UNIVERSIDADE FEDERAL DO CEARÁ

INSTITUTO UFC VIRTUAL

CURSO DE GRADUAÇÃO EM SISTEMAS E MÍDIAS DIGITAIS

JOÃO PAULO BARBOSA AMORIM LEITÃO

ANÁLISE COMPARATIVA DE ALGORITMOS DE APRENDIZAGEM DE MÁQUINA PARA CLASSIFICAÇÃO DE DIFICULDADE DE CURSOS DO TIPO MOOC

FORTALEZA

2023

JOÃO PAULO BARBOSA AMORIM LEITÃO

ANÁLISE COMPARATIVA DE ALGORITMOS DE APRENDIZAGEM DE MÁQUINA PARA CLASSIFICAÇÃO DE DIFICULDADE DE CURSOS DO TIPO MOOC

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas e Mídias Digitais do Instituto UFC Virtual da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas e Mídias Digitais.

Orientador: Prof. Me. Carlos Diego Andrade de Almeida.

FORTALEZA

2023

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)

L548a Leitão, João Paulo Barbosa Amorim.
Análise comparativa de algoritmos de aprendizagem de máquina para classificação de dificuldade de cursos do tipo mooc / João Paulo Barbosa Amorim Leitão. – 2023.
66 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Instituto UFC Virtual, Curso de Sistemas e Mídias Digitais, Fortaleza, 2023.
Orientação: Prof. Me. Carlos Diego Andrade de Almeida.

1. MOOCs. 2. Multiclass Classification. 3. Machine Learning. I. Título.

CDD 302.23

JOÃO PAULO BARBOSA AMORIM LEITÃO

ANÁLISE COMPARATIVA DE ALGORITMOS DE APRENDIZAGEM DE MÁQUINA PARA CLASSIFICAÇÃO DE DIFICULDADE DE CURSOS DO TIPO MOOC

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas e Mídias Digitais do Instituto UFC Virtual da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas e Mídias Digitais.

Aprovada em: 13/07/2023.

BANCA EXAMINADORA

Prof. Me. Carlos Diego Andrade de Almeida (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Rafael Augusto Ferreira do Carmo
Universidade Federal do Ceará (UFC)

Prof. Dr. Ernesto Trajano de Lima Neto
Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

Agradeço de coração às seguintes pessoas que desempenharam papéis fundamentais em minha jornada acadêmica e na conclusão deste trabalho:

À minha amada mãe e meu pai, sou imensamente grato por todo o amor, apoio incondicional e sacrifícios que fizeram ao longo dos anos. Seu encorajamento constante e crença em mim foram fontes de inspiração. Sou abençoado por ter vocês.

A meus irmãos, agradeço por serem minha rede de apoio e por sempre me lembrarem da importância de encontrar equilíbrio entre os estudos e a vida pessoal. Suas palavras de incentivo e momentos compartilhados me ajudaram a superar desafios e a manter a motivação.

A B, por estar sempre presente compartilhando risadas, desafios e momentos inesquecíveis ao longo dos anos.

A G, agradeço por você ser a pessoa incrível que é, sua presença constante e seu carinho são tesouros que guardo sempre comigo.

Yo e Jo, minha gratidão por a vida ter me presenteado com sua existência e proximidade, mesmo a tanto tempo atrás.

And e Gitana, suas amizades representam muito. Agradeço por existirem e por serem fontes de apoio e motivação.

A Jon, agradeço pela admiração mútua. Seu apoio é valioso.

Agradeço muito ao meu orientador Carlos Diego por sua orientação perspicaz, expertise e dedicação em orientar meu trabalho. Suas sugestões e *feedbacks* foram inestimáveis para o desenvolvimento e aprimoramento desta pesquisa.

Expresso minha gratidão aos membros da banca examinadora por dedicarem seu tempo e conhecimento para avaliar este trabalho. Suas contribuições e comentários foram valiosos para o aprimoramento do estudo.

Aos professores do curso de Sistemas e Mídias Digitais, em especial a Gilvan, que sempre reforçou o que é brio e Inga, que sempre demonstrou grande dedicação à docência, agradeço por compartilharem seus conhecimentos e experiências.

A Lula, agradeço pela sua força e garra de mudar o mundo mesmo que aos poucos, sem as políticas afirmativas não estaria onde estou hoje. Que seja sempre uma inspiração à busca de justiça e a igualdade para todos.

Agradeço a Deus por me conceder força, perseverança e sabedoria ao longo desta jornada acadêmica. Sua presença divina foi meu amparo em momentos de dificuldade e minha

fonte de esperança.

Por fim, agradeço a Duna, que sempre esteve ao meu lado, me acolhendo e me dando coragem para enfrentar os desafios. Sua existência tornou possível a minha.

Meu mais profundo agradecimento a todas as pessoas mencionadas e a todas as outras que, de alguma forma, contribuíram para o meu crescimento acadêmico e pessoal. Vocês fizeram a diferença e sou grato por ter cada um de vocês em minha vida.

"Pane no sistema, alguém me desconfigurou. Aonde estão meus olhos de robô? Eu não sabia, eu não tinha percebido. Eu sempre achei que era vivo." (Pitty, 2003)

RESUMO

O trabalho atual envolve o desenvolvimento de uma aplicação de Análise de Dados e Aprendizado de Máquina para classificar *Massive Open Online Courses (MOOCs)*, que são cursos *online*, abertos e massivos, em diferentes níveis de dificuldade. São utilizados algoritmos como *Random Forest Classifier*, *Logistic Regression* e *K-neighbors classifier* para determinar a melhor abordagem de classificação. O objetivo é encontrar as configurações ideais dos algoritmos, considerando diferentes métricas. A classificação adequada dos cursos é crucial para evitar frustrações e perda de confiança por parte dos participantes. A aplicação foi projetada para integrar-se a uma plataforma de cursos *online* futuramente, com a Coursera sendo escolhida como referência internacional. Utilizando o *dataset* da Coursera, a aplicação fornece uma classificação mais correta do nível de dificuldade dos cursos com base nas habilidades necessárias, no nome do curso e na instituição responsável, auxiliando os criadores de cursos na classificação adequada e garantindo a adequação dos cursos aos participantes.

Palavras-chave: MOOCs; Multiclass Classification; Machine Learning.

ABSTRACT

The current work involves the development of a Data Analysis and Machine Learning application to classify Massive Open Online Courses (MOOCs), which are online, open, and massive courses, into different levels of difficulty. Algorithms such as Random Forest Classifier, Logistic Regression, and K-neighbors classifier are used to determine the best classification approach. The objective is to find the optimal algorithm settings, considering different metrics. Proper classification of the courses is crucial to avoid frustrations and loss of trust from the participants. The application was designed to integrate into an online course platform in the future, with Coursera being chosen as the international reference. Using Coursera's dataset, the application provides more correct classification of the courses' difficulty level based on the required skills, course name, and responsible institution, assisting course creators in proper classification and ensuring course suitability for participants.

Keywords: MOOCs; Multiclass Classification; Machine Learning.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A educação a distância (EaD) é uma modalidade de ensino que tem crescido rapidamente, nacional e internacionalmente, especialmente em economias emergentes (GUERREIRO; BARROS, 2019). O mercado no Brasil cresceu nos últimos anos, principalmente após a facilitação do uso da *internet* nas residências pelo país, além da massificação de dispositivos computacionais (INEP, 2022).

Existe um aumento na oferta dos *MOOCs*, que são cursos *online*, abertos e massivos. Embora não haja uma definição consensual, esses cursos são geralmente abertos a todos, realizados totalmente *online* e disponibilizados para um grande número de alunos, sem pré-requisitos obrigatórios para participação. Esses cursos são comumente disponibilizados por provedores, sendo exemplos populares de provedores a Udacity, edX, Udemy, Khan Academy e Coursera. Esses provedores oferecem diferentes características nos *MOOCs*, como interatividade e aprendizagem colaborativa (*cMOOC*) ou transmissão de conteúdos (*xMOOC*). Além disso, na maioria dos provedores, busca-se disponibilizar os cursos de forma a possibilitar que os participantes do curso realizem o curso não supervisionados por professores ou tutores. (BATTESTIN; SANTOS, 2022).

Cursos com classificação de dificuldade inadequada podem resultar em experiências frustrantes para os participantes, levando a consequências indesejadas, como desistência e perda de confiança na plataforma. A precisão de uma classificação eficaz é motivada pela demanda crescente por plataformas de cursos *online*, abertos e massivos e sua responsabilidade em fornecer uma indicação mais correta e contextualizada do nível de dificuldade de cada curso. Neste trabalho, ao contribuir para o desenvolvimento de uma solução confiável e precisa na classificação de *MOOCs*, pode ser possível proporcionar uma experiência de aprendizado mais eficiente e satisfatória aos participantes.

O presente trabalho teve como objetivo principal projetar uma aplicação no campo da análise de dados e aprendizado de máquina, com o propósito de investigar o desempenho de algoritmos de classificação aplicados a um conjunto de dados específico e determinar a abordagem mais eficiente para classificar cursos *online* em diferentes níveis de dificuldade. A aplicação foi projetada com a intenção de ser posteriormente integrada a uma plataforma de cursos *online*, sendo o Coursera uma escolha específica devido à sua reputação internacional.

Foram analisados diferentes algoritmos e métricas buscando encontrar as configurações ideais que permitam a correta classificação dos cursos em três níveis de dificuldade:

avançado, intermediário e iniciante, com base nas habilidades necessárias para sua conclusão, na descrição do curso, na instituição que o disponibiliza e no nome do curso. O uso de um modelo de aprendizado de máquina foi escolhido como uma opção útil para evitar direcionar os participantes de forma incorreta para cursos que não correspondam ao seu nível de habilidade.

Além disso, o sistema desenvolvido identificou quais configurações de parâmetros são mais adequadas para cada modelo na busca pelo melhor desempenho e corretude possível, visando a obtenção de resultados otimizados considerando diferentes métricas indicadas para o contexto de classificação, a serem utilizadas na comparação entre os modelos.

Objetivo geral:

- Realizar o *benchmark* de modelos de aprendizado de máquina para classificação de cursos online em níveis de dificuldade.

Objetivos específicos:

- Identificar um conjunto de dados que possa ser utilizado de forma a atingir os objetivos visados neste trabalho.
- Identificar, para cada modelo de aprendizado de máquina selecionado para utilização neste trabalho, qual a configuração do modelo que produz os melhores resultados de desempenho de acordo com as métricas de desempenho escolhidas.
- Identificar entre os modelos selecionados para análise neste trabalho, o modelo de aprendizado de máquina que melhor realiza a tarefa de fazer a classificação de cursos online de acordo com seu nível de dificuldade.
- Identificar um conjunto de dados amplamente usado na comunidade de aprendizado de máquina que possa ser utilizado para validação do processo através da replicação do processo utilizado no conjunto de dados principal.
- Realizar a validação do processo com um conjunto de dados amplamente utilizado na comunidade de aprendizado de máquina.

O restante do trabalho está organizado da seguinte maneira:

- Capítulo 2 - Fundamentação teórica: São apresentados os principais conceitos e informações necessárias ao entendimento das bases deste trabalho. São estes: do que se trata a plataforma Coursera e como a mesma se organiza, conceitos de Machine Learning, aprendizado supervisionado multiclasse, dados desbalanceados, técnica SMOTE e do que se trata a biblioteca Scikit-Learn.
- Capítulo 3 - Metodologia: São apresentados os passos realizados para o desenvolvimento

do produto, desde o processo inicial de coleta de informações até a obtenção dos resultados finais.

- Capítulo 4 - Relato histórico e resultados: É explicada a sequência de eventos que levou a produção deste trabalho, assim como apresentados os resultados e realizada uma análise dos mesmos.
- Capítulo 5 - Conclusões e trabalhos futuros: É feita uma síntese acerca do que se pode concluir dos resultados em relação aos objetivos estabelecidos previamente, assim como esclarece-se acerca dos possíveis desdobramentos futuros de possíveis projetos que possam utilizar este trabalho como base.
- Referências - São listadas as referências utilizadas para embasar este trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Guiada pelos objetivos deste trabalho, o capítulo de Fundamentação Teórica está organizado para tentar prover o leitor com os conhecimentos básicos acerca de:

- Conceitos de *Machine Learning*;
- Modelos de *Machine Learning*;
- *Dataset* e técnica *SMOTE*;
- Métricas de desempenho;
- *Hyperparameter tunning*;
- Plataforma Coursera.

2.1 Conceitos de *Machine Learning*

O conceito de *Machine Learning* ou aprendizado de máquina descreve a capacidade dos sistemas de aprender a partir de dados de treinamento específicos do problema, automatizando o processo de construção de modelos analíticos e solucionando tarefas associadas (MITCHELL, 1997)

É uma abordagem orientada a dados em que os algoritmos aprendem automaticamente a partir de grandes volumes de dados. Em resumo, a qualidade dos dados influencia a precisão das generalizações. Existem três principais tipos de aprendizado de máquina: supervisionado, não supervisionado e por reforço (MORALES; ESCALANTE, 2022).

No aprendizado supervisionado, cada exemplo de treinamento possui uma resposta desejada, permitindo que o algoritmo construa um classificador capaz de determinar a classe de novos exemplos não rotulados. No aprendizado não supervisionado, os exemplos são fornecidos sem rótulos, e o algoritmo busca agrupá-los com base em suas similaridades. No aprendizado por reforço, o algoritmo recebe sinais de recompensa ou punição, ajustando suas hipóteses com base no desempenho, utilizando tentativa e erro (GOODFELLOW *et al.*, 2016).

A classificação multiclasse envolve inferir a classe correta dentre várias opções. Alguns algoritmos podem ser usados diretamente para problemas multiclasse, enquanto outros exigem adaptações (CARVALHO, 2021).

Para a escolha de modelos de aprendizado de máquina multiclasse para este projeto com a finalidade de determinar qual deles apresenta o melhor desempenho na classificação de cursos *online* em níveis de dificuldade, foi levado em conta durante o processo de desenvolvi-

mento a semelhança entre os nomes de funções padrão dos modelos na implementação realizada dos mesmos pelo *Scikit Learn*. Também decidiu-se, devido ao tempo limitado para a produção deste trabalho, limitar a quantidade de modelos analisados a três. Assim sendo, por um processo de eliminação, chegou-se à escolha de três modelos de algoritmos de classificação, sendo eles o *Random Forest Classifier*, o *Logistic Regression* e o *K-nearest neighbors*.

2.2 Modelos de Machine Learning

O modelo *Random Forest Classifier* é um método que se tornou popular devido à sua simplicidade e versatilidade, sendo um poderoso algoritmo de classificação supervisionado por conjunto. Ele combina várias Árvores de Decisão para formar uma floresta, dividindo o conjunto de dados em subconjuntos aleatórios. O método é adequado tanto para problemas de classificação quanto para regressão. Devido à sua precisão superior, robustez e capacidade de fornecer insights através da classificação de suas características, o *Random Forest Classifier* tem sido amplamente aplicado em várias aplicações de aprendizado de máquina (PETKOVIC *et al.*, 2018).

Já o *Logistic Regression* é um modelo estatístico amplamente utilizado em várias áreas, como medicina, instituições financeiras e econometria. (CABRERA, 1994). Pode ser implementado de forma a ser utilizado para classificação multiclasse, onde o problema de predição é tratado como uma regressão de múltiplos retornos, e a classe prevista pelo modelo é correspondente ao retorno com o valor mais alto (Scikit Learn, 2023d).

O modelo *K-nearest neighbors* é simples, eficiente e efetivo em áreas como reconhecimento de padrões, categorização de textos e reconhecimento de objetos. Ele realiza inferência com base na proximidade das instâncias de teste com as instâncias mais próximas no conjunto de dados. É um dos muitos algoritmos de aprendizagem supervisionada utilizados no campo de mineração de dados e aprendizado de máquina (Scikit Learn, 2023e).

Para a implementação deste projeto, foi utilizado o *SciKit-Learn*, que se trata de uma biblioteca em *Python* que oferece uma interface padrão para implementar algoritmos de aprendizado de máquina. Além disso, ele inclui outras funções auxiliares que são essenciais para a construção de um *pipeline* (um conjunto de passos a serem realizados em sequência) de aprendizado de máquina, como etapas de pré-processamento de dados, técnicas de reamostragem de dados, parâmetros de avaliação e interfaces de busca para ajustar/otimizar o desempenho de um algoritmo (HEIDARI *et al.*, 2022).

Dada a escolha dos modelos acima, convém observar que todos os modelos citados conseguem lidar com conjuntos de dados desbalanceados na implementação dos modelos disponibilizada na biblioteca do Scikit Learn, possuindo parâmetros ou configurações que possibilitam essa capacidade, como o parâmetro *class_weight="balanced"* dos modelos *Random Forest Classifier* e *Logistic Regression* e o parâmetro *weights="distance"* do modelo *K-nearest neighbors* (Scikit Learn, 2023j). Mesmo assim, é conveniente a realização de pre-processamento dos dados do conjunto de dados utilizado, de forma a reduzir de antemão os possíveis problemas que podem surgir pela presença de desbalanceamento dos dados.

