NN)	0.3690	89	0.3817	SAM-KNN	0.1858	L++.NSE	0.4036
Weather	54 (7-NN)	0.2232	20	0.2590	KNN_s	0.2153	DACC	0.2678

number of prototypes added to the model, all but this hyperparameter and the sparsity level ν_1 were kept constant. Just the first 1000 samples were chosen to execute a crude hyperparameter optimization, since there is no information about drifts, and just a little information about the initial data distribution. For sequential learning problems, one can always collect some data before training the model in an online fashion.

5.5 Evaluating sparsification methods in the SPOK model

In this section, motivated by the results obtained with the SPOK framework using the ALD sparsification procedure (as in Chapter 4, when applying the SPARK framework to batch datasets), other sparsification procedures were investigated to build sparse kernel prototype-based classifiers for streaming datasets. As in Section 5.3, the SPOK framework performance is evaluated across three artificial and six real-world datasets, but know using four sparsification procedures and eight kernel functions. The training and evaluation methodology remains the same as in Section 5.3. Additionally, the KWKNN strategy is applied to further improve classification. Detailed results for each dataset and SPOK model configurations are provided in Appendix D.

As shown in Table 51, using alternative sparsification procedures combined with the KWKNN strategy improved the overall accuracy of the SPOK models. Two datasets are particularly noteworthy. For the Outdoor dataset, 435 prototypes were required to nearly reach the best result. For the Weather dataset, 54 prototypes were sufficient, with the SPOK performance differing from the best result by less than 1%.

5.6 Final Considerations

In this chapter, a novel adaptive prototype-based kernel classifier was introduced aimed at efficient online processing of stream data. The proposed classifier, named SPOK-NN, was designed to balance the trade-off between accuracy (i.e. small error) and complexity (size of the dictionary). This trade-off is particularly important for tasks involving continuous online learning with gradual and abrupt concept drift.

The design of the SPOK-NN is inspired by competitive learning algorithms and in how they deal with the stability-plasticity dilemma and, hence, with the issue of catasthophic forgetting. In this regard, the SPOK-NN classifier is stable since it only keeps relevant prototypes in a dictionary. It is adaptive since it keeps learning relevant information from data even if the sample is not inserted into the dictionary. Stable unlearning of information that became irrelevant for current state of the task is also possible by carefully removing items from the dictionary. Finally, the proposed SPOK-NN classifier is accurate (i.e., it achieves acceptable low error rates) in classifying stream data in nonstationary scenarios.

These characteristics of the SPOK-NN classifier were evaluated, firstly, using benchmarking artificial data sets, namely, Moving Squares, Interchanging RBF, and Transient Chessboard, which present, respectively, gradual, abrupt and recurring drifts. The error rates and the number of prototypes for these artificial data sets were, in the best scenario, respectively, 0.0429 and 8; 0.0171 and 19; 0.1487 and 78. Then, the proposed algorithm was also evaluated using benchmarking real-world data sets, such as the Forest Cover Type, Electricity, Outdoor, Poker Hand, Weather and Rialto. The resulting classifier was able to achieve equivalent or smaller error rates (in comparison to other state of the art algorithms) with lower numbers of prototypes.

In the next chapter, the thesis conclusions are made, and some open problemas are pointed out.

6 CONCLUSIONS AND OPEN PROBLEMS

In this thesis, new contributions were made to the development of prototype-based classifiers using kernel methods. Firstly, the application of the kernel trick to the self-organizing map (SOM) in the classification of batch datasets has shown that computing both the distance measure and the winning prototype in the feature space is the best strategy. Additionally, using alternative kernel functions, instead of only linear and Gaussian, can lead to better results.

Then, as defining the number of prototypes is very important for the accuracy of prototype-based classifiers, the use of sparsification procedures, such as ALD, novelty, surprise and coherence, is proposed for the automatic selection of this quantity, in classification problems with batch data sets, applying the weighted K-nearest neighbors strategy. The results showed that the SPARK method can effectively handle datasets in batch format, building models with a reasonable number or prototypes and, in most cases, achieving higher accuracy compared with the KSOM model with a fixed number of prototypes. Moreover, no single sparsification method or kernel function consistently outperformed the others across all datasets, reinforcing the need for careful selection of sparsification criteria, kernel functions, and hyperparameters tailored to each specific task.

Finally, a novel method, named SPOK, for designing sparse kernel prototype-based classifiers is introduced, capable of handling the challenges posed by the processing of streaming data. This is achieved by allowing items in the dictionary to be removed or updated. The results with real-world and synthetic datasets show that this model is competitive with state-of-the-art approaches while using a small number of prototypes.

It is important to mention that many open questions arose from this work. Firstly, one can investigate why a particular sparsification method or kernel function performs better for a specific dataset. Additionally, other distance measures can be used, such as the Mahalanobis distance, fuzzy distance or the fractional order minkowiski distance. Moreover, the SPARK model can be used as a strategy to select the number of prototypes for other prototype-based classifiers, such as SOM and LVQ. Furthermore, more complex models can be built using the prototypes generated by these methods. Also, an ensemble of SPARK models, employing different sparsification strategies, can also be applied for batch learning. Finally, the SPARK and SPOK models can be adapted and evaluated in regression tasks.

BIBLIOGRAPHY

ALBUQUERQUE, R. F.; OLIVEIRA, P. D. de; BRAGA, A. P. d. S. Adaptive fuzzy learning vector quantization (AFLVQ) for time series classification. In: SPRINGER. **North American Fuzzy Information Processing Society Annual Conference (NAFIPS'2018)**. Springer, 2018. p. 385–397. Available at: https://doi.org/10.1007/978-3-319-95312-0_33.

ALIYU, A.; ABDULLAH, A. H.; KAIWARTYA, O.; CAO, Y.; LLORET, J.; ASLAM, N.; JODA, U. M. Towards video streaming in IoT environments: Vehicular communication perspective. **Computer Communications**, v. 118, p. 93–119, 2018. Available at: https://doi.org/10.1016/j.comcom.2017.10.003.

ANDRAS, P. Kernel-kohonen networks. **International Journal of Neural Systems**, World Scientific, v. 12, n. 02, p. 117–135, 2002. Available at: https://doi.org/10.1142/S0129065702001084.

AUGENSTEIN, C.; SPANGENBERG, N.; FRANCZYK, B. Applying machine learning to big data streams: An overview of challenges. In: IEEE. **2017 IEEE 4th International Conference on Soft Computing & Machine Intelligence (ISCMI)**. Mauritius: IEEE, 2017. p. 25–29. Available at: https://doi.org/10.1109/ISCMI.2017.8279592.

AURENHAMMER, F. Voronoi diagrams—a survey of a fundamental geometric data structure. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 23, n. 3, p. 345–405, 1991. Available at: https://doi.org/10.1145/116873.116880.

AYAT, N.-E.; CHERIET, M.; SUEN, C. Y. KMOD - a two-parameter SVM kernel for pattern recognition. In: IEEE. **2002 International Conference on Pattern Recognition**. Quebec City, QC, Canada, 2002. v. 3, p. 331–334. Available at: https://doi.org/10.1109/ICPR.2002.1047860.

BABÜROĞLU, E. S.; DURMUŞOĞLU, A.; DERELI, T. Concept drift from 1980 to 2020: a comprehensive bibliometric analysis with future research insight. **Evolving Systems**, Springer, v. 15, n. 3, p. 789–809, 2024. Available at: https://doi.org/10.1007/s12530-023-09503-2.

BAN, T.; ZHANG, R.; PANG, S.; SARRAFZADEH, A.; INOUE, D. Referential kNN regression for financial time series forecasting. In: SPRINGER. **Neural Information Processing: 20th International Conference, ICONIP 2013, Daegu, Korea, November 3-7, 2013. Proceedings, Part I 20**. Daegu, Korea, 2013. p. 601–608. Available at: https://doi.org/10.1007/978-3-642-42054-2_75.

BERGSTRA, J.; BENGIO, Y. Random search for hyper-parameter optimization. **Journal of machine learning research**, v. 13, n. 2, p. 281–305, 2012. Available at: https://dl.acm.org/doi/abs/10.5555/2188385.2188395.

BIEHL, M.; HAMMER, B.; VILLMANN, T. Prototype-based models in machine learning. **Wiley Interdisciplinary Reviews: Cognitive Science**, Wiley Online Library, v. 7, n. 2, p. 92–111, 2016. Available at: https://doi.org/10.1002/wcs.1378.

BIFET, A.; GAVALDA, R. Learning from time-changing data with adaptive windowing. In: SIAM. **Proceedings of the 2007 SIAM international conference on data mining**. Minnesota, USA, 2007. p. 443–448. Available at: https://doi.org/10.1137/1.9781611972771.42.

BIFET, A.; HOLMES, G.; PFAHRINGER, B. Leveraging bagging for evolving data streams. In: BALCÁZAR, J. L.; BONCHI, F.; GIONIS, A.; SEBAG, M. (Ed.). **Machine Learning and Knowledge Discovery in Databases**. Berlin, Heidelberg, 2010. p. 135–150. Available at: https://doi.org/10.1007/978-3-642-15880-3_15.

BIFET, A.; HOLMES, G.; PFAHRINGER, B.; KRANEN, P.; KREMER, H.; JANSEN, T.; SEIDL, T. MOA: Massive online analysis, a framework for stream classification and clustering. In: DIETHE, T.; CRISTIANINI, N.; SHAWE-TAYLOR, J. (Ed.). **Proceedings of the First Workshop on Applications of Pattern Analysis**. Cumberland Lodge, Windsor, UK: PMLR, 2010. (Proceedings of Machine Learning Research, v. 11), p. 44–50. Available at: https://proceedings.mlr.press/v11/bifet10a.html.

BIFET, A.; PFAHRINGER, B.; READ, J.; HOLMES, G. Efficient data stream classification via probabilistic adaptive windows. In: **Proceedings of the 28th Annual ACM Symposium on Applied Computing**. ACM, 2013. (SAC '13), p. 801–806. Available at: http://dx.doi.org/10.1145/2480362.2480516.

BIGDELI, A.; MAGHSOUDI, A.; GHEZELBASH, R. Application of self-organizing map (SOM) and K-means clustering algorithms for portraying geochemical anomaly patterns in moalleman district, NE iran. **Journal of Geochemical Exploration**, Elsevier BV, v. 233, p. 106923, fev. 2022. ISSN 0375-6742. Available at: https://doi.org/10.1016/j.gexplo.2021.106923.

BISHOP, C. M. **Pattern Recognition and Machine Learning**. New York: Springer, 2006. (Information Science and Statistics). ISBN 978-0-387-31073-2. Available at: https://www.springer.com/gp/book/9780387310732.

BOUGHORBEL, S.; TAREL, J.-P.; BOUJEMAA, N. Conditionally positive definite kernels for SVM based image recognition. In: **Proceedings of the IEEE International Conference on Multimedia & Expo (ICME'2005)**. IEEE, 2005. p. 113–116. Available at: http://dx.doi.org/10.1109/ICME.2005.1521373.

BRNA, A. P.; BROWN, R. C.; CONNOLLY, P. M.; SIMONS, S. B.; SHIMIZU, R. E.; AGUILAR-SIMON, M. Uncertainty-based modulation for lifelong learning. **Neural Networks**, Elsevier BV, v. 120, p. 129–142, dez. 2019. ISSN 0893-6080. Available at: https://www.sciencedirect.com/science/article/pii/S0893608019302722.

CARPENTER, G. A.; GROSSBERG, S.; ROSEN, D. B. Fuzzy ART: Fast stable learning, categorization of analog patterns by an adaptive resonance system. **Neural Networks**, Elsevier BV, v. 4, n. 6, p. 759–771, jan. 1991. ISSN 0893-6080. Available at: https://www.sciencedirect.com/science/article/pii/089360809190056B.

CARRINGTON, A. M.; FIEGUTH, P. W.; CHEN, H. H. A new mercer sigmoid kernel for clinical data classification. In: **2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society**. IEEE, 2014. p. 6397–6401. Available at: http://dx.doi.org/10.1109/EMBC.2014.6945092.

CHOW, C. On optimum recognition error and reject tradeoff. **IEEE Transactions on Information Theory**, Institute of Electrical and Electronics Engineers (IEEE), v. 16, n. 1, p. 41–46, jan. 1970. ISSN 1557-9654. Available at: http://dx.doi.org/10.1109/TIT.1970.1054406.

CHUA, S.-L.; MARSLAND, S.; GUESGEN, H. W. Unsupervised learning of patterns in data streams using compression and edit distance. In: **Twenty-Second International Joint**

Conference on Artificial Intelligence. Barcelona, Spain: AAAI Press, 2011. v. 1, p. 1231–1236. Available at: https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-209.

COELHO, D. N. **Detecção neural de falhas incipientes de curto circuito no estator do motor de indução trifásico gaiola de esquilo**. Dissertação (Bachelor's thesis) – Universidade Federal do Ceará–UFC, 2012.

COELHO, D. N.; BARRETO, G.; MEDEIROS, C. M.; SANTOS, J. D. A. Performance comparison of classifiers in the detection of short circuit incipient fault in a three-phase induction motor. In: **2014 IEEE Symposium on Computational Intelligence for Engineering Solutions (CIES)**. IEEE, 2014. p. 42–48. Available at: http://dx.doi.org/10.1109/CIES.2014.7011829.

COELHO, D. N.; BARRETO, G. A.; MEDEIROS, C. M. Detection of short circuit faults in 3-phase converter-fed induction motors using kernel SOMs. In: **2017 12th International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM)**. IEEE, 2017. p. 1–7. Available at: http://dx.doi.org/10.1109/WSOM.2017.8020016.

COVER, T.; HART, P. Nearest neighbor pattern classification. **IEEE Transactions on Information Theory**, Institute of Electrical and Electronics Engineers (IEEE), v. 13, n. 1, p. 21–27, jan. 1967. ISSN 1557-9654. Available at: http://dx.doi.org/10.1109/TIT.1967.1053964.

COX, D. R. The regression analysis of binary sequences. **Journal of the Royal Statistical Society Series B: Statistical Methodology**, Oxford University Press (OUP), v. 20, n. 2, p. 215–232, jul. 1958. ISSN 1467-9868. Available at: http://dx.doi.org/10.1111/j.2517-6161.1958.tb00292.x.

CUI, B.; DING, Y. Accurate identification of human phosphorylated proteins by ensembling supervised kernel self-organizing maps. **Molecular Informatics**, Wiley Online Library, v. 39, n. 7, p. 1900141, mar. 2020. ISSN 1868-1751. Available at: http://dx.doi.org/10.1002/minf.201900141.

DE SOUZA, C. R. **Kernel Functions for Machine Learning Applications**. 2010. Available at: https://davejingtian.org/2010/09/10/kernel-functions-in-machine-learning-transfered/.

DITZLER, G.; ROVERI, M.; ALIPPI, C.; POLIKAR, R. Learning in nonstationary environments: A survey. **IEEE Computational Intelligence Magazine**, Institute of Electrical and Electronics Engineers (IEEE), v. 10, n. 4, p. 12–25, nov. 2015. ISSN 1556-603X. Available at: http://dx.doi.org/10.1109/MCI.2015.2471196.

DUDA, R. O.; HART, P. E.; STORK, D. G. **Pattern Classification**. 2nd. ed. New York: Wiley-Interscience, 2001. ISBN 978-0-471-05669-0. Available at: https://www.wiley.com/en-us/Pattern+Classification%2C+2nd+Edition-p-9780471056690.

DUDANI, S. A. The distance-weighted k-nearest-neighbor rule. **IEEE Transactions on Systems, Man, and Cybernetics**, Institute of Electrical and Electronics Engineers (IEEE), SMC-6, n. 4, p. 325–327, abr. 1976. ISSN 0018-9472. Available at: https://ieeexplore.ieee.org/abstract/document/5408784.

ELWELL, R.; POLIKAR, R. Incremental learning of concept drift in nonstationary environments. **IEEE Transactions on Neural Networks**, Institute of Electrical and Electronics Engineers (IEEE), v. 22, n. 10, p. 1517–1531, out. 2011. ISSN 1941-0093. Available at: http://dx.doi.org/10.1109/TNN.2011.2160459.

ENGEL, Y.; MANNOR, S.; MEIR, R. The kernel recursive least-squares algorithm. **IEEE Transactions on Signal Processing**, Institute of Electrical and Electronics Engineers (IEEE), v. 52, n. 8, p. 2275–2285, ago. 2004. ISSN 1053-587X. Available at: http://dx.doi.org/10.1109/TSP.2004.830985.

FAWCETT, T. An introduction to ROC analysis. **Pattern Recognition Letters**, Elsevier BV, v. 27, n. 8, p. 861–874, jun. 2006. ISSN 0167-8655. Available at: http://dx.doi.org/10.1016/j.patrec.2005.10.010.

FIX, E.; HODGES, J. L. Discriminatory analysis. nonparametric discrimination: Consistency properties. **International Statistical Review / Revue Internationale de Statistique**, JSTOR, v. 57, n. 3, p. 238, dez. 1989. ISSN 0306-7734. Available at: http://dx.doi.org/10.2307/1403797.

FREIRE, A. L.; BARRETO, G. A.; VELOSO, M.; VARELA, A. T. Short-term memory mechanisms in neural network learning of robot navigation tasks: A case study. In: IEEE. **2009 6th Latin American robotics symposium (LARS 2009)**. IEEE, 2009. p. 1–6. Available at: http://dx.doi.org/10.1109/LARS.2009.5418323.

FRÍAS, M. P.; MARTÍNEZ, F. An ensemble for automatic time series forecasting with K-nearest neighbors. **IEEE Access**, IEEE, v. 13, p. 4117–4125, 2025. Available at: https://doi.org/10.1109/ACCESS.2025.3525561.

GAMA, J.; ŽLIOBAITė, I.; BIFET, A.; PECHENIZKIY, M.; BOUCHACHIA, A. A survey on concept drift adaptation. **ACM Computing Surveys**, Association for Computing Machinery (ACM), v. 46, n. 4, p. 1–37, mar. 2014. ISSN 1557-7341. Available at: http://dx.doi.org/10.1145/2523813.

GARCIA, K.; FORSTER, C. H. Q. Supervised growing neural gas. In: ____. **Intelligent Data Engineering and Automated Learning - IDEAL 2012**. Springer Berlin Heidelberg, 2012. p. 502–507. ISBN 9783642326394. Available at: http://dx.doi.org/10.1007/978-3-642-32639-4_61.

GOMES, H. M.; BARDDAL, J. P.; ENEMBRECK, F.; BIFET, A. A survey on ensemble learning for data stream classification. **ACM Computing Surveys**, Association for Computing Machinery (ACM), v. 50, n. 2, p. 1–36, mar. 2017. ISSN 1557-7341. Available at: http://dx.doi.org/10.1145/3054925.

GOMES, H. M.; BIFET, A.; READ, J.; BARDDAL, J. P.; ENEMBRECK, F.; PFHARINGER, B.; HOLMES, G.; ABDESSALEM, T. Adaptive random forests for evolving data stream classification. **Machine Learning**, Springer Science and Business Media LLC, v. 106, n. 9–10, p. 1469–1495, jun. 2017. ISSN 1573-0565. Available at: http://dx.doi.org/10.1007/s10994-017-5642-8.