2.3 *Dataset* e técnica *SMOTE*

O *dataset* (ou conjunto de dados) utilizado possui dados desbalanceados, por conta disso se faz necessário descrevermos de forma mais aprofundada o que isso representa. Dados desbalanceados são caracterizados pela presença de uma categoria (ou classe) minoritária (que possui menos amostras) em comparação com as categorias majoritárias (que possuem mais amostras) em um conjunto de dados. Isso pode causar problemas na construção de modelos de previsão, levando a resultados enviesados. Se não tratados adequadamente, os dados desbalanceados podem levar a modelos que não conseguem distinguir a classe minoritária, resultando em erros significativos (AZANK, 2022).

Há algumas possíveis abordagens para reduzir esse problema, as principais sendo a de *oversampling*, onde são adicionadas ao *dataset* amostras sintéticas criadas a partir dos padrões e características existentes nas classes com menor quantidade de amostras presentes no dataset (classes minoritárias), de forma que as classes que antes estavam subrepresentadas passem a não mais ser, e a de *undersampling*, onde são desconsideradas e removidas do *dataset* amostras das classes com maior representação (classes majoritárias), de forma a aproximar sua quantidade da quantidade de amostras de classes minoritárias e equilibrar a presença de todas as classes no *dataset* (LEMAÎTRE *et al.*, 2017).

Para lidar com o problema de desbalanceamento, foi utilizada neste trabalho a técnica *SMOTE*. De acordo com (TORRES-VÁSQUEZ *et al.*, 2019), a técnica de pré-processamento *SMOTE* ou *Synthetic Minority Over-sampling TEchnique* é utilizada para lidar com conjuntos de dados desbalanceados, onde a classe minoritária é ampliada até igualar a classe majoritária. O *SMOTE* é um método de sobreamostragem de minorias sintéticas que cria dados sintéticos com base nas características dos exemplos da classe minoritária, em vez de simplesmente substituir

os dados de forma aleatória. O *SMOTE* seleciona cada exemplo da classe minoritária e introduz exemplos sintéticos ao longo dos segmentos de linha que conectam os vizinhos mais próximos da classe minoritária. Dependendo do grau de sobreamostragem desejado, os vizinhos dos vizinhos mais próximos são escolhidos aleatoriamente. Essa técnica tem sido amplamente utilizada e com sucesso em problemas de desbalanceamento de classes binárias.

2.4 Métricas de desempenho

Foi determinada uma métrica padrão para avaliação de modelos de machine learning. Para isso se escolheu a métrica *f1 (f1 score*), também conhecida como *F-score balanced* ou *F-measure*, com o parâmetro *average="weighted*". O *f1-score* é interpretado como a média harmônica da *precisão* e do *recall*, onde o valor máximo do *f1-score* é 1 e o valor mínimo é 0. A contribuição relativa da *precision* e do *recall* para o *f1-score* é igual. A fórmula para o cálculo do *f1-score* leva em consideração tanto as métricas *precision* quanto *recall*: *f1 score* = 2 x [(*precision* x *recall*) / (*precision* + *recall*)] (Scikit Learn, 2023j).

A métrica *precision* mede a proporção de instâncias classificadas corretamente como positivas em relação a todas as instâncias classificadas como positivas. O *recall*, por outro lado, mede a proporção de instâncias positivas corretamente classificadas em relação a todas as instâncias verdadeiramente positivas (Scikit Learn, 2023f).

O *f1-score* leva em consideração tanto a capacidade de identificar corretamente as instâncias positivas quanto a capacidade de evitar falsos positivos. Quando o parâmetro "*average*"é definido como "*weighted*", a métrica *f1* é calculada considerando a média ponderada das métricas de *precisão* e *recall*, levando em conta o suporte de cada classe. Isso significa que as classes com maior número de amostras têm uma influência maior no cálculo do *f1 score*(Scikit Learn, 2023f).

No contexto do trabalho atual, onde há o uso de um *dataset* com desbalanceamento de classes, o *f1 score* contribui como uma métrica mais "justa" de avaliação dos modelos, já que o modo como o *f1 score* funciona leva em conta a característica do *dataset* (presença de quantidade de amostras diferentes para cada classe).

Além disso, foi obtido os resultados da métrica *ROC AUC curve* (*Receiver Operating Characteristic Area Under the Curve*), que se trata de uma métrica possível de ser visualizada através de um gráfico representando a taxa de verdadeiros positivos (*TPR*) em relação à taxa de falsos positivos (*FPR*) para diferentes limiares de classificação. Cada ponto na curva representa

um ponto de corte diferente para a classificação binária. Quanto mais próximo a curva estiver do canto superior esquerdo do gráfico, melhor será o desempenho do modelo (Scikit Learn, 2023h).

AUC é a medida da área sob a curva *ROC*. Ela varia entre 0 e 1, o valor 1 indicando um modelo perfeito e um valor de 0,5 indicando um modelo que é equivalente a uma escolha aleatória. Foi utilizada a função *roc_auc_score*(), que recebe como parâmetros as classes corretas para cada *feature*, as probabilidades previstas, e o parâmetro *multi_class* com o valor "*ovr*", que determina que a pontuação de desempenho deve levar em consideração o contexto multiclasse e a aplicação da estratégia "*One-vs-rest*", ou seja, o *ROC AUC score* deve ser calculado para cada classe em relação às outras classes, sendo apropriado para o contexto multiclasse pois leva em conta o desbalanceamento de classes do *dataset*(Scikit Learn, 2023h).

Foi obtida também a *confusion matrix* para cada modelo. A *confusion matrix* se trata de uma tabela que descreve o desempenho de um modelo, exibindo a contagem de classificações corretas e incorretas por classe. A tabela apresenta as seguintes informações: verdadeiro positivo (*True Positive* ou *TP*), que indica o número de classificações corretamente determinadas como positivas pelo modelo. Falso positivo (*False Positive* ou *FP*), que indica o número de classificações incorretamente determinadas como positivas pelo modelo, quando na verdade são negativas. Verdadeiro negativo (*True Negative* ou *TN*), que indica o número de classificações corretamente determinadas como negativas pelo modelo. E por último falso negativo (*False Negative* ou *FN*), que indica o número de classificações incorretamente determinadas como negativas pelo modelo, quando na verdade são positivas (Scikit Learn, 2023a) .

A *confusion matrix* possibilita entender melhor o desempenho do modelo, mostrando não apenas a acurácia geral, mas também informações sobre os erros de classificação e a capacidade do modelo em distinguir corretamente as diferentes classes (Scikit Learn, 2023a).

2.5 Hyperparameter tunning

Para encontrar a combinação de parâmetros para cada modelo que pudesse resultar no melhor desempenho de cada um deles, foi utilizado *hyperparameter tunning*. Ele consiste em utilizar técnicas de combinação de parâmetros para obtenção de uma combinação final de parâmetros que possua o melhor *score*, passou a ser utilizada. Hiperparâmetros são parâmetros que não são diretamente aprendidos dentro dos modelos. São passados como parâmetros para os modelos, correspondendo ao equivalente a uma configuração do modelo para determinado propósito. Na técnica de *hyperparameter tunning*, é feita uma busca por valores de parâmetros

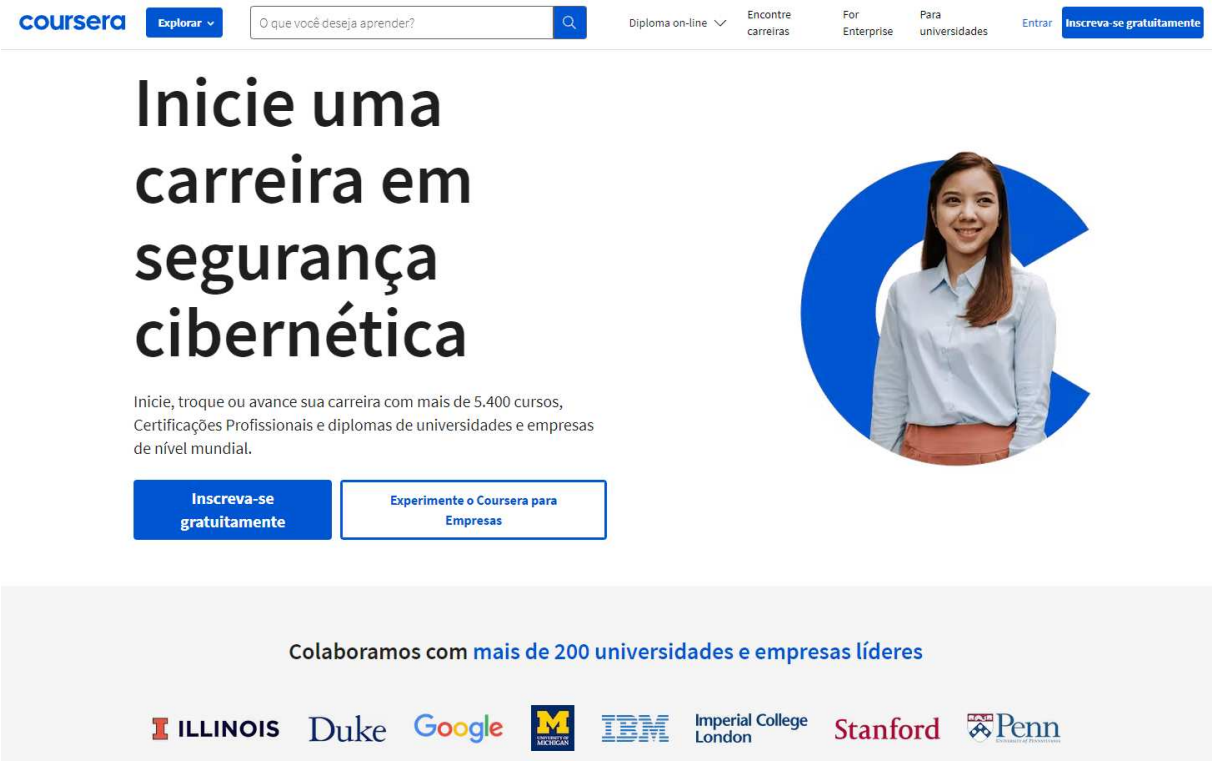
que possibilitem melhorar o desempenho do modelo para determinado propósito. Essa busca pode ser feita manualmente, ajustando os valores dos parâmetros antes do treinamento do modelo e posteriormente validando se ocorreu uma melhora nas métricas, ou de forma automatizada (Scikit Learn, 2023k).

2.6 Plataforma Coursera

Os *MOOCs*, que são cursos *online*, abertos e massivos, são ambientes de aprendizagem que se destacam pela sua escala global e diversidade de participantes. No entanto, a criação de ambientes de aprendizagem para um público tão amplo apresenta desafios complexos devido às diferentes origens, habilidades e experiências dos participantes (AGONÁCS; MATOS, 2020).

Criado em 2012, o Coursera, que tem sua página inicial que pode ser vista na Figura 1, é uma plataforma de ensino online que possui aproximadamente 124 milhões de alunos registrados, parceria com mais de 300 universidades e parceiros industriais, disponibilizando cursos, especializações, certificados profissionais, projetos guiados, cursos de bacharelado e mestrado (Coursera, 2023)

Figura 1 – Plataforma Coursera



Fonte: elaborada pelo autor.

O conteúdo de um curso possui diferentes aulas em vídeo, notas de texto, atividades

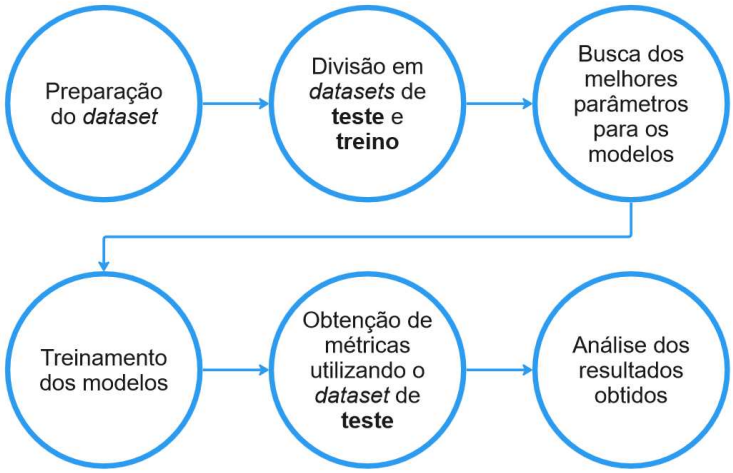
a serem realizadas pelo aluno, testes e exames finais. Há um tempo limite para execução dos testes e atividades, que sempre que possível são avaliados de forma automatizada pelo sistema da Coursera, reduzindo a dependência de avaliadores humanos. Ao se obter desempenho satisfatório ao longo do curso, quando da finalização do mesmo é possível obter o certificado de conclusão (KORABLEVA *et al.*, 2019).

O conteúdo de um curso é organizado em capítulos ou módulos. Cada módulo é dividido em seções ou lições, que por sua vez são organizadas em sequências de aprendizagem. No Coursera, cada semana do curso corresponde a um módulo. As seções ou lições de aprendizagem representam os tópicos a serem estudados a cada semana. Uma lição está associada a um módulo específico. Por fim, uma sequência de aprendizagem é um conjunto de atividades propostas pelo professor para os alunos como parte de uma lição, como avaliações, videoaulas, leituras, tarefas programadas, entre outras. As atividades incluídas nas sequências de aprendizagem são as mais relevantes para o acompanhamento dentro da plataforma, pois correspondem diretamente às atividades realizadas pelo aluno durante seu processo de aprendizagem no curso. O acompanhamento dessas atividades permite obter informações relevantes para aplicar técnicas de análise de aprendizado. O Coursera utiliza 16 tipos de atividades de aprendizagem que o professor pode adicionar ao projetar uma lição na plataforma. Os fóruns de discussão não fazem parte desse grupo de atividades, pois são tratados separadamente no Coursera (PÉREZ-ÁLVAREZ *et al.*, 2018). Para este trabalho, foi utilizado o *dataset* da Coursera, pois o mesmo apresentava características que o tornavam mais adequado aos objetivos estabelecidos. No próximo capítulo, se dará seguimento ao aprofundamento acerca dessas características e processo utilizado em busca de atingir os objetivos.

3 METODOLOGIA

A partir deste ponto, a apresentação da metodologia aplicada para o desenvolvimento do produto será realizada, abrangendo desde o processo inicial de coleta de informações até a obtenção dos resultados finais. Os principais passos realizados podem ser visualizados na Figura 2, são eles: preparação do *dataset*, divisão em *datasets* de teste e treino, busca dos melhores parâmetros para os modelos, treinamento do modelos, obtenção de métricas utilizando o *dataset* de testes e análise dos resultados obtidos.

Figura 2 – Principais processos realizados



Fonte: (Scikit Learn, 2023i)

O código fonte utilizado foi disponibilizado no Github , plataforma de hospedagem de código fonte e arquivos (Github Inc, 2023). Os *links* de acesso para cada arquivo individual de código-fonte citado neste trabalho estão disponíveis no Apêndice A. O diretório raiz para o projeto se encontra no *link https://github.com/joaopaulobarbosa/TCC_2023*.

3.1 Configuração do ambiente de desenvolvimento

O hardware e Sistema Operacional utilizados foram um *notebook Lenovo Ideapad 330* com processador *Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz*, com 12GB de memória *RAM*, 240gb de armazenamento *SSD* e sistema operacional *Windows 11 Pro* 64bits.

Foi utilizada a *IDE Visual Studio Code* e a biblioteca *Jupyter* para suporte básico a *notebooks* python. Foi também usado um editor de texto e planilhas para abertura para visualização e correções dos arquivos *.csv* contendo o *dataset*, ou seja, o conjunto de dados utilizados para treinamento de modelos, bem como o teste de tais modelos.

Além disso, foi usado o Anaconda(Anaconda, 2023), plataforma de código aberto e distribuição de *software* utilizada para ciência de dados, análise de dados e computação científica. O Anaconda disponibiliza um ambiente de desenvolvimento integrado, onde reúne diversas ferramentas e recursos em um único ambiente para facilitar o desenvolvimento, depuração e testes de código (Hock-Chuan, Chua, 2023).

O *Anaconda* também possui um gerenciador de pacotes chamado Conda. Quando ocorre a instalação do Anaconda, são instalados automaticamente pacotes de ciência de dados e aprendizado de máquina de código aberto mais utilizados. Entre esses pacotes, os mais importantes para o processo de desenvolvimento foram:

- Pandas , biblioteca de código aberto que fornece estruturas de dados, além de ferramentas de análise de dados para a linguagem de programação *Python* (Pandas, 2023).
- NumPy, biblioteca para computação científica em Python, que fornece um objetos de *arrays* multidimensionais, vários objetos derivados (como matrizes mascaradas e matrizes), e uma variedade de rotinas para operações rápidas em matrizes. Isso inclui operações matemáticas, lógicas, manipulação de forma, ordenação, seleção, entrada/saída, transformadas discretas de *Fourier*, álgebra linear básica, operações estatísticas básicas e simulação aleatória (NumPy, 2023).
- Scikit-learn, biblioteca de aprendizado de máquina de código aberto para a linguagem de programação *Python*. Também conhecido como *sklearn*, ele fornece uma ampla gama de algoritmos e ferramentas para tarefas de aprendizado de máquina, incluindo classificação, regressão, clusterização, redução de dimensionalidade, pré-processamento de dados e avaliação de modelos (Scikit Learn, 2023j).
- Matplotlib: biblioteca para criar visualizações estáticas, animadas ou interativas em *Python* (Matplotlib, 2023).

3.2 Identificação de *datasets* apropriados

Em seguida, foi realizada a etapa de coleta de dados para determinar quais *datasets* poderiam ser utilizados para o treinamento dos modelos. Para isso, foi feita uma busca por *datasets* na base de dados do Kaggle, plataforma de competição de ciência de dados e comunidade *online* de cientistas de dados e praticantes de aprendizado de máquina (Kaggle, 2023).

Foi escolhido o *dataset* do Coursera disponibilizado por Khushee Kapoor no Kaggle, que apresentava características adequadas para o treinamento de modelos, pois possuía uma

estrutura onde as classes alvo definindo o nível de dificuldade de cada curso estavam bem definidas, contendo as classes “*Advanced*”, “*Intermediary*”, “*Begginer*”, com colunas (*features*) diversas que poderiam ser combinadas para que o modelo pudesse identificar a que classe pertenciam.

O *dataset* também possuía um tamanho reduzido, consistindo de 3522 amostras ou *samples* e 7 colunas, sendo as colunas nomeadas “*Course Name*”, ”*University*”, ”*Difficulty Level*”, “*Course Rating*”, “*Course URL*”, "*Course Description*” e ”*Skills*”. Todas as colunas consistiam em texto.

3.3 Preparação manual dos dados

Foi realizado o *download* do *dataset* em *.csv*, mas foi observado que ele estava configurado para separação entre colunas somente utilizando separação por vírgulas. Foi aberto o arquivo utilizando um editor de planilha, e tentou-se realizar a separação de colunas utilizando a função “Texto para Colunas”, mas não havia possibilidade de separar corretamente porque a coluna de “*Skills*” possuía texto separados por vírgulas, ou seja, a função separaria os dados da coluna “*Skills*” em todos os pontos que houvessem vírgulas, gerando diversas colunas extras sem nome.

Foi constatado um problema nos dados da coluna "*Skills*": as vírgulas estavam acompanhadas por um espaço em branco. Para corrigir essa situação, uma série de passos foi realizada. Primeiramente, utilizou-se a ferramenta de localização e substituição de termos do editor para identificar as vírgulas com espaçamento adicional. Cada uma dessas vírgulas foi substituída pelo símbolo "#"como medida temporária.

Em seguida, realizou-se uma nova busca para encontrar as vírgulas que indicam a separação das colunas. Essas vírgulas foram substituídas pelo caractere ";"e, em seguida, fez-se o processo inverso: substituiu-se o "#"pelo caractere ",", restaurando assim o formato original dos dados na coluna "*Skills*". Após essa etapa, foi realizada a separação dos dados em colunas, utilizando o caractere ";"como delimitador, organizando os dados corretamente. Com o objetivo de preservar o arquivo original, foi feita cópia do mesmo, nomeando-o como "*Coursera_preparacao.csv*". Por fim, realizou-se uma alteração no arquivo "*Coursera_preparacao.csv*", renomeando a coluna "*Difficulty Level*"para "*Difficulty_Level*".

Foi identificado que o arquivo contendo o *dataset* havia sido disponibilizado no site contendo erros de codificação de caracteres acentuados, onde os mesmos haviam sido

substituídos pelos caracteres "***ï¿½***". Convém observar que foi realizado um processo de *debug* minucioso para verificar se não se tratava de um erro de codificação do editor de planilha ou mesmo do ambiente de desenvolvimento, processo esse encerrado somente após diversas abordagens terem sido testadas sem sucesso. Como não foi possível verificar facilmente a quais caracteres o caractere "***ï¿½***" correspondia, já que é um símbolo genérico para identificar um caractere é desconhecido ou não pode ser exibido, foi feita uma análise cuidadosa do *dataset* para identificar pontos onde fosse possível a correção em lote utilizando a ferramenta de busca e substituição do editor de planilha, de forma a obter dados acerca da quantidade de ocorrências de cada termo a ser corrigido, sem considerar o termo em conjunto com outros termos na mesma célula, e selecionando uma coluna por vez.

Foi realizada a correção no texto da coluna "*University*", por a mesma possuir uma quantidade reduzida de nomes diferentes de universidades que estão disponíveis de forma correta online viabilizando assim uma correção manual. Para isso foi realizada a busca utilizando uma ferramenta de pesquisa na internet, tendo como termo de busca cada nome incorreto, removendo os caracteres "***ï¿½***" do termo de pesquisa, e ao se identificar no retorno da pesquisa o nome correto, foi realizada a substituição no arquivo .csv dos nomes de universidades incorretos por sua versão correta. Foi também coletada a quantidade de amostras modificadas para cada universidade corrigida.

A Tabela 1, mostra os ajustes realizados em 96 amostras com caracteres incorretos ou incompatíveis. Após as correções feitas, foi renomeado o arquivo *Coursera_preparacao.csv* paro *Coursera_original_university_name_fixed.csv*.

Tabela 1 – Correções de caracteres incorretos

Original	Correção
ï¿½cole Polytechnique	École Polytechnique
ï¿½cole Polytechnique Fï¿½dï¿½rale de Lausanne	École Polytechnique Fédérale de Lausanne
ï¿½cole des Ponts ParisTech	École des Ponts ParisTech
Universitï¿½ Bocconi	Università Bocconi
Universitat Autï¿½noma de Barcelona	Universitat Autònoma de Barcelona
Ludwig-Maximilians-Universitï¿½t Mï¿½nchen (LMU)	Ludwig-Maximilians-Universität München
ï¿½cole normale supï¿½rieure	École normale supérieure
CentraleSupï¿½lec	CentraleSupélec
Universidad Nacional Autï¿½noma de Mï¿½xico	Universidad Nacional Autónoma de México
Technische Universitï¿½t Mï¿½nchen (TUM)	Technische Universität München
Fundaï¿½ï¿½o Instituto de Administraï¿½ï¿½o	Fundação Instituto de Administração
Institut Mines-Tï¿½lï¿½com (LMU)	Institut Mines-Télécom
(ISC)ï¿½	ISC
Pontificia Universidad Catï¿½lica de Chile	Pontificia Universidad Católica de Chile

Fonte: elaborada pelo autor.

3.4 Escolha dos modelos

Após finalizadas as operações de ajuste no *dataset*, foi dada continuidade com o desenvolvimento do código. Foram escolhidos os modelos *Random Forest Classifier*, *Logistic Regression* e *K-Nearest Neighbors* do *Scikit Learn*. Esses modelos foram selecionados pois os mesmos apresentavam características de implementação em comum, como por exemplo a de que em todos a implementação da função de predição utilizada para calcular as probabilidades para cada classe no conjunto de testes ter o mesmo nome. Essa padronização reduziu a necessidade de inserção de mais complexidade no código para a chamada da função correta para cada modelo a ser testado, caso o cenário fosse de uso de modelos com implementações diferentes.

Outro ponto é que os três modelos estavam definidos na documentação oficial do *ScikitLearn* como sendo inerentemente multiclasse, ou seja, eram modelos que não precisavam ser adaptados ao contexto de classificação multiclasse, pois já eram adequados para essa tarefa (Scikit Learn, 2023j).

3.5 Abordagem inicial de implementação

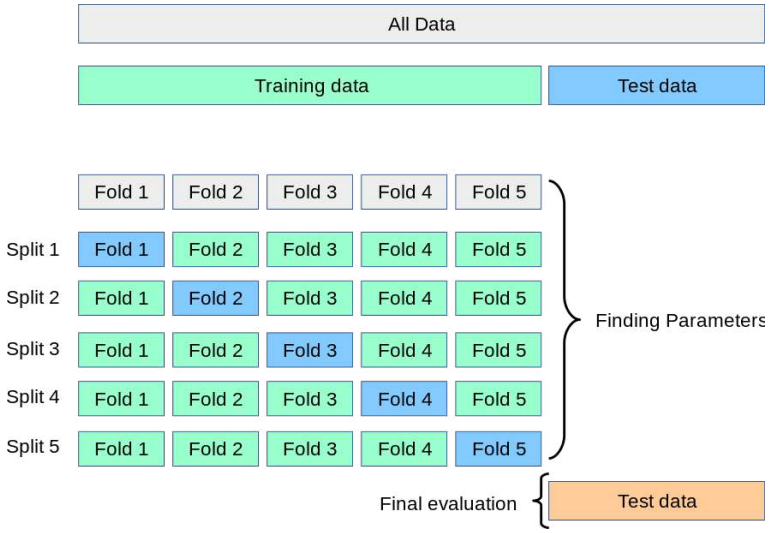
Na primeira abordagem de implementação do projeto, não foi utilizada a técnica de *hyperparameter tunning*, além disso, foram utilizadas somente as colunas *Difficulty_Level* e *Skills*. O código fonte para a preparação dos dados encontra-se no Apêndice A, na subseção "Código-fonte 1 – Preparação de dados, primeira abordagem". As tarefas realizadas por ele foram:

1. Obtenção do *dataset* a partir do armazenamento do arquivo;
2. Remoção de amostras contendo valores vazios na coluna "*Skills*" e na coluna "*Difficulty_Level*";
3. Seleção para uso somente das colunas *Skills* e *Difficulty_Level*;
4. Geração de uma coluna extra chamada "*category_id*", utilizando a função *factorize()* para que cada valor dessa coluna correspondesse à classe da amostra correspondente, presente na coluna "*Difficulty_Level*".

Após a preparação de dados, foi realizada a comparação entre modelos utilizando o procedimento de validação cruzada (*cross-validation*). Validação cruzada se trata de um processo onde o *dataset* de treino é dividido em tempo real em k conjuntos menores, chamados *k-folds*. Em seguida, o modelo é treinado usando um dos conjuntos menores (k-1) como *dataset* de

treinamento. Na Figura 3, é ilustrado como se dão essas divisões.

Figura 3 – Funcionamento da *cross-validation*



Fonte: (Scikit Learn, 2023i)

O modelo é então validado utilizando o resto do *dataset* como conjunto de teste, para que se obtenha uma medida de desempenho, como por exemplo *f1 score*. Esse processo é repetido com cada um dos *k-folds* e conjunto de dados restante, e a cada repetição, são coletadas as medida de desempenho. Ao final, é calculada a média das medidas de desempenho coletadas, além da medida de desempenho final.

A principal vantagem é a obtenção de uma medida de desempenho mais confiável, já que ao longo da execução da *cross-validation*, diferentes configurações de conjuntos de treino e teste são utilizadas para treino do modelo e cálculo das medidas de desempenho.

A principal desvantagem é o uso aumentado de recursos de processamento e maior tempo de execução, já que serão realizadas diversas operações de divisão *(split)* de conjuntos em treino e teste, treinamento do modelo e coleta de medidas de desempenho.

Uma das preocupações quando se realiza o treinamento de modelos e a coleta de medidas de desempenho é o risco de *data leakage*. *Data leakage* é um termo que significa vazamento de dados. No contexto de aprendizagem de máquina supervisionado, se trata da situação onde informação que deveria ser disponibilizada ao modelo somente na etapa de predição é disponibilizada na etapa de treinamento (*fitting*) (Scikit Learn, 2023b).

Como consequência, o modelo terá um desempenho artificialmente aprimorado, pois estará sendo requisitado na etapa de testes a fazer uma correlação entre os valores de teste e suas respectivas classes, mas já teve acesso a quais correlações são corretas na etapa de treino. Ou

seja, se torna desnecessária a capacidade preditiva real, já que o modelo já sabe as respostas corretas.

Para dificultar a ocorrência de *data leakage*, na etapa do código de divisão do *dataset* em treino e teste, foi utilizada a estratégia padrão de realizar essa divisão usando *train_test_split*(), uma função que realiza a divisão de um *dataset* em *datasets* de treino e teste de forma automatizada.

A primeira abordagem utilizada foi realizar a divisão do *dataset* utilizando somente a coluna de *Skills* como *features* e a coluna de *Difficulty_Level* como classes. O *dataset* dessa forma foi dividido nos *dataframes X_train (Skills)*, *X_test (Skills)*, *y_train (Skills)*, *y_test (Difficulty_Level)*. Um *dataframe* é uma estrutura de dados tabular de duas dimensões, gerada pela biblioteca Pandas. Consiste em linhas e colunas, com cada coluna contendo um tipo de dados. *Dataframes* são utilizados normalmente para tarefas como limpeza de dados, pré-processamento, exploração e análise. O conjunto de treino ficou com a proporção de 80% do *dataset* original, e o conjunto de teste ficou com a proporção de 20% do *dataset* original.

3.6 Segunda abordagem: Separação de *datasets* em arquivos

Logo foi identificado que um possível problema seria se os conjuntos de treino e teste tivessem amostras diferentes para diferentes modelos durante o uso da função *train_test_split()*, já que devido ao contexto do projeto, onde foram concentradas tarefas específicas em arquivos separados no projeto, como os diferentes arquivos de *tunning* e o arquivo de modelo, cada arquivo de código realizava a importação do arquivo *"Coursera_original_university_name_fixed.csv"* e a partir desse *dataset* original, dividiria em treino e teste.

Isso resultaria na possibilidade de que em cada divisão as amostras contidas em treino e teste fossem diferentes para cada algoritmo, levando os mesmos a serem aprimorados no processo de *tunning* e comparados com combinações de amostras (*samples*) diferentes, levando a um possível viés de desempenho de um modelo em relação a outro, produzindo resultados diferentes pois não estariam sujeitos às mesmas condições de treinamento e comparação.

Para corrigir esse problema, adotou-se a estratégia de separar o *dataset* "fisicamente"em dois arquivos *.csv*, de treino e teste, dessa vez 60% treino e 40% teste (como mencionado anteriormente, na primeira abordagem a divisão foi feita em 80% treino e 20% teste), pois se observou que um possível motivo para o baixo desempenho dos modelos com o *dataset* anterior poderia ser de que houvesse uma quantidade pequena de amostras disponíveis para teste.

Os arquivos foram nomeados *train.csv* e *test.csv.* Nesse código também se concentrou a limpeza automatizada de dados, possibilitando a remoção da etapa de limpeza de dados dos códigos de *tunning* e comparação de modelos.

Além disso, como uma forma de disponibilizar mais dados para o treinamento dos modelos, decidiu-se incluir as colunas *"Course_Name", "University"e "Course_Description"* nos *datasets* de treino e teste, além das colunas já presentes, "*Difficulty_Level*"e "*Skills*". O *link* de acesso ao código fonte específico para essas tarefas está disponível no Apêndice A, na subseção "Código-fonte 2 – dataset_split_dataclean.py".

O código-fonte realiza uma série de tarefas para processar e estruturar um *dataset*. Inicialmente, são importadas as bibliotecas e métodos necessários para o processamento dos dados. Em seguida, é feita a leitura do arquivo contendo o *dataset*, e os dados são armazenados em um *dataframe*.

Antes de realizar os procedimentos de limpeza e estruturação dos dados, o código exibiu informações sobre o dataset, como as colunas presentes, as dimensões e as classes únicas na coluna "*Difficulty_Level*". Em seguida, foram selecionadas somente as classes desejadas, "*Advanced*", "*Intermediary*"e "*Beginner*", ignorando amostras correspondentes às classes "*Not Calibrated*", "*Goldman Sachs*"e "*nan*"ou seja, cursos não classificados em uma classe válida e "*Conversant*", uma classe correspondente a um nível de curso abaixo de "*Begginer*"mas que possuía poucas amostras e não se encaixaria nos requisitos estabelecidos nesse projeto.

Para garantir a qualidade dos dados, foram realizadas algumas etapas de remoção. Amostras que possuem valores vazios ou booleanos nas colunas "*Course_Name*", "*University*", "*Difficulty_Level*", "*Course_Description*"e "*Skills*"são removidas do *dataset*. Além disso, caracteres de aspas duplas no início e no final do texto são eliminados, quando presentes. Também são descartadas as amostras que contêm apenas valores numéricos, pois o objetivo é classificar corretamente os cursos com base nas informações de texto disponíveis nas colunas selecionadas.

Por fim, é gerada uma nova coluna chamada "*category_id*", utilizando a função "*factorize()*". Essa coluna atribui um valor correspondente a cada classe da coluna "*Difficulty_Level*", facilitando a classificação dos cursos. Assim, o código realiza todas essas etapas de processamento e limpeza, preparando o *dataset* para análises e classificações futuras.

Foi identificado ao analisar manualmente os arquivos finais de treino/ teste utilizando um editor de planilhas que a função de remoção de caractere “ (aspas duplas) produziu um valor vazio em uma das amostras do *dataset* de treino, então se procedeu novamente com a chamada e

execução da função de remoção de amostras com valores vazios novamente.

Segue amostra identificada: *Course_Name: Social Marketing Capstone Project University: Northwestern University Difficulty_Level: Advanced Course_Description: Your markets are on social and you need to be there. However, your social strategy needs to be based on the business metrics which define your success. This final Capstone Project in the Social Marketing Specialization will put the methodologies, tools, and insights you have learned to the test as you create a multifaceted plan to assure effective social marketing is an integral part of your business strategy. Whether your company has a sophisticated Engagement Strategy or you are a new start-up, you will learn to harness the full power of social marketing to grow provable market share and build stronger relationships with your high value markets. For success in today's digital world, you must have a plan to integrate your social and mobile marketing strategies into your business strategy. In this Capstone you will use the practical skills that youï¿½ve mastered through the Specialization to demonstrate your ability to integrate social and mobile marketing strategies into a companyï¿½s business strategy. 2016 Capstone Schedule: 1st offering begins on February 29 (2/29/16). It will be offered again starting on May 23 (5/23/16), August 29 (8/29/16), and November 28 (11/28/16) Skills:* (vazio, não possuía nenhum valor) *category_id*: 1

Após as etapas de processamento e estruturação dos dados do *dataset*, foi novamente chamada a função para exibir as características do *dataset*. A função mostrou as colunas presentes no *dataset*, as dimensões do *dataset*, as classes únicas na coluna "*Difficulty_Level*"e o *ID* correspondente a cada classe na coluna "*category_id*".