GROSSBERG, S. Competitive learning: From interactive activation to adaptive resonance. **Cognitive Science**, Wiley, v. 11, n. 1, p. 23–63, jan. 1987. ISSN 1551-6709. Available at: http://dx.doi.org/10.1111/j.1551-6708.1987.tb00862.x.

HAASDONK, B.; PęKALSKA, E. Classification with kernel mahalanobis distance classifiers. In: ____. **Advances in Data Analysis, Data Handling and Business Intelligence**. Springer Berlin Heidelberg, 2009. p. 351–361. ISBN 9783642010446. Available at: http://dx.doi.org/10.1007/978-3-642-01044-6_32.

HAMMER, B.; HOFMANN, D.; SCHLEIF, F.-M.; ZHU, X. Learning vector quantization for (dis-)similarities. **Neurocomputing**, Elsevier BV, v. 131, p. 43–51, maio 2014. ISSN 0925-2312. Available at: http://dx.doi.org/10.1016/j.neucom.2013.05.054.

HARKAT, M.-F.; KOUADRI, A.; FEZAI, R.; MANSOURI, M.; NOUNOU, H.; NOUNOU, M. Machine learning-based reduced kernel PCA model for nonlinear chemical process monitoring. **Journal of Control, Automation and Electrical Systems**, Springer Science and Business Media LLC, v. 31, n. 5, p. 1196–1209, maio 2020. ISSN 2195-3899. Available at: http://dx.doi.org/10.1007/s40313-020-00604-w.

HARRIES, M. Splice-2 comparative evaluation: Electricity pricing. University of New South Wales, School of Computer Science and Engineering, 1999. Available at: https://cgi.cse.unsw.edu.au/~reports/papers/9905.pdf.

HAYKIN, S.; LI, L. Nonlinear adaptive prediction of nonstationary signals. **IEEE Transactions on Signal Processing**, Institute of Electrical and Electronics Engineers (IEEE), v. 43, n. 2, p. 526–535, 1995. ISSN 1053-587X. Available at: http://dx.doi.org/10.1109/78.348134.

HECHENBICHLER, K.; SCHLIEP, K. **Weighted k-Nearest-Neighbor Techniques and Ordinal Classification**. 2004. (sfb386, v. 399). Available at: https://epub.ub.uni-muenchen.de/1769.

HEUSINGER, M.; RAAB, C.; SCHLEIF, F.-M. Passive concept drift handling via variations of learning vector quantization. **Neural Computing and Applications**, Springer Science and Business Media LLC, v. 34, n. 1, p. 89–100, ago. 2020. ISSN 1433-3058. Available at: http://dx.doi.org/10.1007/s00521-020-05242-6.

HOFMANN, D.; HAMMER, B. Sparse approximations for kernel learning vector quantization. In: **21st European Symposium on Artificial Neural Networks, ESANN 2013, Bruges, Belgium, April 24-26, 2013**. [*S. n.*], 2013. p. 549–554. Available at: https://www.esann.org/sites/default/files/proceedings/legacy/es2013-63.pdf.

HOFMANN, D.; SCHLEIF, F.-M.; PAASSEN, B.; HAMMER, B. Learning interpretable kernelized prototype-based models. **Neurocomputing**, Elsevier BV, v. 141, p. 84–96, out. 2014. ISSN 0925-2312. Available at: http://dx.doi.org/10.1016/j.neucom.2014.03.003.

HUTTER, F.; KOTTHOFF, L.; VANSCHOREN, J. **Automated machine learning: methods, systems, challenges**. Springer International Publishing, 2019. ISSN 2520-1328. ISBN 9783030053185. Available at: http://dx.doi.org/10.1007/978-3-030-05318-5.

IWASHITA, A. S.; PAPA, J. P. An overview on concept drift learning. **IEEE Access**, Institute of Electrical and Electronics Engineers (IEEE), v. 7, p. 1532–1547, 2018. ISSN 2169-3536. Available at: http://dx.doi.org/10.1109/ACCESS.2018.2886026.

JABER, G.; CORNUÉJOLS, A.; TARROUX, P. Online learning: Searching for the best forgetting strategy under concept drift. In: **International Conference on Neural Information Processing**. Springer Berlin Heidelberg, 2013. p. 400–408. ISBN 9783642420429. ISSN 1611-3349. Available at: http://dx.doi.org/10.1007/978-3-642-42042-9_50.

JAFARI, F.; NASSERY, H. R.; ALIJANI, F.; GILANI, S. M. Identification of salinity sources in groundwater at golgohar mine using self-organizing maps (SOM) and correlation analysis: a hydrogeochemical and isotopic approach, south-central iran. **Environmental Geochemistry**

and Health, Springer Science and Business Media LLC, v. 47, n. 4, p. 1–25, mar. 2025. ISSN 1573-2983. Available at: http://dx.doi.org/10.1007/s10653-025-02414-y.

JäKEL, F.; SCHöLKOPF, B.; WICHMANN, F. A. A tutorial on kernel methods for categorization. **Journal of Mathematical Psychology**, Elsevier BV, v. 51, n. 6, p. 343–358, dez. 2007. ISSN 0022-2496. Available at: http://dx.doi.org/10.1016/j.jmp.2007.06.002.

JANTZEN, J.; NORUP, J.; DOUNIAS, G.; BJERREGAARD, B. Pap-smear benchmark data for pattern classification. **Nature inspired Smart Information Systems (NiSIS 2005)**, p. 1–9, 2005. Available at: https://orbit.dtu.dk/en/publications/pap-smear-benchmark-data-for-pattern-classification/.

JUÁREZ-RUIZ, E.; CORTÉS-MALDONADO, R.; PÉREZ-RODRÍGUEZ, F. Relationship between the inverses of a matrix and a submatrix. **Computación y Sistemas**, Instituto Politecnico Nacional/Centro de Investigacion en Computacion, v. 20, n. 2, p. 251–262, jul. 2016. ISSN 1405-5546. Available at: http://dx.doi.org/10.13053/cys-20-2-2083.

KOHONEN, T. Improved versions of learning vector quantization. In: **Proceedings of the 1990 International Joint Conference on Neural Networks (IJCNN'90)**. IEEE, 1990. p. 545–550. Available at: http://dx.doi.org/10.1109/IJCNN.1990.137622.

KOHONEN, T. The self-organizing map. **Proceedings of the IEEE**, Institute of Electrical and Electronics Engineers (IEEE), v. 78, n. 9, p. 1464–1480, 1990. ISSN 0018-9219. Available at: http://dx.doi.org/10.1109/5.58325.

KOHONEN, T. Essentials of the self-organizing map. **Neural Networks**, Elsevier BV, v. 37, p. 52–65, jan. 2013. ISSN 0893-6080. Available at: http://dx.doi.org/10.1016/j.neunet.2012.09.018.

KRISHNA, K.; RAMAKRISHNAN, K.; THATHACHAR, M. Vector quantization using genetic K-means algorithm for image compression. In: **Proceedings of ICICS, 1997 International Conference on Information, Communications and Signal Processing. Theme: Trends in Information Systems Engineering and Wireless Multimedia Communications (Cat. No.97TH8237)**. IEEE, 1997. (ICICS-97, v. 3), p. 1585–1587. Available at: http://dx.doi.org/10.1109/ICICS.1997.652261.

LAU, K. W.; YIN, H.; HUBBARD, S. Kernel self-organising maps for classification. **Neurocomputing**, Elsevier BV, v. 69, n. 16–18, p. 2033–2040, out. 2006. ISSN 0925-2312. Available at: http://dx.doi.org/10.1016/j.neucom.2005.10.003.

LEE, D.-T. Two-dimensional voronoi diagrams in the lp-metric. **Journal of the ACM (JACM)**, Association for Computing Machinery (ACM), v. 27, n. 4, p. 604–618, out. 1980. ISSN 1557-735X. Available at: http://dx.doi.org/10.1145/322217.322219.

LI, X.; YU, W. Data stream classification for structural health monitoring via on-line support vector machines. In: **2015 IEEE First International Conference on Big Data Computing Service and Applications**. IEEE, 2015. p. 400–405. Available at: http://dx.doi.org/10.1109/BigDataService.2015.17.

LI, Z.; HUANG, W.; XIONG, Y.; REN, S.; ZHU, T. Incremental learning imbalanced data streams with concept drift: The dynamic updated ensemble algorithm. **Knowledge-Based Systems**, Elsevier BV, v. 195, p. 105694, maio 2020. ISSN 0950-7051. Available at: http://dx.doi.org/10.1016/j.knosys.2020.105694.

LIASHCHYNSKYI, P.; LIASHCHYNSKYI, P. Grid search, random search, genetic algorithm. a big comparison for NAS. **arXiv preprint:1912.06059**, arXiv, 2019. Available at: https://arxiv.org/abs/1912.06059.

LIU, W.; PARK, I.; PRINCIPE, J. C. An information theoretic approach of designing sparse kernel adaptive filters. **IEEE Transactions on Neural Networks**, Institute of Electrical and Electronics Engineers (IEEE), v. 20, n. 12, p. 1950–1961, dez. 2009. ISSN 1941-0093. Available at: http://dx.doi.org/10.1109/TNN.2009.2033676.

LIU, W.; POKHAREL, P. P.; PRINCIPE, J. C. The kernel least-mean-square algorithm. **IEEE Transactions on Signal Processing**, Institute of Electrical and Electronics Engineers (IEEE), v. 56, n. 2, p. 543–554, fev. 2008. ISSN 1053-587X. Available at: http://dx.doi.org/10.1109/TSP.2007.907881.

LOSING, V.; HAMMER, B.; WERSING, H. Interactive online learning for obstacle classification on a mobile robot. In: **2015 International Joint Conference on Neural Networks (IJCNN'2015)**. IEEE, 2015. p. 1—-8. Available at: http://dx.doi.org/10.1109/IJCNN.2015.7280610.