Em seguida, foi realizado o cálculo e exibição do balanceamento entre as classes do *dataset*. Essa análise mostrou a quantidade de amostras presente em cada classe, apresentando os resultados em forma de gráfico. Além disso, foram calculadas as estatísticas de balanceamento de classes no *dataset*. Essas estatísticas incluíram informações como a classe, a quantidade de amostras pertencentes a cada classe e a porcentagem dessa quantidade em relação ao total de amostras.

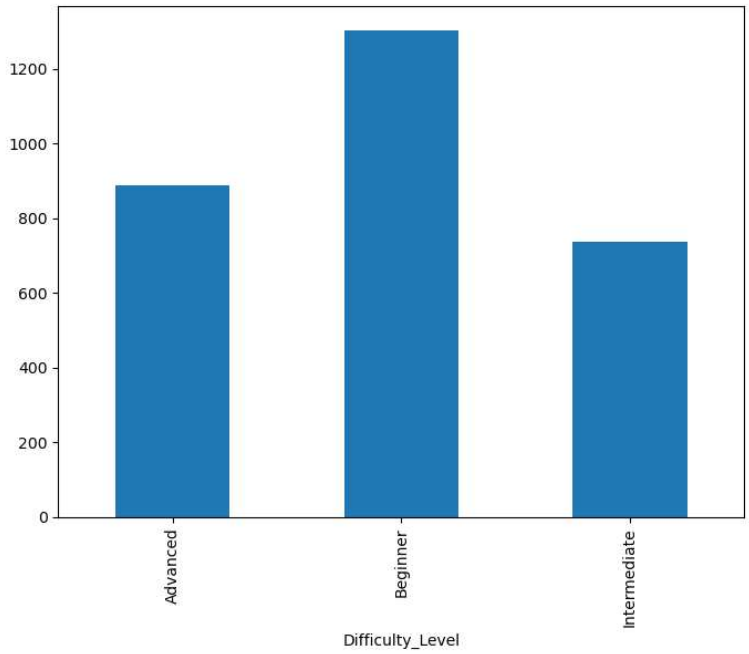
Para a preparação do treinamento e teste do modelo, o *dataset* foi dividido em dois conjuntos. A divisão considerou a proporção de cada classe nos conjuntos de treino e teste, utilizando o parâmetro "*stratify*"com base na coluna "*Difficulty_Level*"do *dataframe* "coursera". Em seguida, foi definido o caminho no sistema de arquivos onde seriam armazenados os conjuntos de teste e treino. Os conjuntos foram salvos em arquivos .csv com os nomes

"train.csv"e "test.csv", respectivamente.

Por fim, os *datasets* de treino e teste foram persistidos no armazenamento local, em arquivos separados por ponto e vírgula (;). Essa ação garantiu a preservação dos conjuntos de dados para uso posterior.

Antes da execução do código, o *dataset* possuía as seguintes dimensões: 3532 linhas (amostras) e 7 colunas (1 coluna de classes e 6 colunas de *features*). Após o processo de limpeza e preparação de dados, o *dataset* possuía 2938 linhas (amostras) e 6 colunas (1 coluna de classes e 6 colunas de *features*).

Figura 4 – *Class Balancing*



Fonte: elaborada pelo autor.

Tabela 2 – *Class Imbalance Summary*

Class	**Count**	**Percentage (%)**
Beginner	1303	44.50
Advanced	887	30.29
Intermediate	738	25.20

Fonte: elaborada pelo autor.

Como se pode observar na Figura 4 e na Tabela 2, há uma grande desproporção entre a quantidade de amostras classificadas como *Beginner* e *Intermediate*. A partir das informações acima, pode-se concluir que o *dataset* possui um desbalanceamento claro. O desbalanceamento entre classes se dá quando há uma diferença significativa da quantidade de amostras presentes em

cada classe. Isso pode levar o modelo a dar um peso menor às classes com menos amostras, e um peso maior às classes com maior quantidade de amostras, gerando durante o treinamento um viés que pode reduzir o desempenho de predição do modelo (RAMYACHITRA; MANIKANDAN, 2014).

3.6.1 Hyperparameter tunning

Nas primeiras execuções do código de comparação entre modelos, foi observado que as medidas de desempenho estavam muito abaixo do esperado (*50% f1 score 70% ROC AUC curve*), girando em torno de 40% para o modelo com melhor desempenho (*Random Forest Classifier*). Assim, buscou-se alternativas que levassem a um desempenho melhor. Uma alternativa encontrada foi o uso da técnica de *hyperparameter tunning*, que pode ser utilizada de forma automatizada com funções *GridSearchCV*, *RandomizedSearchCV* ou outras alternativas. Neste trabalho, se utilizou inicialmente o *GridSearchCV*.

O mesmo tem como característica realizar uma busca exaustiva, realizando validação cruzada de cada combinação entre um conjunto de opções de parâmetros do modelo, coletando métricas de desempenho para avaliar o quão adequada foi essa combinação e comparando com a métrica de desempenho obtida no passo anterior. A cada execução, é armazenado o *score* mais alto e a combinação que o produziu, até todas as combinações terem sido realizadas e se obtenha a melhor métrica de desempenho obtida. O *GridSearchCV* retorna então as informações de qual a melhor combinação de hiperparâmetros, assim como os valores de métricas obtidos (Scikit Learn, 2023c).

O processo utilizado pelo *GridSearcCV* para identificar a melhor combinação de parâmetros leva a um uso exaustivo de recursos de processamento. Quanto maior a combinação possível de parâmetros e seus valores, maior o tempo necessário para testar todas as combinações, pois durante o *cross-validation*, é realizado o split do *dataset* em teste e treino, é treinado o modelo com o conjunto de treino e avaliado o mesmo com o *dataset* de teste, esses passos sendo realizados a cada combinação de parâmetros possível.

Por isso, passou-se a utilizar o *RandomizedSearchCV*. O mesmo tem como característica extrair uma amostra das combinações possíveis, e só então realizar a testagem exaustiva, mas abrangendo somente essa quantidade limitada de amostras e não todas as possíveis combinações de parâmetros (Scikit Learn, 2023g).

Sendo assim, foi refatorado o código de *tunning* de cada modelo, para que o mesmo

utilizasse *RandomizedSearchCV* como abordagem na busca por combinações melhores de parâmetros.

A utilização inicial do *GridSearchCV* e a observação de sua limitação no contexto do trabalho atual também motivou a adoção da estratégia de separar diversos arquivos *.py* (extensão de arquivo de código-fonte na linguagem *Python*) contendo o código de *hyperparameter tunning* de cada modelo e outro arquivo no formato *Python* para comparação entre os modelos. Isso também possibilitou focar em problemas mais específicos por vez, facilitando o *debug*.

3.6.1.1 Random Forest Classifier

Buscou-se tentar identificar os melhores parâmetros possíveis para o modelo *RandomForestClassifer*. O *link* de acesso ao código fonte responsável por essa tarefa está disponível no Apêndice A, na subseção "Código-fonte 3 – tunning_randomforestclassifier.py". Foi realizada a importação de bibliotecas e principais métodos e inicializado um contador em segundos, de forma a monitorar em quanto tempo se daria o processo de *tunning* e exibição dos resultados.

Em seguida, foi realizada a leitura do *dataset* de treino armazenado no arquivo "train.csv"e realizada a preparação dos dados, obtendo como resultado as *features*, *targets* (classes alvo) codificados em valores numéricos e a relação entre nome das classes e seus valores codificados. Para a realização das tarefas acima, houve a conversão do *dataset* em um *dataframe*, utilizando a biblioteca Pandas. Foi realizada a codificação da coluna *University* para valores numéricos, pois os modelos compreendem somente valores numéricos, e a coluna *University*, assim como outras colunas do *dataframe*, possui dados em texto. Foi gerado um *dataframe* contendo a coluna *University* codificada, para posteriormente uni-lo ao *dataframe* final. Em seguida, foi gerado também um *dataframe* onde foram unidas em uma só coluna as colunas "*Course_Name*", "*Course_Description*" e "*Skills*".

Para determinar quais termos eram mais frequentes na nova coluna gerada, foi utilizada a conversão da coleção de documentos de texto da coluna do novo *dataframe* para uma matriz numérica de contagem de *tokens*, contendo somente a ocorrência dos termos mais frequentes. A quantidade máxima de termos mais frequentes foi determinada através de diversas testagens manuais com vários valores diferentes até obter o que melhor produzisse resultados satisfatórios das métricas dos modelos. Além da contagem de termos, também foi considerado combinações de termos únicos e de dois termos. Também foi realizado o processo de colocar em minúsculas todas as letras dos termos. Essas operações foram realizadas com o intuito de

melhorar o desempenho dos modelos.

A matriz numérica resultante foi convertida em um *dataframe* e seus nomes de colunas foram convertidos para texto, de forma a evitar o erro de tempo de execução que informa que nomes de colunas precisam ser do tipo texto, e não inteiros. Foi então realizada a união dos *dataframes* contendo a coluna *University* e a matriz numérica em um único *dataframe*, em seguida codificada a coluna com as classes para valores numéricos e adicionada a mesma em outro *dataframe*. O resultado, como já mencionado anteriormente, foram três conjuntos essenciais para o funcionamento dos modelos e do cálculo de métricas: Um *dataset* com as *features*, um *dataset* com as classes, e uma variável contendo a relação entre quais classes se referem a quais valores codificados. O mesmo processo descrito nos passos anteriores foi aplicado ao dataset de testes armazenado em *"test.csv"*.

Em seguida, foi definido um conjunto de parâmetros e seus possíveis valores, a serem utilizados na testagem de forma a identificar quais combinações resultariam em melhor desempenho do modelo.

Foi então definido o modelo e criada uma *pipeline*, com o objetivo de realizar as seguintes ações: aplicação da técnica *SMOTE* no *dataset*, aplicação de *scaling* no *dataset* e por último, instanciação do modelo Random Forest Classifier. *Scaling* é o processo de transformar *features* numéricas de forma que as mesmas possuam um tamanho consistente. É aplicada a transformação dos valores das *features* de forma que eles fiquem em um tamanho ou distribuição específica. Esse processo é importante pois ajuda a assegurar que *features* com diferentes escalas sejam tratadas com igualdade durante o treinamento do modelo (WAN, 2019).

Foi utilizado em seguida o método *StratifiedKFold*. Ele se trata de um *cross-validator*, ou seja, sua função é gerar conjuntos de treino e teste que contenham a mesma distribuição de classes que existe no *dataset* original, ou o mais próximo possível dessa distribuição, em que a diferença seja de no máximo uma amostra entre cada conjunto, de forma a esses conjuntos sejam usados em *cross-validation*. Foi determinado que seja realizada a geração de conjuntos de treino e teste 5 vezes, que as amostras de cada classe devem ser embaralhadas antes de serem divididas em lotes,e que fosse possível obter uma saída reproduzível em várias chamadas de *StratifiedKFold*.

Para realizar a busca da melhor combinação de parâmetros, foi utilizado o método *RandomizedSearchCV*, passando para o mesmo a *pipeline* criada, o conjunto de parâmetros e seus possíveis valores, a quantidade máxima de amostras de combinações de parâmetros/ valores

que deveria ser testada, de forma a não testar todas as combinações possíveis, o que demandaria um tempo mais longo, foi determinado que a busca pela melhor combinação deveria levar em conta a obtenção do melhor *score f1*, e o *StratifiedKFold* previamente configurado. Também foi determinado que não fossem exibidas mensagens em tela do andamento do processo de busca das melhores combinações de parâmetros, pois devido a ser um processo longo e que estava gerando uma quantidade muito alta de mensagens, estava ocorrendo o travamento da janela interativa de exibição.

Foi realizado em seguida o treinamento do modelo, e obtidos os resultados contendo as melhores pontuações de desempenho e a melhor combinação de parâmetros encontrada. Então, foi realizada a avaliação individual do modelo em relação ao *dataframe* de testes, para validar o desempenho do modelo em relação a dados aos quais o mesmo não tinha tido acesso anteriormente, ou seja, "perguntas", conjuntos de *features* para os quais o mesmo ainda não possuía "respostas", ou a qual classe cada conjunto de *features* pertencia. O resultado das predições foi utilizado para gerar um relatório de classificações, comparando o resultado das predições com as classes corretas para cada *feature*.

Em seguida, foi calculada a *ROC AUC curve* e exibida a mesma como um gráfico. Para isso, foi utilizada a função "*predict_proba*", que retorna as probabilidades previstas para cada classe do modelo. Essas probabilidades indicam a probabilidade estimada de cada classe ser a classe correta para cada amostra no conjunto de teste. Calculou-se também a *confusion matrix* e exibiu-se a mesma como um gráfico.

Por último, foi definido um caminho de armazenagem de arquivo no sistema de arquivos local, e persistidas em um arquivo chamado *results_tunning_randomforestclassifier.txt* as seguintes informações: a melhor combinação encontrada de parâmetros do modelo, a melhor pontuação de desempenho do modelo identificada, o relatório de classificação, a pontuação de desempenho *ROC AUC*, uma matriz numérica correspondente aos valores da *confusion matrix*, e o tempo de execução do processo em segundos, desde o início até o ponto antes de começar o processo de configuração e escrita do arquivo.

3.6.1.2 Logistic Regression

Buscou-se tentar identificar os melhores parâmetros possíveis para o modelo *Logistic Regression*. O *link* de acesso ao código fonte está disponível no Apêndice A, na subseção "Código-fonte 4 – tunning_logisticregression.py". As tarefas realizadas por ele são semelhantes

às realizadas no código-fonte de *tunning* do modelo *Random Forest Classifier*, exceto pelos pontos relacionados abaixo:

1. É disponibilizado um conjunto diferente de parâmetros e seus possíveis valores a serem utilizados no *RandomizedSearchCV*. Se trata de um conjunto de parâmetros apropriados para o modelo *Logistic Regression*, tendo em vista que diferentes modelos possuem tipos de parâmetros diferentes.
2. O modelo instanciado na pipeline foi o *Logistic Regression*.
3. Os resultados foram persistidos em um arquivo chamado *results_tunning_logisticregression.txt*.

3.6.1.3 KNeighbors Classifier

Buscou-se tentar identificar os melhores parâmetros possíveis para o modelo *KNeighbors Classifier*. O *link* de acesso ao código fonte está disponível no Apêndice A, na subseção "Código-fonte 5 – tunning_kneighborsclassifier.py". As tarefas realizadas por ele são semelhantes às realizadas no código-fonte de tunning do modelo Logistic Regression, exceto pelos pontos relacionados abaixo:

1. É disponibilizado um conjunto diferente de parâmetros e seus possíveis valores a serem utilizados no *RandomizedSearchCV*. Se trata de um conjunto de parâmetros apropriados para o modelo *KNeighbors Classifier*, tendo em vista que diferentes modelos possuem tipos de parâmetros diferentes.
2. O modelo instanciado na pipeline foi o *KNeighbors Classifier*
3. Os resultados foram persistidos em um arquivo chamado *results_tunning_kneighborsclassifier.txt*.

3.6.2 Comparação dos modelos

Utilizando os resultados do processo anterior de busca da combinação ideal de hiperparâmetros para os modelos, foi desenvolvido um código fonte para realizar a comparação de desempenho entre todos os modelos, de forma a identificar qual deles obteria pontuações maiores de *f1 score* e *ROC AUC curve*. O *link* de acesso ao código citado encontra-se no Apêndice A, na subseção "Código-fonte 6 – model_comparison.py". Foi realizada a importação de bibliotecas e principais métodos e inicializado um contador em segundos, de forma a monitorar em quanto tempo se daria o processo de *tunning* e exibição dos resultados.

Em seguida, foi realizada a leitura do *dataset* de treino armazenado no arquivo "train.csv"e realizada a preparação dos dados, obtendo como resultado as *features*, *targets*

(classes alvo) codificados em valores numéricos e a relação entre nome das classes e seus valores codificados. Para a realização das tarefas acima, houve a conversão do *dataset* em um *dataframe*, utilizando a biblioteca Pandas. Foi realizada a codificação da coluna *University* para valores numéricos, pois os modelos compreendem somente valores numéricos, e a coluna *University*, assim como outras colunas do *dataframe*, possui dados em texto. Foi gerado um *dataframe* contendo a coluna *University* codificada, para posteriormente uni-lo ao *dataframe* final. Em seguida, foi gerado também um *dataframe* onde foram unidas em uma só coluna as colunas "*Course_Name*", "*Course_Description*" e "*Skills*".

Para determinar quais termos eram mais frequentes na nova coluna gerada, foi utilizada a conversão da coleção de documentos de texto da coluna do novo *dataframe* para uma matriz numérica de contagem de *tokens*, contendo somente a ocorrência dos termos mais frequentes. A quantidade máxima de termos mais frequentes foi determinada através de diversas testagens manuais com vários valores diferentes até obter o que melhor produzisse resultados satisfatórios das métricas dos modelos. Além da contagem de termos, também foi considerado combinações de termos únicos e de dois termos. Também foi realizado o processo de colocar em minúsculas todas as letras dos termos. Essas operações foram realizadas com o intuito de melhorar o desempenho dos modelos.

A matriz numérica resultante foi convertida em um *dataframe* e seus nomes de colunas foram convertidos para texto, de forma a evitar o erro de tempo de execução que informa que nomes de colunas precisam ser do tipo texto, e não inteiros. Foi então realizada a união dos *dataframes* contendo a coluna *University* e a matriz numérica em um único *dataframe*, em seguida codificada a coluna com as classes para valores numéricos e adicionada a mesma em outro *dataframe*. O resultado, como já mencionado anteriormente, foram três conjuntos essenciais para o funcionamento dos modelos e do cálculo de métricas: Um *dataset* com as *features*, um *dataset* com as classes, e uma variável contendo a relação entre quais classes se referem a quais valores codificados. O mesmo processo descrito nos passos anteriores foi aplicado ao dataset de testes armazenado em *"test.csv"*.

Foram definidos os modelos, cada um deles tendo atribuídos a seus parâmetros os valores obtidos durante o processo de *tunning* realizados anteriormente, valores esses que correspondem à melhor pontuação de desempenho alcançada. Os modelos definidos foram *Random Forest Classifier*, *Logistic Regression* e *KNeighbors Classifier*.

Foi então utilizado o método *StratifiedKFold*, com as mesmas configurações utiliza-

das nos processos de *tunning* dos modelos. Em seguida, foi realizado o seguinte processo para cada modelo, de forma sequencial: exibição do nome do modelo e do conjunto de parâmetros do mesmo, criação de uma pipeline com os passos de aplicação de técnica *SMOTE*, aplicação de *scaling* no *dataset*, instanciação do modelo, treinamento do modelo utilizando o *dataframe* de treino, realização de predições para cada amostra do *dataframe*, cálculo do *f1 score* sem a utilização de *cross-validation*, utilizando o parâmetro *average=weighted*, que determina que o *f1 score* seja calculado considerando o cálculo de métricas para cada classe, encontrando a média ponderada das classes pelo suporte (número de instâncias verdadeiras para cada classe). Dessa forma, é levado em consideração o desbalanceamento entre as classes do *dataframe* (presença no *dataframe* de quantidade de amostras diferentes para cada classe). Foi em seguida calculado o *f1 score* utilizando *cross-validation*, passando como parâmetros para a função *cross_val_score*() a *pipeline* com o modelo, o dataframe de treino, o *StratifiedKFold* configurado anteriormente e o *scorer* configurado. A seguir, é então computada a *ROC AUC curve* e utilizadas as funções de visualização de dados da biblioteca *Matplotlib* para configurar um gráfico contendo a *ROC AUC curve* para o modelo. É então calculada a *confusion matrix*. Após os passos anteriores serem finalizados para cada modelo, é exibido um gráfico com a *ROC AUC curve* e outro com a *confusion matrix*, para cada modelo.

3.6.3 Coleta de dados

Foram executados os códigos individuais de *hyperparameter tunning* para os modelos *Random Forest Classifier*, *Logistic Regression* e *KNeighbors Classifer*. Foi codificada solução para armazenagem automática dos resultados em arquivo de texto, contendo as informações abaixo:

- Melhor combinação de parâmetros.
- Melhor pontuação de desempenho *f1 score*.
- Relatório de classificação com pontuações de desempenho por classe e pontuação geral (para todas as classes).
- Melhor pontuação de desempenho para *ROC AUC* score.
- *Confusion matrix*.
- Objeto *JSON cv_results* contendo, para cada parâmetro, um *array* correspondente, e em cada elemento do *array*, a pontuação que o parâmetro obteve em determinada combinação durante a combinação de parâmetros automatizada do *hyperparameter tunning*.

Foi executado o código de comparação de modelos e salvo o notebook da janela interativa do *Jupyter notebooks* para o arquivo "*model_comparison.ipynb*".

3.7 *Dataset* de validação

Para realização da validação do processo, foi utilizado o *dataset 20newsgroups*, disponível na biblioteca do *Scikit Learn*. O mesmo possui um conjunto de e-mails, com categorias correspondentes. O link para o código-fonte citado encontra-se no Apêndice A, na subseção "Código-fonte 7 – Scikit version of dataset_split_dataclean.py". Foi realizado o seguinte processo para tornar o *dataset* o mais semelhante possível ao *dataset* da *Coursera*:

Importação de bibliotecas, criação de uma lista de três categorias (o *dataset* possui mais), para equivaler às três classes do *dataset* da Coursera, *Advanced*, *Intermediary e Beginner* (as categorias escolhidas foram "*rec.motorcicles*", "*rec.autos*"e "*rec.sport.baseball*"), são extraídas da coluna *text* a linha que possui a *organization* (equivalente à coluna *University* da *Coursera*) e a linha que possui o *subject* (equivalente à coluna *Course_Name* da *Coursera*), e retornados três conteúdos: a *organization*, o *subject* e o texto original de onde as duas primeiras colunas foram removidas. É gerado um *dataset* combinado, contendo os dados de treino e teste. São removidas as amostras que possuam valores vazios em alguma das colunas. É removido o caractere especial "(aspas duplas) das colunas São removidas todas as colunas que possuírem apenas dados numéricos. É gerado o gráfico com o balanceamento de classes. São calculados os dados referentes ao estado de balanceamento do *dataset*. O *dataset* combinado é dividido em 60% treino e 40% teste. Por último, os *datasets* de treino e teste são armazenados nos arquivos .csv *train_SCIKIT_DATA.csv* e *test_SCIKIT_DATA.csv*.

3.7.1 Hyperparameter tunning e comparação de modelos

É utilizada a mesma estrutura de código-fonte que foi utilizada no *dataset Coursera* para o *hyperparameter tunning* e para comparação entre modelos. O único ponto de mudança são os locais e arquivos que são carregados, contendo os *datasets* de treino e teste. O *link* de acesso ao código-fonte para o modelo *RandomForestClassifer* se encontra no Apêndice A, na subseção "Código-fonte 8 – Scikit version of tunning_randomforestclassifier.py". O *link* de acesso ao código-fonte para o modelo *Logistic Regression* se encontra no Apêndice A, na subseção "Código-fonte 9 – Scikit version of tunning_logisticregression.py". O *link* de acesso

ao código-fonte para o modelo *KNeighbors Classifier* se encontra no Apêndice A, na subseção "Código-fonte 10 – Scikit version of tunning_kneighborsclassifier.py". Por último, o *link* de acesso ao código-fonte para a comparação entre modelos se encontra no Apêndice A, na subseção "Código-fonte 11 – Scikit version of model_comparison.py".

Como pôde ser visto, neste capítulo se percorreu o processo utilizado na busca de atingir os objetivos previamente estabelecidos. No próximo capítulo, serão apresentados os eventos que levaram à decisão de realizar este trabalho, assim como os resultados obtidos através da aplicação do processo descrito no presente capítulo.

4 RELATO HISTÓRICO E RESULTADOS

Nesta seção, discorre-se acerca dos eventos que levaram a decisão de realizar este trabalho, assim como acerca dos resultados obtidos ao longo do processo descrito no capítulo 3 (Metodologia).

4.1 Relato histórico

Durante a disciplina de Projeto Integrado II, disponibilizada no semestre 2 no ano de 2022 no curso de graduação em Sistemas e Mídias Digitais da Universidade Federal do Ceará, foi concebida e implementada uma solução voltada à indicação de trilhas de migração para profissionais de tecnologia que desejassem mudar de país e trilhas de migração para profissionais que desejassem mudar de área profissional da área que atuam para a área de tecnologia da informação.

Na disciplina foi desenvolvido o *back-end* da aplicação e parte do *frond-end*, assim como o conceito. Para a disciplina de Trabalho de Conclusão de Curso (TCC), se objetivava prosseguir com o desenvolvimento da aplicação como produto, ajustando a mesma para disponibilizar conteúdo de formação (cursos, indicação de canais de *Discord* para ambientação e treinamento de lingua estrangeira, etc).

Ao se debater a viabilidade do projeto, deu-se conta de que devido ao tempo reduzido disponível na disciplina de TCC não seria possível alcançar o resultado almejado, pois uma plataforma desse tipo envolveria o desenvolvimento e testagem de diversas funcionalidades com um certo grau de complexidade, como por exemplo a implementação do uso de algoritmos de recomendação.

Assim sendo, surgiu a ideia de se focar nesse aspecto específico dos algoritmos de recomendação, identificando quais algoritmos poderiam ser mais adequados para a tarefa específica de classificação de cursos de acordo com seu nível de dificuldade. Passou-se à implementação, buscando identificar conteúdos e coletar conhecimento que torna-se viável a concretização dos objetivos estabelecidos para o projeto.

Durante a busca, foi sugerido pelo orientador o uso da biblioteca *Scikit Learn*, pois a mesma apresentava documentação extensa, ampla variedade de modelos de machine *learning* e funcionalidades para a implementação do produto almejado neste projeto. A sugestão foi acatada, e passou-se a utilizar a biblioteca, focando principalmente nos algoritmos identificados na

documentação como inerentemente multiclasse, ou seja, algoritmos que tem como característica serem adequados para tarefas de classificação onde há várias classes possíveis e uma delas precisa ser escolhida como a correta de acordo com um determinado conteúdo disponibilizado ao algoritmo.

4.2 Resultados - *Dataset* Coursera

Por meio da coleta de dados realizada através da automação da exibição de informações em gráficos (*confusion matrix* e *ROC AUC score*: 0.71248351173282), assim como o armazenamento em arquivo de texto dos resultados referentes ao processo de *hyperparameter tunning* dos modelos, foi possível gerar subsídios para uma análise do processo aplicado para a construção do produto.

4.2.1 Hyperparameter tunning

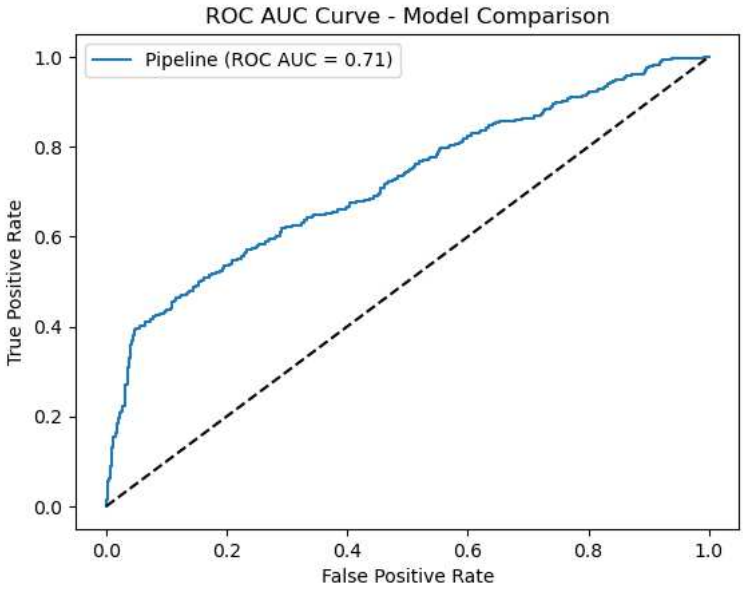
Seguem os resultados obtidos do processo de *hyperparameter tunning* para o modelo *RandomForestClassifer*: Melhor combinação de parâmetros: *n_jobs: -1, n_estimators: 420, min_samples_split: 3, min_samples_leaf: 1, max_features: "log2", max_depth: 13, criterion: "log_loss", class_weight: "balanced", ccp_alpha: 0.0.*. Melhor *f1-score*: 0.6563775170076722. O relatório de classificação pode ser observado na Tabela 3. O gráfico com a ROC AUC curve obtida pode ser observado na Figura 5. A *confusion matrix* obtida pode ser observada na Figura 6.

Tabela 3 – *Dataset* Coursera - *Hyperparameter tunning* do modelo *Random Forest Classifier: Classification Report*

Class	Precision	Recall	F1-Score	Support
0	0.50	0.52	0.51	355
1	0.69	0.53	0.60	522
2	0.38	0.52	0.44	295
Accuracy			0.52	1172
Macro Avg	0.52	0.52	0.52	1172
Weighted Avg	0.56	0.52	0.53	1172

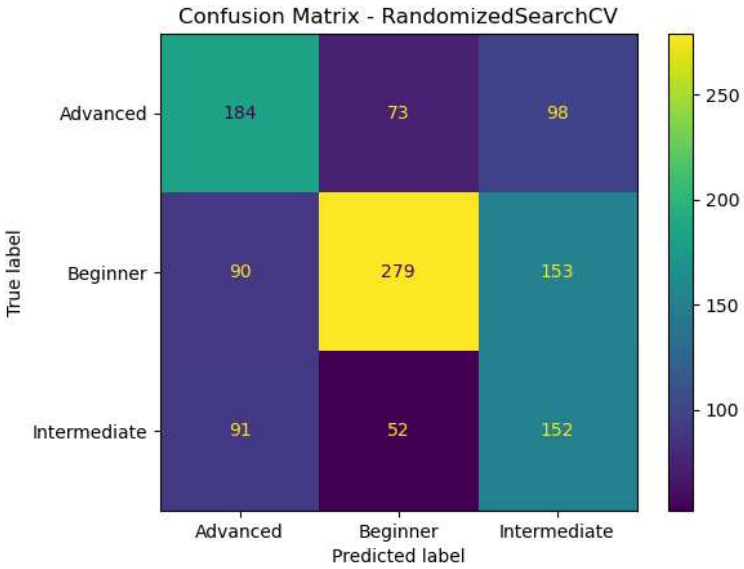
Fonte: elaborada pelo autor.

Figura 5 – *Dataset* Coursera - *Hyperparameter tunning* do modelo *Random Forest Classifier: ROC AUC curve*



Fonte: elaborada pelo autor.

Figura 6 – *Dataset* Coursera - *Hyperparameter tunning* do modelo *Random Forest Classifier: Confusion matrix*



Fonte: elaborada pelo autor.

Seguem os resultados obtidos do processo de *hyperparameter tunning* para o modelo *Logistic Regression*: melhor combinação de parâmetros: *solver: "lbfgs", penalty: l2, n_jobs: -1, multi_class: "ovr", max_iter: 120, fit_intercept: True, class_weight: "balanced", C: 0.01*. O melhor f1-score: 0.5220648604167668. O relatório de classificação pode ser observado na Tabela 4. O gráfico com a ROC AUC curve obtida pode ser observado na Figura 7. A *confusion*

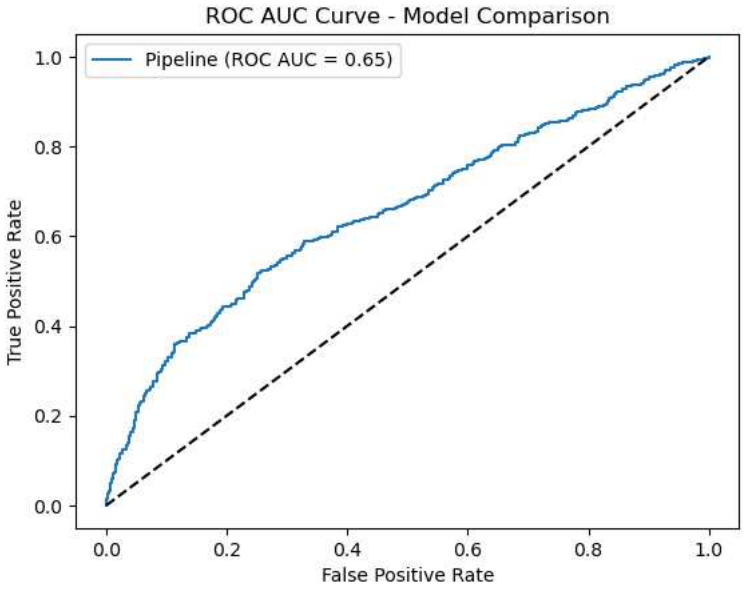
matrix obtida pode ser observada na Figura 8.

Tabela 4 – *Dataset* Coursera - *Hyperparameter tunning* do modelo *Logistic Regression: Classification Report*

Class	Precision	Recall	F1-Score	Support
0	0.45	0.38	0.41	355
1	0.60	0.54	0.57	522
2	0.36	0.50	0.42	295
Accuracy			0.48	1172
Macro Avg	0.47	0.47	0.47	1172
Weighted Avg	0.49	0.48	0.48	1172

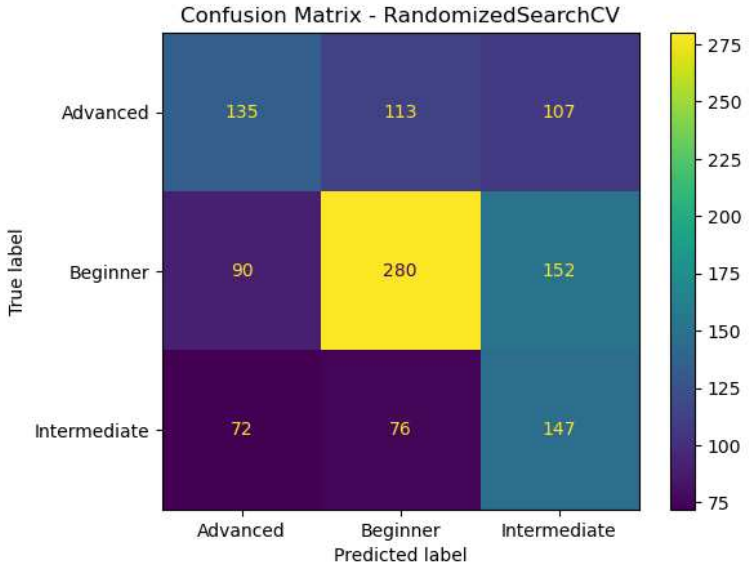
Fonte: elaborada pelo autor.