LOSING, V.; HAMMER, B.; WERSING, H. KNN classifier with self adjusting memory for heterogeneous concept drift. In: **2016 IEEE 16th international conference on data mining (ICDM)**. IEEE, 2016. p. 291—-300. Available at: http://dx.doi.org/10.1109/ICDM.2016.0040.

LOSING, V.; HAMMER, B.; WERSING, H. Incremental on-line learning: A review and comparison of state of the art algorithms. **Neurocomputing**, Elsevier BV, v. 275, p. 1261–1274, jan. 2018. ISSN 0925-2312. Available at: http://dx.doi.org/10.1016/j.neucom.2017.06.084.

MACQUEEN, J. B. Some methods for classification and analysis of multivariate observations. In: **Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability**. University of California Press, 1967. v. 5, p. 281–297. Available at: https://digicoll.lib.berkeley.edu/record/113015/files/math_s5_v1_article-17.pdf.

MAIRAL, J.; BACH, F.; PONCE, J.; SAPIRO, G. Online learning for matrix factorization and sparse coding. **Journal of Machine Learning Research**, v. 11, n. 2, p. 19–60, 2010. Available at: http://jmlr.org/papers/v11/mairal10a.html.

MARTINETZ, T. M.; BERKOVICH, S. G.; SCHULTEN, K. J. Neural-gas network for vector quantization and its application to time-series prediction. **IEEE Transactions on Neural Networks**, Institute of Electrical and Electronics Engineers (IEEE), v. 4, n. 4, p. 558–569, jul. 1993. ISSN 1045-9227. Available at: http://dx.doi.org/10.1109/72.238311.

MATTOS, C. L. C.; BARRETO, G. A. ARTIE and MUSCLE models: building ensemble classifiers from fuzzy ART and SOM networks. **Neural Computing and Applications**, Springer Science and Business Media LLC, v. 22, n. 1, p. 49–61, out. 2011. ISSN 1433-3058. Available at: http://dx.doi.org/10.1007/s00521-011-0747-7.

MERMILLOD, M.; BUGAISKA, A.; BONIN, P. The stability-plasticity dilemma: investigating the continuum from catastrophic forgetting to age-limited learning effects. **Frontiers in Psychology**, Frontiers Media SA, v. 4, 2013. ISSN 1664-1078. Available at: http://dx.doi.org/10.3389/fpsyg.2013.00504.

MINKU, L. L.; YAO, X. DDD: A new ensemble approach for dealing with concept drift. **IEEE Transactions on Knowledge and Data Engineering**, Institute of Electrical and Electronics Engineers (IEEE), v. 24, n. 4, p. 619–633, abr. 2012. ISSN 1041-4347. Available at: http://dx.doi.org/10.1109/TKDE.2011.58.

MOORE, E. H. On the reciprocal of the general algebraic matrix. **Bull. Amer. Math. Soc.**, v. 26, p. 394–395, 1920.

MORENO-TORRES, J. G.; RAEDER, T.; ALAIZ-RODRÍGUEZ, R.; CHAWLA, N. V.; HERRERA, F. A unifying view on dataset shift in classification. **Pattern Recognition**, Elsevier BV, v. 45, n. 1, p. 521–530, jan. 2012. ISSN 0031-3203. Available at: http://dx.doi.org/10.1016/j.patcog.2011.06.019.

NADARAYA, E. On estimating regression. **Theory of Probability and Its Applications**, Society for Industrial and Applied Mathematics (SIAM), v. 9, n. 1, p. 141–142, jan. 1964. ISSN 1095-7219. Available at: http://dx.doi.org/10.1137/1109020.

NGU, J. C. Y.; YEO, C. A comparative study of different kernel functions applied to LW-KPLS model for nonlinear processes. **Biointerface Research in Applied Chemistry**, AMG Transcend Association, v. 13, n. 2, p. 184, abr. 2022. ISSN 2069-5837. Available at: http://dx.doi.org/10.33263/BRIAC132.184.

NGUYEN, H.-D.; TO, T.-P.; NGUYEN, D. Q.; HUYNH, T.-T.; TRAN, T.; BUI, C.-T.; QUAN, T. A novel approach for non-query fake news detection using K-SOM and graph neural networks. In: **2023 15th International Conference on Knowledge and Systems Engineering (KSE)**. IEEE, 2023. p. 1–6. Available at: http://dx.doi.org/10.1109/KSE59128.2023.10299416.

PENROSE, R. A generalized inverse for matrices. In: . Cambridge University Press (CUP), 1955. v. 51, n. 3, p. 406–413. ISSN 1469-8064. Available at: http://dx.doi.org/10.1017/S0305004100030401.

PLATT, J. **A Resource-Allocating Network for Function Interpolation**. MIT Press - Journals, 1991. v. 3. 213–225 p. ISSN 1530-888X. Available at: http://dx.doi.org/10.1162/neco.1991.3.2.213.

PONTE, J. U. **Revisitando a estimação do número de neurônios ocultos da rede MLP usando análise de componentes principais baseada em kernel**. Dissertação (Dissertação de Mestrado) – Universidade Federal do Ceara, Fortaleza, Brasil, 2020. Available at: http://www.repositorio.ufc.br/handle/riufc/52214.

QIN, A. K.; SUGANTHAN, P. N. Kernel neural gas algorithms with application to cluster analysis. In: **Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.** IEEE, 2004. v. 4, p. 617–620. Available at: http://dx.doi.org/10.1109/ICPR.2004.1333848.

QIN, A. K.; SUGANTHAN, P. N. A novel kernel prototype-based learning algorithm. In: **Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.** IEEE, 2004. v. 4, p. 621–624. Available at: http://dx.doi.org/10.1109/ICPR.2004.1333849.

RICHARD, C.; BERMUDEZ, J. C. M.; HONEINE, P. Online prediction of time series data with kernels. **IEEE Transactions on Signal Processing**, Institute of Electrical and Electronics Engineers (IEEE), v. 57, n. 3, p. 1058–1067, mar. 2009. ISSN 1941-0476. Available at: http://dx.doi.org/10.1109/TSP.2008.2009895.

RICHARDSON, F. M.; THOMAS, M. S. Critical periods and catastrophic interference effects in the development of self-organizing feature maps. **Developmental Science**, Wiley, v. 11, n. 3, p. 371–389, maio 2008. ISSN 1467-7687. Available at: http://dx.doi.org/10.1111/j.1467-7687.2008.00682.x.

ROCHA NETO, A. R. d. **SINPATCO II. Novas estratégias de aprendizado de máquina para classificação de patologias da coluna vertebral**. Tese (Doutorado) – Universidade Federal do Ceará–UFC, 2011.

ROSSIUS, R.; ZENKER, G.; ITTNER, A.; DILGER, W. A short note about the application of polynomial kernels with fractional degree in support vector learning. In: **Machine Learning: ECML-98: 10th European Conference on Machine Learning Chemnitz, Germany, April 21–23, 1998 Proceedings 10**. Springer Berlin Heidelberg, 1998. p. 143–148. ISBN 9783540697817. ISSN 1611-3349. Available at: http://dx.doi.org/10.1007/BFb0026684.

RUBIO, G.; HERRERA, L. J.; POMARES, H.; ROJAS, I.; GUILLÉN, A. Design of specific-to-problem kernels and use of kernel weighted K-nearest neighbours for time series modelling. **Neurocomputing**, Elsevier BV, v. 73, n. 10–12, p. 1965–1975, jun. 2010. ISSN 0925-2312. Available at: http://dx.doi.org/10.1016/j.neucom.2009.11.029.

RUSTAM, Z.; RIKA. Face recognition using fuzzy kernel learning vector quantization. In: . IOP Publishing, 2018. v. 1108, n. 1, p. 012068. ISSN 1742-6596. Available at: http://dx.doi.org/10.1088/1742-6596/1108/1/012068.

SATO, A.; YAMADA, K. Generalized learning vector quantization. MIT Press, v. 8, 1995. Available at: https://proceedings.neurips.cc/paper_files/paper/1995/file/9c3b1830513cc3b8fc4b76635d32e692-Paper.pdf.

SHERMAN, J.; MORRISON, W. J. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. **The Annals of Mathematical Statistics**, Institute of Mathematical Statistics, v. 21, n. 1, p. 124–127, mar. 1950. ISSN 0003-4851. Available at: http://dx.doi.org/10.1214/aoms/1177729893.

Soares Filho, L. A.; BARRETO, G. A. On the efficient design of a prototype-based classifier using differential evolution. In: **2014 IEEE Symposium on Differential Evolution (SDE)**. IEEE, 2014. p. 1–8. Available at: http://dx.doi.org/10.1109/SDE.2014.7031535.

SOUSA, D. P.; BARRETO, G. A.; CAVALCANTE, C. C.; MEDEIROS, C. LVQ-type classifiers for condition monitoring of induction motors: A performance comparison. In: **Advances in Self-Organizing Maps, Learning Vector Quantization, Clustering and Data Visualization**. Springer International Publishing, 2019. p. 130–139. ISBN 9783030196424. ISSN 2194-5365. Available at: http://dx.doi.org/10.1007/978-3-030-19642-4_13.

SPANGENBERG, N.; AUGENSTEIN, C.; FRANCZYK, B.; WAGNER, M.; APITZ, M.; KENNGOTT, H. Method for intra-surgical phase detection by using real-time medical device data. In: **2017 IEEE 30th International Symposium on Computer-Based Medical Systems (CBMS)**. IEEE, 2017. p. 254–259. Available at: http://dx.doi.org/10.1109/CBMS.2017.65.

SUYKENS, J.; VANDEWALLE, J. Least squares support vector machine classifiers. **Neural Processing Letters**, Springer Science and Business Media LLC, v. 9, n. 3, p. 293–300, jun. 1999. ISSN 1573-773X. Available at: http://dx.doi.org/10.1023/A:1018628609742.

SYRIOPOULOS, P. K.; KALAMPALIKIS, N. G.; KOTSIANTIS, S. B.; VRAHATIS, M. N. kNN classification: a review. **Annals of Mathematics and Artificial Intelligence**, Springer Science and Business Media LLC, v. 93, n. 1, p. 43–75, set. 2023. ISSN 1573-7470. Available at: http://dx.doi.org/10.1007/s10472-023-09882-x.

TSYMBAL, A. **The problem of concept drift: definitions and related work**. [*S. l.*], 2004. Available at: www.scss.tcd.ie/publications/tech-reports/.

VALYON, J. **Extended LSSVM for System Modeling**. Tese (Doutorado) – Budapest University of Technology and Economics, Department of Measurement and Information Systems, 2006. Available at: http://hdl.handle.net/10890/648.

Van Vaerenbergh, S.; SANTAMARÍA, I. Online regression with kernels. **Regularization, Optimization, Kernels, and Support Vector Machines**, Chapman & Hall, p. 477–501, 2014. Available at: https://gtas.unican.es/files/pub/ch21_online_regression_with_kernels.pdf.

WADEWALE, K.; DESAI, S. Survey on method of drift detection and classification for time varying data set. **International Research Journal of Engineering and Technology IRJET**, v. 2, n. 9, p. 709–713, 2015. Available at: https://www.irjet.net/archives/V2/i9/IRJET-V2I9117.pdf.

WATSON, G. S. Smooth regression analysis. **Sankhyā: The Indian Journal of Statistics, Series A**, JSTOR, v. 26, n. 4, p. 359–372, dez. 1964. Available at: https://www.jstor.org/stable/25049340.

WEBB, G. I.; HYDE, R.; CAO, H.; NGUYEN, H. L.; PETITJEAN, F. Characterizing concept drift. **Data Mining and Knowledge Discovery**, Springer Science and Business Media LLC, v. 30, n. 4, p. 964–994, abr. 2016. ISSN 1573-756X. Available at: http://dx.doi.org/10.1007/s10618-015-0448-4.

YANG, L.; SHAMI, A. On hyperparameter optimization of machine learning algorithms: Theory and practice. **Neurocomputing**, Elsevier BV, v. 415, p. 295–316, nov. 2020. ISSN 0925-2312. Available at: http://dx.doi.org/10.1016/j.neucom.2020.07.061.

YIN, H. On the equivalence between kernel self-organising maps and self-organising mixture density networks. **Neural Networks**, Elsevier BV, v. 19, n. 6–7, p. 780—-784, jul. 2006. ISSN 0893-6080. Available at: http://dx.doi.org/10.1016/j.neunet.2006.05.007.

ZHANG, G. P. Neural networks for classification: a survey. **IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)**, IEEE, v. 30, n. 4, p. 451–462, 2000. ISSN 1094-6977. Available at: http://dx.doi.org/10.1109/5326.897072.

ZHANG, R.; RUDNICKY, A. A large scale clustering scheme for kernel K-means. In: **Object recognition supported by user interaction for service robots**. IEEE Comput. Soc, 2002. (ICPR-02, v. 4), p. 289–292. Available at: http://dx.doi.org/10.1109/ICPR.2002.1047453.

ZHANG, S. Challenges in KNN classification. **IEEE Transactions on Knowledge and Data Engineering**, Institute of Electrical and Electronics Engineers (IEEE), v. 34, n. 10, p. 4663–4675, out. 2022. ISSN 2326-3865. Available at: http://dx.doi.org/10.1109/TKDE.2021.3049250.

ŽLIOBAITė, I.; PECHENIZKIY, M.; GAMA, J. An overview of concept drift applications. In: ____. **Big Data Analysis: New Algorithms for a New Society**. Springer International Publishing, 2015. p. 91–114. ISBN 9783319269894. Available at: http://dx.doi.org/10.1007/978-3-319-26989-4_4.

APPENDIX A – DISTANCE MEASURES

At this appendix, some distances that can be used as dis(similarity) measures for prototype-based models are described.

A.1 Dot Product

It is a measure of similarity.

$$\left\langle \mathbf{x}_i, \mathbf{x}_j \right\rangle = \mathbf{x}_i^T \mathbf{x}_j$$

A.2 Minkowski

Generalized measure of dissimilarity

$$\left\| \mathbf{x}_i - \mathbf{x}_j \right\|_M = \left(\sum_{p=1}^{P} \left| x_{ip} - x_{jp} \right|^M \right)^{\frac{1}{M}}$$

A.3 Manhattan

Also knonw as *city-block* measure.

$$\left\| \mathbf{x}_i - \mathbf{x}_j \right\|_1 = \sum_{p=1}^{P} \left| x_{ip} - x_{jp} \right|$$

A.4 Euclidean

Most commonly used distance measure between two vectors.

$$\left\| \mathbf{x}_i - \mathbf{x}_j \right\|_2 = \left(\sum_{p=1}^{P} \left| x_{ip} - x_{jp} \right|^2 \right)^{\frac{1}{2}} = \sqrt[2]{\left(\mathbf{x}_i - \mathbf{x}_j \right)^T \left(\mathbf{x}_i - \mathbf{x}_j \right)}$$

A.5 Chebyshev

Considers the maximum value between the absolute distance of two vectors' attributes.

$$\left\| \mathbf{x}_i - \mathbf{x}_j \right\|_\infty = \lim_{M \to \infty} \left(\sum_{p=1}^{P} \left| x_{ip} - x_{jp} \right|^M \right)^{\frac{1}{M}} = \max_{\forall p} \left| x_{ip} - x_{jp} \right|$$

A.6 Mahalanobis

This distance is based on the attributes correlation, where **S** is the covariance matrix.

$$\left\|\mathbf{x}_i - \mathbf{x}_j\right\|_{\mathbf{S}} = \sqrt[2]{\left(\mathbf{x}_i - \mathbf{x}_j\right)^T \mathbf{S}^{-1} \left(\mathbf{x}_i - \mathbf{x}_j\right)}$$

A.7 Quadratic

It is a generalization of the Mahalanobis distance, where **A** is a symmetric and positive-definite matrix.

$$\left\|\mathbf{x}_i - \mathbf{x}_j\right\|_{\mathbf{A}} = \sqrt[2]{\left(\mathbf{x}_i - \mathbf{x}_j\right)^T \mathbf{A} \left(\mathbf{x}_i - \mathbf{x}_j\right)}.$$

If $\mathbf{A} = \mathbf{S}^{-1}$, this measure is equal to the Mahalanobis distance. On the other hand, if $\mathbf{A} = \mathbf{I}$, this measure is equal to the Euclidean distance.

A.8 Cosine

Like the dot product, this is another measure of similarity, however, this distance does not depend on the vectors magnitudes.

$$\frac{\left\langle \mathbf{x}_i, \mathbf{x}_j \right\rangle}{\left\|\mathbf{x}_i\right\|_2 \cdot \left\|\mathbf{x}_j\right\|_2} = \frac{\mathbf{x}_i^T \mathbf{x}_j}{\left\|\mathbf{x}_i\right\|_2 \cdot \left\|\mathbf{x}_j\right\|_2}$$

APPENDIX B – KERNEL FUNCTIONS

At this appendix, the equations of the kernel distances and its gradients of the kernel functions used at this thesis are developed.

B.1 Functions

B.1.1 Linear

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle + \theta = \mathbf{x}_i^T \mathbf{x}_j + \theta \tag{B.1}$$

B.1.2 Gaussian

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = exp\left(-\frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2}{2\gamma^2}\right) \tag{B.2}$$

B.1.3 Polynomial

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \left(\alpha . \langle \mathbf{x}_i, \mathbf{x}_j \rangle + \theta\right)^\gamma = \left(\alpha \mathbf{x}_i^T \mathbf{x}_j + \theta\right)^\gamma \tag{B.3}$$

B.1.4 Laplacian or Exponential

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = exp\left(-\frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|}{\gamma}\right) \tag{B.4}$$

B.1.5 Cauchy

$$\kappa\left(\mathbf{x}_i, \mathbf{x}_j\right) = \left(1 + \frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2}{\gamma^2}\right)^{-1} \tag{B.5}$$

B.1.6 Log

$$\kappa\left(\mathbf{x}_i, \mathbf{x}_j\right) = -log\left(1 + \frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2}{\gamma^2}\right) \tag{B.6}$$

B.1.7 Sigmoid

$$\kappa\left(\mathbf{x}_i,\mathbf{x}_j\right) = tgh\left(\alpha\left\langle\mathbf{x}_i,\mathbf{x}_j\right\rangle+\theta\right) = tgh\left(\alpha\mathbf{x}_i^T\mathbf{x}_j+\theta\right) \tag{B.7}$$

B.1.8 Kmod (moderated decreasing)

$$\kappa\left(\mathbf{x}_i,\mathbf{x}_j\right) = \frac{1}{exp\left(\theta/\gamma^2\right)-1}\left[exp\left(\frac{\theta}{\left\|\mathbf{x}_i-\mathbf{x}_j\right\|^2+\gamma^2}\right)-1\right] \tag{B.8}$$