Figura 7 – *Dataset* Coursera - *Hyperparameter tunning* do modelo *Logistic Regression: ROC AUC curve*



Fonte: elaborada pelo autor.

Figura 8 – *Dataset* Coursera - *Hyperparameter tunning* do modelo *Logistic Regression: Confusion matrix*



Fonte: elaborada pelo autor.

Seguem os resultados obtidos do processo de *hyperparameter tunning* para o modelo *KNeighbors Classifier*. A melhor combinação de parâmetros: *weights: "distance", p: 2, n_neighbors: 2, n_jobs: -1, metric: "manhattan", leaf_size: 34, algorithm: "auto"*. O melhor f1-score: 0.6694382511189548. O relatório de classificação segue anexado na

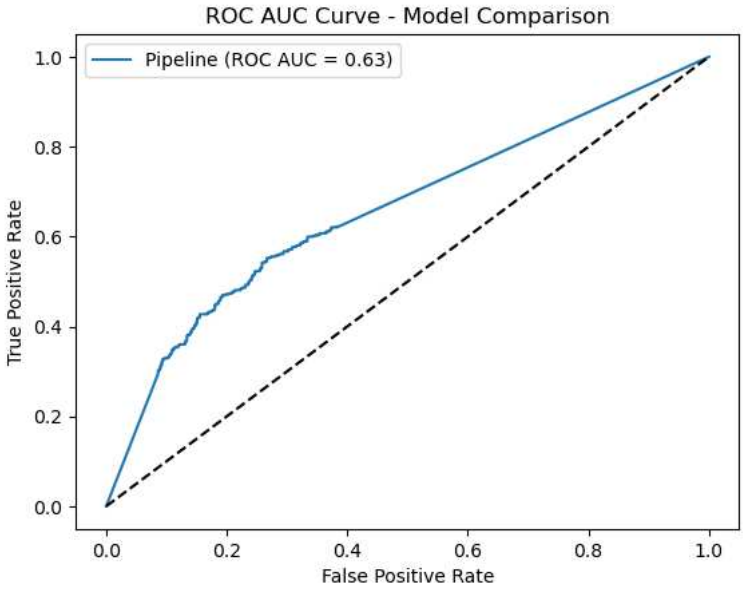
O relatório de classificação pode ser observado na Tabela 5. O gráfico com a ROC AUC curve obtida pode ser observado na Figura 9. A *confusion matrix* obtida pode ser observada na Figura 10.

Tabela 5 – *Dataset* Coursera - *Hyperparameter tunning* do modelo *KNeighbors Classifier: Classification Report*

Class	Precision	Recall	F1-Score	Support
0	0.42	0.41	0.41	355
1	0.63	0.49	0.55	522
2	0.33	0.47	0.39	295
Accuracy			0.46	1172
Macro Avg	0.46	0.46	0.45	1172
Weighted Avg	0.49	0.46	0.47	1172

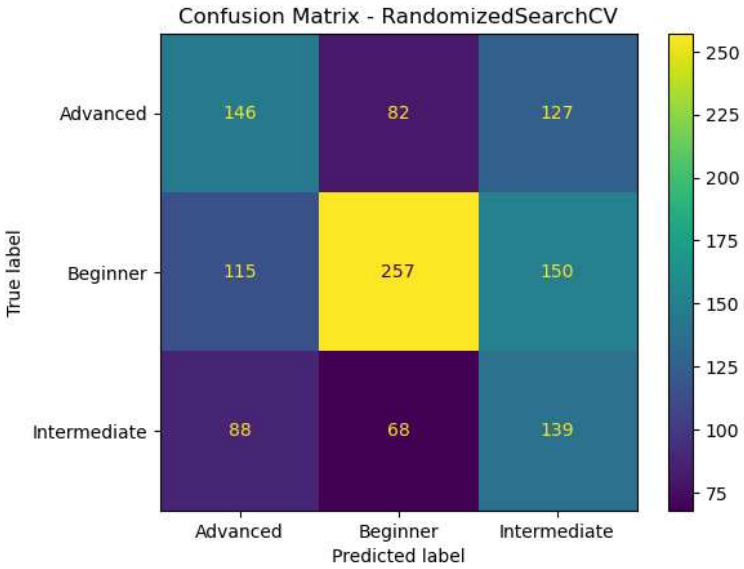
Fonte: elaborada pelo autor.

Figura 9 – *Dataset* Coursera - *Hyperparameter tunning* do modelo *KNeighbors Classifier: ROC AUC curve*



Fonte: elaborada pelo autor.

Figura 10 – *Dataset* Coursera - *Hyperparameter tunning* do modelo *KNeighbors Classifier: Confusion matrix*



Fonte: elaborada pelo autor.

Pode-se concluir das diversas testagens e processo de desenvolvimento, além dos dados obtidos que o processo de limpeza dos dados demonstrou ser útil para melhorar o desempenho dos modelos. A remoção de amostras com dados vazios evitou que houvesse amostras inconsistentes nos conjuntos de treino e teste. Os modelos compreendem dados numéricos, por isso foi utilizada a técnica de vetorização para corrigir erros relacionados que ocorriam em de tempo de

execução durante o treino do modelo, como no exemplo de erro "*RandomForestClassfier.fit(): ValueError: could not convert string to float*".

Essa mensagem de erro informa que o modelo recebeu um *dataframe* contendo dados em formato texto, e que a conversão para valores de texto não foi possível durante o tempo de execução do modelo. É um erro crítico que interrompe o processo de treinamento e interrompe a execução do *Jupyter notebook*.

O processo de *hyperparameter tuning* demonstrou ser eficiente para encontrar uma combinação de parâmetros adequada. Os modelos apresentaram pontuações de desempenho muito inferiores quando da utilização dos seus parâmetros padrão. Com o processo de *tunning*, foi possível obter um aumento no desempenho do modelo *Random Forest Classifier* até o limiar desejável, acima de 50% f1 score e acima de 70% *ROC AUC curve*.

A validação do desempenho dos modelos utilizando um *dataset* de testes se mostrou uma estratégia adequada, pois dessa forma foi possível identificar se o modelos apresentariam um bom desempenho ao classificar amostras as quais não conheciam previamente a quais classes pertenciam, evitando *data leakage*.

Sobre a questão de *data leakage*, foi uma preocupação muito presente ao longo do trabalho, pois representava um grande risco de enviesar os resultados, obtendo pontuações irreais. Para isso se utilizou a divisão dos *datasets* de treino e teste em arquivos separados, se utilizou *SMOTE* e se utilizou *scaling*.

A estratégia de separação do *dataset* principal em dois arquivos individuais contendo respectivamente as amostras de treino e de teste se mostrou uma estratégia adequada, pois reduziu o risco de ocorrer um *splitting* dos *datasets* em tempo de execução diferente para cada modelo durante o *tuning*, e posteriormente durante a comparação de modelos, levando a uma contaminação dos resultados coletados.

4.2.2 Comparação de modelos

O modelo *RandomForestClassifer* apresentou as melhores pontuações de desempenho. Isso se deve ao processo de identificação dos melhores parâmetros para o mesmo, assim como os processos de limpeza de dados e vetorização das *features*. Acerca dos resultados obtidos com o modelo *Random Forest Classifier*, algumas considerações:

Foi observado que a classe 1 (*Advanced*) tem a maior precisão (0.69), indicando que, das instâncias classificadas como classe 1, 69% delas são realmente da classe 1. As

classes 0 (*Beginner*) e 2 (*Intermediate*) têm precisões menores, de 0.50 e 0.38, respectivamente. Foi observado que a maior quantidade de classificações incorretas para a classe *Begginer* foi a classificando como *Intermediate*. Considerando a estrutura do *dataset* e o fato de que comumente cursos de nível iniciante (*Begginer*) compartilham uma exigência de habilidades mais semelhantes a cursos intermediários (*Intermediate*) do que do conjunto de habilidades cursos avançados (*Advanced*), pode-se levantar a hipótese de que devido a essa possível similaridade entre termos utilizados na coluna *Skills* das amostras pertencentes a essas duas classes, tenha havido uma maior quantidade de predições incorretas.

Quanto aos modelos *Logistic Regression* e *KNeighbors Classifier*, devido a não terem alcançado os requisitos necessários para esse trabalho, não se mostraram adequados para o uso posterior em uma plataforma de classificação de cursos online, sendo assim descartados.

Que a corretude dos scores obtidos durante o tunning se confirmaram pela realização de outra avaliação independente utilizando somente os parâmetros obtidos durante o *tunning*. Essa avaliação foi feita utilizando a comparação de modelos, obtendo-se e os seguintes resultados:

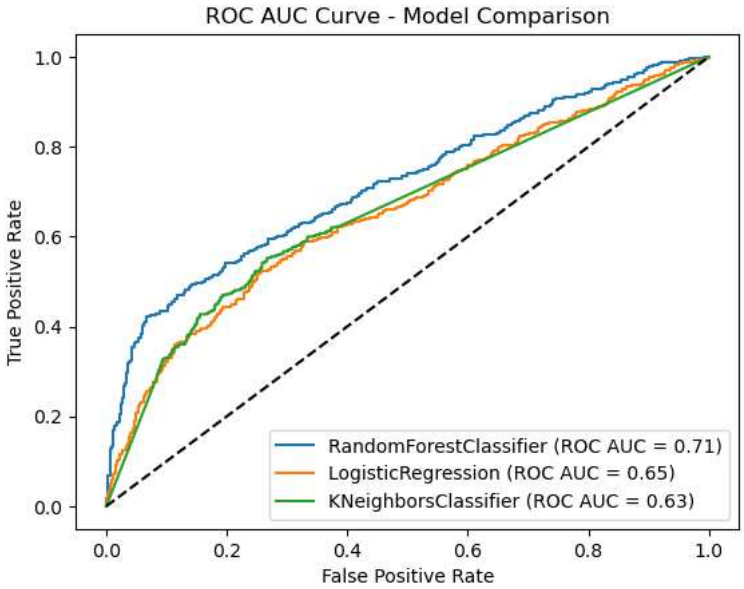
Para o modelo Random Forest Classifier, foi obtido o f1 score de 0.533212477959957, a *accuracy* de 0.53 com um desvio padrão de 0.03 e o *ROC AUC score* de 0.7108220830394365.

Para o modelo Logistic Regression, foi obtido o *f1 score* de 0.4826853480435417, a accuracy de 0.47 com um desvio padrão de 0.01 e o *ROC AUC score* de 0.6496855523757935.

Para o modelo *KNeighbors Classifier* , foi obtido o f1 score de 0.4704801036940513, a accuracy de 0.48 com um desvio padrão de 0.02 e o ROC AUC score de 0.6340839751952095.

Na Figura 11, pode ser observado o gráfico comparativo entre os três modelos avaliados, assim como suas pontuações de desempenho *ROC AUC curve*.

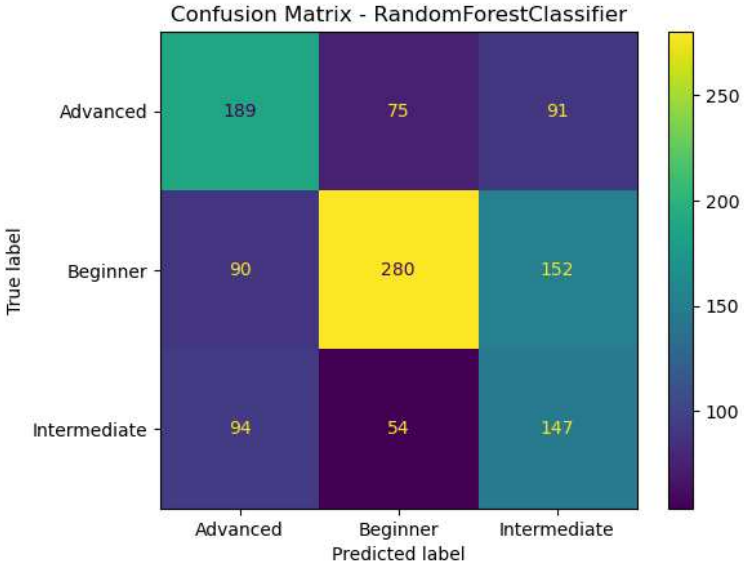
Figura 11 – *Dataset* Coursera - Comparação de modelos: *ROC AUC curve*



Fonte: elaborada pelo autor.

Na Figura 12, pode ser observado a *confusion matrix* referente ao modelo *Random Forest Classifier*.

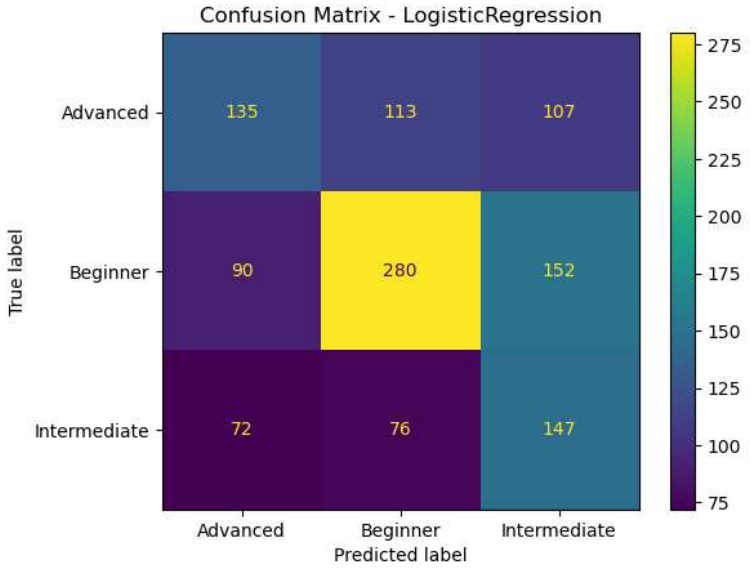
Figura 12 – *Dataset* Coursera - Comparação de modelos, *Random Forest Classifier: Confusion matrix*



Fonte: elaborada pelo autor.

Na Figura 13, pode ser observado a *confusion matrix* referente ao modelo *Logistic Regression*.

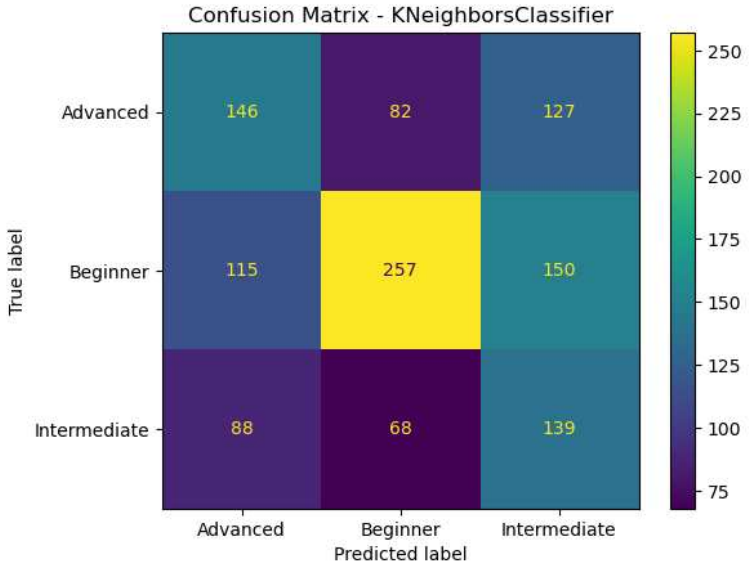
Figura 13 – *Dataset* Coursera - Comparação de modelos, *Logistic Regression: Confusion matrix*



Fonte: elaborada pelo autor.

Na Figura 14, pode ser observado a *confusion matrix* referente ao modelo *KNeighbors Classifier*.

Figura 14 – *Dataset* Coursera - Comparação de modelos, *KNeighbors Classifier: Confusion matrix*



Fonte: elaborada pelo autor.

Acerca desses dados, é possível observar que embora tenha havido uma variação na pontuação *f1 score* se comparado à pontuação obtida durante o processo de *hyperparameter tunning*, a pontuação de *ROC AUC curve* e a configuração da *confusion matrix* se mantém semelhante. Isso demonstra a consistência do processo.

4.3 Resultados - *Dataset* de validação

Acerca do *dataset* de validação pertencente à biblioteca do *Scikit Learn*, foram obtidos os seguintes dados:

4.3.1 Hyperparameter tunning

Os resultados obtidos do processo de *hyperparameter tunning* para o modelo *RandomForestClassifer*: a melhor combinação de parâmetros: *n_jobs: -1, n_estimators: 420, min_samples_split: 3, min_samples_leaf: 1, max_features: log2, max_depth: 13, criterion: log_loss, class_weight: balanced, ccp_alpha: 0.0*. O melhor f1-score: 0.7013594704503539. O relatório de classificação encontra-se disponível na Tabela 6.