B.2 Kernel Distances and its Gradients

B.2.1 Linear

$$J_i(\mathbf{x}) = (\mathbf{x}^T\mathbf{x}+\theta)-2(\mathbf{w}_i^T\mathbf{x}+\theta)+(\mathbf{w}_i^T\mathbf{w}_i+\theta)$$

$$J_i(\mathbf{x}) = \mathbf{x}^T\mathbf{x}-2\mathbf{w}_i^T\mathbf{x}+\mathbf{w}_i^T\mathbf{w}_i \tag{B.9}$$

$$\nabla J_i(\mathbf{x}) = 2(\mathbf{w}_i-\mathbf{x}) \tag{B.10}$$

B.2.2 Gaussian

$$J_i\left(\mathbf{x}\right) = exp\left(-\frac{\|\mathbf{x}-\mathbf{x}\|_2^2}{2\gamma^2}\right)-2.exp\left(-\frac{\|\mathbf{w}_i-\mathbf{x}\|_2^2}{2\gamma^2}\right)+exp\left(-\frac{\|\mathbf{w}_i-\mathbf{w}_i\|_2^2}{2\gamma^2}\right)$$

$$J_i\left(\mathbf{x}\right) = 2-2.exp\left(-\frac{\|\mathbf{w}_i-\mathbf{x}\|_2^2}{2\gamma^2}\right) \tag{B.11}$$

$$\nabla J_i(\mathbf{x}) = -2.exp\left(-\frac{\|\mathbf{w}_i-\mathbf{x}\|_2^2}{2\gamma^2}\right)\left(-\frac{1}{2\gamma^2}\right)2(\mathbf{w}_i-\mathbf{x})$$

$$\nabla J_i(\mathbf{x}) = \left(\frac{2}{\gamma^2}\right)exp\left(-\frac{\|\mathbf{w}_i-\mathbf{x}\|_2^2}{2\gamma^2}\right)(\mathbf{w}_i-\mathbf{x}) \tag{B.12}$$

B.2.3 Polynomial

$$J_i(\mathbf{x}) = (\alpha \mathbf{x}^T \mathbf{x} + \theta)^\gamma + (\alpha \mathbf{w}_i^T \mathbf{w}_i + \theta)^\gamma - 2(\alpha \mathbf{w}_i^T \mathbf{x} + \theta)^\gamma \tag{B.13}$$
$$\nabla J_i(\mathbf{x}) = \gamma(\alpha \mathbf{w}_i^T \mathbf{w}_i + \theta)^{\gamma-1} 2\alpha \mathbf{w}_i - 2\gamma(\alpha \mathbf{w}_i^T \mathbf{x} + \theta)^{\gamma-1} \alpha \mathbf{x}$$

$$\nabla J_i(\mathbf{x}) = 2\alpha\gamma \left[\mathbf{w}_i (\alpha \mathbf{w}_i^T \mathbf{w}_i + \theta)^{\gamma-1} - \mathbf{x}(\alpha \mathbf{w}_i^T \mathbf{x} + \theta)^{\gamma-1} \right] \tag{B.14}$$

B.2.4 Laplacian or Exponential

$$J_i(\mathbf{x}) = exp\left(-\frac{\|\mathbf{x}-\mathbf{x}\|}{\gamma}\right) - 2exp\left(-\frac{\|\mathbf{w}_i-\mathbf{x}\|}{\gamma}\right) + exp\left(-\frac{\|\mathbf{w}_i-\mathbf{w}_i\|}{\gamma}\right)$$

$$J_i(\mathbf{x}) = 2 - 2exp\left(-\frac{\|\mathbf{w}_i-\mathbf{x}\|}{\gamma}\right) \tag{B.15}$$
$$\nabla J_i(\mathbf{x}) = -2exp\left(-\frac{\|\mathbf{w}_i-\mathbf{x}\|}{\gamma}\right)\left(-\frac{1}{\gamma}\right)\left(\frac{1}{2}\frac{1}{\|\mathbf{w}_i-\mathbf{x}\|}\right) 2(\mathbf{w}_i-\mathbf{x})$$

$$\nabla J_i(\mathbf{x}) = \left(\frac{2}{\gamma\|\mathbf{w}_i-\mathbf{x}\|}\right) exp\left(-\frac{\|\mathbf{w}_i-\mathbf{x}\|}{\gamma}\right)(\mathbf{w}_i-\mathbf{x}) \tag{B.16}$$

B.2.5 Cauchy

$$J_i(\mathbf{x}) = \left(1+\frac{\|\mathbf{x}-\mathbf{x}\|^2}{\gamma^2}\right)^{-1} - 2\left(1+\frac{\|\mathbf{w}_i-\mathbf{x}\|^2}{\gamma^2}\right)^{-1} + \left(1+\frac{\|\mathbf{w}_i-\mathbf{w}_i\|^2}{\gamma^2}\right)^{-1}$$

$$J_i(\mathbf{x}) = 2 - 2\left(1+\frac{\|\mathbf{w}_i-\mathbf{x}\|^2}{\gamma^2}\right)^{-1} = 2 - 2\left(\frac{\gamma^2}{\gamma^2+\|\mathbf{w}_i-\mathbf{x}\|^2}\right) \tag{B.17}$$
$$\nabla J_i(\mathbf{x}) = -2\gamma^2\left[-\frac{1}{\left(\gamma^2+\|\mathbf{w}_i-\mathbf{x}\|^2\right)^2}\right] 2(\mathbf{w}_i-\mathbf{x})$$

$$\nabla J_i(\mathbf{x}) = \left[\frac{4\gamma^2}{\left(\gamma^2+\|\mathbf{w}_i-\mathbf{x}\|^2\right)^2}\right](\mathbf{w}_i-\mathbf{x}) \tag{B.18}$$

B.2.6 Log

$$J_i(\mathbf{x}) = -log\left(1+\frac{\|\mathbf{x}-\mathbf{x}\|^2}{\gamma^2}\right)+2log\left(1+\frac{\|\mathbf{w}_i-\mathbf{x}\|^2}{\gamma^2}\right)-log\left(1+\frac{\|\mathbf{w}_i-\mathbf{w}_i\|^2}{\gamma^2}\right)$$

$$J_i(\mathbf{x}) = 2log\left(1+\frac{\|\mathbf{w}_i-\mathbf{x}\|^2}{\gamma^2}\right) = 2log\left(\frac{\gamma^2+\|\mathbf{w}_i-\mathbf{x}\|^2}{\gamma^2}\right) \tag{B.19}$$

$$\nabla J_i(\mathbf{x}) = 2\frac{\gamma^2}{\gamma^2+\|\mathbf{x}-\mathbf{w}_i\|^2}\frac{2(\mathbf{w}_i-\mathbf{x})}{\gamma^2} = \frac{4}{\gamma^2+\|\mathbf{x}-\mathbf{w}_i\|^2}(\mathbf{w}_i-\mathbf{x}) \tag{B.20}$$

B.2.7 Sigmoid

$$J_i(\mathbf{x}) = tgh\left(\alpha\mathbf{x}^T\mathbf{x}+\theta\right)+tgh\left(\alpha\mathbf{w}_i^T\mathbf{w}_i+\theta\right)-2tgh\left(\alpha\mathbf{w}_i^T\mathbf{x}+\theta\right) \tag{B.21}$$

Obs:

$$\frac{\partial\, tgh(f(\mathbf{w}))}{\partial\mathbf{w}} = \left[1-tgh^2\left(f(\mathbf{w})\right)\right]f^{'}(w)$$

$$\nabla J_i(\mathbf{x}) = \left[1-tgh^2(\alpha\mathbf{w}_i^T\mathbf{w}_i+\theta)\right](2\alpha\mathbf{w}_i)-2\left[1-tgh^2(\alpha\mathbf{w}_i^T\mathbf{x}+\theta)\right](\alpha\mathbf{x})$$

$$\nabla J_i(\mathbf{x}) = 2\alpha\left[(\mathbf{w}_i-\mathbf{x})-\left(tgh^2(\alpha\mathbf{w}_i^T\mathbf{w}_i+\theta)\mathbf{w}_i-tgh^2(\alpha\mathbf{w}_i^T\mathbf{x}+\theta)\mathbf{x}\right)\right] \tag{B.22}$$

B.2.8 Kmod (moderated decreasing)

$$\begin{aligned} J_i(\mathbf{x}) \;=\; & \frac{1}{exp\left(\theta/\gamma^2\right)-1}\left[exp\left(\frac{\theta}{\|\mathbf{x}-\mathbf{x}\|^2+\gamma^2}\right)-1\right]+ \\ & \frac{1}{exp\left(\theta/\gamma^2\right)-1}\left[exp\left(\frac{\theta}{\|\mathbf{w}_i-\mathbf{w}_i\|^2+\gamma^2}\right)-1\right]- \\ & 2\frac{1}{exp\left(\theta/\gamma^2\right)-1}\left[exp\left(\frac{\theta}{\|\mathbf{w}_i-\mathbf{x}\|^2+\gamma^2}\right)-1\right] \end{aligned}$$

$$J_i(\mathbf{x}) = 2-\frac{2}{exp\left(\theta/\gamma^2\right)-1}\left[exp\left(\frac{\theta}{\|\mathbf{w}_i-\mathbf{x}\|^2+\gamma^2}\right)-1\right] \tag{B.23}$$

$$\nabla J_i(\mathbf{x}) = \frac{-2}{exp\left(\theta/\gamma^2\right)-1}exp\left(\frac{\theta}{\|\mathbf{w}_i-\mathbf{x}\|^2+\gamma^2}\right)\frac{-\theta}{\left(\|\mathbf{w}_i-\mathbf{x}\|^2+\gamma^2\right)^2}2\left(\mathbf{w}_i-\mathbf{x}\right)$$

$$\nabla J_i(\mathbf{x}) = \frac{4\theta\; exp\left(\frac{\theta}{\|\mathbf{w}_i-\mathbf{x}\|^2+\gamma^2}\right)}{\left(exp\left(\theta/\gamma^2\right)-1\right)\left(\|\mathbf{w}_i-\mathbf{x}\|^2+\gamma^2\right)^2}\left(\mathbf{w}_i-\mathbf{x}\right) \tag{B.24}$$

APPENDIX C – OPTIMUM HYPERPARAMETERS

At this appendix, the best combination of hyperparameters, for each model configuration is disposed in tables.