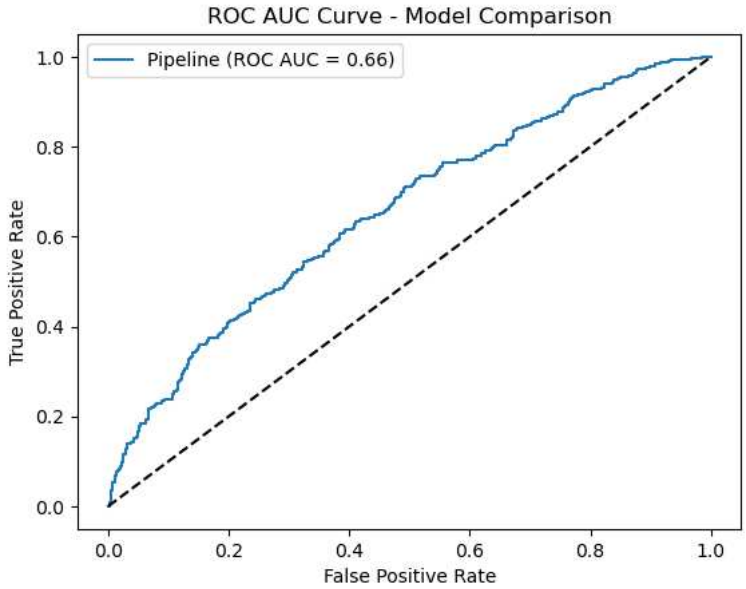
Tabela 6 – *Dataset* de validação - *Hyperparameter tunning* do modelo *Random Forest Classifier: Classification Report*

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.37	0.56	0.45	389
1	0.44	0.45	0.44	393
2	0.64	0.30	0.41	394
Accuracy			0.44	
Macro Avg	0.48	0.44	0.43	1176
Weighted Avg	0.48	0.44	0.43	1176

Fonte: elaborada pelo autor.

Na Figura 15, pode ser observado o gráfico da *ROC AUC curve* referente ao modelo *Random Forest Classifier*.

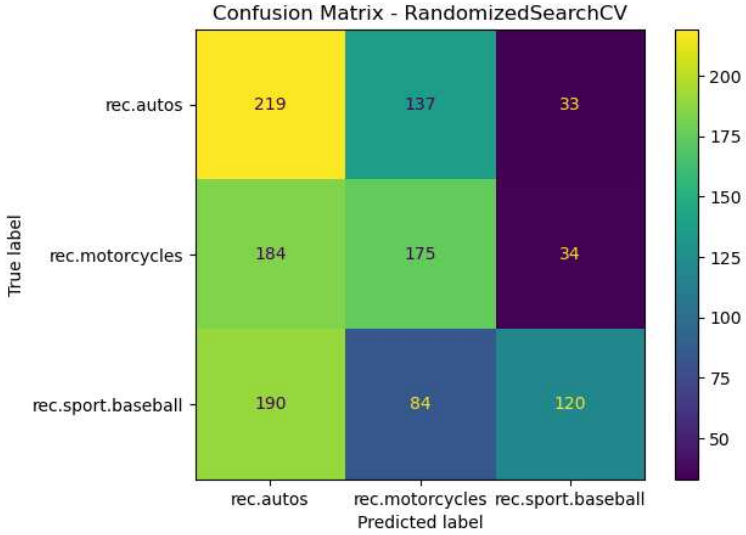
Figura 15 – *Dataset* de validação - *Hyperparameter tunning* do modelo *Random Forest Classifier: ROC AUC curve*



Fonte: elaborada pelo autor.

Na Figura 16, pode ser observada a *confusion matrix* referente ao modelo *Random Forest Classifier*.

Figura 16 – *Dataset* de validação - *Hyperparameter tunning* do modelo *Random Forest Classifier: Confusion matrix*



Fonte: elaborada pelo autor.

O resultados obtidos do processo de *hyperparameter tunning* para o modelo *Logistic Regression*: a melhor combinação de parâmetros: solver: *lbfgs, penalty: l2, n_jobs: -1, multi_class: ovr, max_iter: 120, fit_intercept: True, class_weight: balanced, C: 10*. O melhor f1-score: 0.6722292113917094. O relatório de classificação encontra-se disponível na Tabela 7.

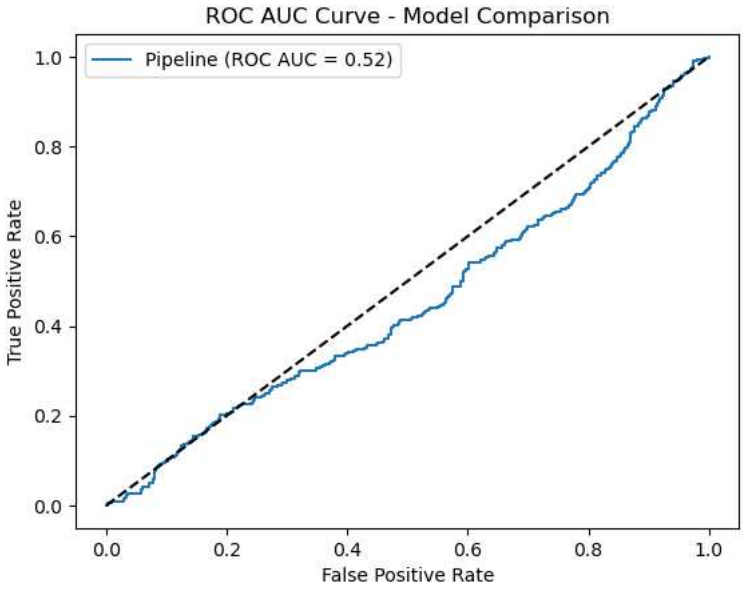
Tabela 7 – *Dataset* de validação - *Hyperparameter tunning* do modelo *Logistic Regression: Classification Report*

Class	Precision	Recall	F1-Score	Support
0	0.34	0.41	0.37	396
1	0.31	0.30	0.31	398
2	0.50	0.40	0.44	398
Accuracy			0.37	
Macro Avg	0.38	0.37	0.37	1192
Weighted Avg	0.38	0.37	0.37	1192

Fonte: elaborada pelo autor.

Na Figura 17, pode ser observado o gráfico da *ROC AUC curve* referente ao modelo *Logistic Regression*.

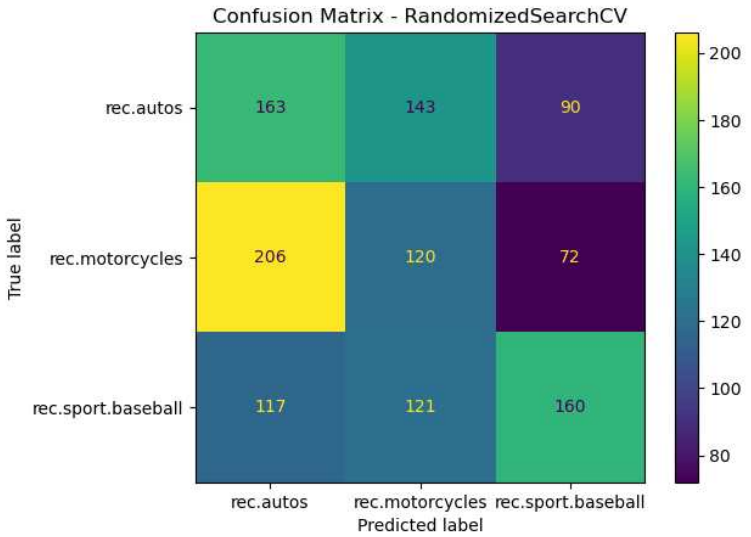
Figura 17 – *Dataset* de validação - *Hyperparameter tunning* do modelo *Logistic Regression: ROC AUC curve*



Fonte: elaborada pelo autor.

Na Figura 18, pode ser observada a *confusion matrix* referente ao modelo *Logistic Regression*.

Figura 18 – *Dataset* de validação - *Hyperparameter tunning* do modelo *Logistic Regression: Confusion matrix*



Fonte: elaborada pelo autor.

Resultados obtidos do processo de *hyperparameter tunning* para o modelo *KNeighbors Classifier*: a melhor combinação de parâmetros: *weights: distance, p: 2, n_neighbors: 6, n_jobs: -1, metric: manhattan, leaf_size: 13, algorithm: auto*. O melhor *f1-score*: 0.573229761416023. O relatório de classificação encontra-se disponível na Tabela 8.

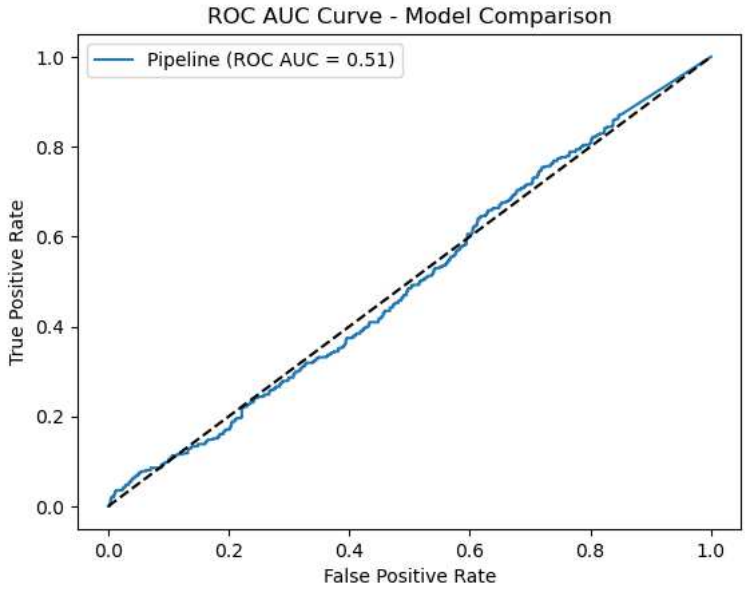
Tabela 8 – *Dataset* de validação - *Hyperparameter tunning* do modelo *KNeighbors Classifier: Classification Report*

Class	Precision	Recall	F1-Score	Support
0	0.39	0.20	0.26	396
1	0.31	0.23	0.26	398
2	0.34	0.61	0.44	398
Accuracy			0.34	
Macro Avg	0.35	0.34	0.32	1192
Weighted Avg	0.35	0.34	0.32	1192

Fonte: elaborada pelo autor.

Na Figura 19, pode ser observado o gráfico da *ROC AUC curve* referente ao modelo *KNeighbors Classifier*.

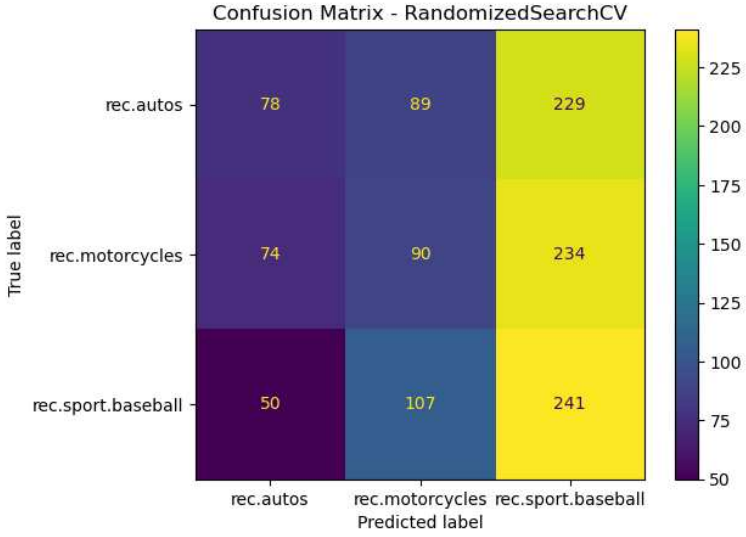
Figura 19 – *Dataset* de validação - *Hyperparameter tunning* do modelo *KNeighbors Classifier: ROC AUC curve*



Fonte: elaborada pelo autor.

Na Figura 20, pode ser observada a *confusion matrix* referente ao modelo *KNeighbors Classifier*.

Figura 20 – *Dataset* de validação - *Hyperparameter tunning* do modelo *KNeighbors Classifier: Confusion matrix*



Fonte: elaborada pelo autor.

4.3.2 Comparação de modelos

Segue resultados da comparação de modelos utilizando as combinações de parâmetros identificadas através do processo de *hyperparameter tunning*: *Random Forest Classifier*: *F1*

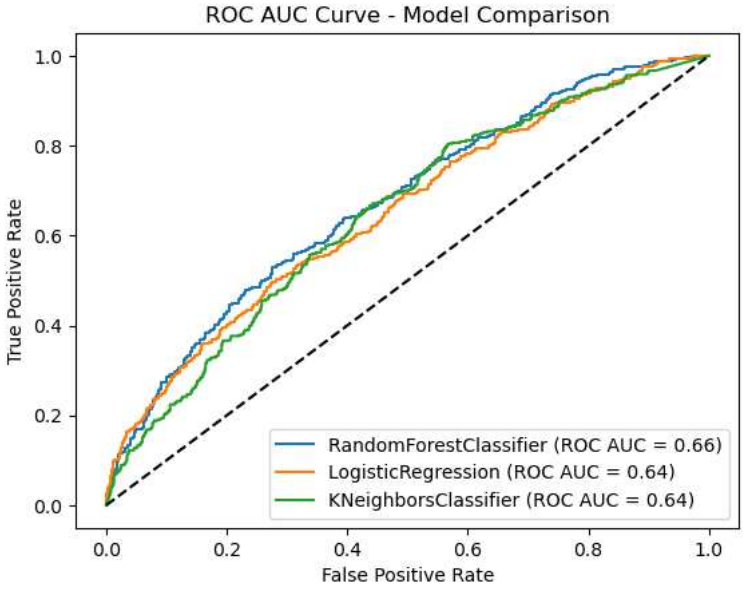
score: 0.4324657696102665 0.70 *accuracy with a standard deviation of* 0.03 *ROC AUC score*: 0.663115394363235

Logistic Regression: *F1 score*: 0.425721227561154 0.68 *accuracy with a standard deviation of* 0.02 ROC AUC score: 0.6367061543676801

KNeighbors Classifier: *F1 score*: 0.43219963524811045 0.56 *accuracy with a standard deviation of* 0.04 *ROC AUC score*: 0.6403657258351122

Na Figura 21, pode ser observado o gráfico comparativo entre os três modelos avaliados, assim como suas pontuações de desempenho *ROC AUC curve*.

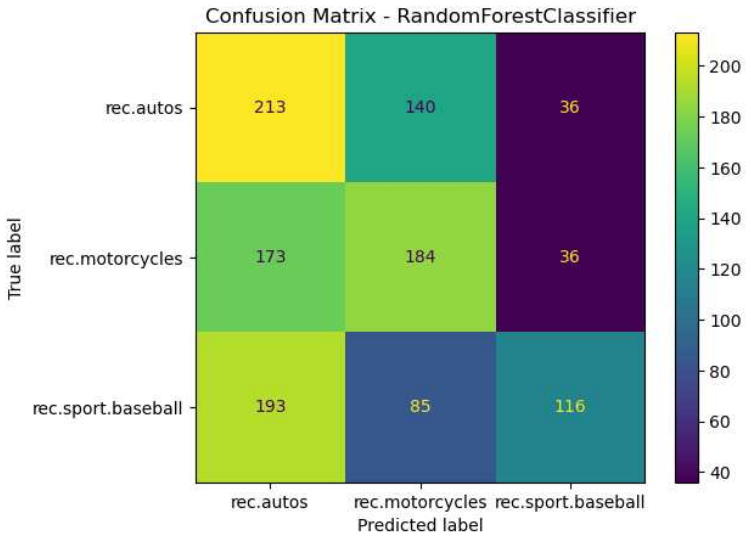
Figura 21 – *Dataset* de validação - Comparação de modelos: *ROC AUC curve*



Fonte: elaborada pelo autor.

Na Figura 22, pode ser observada a *confusion matrix* referente ao modelo *Random Forest Classifier* quando da comparação entre modelos.

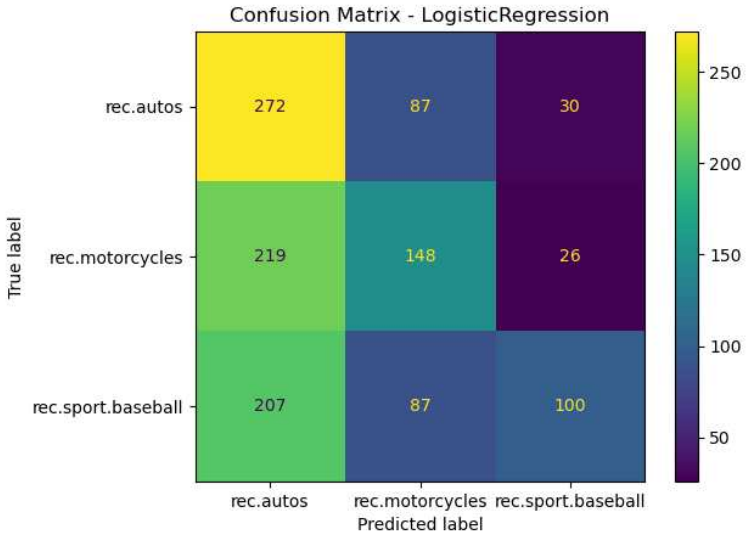
Figura 22 – *Dataset* de validação - Comparação de modelos, *Random Forest Classifier: Confusion matrix*



Fonte: elaborada pelo autor.