Table 52 – Best Hyperparameters of KSOM applied to the Motor Failure Dataset

		linear	gaus	poly			exp
Algorithm	Labeling	theta	sigma	alpha	theta	gamma	sigma
KSOM-EF	Majority Voting	0,0010	512	0,2	8	128	2,6
	Average Distance	0,125	256	0,8	128	16	0,6
	Minimum Distance	0,0039	32	0,2	512	512	0,4
KSOM-GD	Majority Voting	0,0010	512	0,2	8	4	0,6
	Average Distance	0,5	16	0,2	4	128	0,4
	Minimum Distance	0,0156	4	2,4	32	512	1

		cauchy	log		sigm		kmod	
Algorithm	Labeling	sigma	gamma	sigma	alpha	theta	gamma	sigma
KSOM-EF	Majority Voting	128	0,0313	-4	0,0313	128	0,03125	128
	Average Distance	0,5	0,0625	-1	1	8	1	8
	Minimum Distance	256	0,0039	-4	0,25	0,5	0,25	0,5
KSOM-GD	Majority Voting	128	0,0313	-1024	256	4	256	4
	Average Distance	1	0,5	256	8	8	8	8
	Minimum Distance	512	0,0625	-1024	1024	1024	1024	1024

Table 53 – Best Hyperparameters of KSOM applied to the Cervical Cancer Dataset

		linear	gauss	poly			expo
Algorithm	Labeling	theta	sigma	alpha	theta	gamma	sigma
KSOM-EF	Majority Voting	512	64	0,0156	0,5	1	256
	Average Distance	1	8	0,5	0,5	1	512
	Minimum Distance	2	64	0,0625	32	0,8	8
KSOM-GD	Majority Voting	0	64	0,0078	2	2	512
	Average Distance	1024	32	0,25	16	0,6	16
	Minimum Distance	0,0039	64	0,0039	32	0,8	64

		cauchy	log		sigm		kmod	
Algorithm	Labeling	sigma	gamma	sigma	alpha	theta	gamma	sigma
KSOM-EF	Majority Voting	4	2,2	0,0010	0,0039	0,0156	0,25	256
	Average Distance	64	2,4	64	0,0156	0,5	256	32
	Minimum Distance	8	0,8	0,0010	0,0039	0,0156	0,5	4
KSOM-GD	Majority Voting	64	0,8	1	0,0039	0,0313	0,25	64
	Average Distance	4	0,4	8	0,0313	-0,0020	16	8
	Minimum Distance	64	2,4	0,0039	0,0039	0,0020	4	8

Table 54 – Best Hyperparameters of KSOM applied to the Vertebral Column Dataset

		linear	gauss	poly			expo
Algorithm	Labeling	theta	sigma	alpha	theta	gamma	sigma
KSOM-EF	Majority Voting	128	4	0,125	256	0,2	32
	Average Distance	128	1	8	256	0,6	0,5
	Minimum Distance	64	1	0,0039	1	2	8
KSOM-GD	Majority Voting	0,0078	1	0,0078	512	1	1
	Average Distance	16	1	4	128	0,6	1
	Minimum Distance	512	4	0,125	32	0,8	16

		cauchy	log		sigm		kmod	
Algorithm	Labeling	sigma	gamma	sigma	alpha	theta	gamma	sigma
KSOM-EF	Majority Voting	1	2,6	0,5	0,0625	0,0039	4	2
	Average Distance	0,5	0,6	0,25	0,125	1	8	2
	Minimum Distance	2	2	0,5	0,125	-0,125	512	8
KSOM-GD	Majority Voting	0,5	0,2	2	0,0625	-0,0156	1	2
	Average Distance	4	1	0,5	0,0039	-0,1250	0,0020	2
	Minimum Distance	8	0,6	4	0,0039	-0,0020	0,0156	4

Table 55 – Best Hyperparameters of KSOM applied to the Wall Following Dataset

		linear	gauss	poly			expo
Algorithm	Labeling	theta	sigma	alpha	theta	gamma	sigma
KSOM-EF	Majority Voting	0,0156	32	0,0020	256	1	1024
	Average Distance	512	0,5	4	64	0,4	32
	Minimum Distance	0,0313	16	0,5	512	0,4	8
KSOM-GD	Majority Voting	0,0313	32	0,0078	64	0,8	1
	Average Distance	1024	0,5	0,125	32	1	256
	Minimum Distance	0,0020	8	0,0020	1	1	1024

		cauchy	log		sigm		expo	
Algorithm	Labeling	sigma	gamma	sigma	alpha	theta	gamma	sigma
KSOM-EF	Majority Voting	32	2,2	32	0,0020	0,0078	256	32
	Average Distance	1	0,8	32	0,0078	0,0039	1	1
	Minimum Distance	16	2,6	16	0,0010	-0,0313	2	1
KSOM-GD	Majority Voting	32	2,6	32	0,0020	0,0010	16	32
	Average Distance	16	0,8	2	0,25	1	0,0039	0,5
	Minimum Distance	32	2	32	0,0010	0,5	0,0020	16

APPENDIX D – SPOK RESULTS FOR DIFFERENT SPARSIFICATION PROCEDURES

Table 56 – Evaluation of the SPOK model and the Moving Squares Dataset

Sm	Dm	NN	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	2	1	0.954± 0.003	0.949± 0.002	0.371± 0.061	0.949± 0.002	0.955± 0.002	0.576± 0.082	0.247± 0.035	0.952± 0.003
		k	0.970± 0.009	0.659± 0.010	0.327± 0.031	0.723± 0.003	0.445± 0.009	0.833± 0.148	0.249± 0.006	0.683± 0.005
COH	2	1	0.981± 0.002	0.952± 0.002	0.982± 0.002	0.913± 0.004	0.949± 0.003	0.413± 0.039	0.664± 0.032	0.951± 0.002
		k	0.995± 0.001	0.625± 0.006	0.984± 0.002	0.686± 0.007	0.955± 0.012	0.354± 0.010	0.946± 0.008	0.959± 0.010
NOV	2	1	0.997± 0.000	0.334± 0.076	0.968± 0.004	0.369± 0.109	0.997± 0.000	0.997± 0.000	0.250± 0.002	0.368± 0.105
		k	0.638± 0.041	1.000± 0.000	0.422± 0.114	0.371± 0.107	0.350± 0.113	0.338± 0.111	0.380± 0.016	1.000± 0.000
SUR	2	1	0.953± 0.005	0.967± 0.003	0.686± 0.106	0.688± 0.030	0.949± 0.004	0.914± 0.003	0.965± 0.003	0.967± 0.003
		k	0.675± 0.016	0.871± 0.004	0.710± 0.133	0.381± 0.050	0.640± 0.064	0.982± 0.024	0.477± 0.035	0.364± 0.078

Table 57 – Evaluation of the SPOK model and the RBF Interchanging Dataset

Sm	Dm	NN	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	2	1	0.552± 0.131	0.733± 0.084	0.509± 0.125	0.893± 0.033	0.654± 0.104	0.511± 0.121	0.090± 0.007	0.945± 0.016
		k	0.545± 0.138	0.619± 0.111	0.478± 0.126	0.553± 0.104	0.527± 0.135	0.798± 0.060	0.670± 0.058	0.459± 0.149
COH	2	1	0.983± 0.006	0.747± 0.080	0.973± 0.009	0.572± 0.115	0.805± 0.060	0.509± 0.126	0.844± 0.049	0.962± 0.011
		k	0.928± 0.017	0.819± 0.043	0.930± 0.016	0.640± 0.106	0.519± 0.137	0.444± 0.131	0.902± 0.030	0.655± 0.100
NOV	2	1	0.954± 0.013	0.944± 0.004	0.867± 0.023	0.809± 0.051	0.954± 0.013	0.995± 0.001	0.122± 0.008	0.952± 0.004
		k	0.954± 0.013	0.685± 0.096	0.675± 0.100	0.692± 0.092	0.954± 0.013	0.954± 0.013	0.410± 0.012	0.657± 0.101
SUR	2	1	0.892± 0.023	0.414± 0.160	0.443± 0.151	0.746± 0.061	0.947± 0.015	0.526± 0.126	0.484± 0.156	0.599± 0.098
		k	0.538± 0.127	0.634± 0.106	0.817± 0.048	0.688± 0.087	0.756± 0.067	0.827± 0.049	0.815± 0.040	0.762± 0.064

Table 58 – Evaluation of the SPOK model and the Chessboard Dataset

Sm	Dm	NN	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	2	1	0.733± 0.054	0.859± 0.023	0.896± 0.036	0.926± 0.029	0.799± 0.020	0.516± 0.032	0.713± 0.061	0.917± 0.028
		k	0.880± 0.046	0.951± 0.031	0.874± 0.051	0.935± 0.031	0.882± 0.026	0.499± 0.032	0.131± 0.013	0.949± 0.035
COH	2	1	0.886± 0.026	0.932± 0.030	0.589± 0.029	0.919± 0.028	0.915± 0.029	0.888± 0.027	0.668± 0.022	0.915± 0.029
		k	0.913± 0.030	0.960± 0.033	0.880± 0.027	0.942± 0.035	0.950± 0.032	0.919± 0.037	0.631± 0.024	0.956± 0.032
NOV	2	1	0.553± 0.036	0.512± 0.076	0.894± 0.025	0.886± 0.036	0.922± 0.027	0.553± 0.036	0.130± 0.009	0.890± 0.037
		k	0.553± 0.036	0.946± 0.041	0.927± 0.026	0.940± 0.048	0.553± 0.036	0.553± 0.036	0.122± 0.010	0.082± 0.006
SUR	2	1	0.698± 0.055	0.552± 0.036	0.869± 0.053	0.778± 0.022	0.553± 0.035	0.619± 0.042	0.636± 0.037	0.551± 0.035
		k	0.575± 0.030	0.934± 0.031	0.851± 0.083	0.901± 0.111	0.937± 0.030	0.942± 0.033	0.651± 0.047	0.943± 0.032