Na Figura 23, pode ser observada a *confusion matrix* referente ao modelo *Logistic Regression* quando da comparação entre modelos.

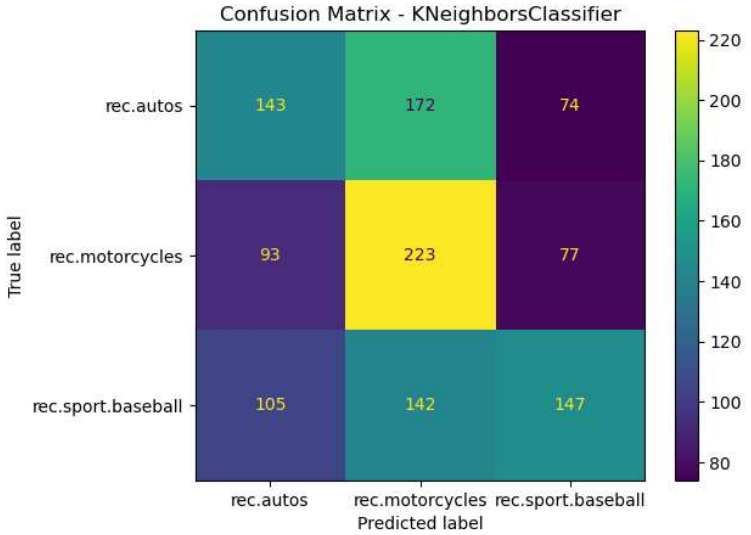
Figura 23 – *Dataset* de validação - Comparação de modelos, *Logistic Regression: Confusion matrix*



Fonte: elaborada pelo autor.

Na Figura 24, pode ser observada a *confusion matrix* referente ao modelo *KNeighbors Classifier* quando da comparação entre modelos.

Figura 24 – *Dataset* de validação - Comparação de modelos, *KNeighbors Classifier: Confusion matrix*



Fonte: elaborada pelo autor.

Acerca desses dados, é possível observar que embora a acurácia seja alta, a *ROC AUC curve* deixou a desejar. A hipótese levantada é de que o provável motivo seja o ruído nos dados da coluna *text*, pois os registros de comunicações de *e-mail* possuem muitos termos comuns à comunicação por esse canal. Ou seja, se tornou mais difícil para o modelo distinguir informações que pudessem realmente classificar corretamente em uma classe específica. Infelizmente, devido ao escassez de tempo, maiores aprofundamentos ficarão para trabalhos futuros que explorem a questão do ruído nos dados em *datasets*.

Neste capítulo, foram apresentados o relato histórico, assim como os resultados dos processos de *hyperparameter tunning* e comparação de modelos aplicados ao *dataset* Coursera e *dataset* de validação, e realizada análises acerca do processo e resultados. No próximo capítulo, Conclusão, serão apresentadas as conclusões alcançadas neste trabalho relacionando-as aos objetivos já estabelecidos no capítulo Introdução, assim como se falará acerca de possíveis trabalhos futuros que possam ter como base o presente trabalho.

5 CONCLUSÕES E TRABALHOS FUTUROS

Conclui-se que os objetivos estabelecidos para este trabalho foram satisfatoriamente alcançados, através da combinação de diferentes técnicas e à implementação de diversas adaptações estratégicas ao longo do processo de desenvolvimento.

Em relação aos objetivos deste trabalho, foi realizado com sucesso o *benchmark* de modelos de aprendizado de máquina para classificação de cursos online em níveis de dificuldade, através da construção de um algoritmo que coletou os dados referentes a métricas de desempenho estabelecidas. Acerca da identificação de um modelo de dados que pudesse ser utilizado para atingir os objetivos estabelecidos neste trabalho, foi identificado, através da busca nas bases de dados disponíveis online, que o conjunto de dados da Coursera poderia ser utilizado pois o mesmo possuía amostras com dados acerca dos níveis de habilidade (*skills*) classificadas de acordo com níveis de dificuldade de cursos (*Advanced*, *Intermediary* e *Begginer*).

Em relação ao objetivo de identificar para cada modelo a configuração que produz os melhores resultados de desempenho de acordo com as métricas escolhidas, o mesmo foi alcançado através da construção de um algoritmo que testou diversas combinações de hiperparâmetros, coletando dados que serviram para determinar qual a melhor combinação encontrada.

Para a identificação do modelo que melhor realiza a tarefa de fazer a classificação de cursos online de acordo com o nível de dificuldade, foi desenvolvido algoritmo que, utilizando as configurações de hiperparâmetros encontradas, realizou a comparação entre os diferentes modelos selecionados e coletou métricas de desempenho utilizadas para determinar que o modelo *Random Forest Classifier* demonstrou ser o mais adequado para a tarefa a que o produto se propõe.

Acerca da identificação de um conjunto de dados amplamente utilizado pela comunidade de aprendizado de máquina, através da realização de busca online, foi escolhido o *dataset 20newsgroups*, disponível na biblioteca do *Scikit Learn*. O mesmo foi utilizado para uma validação bem sucedida do processo aplicado ao conjunto de dados principal (o *dataset* do Coursera).

Embora não tenha sido possível obter pontuações de desempenho (*scores*) mais altos para o *dataset* de validação, o mesmo serviu como uma base para demonstrar que o processo utilizado pode ser aplicado em outros *datasets* com estrutura semelhante sem a ocorrência de falhas críticas, como por exemplo, interrupção da execução dos modelos, não obtenção de resultados válidos ou mesmo a completa impossibilidade de utilização do processo e execução

dos algoritmos implementados. É importante ressaltar que o *dataset* disponibilizado pelo *Scikit Learn* é um *dataset* balanceado, ou seja, nesse aspecto específico, a aplicação da técnica de *SMOTE* não proporcionou um benefício significativo para uma melhoria no desempenho dos modelos.

Esses resultados reforçam a confiabilidade do trabalho realizado, evidenciando a capacidade de implementar uma abordagem bem-sucedida em problemas semelhantes. Além disso, a experiência de desenvolvimento do presente trabalho ressalta a importância de selecionar cuidadosamente os modelos e técnicas apropriados para cada contexto, levando em consideração as particularidades dos conjuntos de dados, dos modelos e dos objetivos estabelecidos.

No futuro, pretende-se utilizar os resultados obtidos neste trabalho para o desenvolvimento de uma aplicação que realize a classificação de cursos *online* ou *MOOCs* de acordo com o nível de dificuldade. Os conhecimentos obtidos durante o processo descrito neste trabalho possuem potencial para contribuir significativamente com o desenvolvimento desse tipo de aplicação, pois ao longo do trabalho se experimentou o uso de diferentes técnicas, a combinação de diferentes abordagens, de forma a obter o melhor resultado possível, apesar das dificuldades encontradas.

REFERÊNCIAS

AGONÁCS, N.; MATOS, J. F. Os cursos on-line abertos e massivos (mooc) como ambientes heutagógicos. **Revista Brasileira de Estudos Pedagógicos**, SciELO Brasil, v. 101, p. 17–35, 2020.

Anaconda. **Anaconda documentation**. 2023. Disponível em: https://www.anaconda.com/open-source. Acesso em: 01 de junho 2023.

AZANK, F. Dados desbalanceados — o que são e como evitá-los. **Turing Talks**, 2022. Disponível em: https://medium.com/turing-talks/dados-desbalanceados-o-que-s%C3%A3o-e-como-evit%C3%A1-los-43df4f49732b.

BATTESTIN, V.; SANTOS, P. Addiem–um processo para criação de cursos mooc. **EaD em Foco**, v. 12, n. 1, 2022.

CABRERA, A. F. Logistic regression analysis in higher education: An applied perspective. **Higher education: Handbook of theory and research**, v. 10, p. 225–256, 1994.

CARVALHO, L. H. T. **Avaliação de algoritmos multi-classe para classificação de solicitações enviadas a Ouvidoria Geral do Estado de Pernambuco**. Dissertação (B.S. thesis) – Brasil, 2021.

Coursera. **Investor relations**. 2023. Disponível em: https://investor.coursera.com/overview/default.aspx. Acesso em: 01 de junho 2023.

Github Inc. **Github about**. 2023. Disponível em: https://github.com/about. Acesso em: 01 de julho 2023.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [*S. l.*]: The Mit Press, 2016.

GUERREIRO, A.; BARROS, D. M. V. Novos desafios da educação a distância: programação e uso de chatbots. 2019.

HEIDARI, A.; NAVIMIPOUR, N. J.; UNAL, M.; TOUMAJ, S. The covid-19 epidemic analysis and diagnosis using deep learning: A systematic literature review and future directions. **Computers in biology and medicine**, Elsevier, v. 141, p. 105141, 2022.

Hock-Chuan, Chua. **Source-Code Editors and IDEs**. 2023. Disponível em: https://www3.ntu.edu.sg/home/ehchua/programming/howto/editoride.html. Acesso em: 01 de junho 2023.

INEP. **Nearest Neighbors**. 2022. Disponível em: https://www.gov.br/inep/pt-br/assuntos/noticias/censo-da-educacao-superior/ensino-a-distancia-cresce-474-em-uma-decada. Acesso em: 01 de junho 2023.

Kaggle. **Coursera Courses *dataset* 2021 | Kaggle**. 2023. Disponível em: https://www.kaggle.com/\protect\unhbox\voidb@x\bgroup\edef.{dataset}\let\futurelet\@let@token\let\itshapedataset\egroups/khusheekapoor/coursera-courses-\protect\unhbox\voidb@x\bgroup\edef.{dataset}\let\futurelet\@let@token\let\itshapedataset\egroup-2021. Acesso em: 14 jun. 2023.

KORABLEVA, O.; DURAND, T.; KALIMULLINA, O. Studying user satisfaction with the mooc platform interfaces using the example of coursera and open education platforms. In: **Proceedings of the 2019 International Conference on Big Data and Education**. [*S. l.*: *s. n.*], 2019. p. 26–30.

LEMAÎTRE, G.; NOGUEIRA, F.; ARIDAS, C. K. imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. In: **The Journal of Machine Learning Research**. [*S. l.*: *s. n.*], 2017. p. 559–563.

Matplotlib. **Matplotlib documentation — Matplotlib 3.7.1 documentation**. 2023. Disponível em: https://matplotlib.org/stable/index.html. Acesso em: 14 jun. 2023.

MITCHELL, T. M. **Machine learning**. [*S. l.*]: McGraw-Hill Science/Engineering/Math, 1997.

MORALES, E.; ESCALANTE, H. A brief introduction to supervised, unsupervised, and reinforcement learning. In: **Biosignal Processing and Classification Using Computational Learning and Intelligence**. [*S. l.*: *s. n.*], 2022. p. 111–129.

NumPy. **NumPy documentation — NumPy v1.24 Manual**. 2023. Disponível em: https://numpy.org/doc/stable/. Acesso em: 14 jun. 2023.

Pandas. **Python Data Analysis Library**. 2023. Disponível em: https://pandas.pydata.org/. Acesso em: 14 jun. 2023.

PÉREZ-ÁLVAREZ, R.; MALDONADO-MAHAUAD, J.; PÉREZ-SANAGUSTÍN, M. How to map learning activities through urls? the case of coursera platform. In: **Proc. 2nd Int. Conf. MOOC-Maker**. [*S. l.*: *s. n.*], 2018. p. 25–34.

PETKOVIC, D.; ALTMAN, R.; WONG, M.; VIGIL, A. Improving the explainability of random forest classifier–user centered approach. In: WORLD SCIENTIFIC. **Pacific symposium on biocomputing 2018: proceedings of the pacific symposium**. [*S. l.*], 2018. p. 204–215.

RAMYACHITRA, D.; MANIKANDAN, P. Imbalanced dataset classification and solutions: a review. **International Journal of Computing and Business Research (IJCBR)**, Scientific Research Publishing, v. 5, n. 4, p. 1–29, 2014.

Scikit Learn. **Confusion matrix**. 2023. Disponível em: https://scikit-learn.org/stable/modules/model_evaluation.html#confusion-matrix. Acesso em: 01 de julho 2023.

Scikit Learn. **Data leakage**. 2023. Disponível em: https://scikit-learn.org/stable/common_pitfalls.html#data-leakage. Acesso em: 01 de julho 2023.

Scikit Learn. **GridSearchCV**. 2023. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn-model-selection-gridsearchcv. Acesso em: 01 de julho 2023.

Scikit Learn. **Logistic regression**. 2023. Disponível em: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression. Acesso em: 01 de junho 2023.

Scikit Learn. **Nearest Neighbors**. 2023. Disponível em: https://scikit-learn.org/stable/modules/neighbors.html#nearest-neighbors-classification. Acesso em: 01 de junho 2023.

Scikit Learn. **Precision, recall and F-measures**. 2023. Disponível em: https://scikit-learn.org/stable/modules/model_evaluation.html#precision-recall-and-f-measures. Acesso em: 01 de julho 2023.

Scikit Learn. **Randomized Parameter Optimization**. 2023. Disponível em: https://scikit-learn.org/stable/modules/grid_search.html#randomized-parameter-optimization. Acesso em: 01 de junho 2023.

Scikit Learn. **Receiver operating characteristic (ROC)**. 2023. Disponível em: https://scikit-learn.org/stable/modules/model_evaluation.html#receiver-operating-characteristic-roc. Acesso em: 01 de julho 2023.

Scikit Learn. **Scikit cross-validation**. 2023. Disponível em: https://scikit-learn.org/stable/modules/cross_validation.html#stratified-k-fold. Acesso em: 01 de junho 2023.

Scikit Learn. **Scikit documentation**. 2023. Disponível em: https://scikit-learn.org. Acesso em: 01 de junho 2023.

Scikit Learn. **Tuning the hyper-parameters of an estimator**. 2023. Disponível em: https://scikit-learn.org/stable/modules/grid_search.html#tuning-the-hyper-parameters-of-an-estimator. Acesso em: 01 de julho 2023.

TORRES-VÁSQUEZ, M.; HERNÁNDEZ-TORRUCO, J.; HERNÁNDEZ-OCANA, B.; CHÁVEZ-BOSQUEZ, O. Balanceo de datos del síndrome de guillain-barré utilizando smote para la clasificación de subtipos. **Res. Comput. Sci.**, v. 148, n. 7, p. 113–125, 2019.

WAN, X. Influence of feature scaling on convergence of gradient iterative algorithm. In: LESHAN VOCATIONAL AND TECHNICAL COLLEGE. **Journal of physics: Conference series**. [*S. l.*], 2019. p. 03.

APÊNDICE A – CÓDIGO DOS DADOS

```
coursera = pd.read_csv("Coursera_original_university_name_fixed.
    csv", sep=";")

coursera = coursera[pd.notnull(coursera["Skills"])]
coursera = coursera[pd.notnull(coursera["Difficulty_Level"])]

col = ["Skills", "Difficulty_Level"]
coursera = coursera[col]

coursera["category_id"] = coursera["Skills"].factorize()[0]
```

Código-fonte 1 – Preparação de dados, primeira abordagem

```
https://github.com/joaopaulobarbosa/TCC_2023/blob/master/
    splitting%20and%20data%20cleaning/dataset_split_dataclean.py
```

Código-fonte 2 – dataset_split_dataclean.py

```
https://github.com/joaopaulobarbosa/TCC_2023/blob/master/
    tunning_randomforestclassifier.py
```

Código-fonte 3 – tunning_randomforestclassifier.py

```
https://github.com/joaopaulobarbosa/TCC_2023/blob/master/
    tunning_logisticregression.py
```

Código-fonte 4 – tunning_logisticregression.py

```
https://github.com/joaopaulobarbosa/TCC_2023/blob/master/
    tunning_kneighborsclassifier.py
```

Código-fonte 5 – tunning_kneighborsclassifier.py

```
https://github.com/joaopaulobarbosa/TCC_2023/blob/master/
    model_comparison.py
```

Código-fonte 6 – model_comparison.py

```
https://github.com/joaopaulobarbosa/TCC_2023/blob/master/
    scikit_data_version/splitting%20and%20data%20cleaning/SCIKIT
    %20DATA/dataset_split_dataclean.py
```

Código-fonte 7 – Scikit version of dataset_split_dataclean.py

```
https://github.com/joaopaulobarbosa/TCC_2023/blob/master/
    scikit_data_version/tunning_randomforestclassifier.py
```

Código-fonte 8 – Scikit version of tunning_randomforestclassifier.py

```
https://github.com/joaopaulobarbosa/TCC_2023/blob/master/
    scikit_data_version/tunning_logisticregression.py
```

Código-fonte 9 – Scikit version of tunning_logisticregression.py

```
https://github.com/joaopaulobarbosa/TCC_2023/blob/master/
    scikit_data_version/tunning_kneighborsclassifier.py
```

Código-fonte 10 – Scikit version of tunning_kneighborsclassifier.py

```
https://github.com/joaopaulobarbosa/TCC_2023/blob/master/
    scikit_data_version/model_comparison.py
```

Código-fonte 11 – Scikit version of model_comparison.py