Table 59 – Evaluation of the SPOK model and the Cover Type Dataset

Sm	Dm	NN	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	2	1	0.881± 0.039	0.890± 0.037	0.878± 0.040	0.887± 0.038	0.888± 0.039	0.866± 0.037	0.884± 0.038	0.882± 0.039
		k	0.881± 0.039	0.886± 0.039	0.880± 0.039	0.872± 0.037	0.877± 0.037	0.839± 0.069	0.881± 0.045	0.879± 0.036
COH	2	1	0.887± 0.037	0.872± 0.045	0.890± 0.035	0.890± 0.035	0.888± 0.037	0.872± 0.036	0.894± 0.034	0.889± 0.035
		k	0.892± 0.035	0.877± 0.035	0.856± 0.039	0.875± 0.038	0.883± 0.038	0.866± 0.037	0.817± 0.030	0.883± 0.038
NOV	2	1	0.885± 0.032	0.887± 0.031	0.886± 0.032	0.885± 0.032	0.885± 0.032	0.885± 0.032	0.889± 0.032	0.885± 0.032
		k	0.863± 0.032	0.855± 0.028	0.865± 0.032	0.869± 0.032	0.881± 0.033	0.872± 0.031	0.837± 0.033	0.855± 0.029
SUR	2	1	0.870± 0.038	0.886± 0.041	0.876± 0.040	0.884± 0.040	0.887± 0.038	0.880± 0.040	0.833± 0.039	0.884± 0.041
		k	0.864± 0.037	0.876± 0.035	0.824± 0.031	0.875± 0.036	0.868± 0.039	0.866± 0.037	0.870± 0.037	0.879± 0.038

Table 60 – Mean Accuracy of SPOK and Electricity Dataset

Sm	Dm	NN	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	2	1	0.744± 0.016	0.727± 0.012	0.744± 0.013	0.740± 0.013	0.742± 0.013	0.741± 0.012	0.715± 0.021	0.734± 0.012
		k	0.745± 0.013	0.696± 0.034	0.726± 0.017	0.728± 0.014	0.725± 0.014	0.512± 0.058	0.674± 0.013	0.737± 0.012
COH	2	1	0.726± 0.011	0.740± 0.015	0.731± 0.013	0.740± 0.015	0.742± 0.014	0.740± 0.012	0.708± 0.013	0.742± 0.014
		k	0.730± 0.011	0.618± 0.056	0.729± 0.008	0.747± 0.017	0.743± 0.015	0.725± 0.022	0.715± 0.013	0.739± 0.009
NOV	2	1	0.745± 0.014	0.700± 0.029	0.744± 0.013	0.737± 0.017	0.745± 0.014	0.745± 0.014	0.725± 0.019	0.747± 0.015
		k	0.722± 0.022	0.747± 0.018	0.720± 0.024	0.740± 0.024	0.734± 0.025	0.750± 0.024	0.678± 0.014	0.753± 0.020
SUR	2	1	0.747± 0.016	0.721± 0.009	0.746± 0.021	0.744± 0.018	0.744± 0.013	0.739± 0.014	0.729± 0.026	0.738± 0.014
		k	0.715± 0.023	0.696± 0.031	0.699± 0.021	0.742± 0.020	0.749± 0.020	0.726± 0.021	0.607± 0.031	0.747± 0.016

Table 61 – Evaluation of the SPOK model and the Outdoor Dataset

Sm	Dm	NN	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	2	1	0.788± 0.030	0.749± 0.062	0.759± 0.053	0.753± 0.062	0.738± 0.063	0.703± 0.044	0.767± 0.030	0.833± 0.027
		k	0.787± 0.030	0.760± 0.030	0.778± 0.031	0.819± 0.029	0.732± 0.067	0.688± 0.053	0.756± 0.037	0.763± 0.059
COH	2	1	0.831± 0.031	0.770± 0.056	0.866± 0.023	0.739± 0.063	0.767± 0.058	0.720± 0.067	0.839± 0.028	0.841± 0.028
		k	0.792± 0.027	0.841± 0.025	0.811± 0.029	0.773± 0.055	0.787± 0.050	0.568± 0.069	0.753± 0.035	0.760± 0.062
NOV	2	1	0.872± 0.021	0.878± 0.021	0.876± 0.022	0.878± 0.021	0.879± 0.020	0.879± 0.020	0.848± 0.024	0.877± 0.021
		k	0.872± 0.021	0.882± 0.022	0.874± 0.021	0.878± 0.021	0.886± 0.022	0.882± 0.020	0.833± 0.026	0.881± 0.021
SUR	2	1	0.745± 0.044	0.782± 0.046	0.742± 0.066	0.684± 0.082	0.759± 0.038	0.757± 0.032	0.737± 0.037	0.743± 0.063
		k	0.792± 0.038	0.866± 0.022	0.713± 0.066	0.823± 0.032	0.889± 0.022	0.767± 0.051	0.765± 0.029	0.861± 0.023

Table 62 – Evaluation of the SPOK model and the Poker Hand Dataset

Sm	Dm	NN	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	2	1	0.715± 0.018	0.698± 0.024	0.711± 0.017	0.738± 0.015	0.730± 0.016	0.699± 0.018	0.737± 0.010	0.725± 0.014
		k	0.759± 0.012	0.742± 0.020	0.767± 0.010	0.780± 0.015	0.784± 0.013	0.742± 0.013	0.641± 0.022	0.774± 0.012
COH	2	1	0.718± 0.019	0.722± 0.016	0.735± 0.017	0.715± 0.019	0.717± 0.019	0.713± 0.017	0.330± 0.061	0.717± 0.020
		k	0.773± 0.012	0.774± 0.015	0.791± 0.013	0.771± 0.014	0.777± 0.011	0.776± 0.012	0.558± 0.019	0.778± 0.012
NOV	2	1	0.743± 0.016	0.749± 0.015	0.742± 0.016	0.749± 0.014	0.742± 0.016	0.742± 0.016	0.738± 0.017	0.746± 0.015
		k	0.780± 0.017	0.780± 0.018	0.783± 0.018	0.786± 0.017	0.772± 0.020	0.789± 0.018	0.728± 0.007	0.789± 0.018
SUR	2	1	0.712± 0.020	0.719± 0.019	0.729± 0.018	0.721± 0.015	0.723± 0.017	0.718± 0.018	0.708± 0.018	0.684± 0.015
		k	0.755± 0.033	0.728± 0.025	0.678± 0.013	0.771± 0.014	0.767± 0.013	0.684± 0.020	0.763± 0.009	0.780± 0.016

Table 63 – Evaluation of the SPOK model and the Rialto Dataset

Sm	Dm	NN	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	2	1	0.622± 0.029	0.615± 0.041	0.627± 0.022	0.624± 0.031	0.624± 0.032	0.573± 0.033	0.629± 0.019	0.626± 0.032
		k	0.640± 0.021	0.575± 0.042	0.613± 0.029	0.611± 0.035	0.610± 0.034	0.575± 0.031	0.149± 0.012	0.614± 0.034
COH	2	1	0.642± 0.033	0.628± 0.031	0.625± 0.031	0.623± 0.032	0.623± 0.032	0.618± 0.032	0.646± 0.032	0.618± 0.033
		k	0.628± 0.032	0.593± 0.040	0.613± 0.033	0.584± 0.042	0.606± 0.033	0.392± 0.029	0.631± 0.017	0.568± 0.046
NOV	2	1	0.651± 0.028	0.654± 0.032	0.654± 0.032	0.655± 0.032	0.648± 0.033	0.652± 0.032	0.654± 0.025	0.654± 0.032
		k	0.661± 0.028	0.478± 0.074	0.658± 0.035	0.658± 0.036	0.650± 0.038	0.655± 0.036	0.631± 0.022	0.657± 0.036
SUR	2	1	0.488± 0.010	0.602± 0.044	0.608± 0.037	0.619± 0.033	0.619± 0.033	0.630± 0.030	0.571± 0.023	0.625± 0.027
		k	0.617± 0.023	0.652± 0.035	0.556± 0.036	0.639± 0.042	0.640± 0.042	0.648± 0.040	0.645± 0.021	0.654± 0.033

Table 64 – Evaluation of the SPOK model and the Weather Dataset

Sm	Dm	NN	linear	gauss	poly	expo	cauchy	log	sigm	kmod
ALD	2	1	0.751± 0.018	0.716± 0.012	0.751± 0.012	0.725± 0.014	0.718± 0.011	0.696± 0.015	0.687± 0.015	0.728± 0.016
		k	0.778± 0.017	0.775± 0.017	0.775± 0.018	0.777± 0.009	0.777± 0.011	0.688± 0.015	0.687± 0.015	0.779± 0.012
COH	2	1	0.700± 0.015	0.735± 0.012	0.722± 0.011	0.746± 0.009	0.739± 0.012	0.750± 0.012	0.425± 0.023	0.734± 0.012
		k	0.679± 0.021	0.777± 0.011	0.739± 0.012	0.779± 0.014	0.779± 0.013	0.763± 0.013	0.687± 0.015	0.776± 0.012
NOV	2	1	0.732± 0.012	0.720± 0.009	0.731± 0.008	0.722± 0.017	0.726± 0.013	0.688± 0.016	0.687± 0.015	0.724± 0.015
		k	0.736± 0.020	0.731± 0.014	0.739± 0.018	0.764± 0.010	0.750± 0.019	0.735± 0.019	0.687± 0.015	0.738± 0.011
SUR	2	1	0.715± 0.015	0.725± 0.014	0.752± 0.017	0.724± 0.014	0.752± 0.011	0.688± 0.020	0.687± 0.015	0.753± 0.012
		k	0.750± 0.015	0.785± 0.016	0.690± 0.020	0.762± 0.013	0.762± 0.016	0.688± 0.020	0.687± 0.015	0.786± 0.017