UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FRANCISCO THOMÁS MENDES DE OLIVEIRA

PREDIÇÃO DE TEMPO DE VIAGEM COM BASE EM TRAJETÓRIAS AUTOMOBILÍSTICAS

QUIXADÁ

2019

FRANCISCO THOMÁS MENDES DE OLIVEIRA

PREDIÇÃO DE TEMPO DE VIAGEM COM BASE EM TRAJETÓRIAS AUTOMOBILÍSTICAS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Dr. Regis Pires Magalhães

QUIXADÁ

2019

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

O47p Oliveira, Francisco Thomás Mendes de.
Predição do tempo de viagem com base em trajetórias de automobilísticas / Francisco Thomás Mendes de Oliveira. – 2019.
50 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Ciência da Computação, Quixadá, 2019.
Orientação: Prof. Dr. Regis Pires Magalhães.

1. Adivinhação. 2. Aprendizado do computador. 3. Controle preditivo. I. Título.

CDD 004

FRANCISCO THOMÁS MENDES DE OLIVEIRA

PREDIÇÃO DE TEMPO DE VIAGEM COM BASE EM TRAJETÓRIAS AUTOMOBILÍSTICAS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Aprovada em: __/__/____.

BANCA EXAMINADORA

Prof. Dr. Regis Pires Magalhães (Orientador)
Universidade Federal do Ceará (UFC)

Profa. Me. Lívia Almada Cruz Rafael
Universidade Federal do Ceará (UFC)

Prof. Tércio Jorge da Silva
Universidade Federal do Ceará (UFC)

Este trabalho é dedicado às crianças adultas que, quando pequenas, sonharam em se tornar cientistas.

AGRADECIMENTOS

À minha família, por sua capacidade de acreditar em mim e investir em mim. Mãe, seu cuidado e dedicação foi que deram, em alguns momentos, a esperança para seguir. Pai, sua presença significou segurança e certeza de que não estou sozinho nessa caminhada.

Ao Prof. Dr. Regis Pires Magalhães por me orientar durante o trabalho de conclusão de curso, e por todos os conselhos e ensinamentos que foram repassados.

Aos amigos do curso de Ciência da Computação, que me incentivaram com palavras motivadoras.

Agradeço a todos os professores por me proporcionar o conhecimento não apenas racional, mas a manifestação do caráter e afetividade da educação no processo de formação profissional, por tanto que se dedicaram a mim, não somente por terem me ensinado, mas por terem me feito aprender.

“Faça as coisas o mais simples que você puder,
porém não se restrinja às mais simples.”

(Albert Einstein)

RESUMO

O objetivo deste trabalho é prever o tempo de viagem usando técnicas de aprendizado supervisionado e algoritmos de regressão, bem como criar um preditor para tentar chegar a uma aproximação precisa do valor de tempo de uma dada rota de um automóvel e, dessa forma, utilizar da competição de predição de tempo de viagem de táxi da plataforma Kaggle[1] que foi apresentada durante a Conferência Europeia sobre Aprendizado de Máquina e Princípios e Prática de Descoberta de Conhecimento em Bancos de Dados como um banco de dados para treinamento de Modelo e como uma das avaliações para nosso preditor. Obtendo assim uma abordagem eficaz que pode ser geralmente utilizada para prever o tempo de viagem.

Palavras-chave: Predição de Tempo de Viagem. Modelos Preditivos. Mobilidade Urbana. Aprendizado de Máquina.

[1] https://www.kaggle.com/c/pkdd-15-taxi-trip-time-prediction-ii

ABSTRACT

The goal of this work is to predict travel time using supervised learning techniques and regression algorithms, as well as to create a predictor to try to arrive at an accurate approximation of the time value of a given route of an automobile and, therefore, to use taxi trip time prediction competition of the Kaggle[2] platform that was presented during the European Conference on Learning of Machine and Principles and Practice of Knowledge Discovery in Databases as a database for Model training and as one of the evaluations for our predictor. Thus obtaining an effective approach that can be generally used to predict travel time.

Keywords: Travel Time Prediction. Predictive Models. Urban Mobility. Machine Learning.

[2] https://www.kaggle.com/c/pkdd-15-taxi-trip-time-prediction-ii

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ECML-PKDD	*The European Conference on MachineLearning and Principles and Practice of Knowledge Discovery in Databases*
GBDT	*Gradient Boosted Decision Trees*
GBM	*Gradient boosting machine*
KNN	*K-nearest neighbors*
LGBM	*Lightgbm*
LR	*Linear Regression*
MAE	*Mean Absolute Error*
MAPE	*Mean Absolute Percentage Error*
MSE	*Mean Squared Error*
RF	*Random Forest*
RMSE	*root mean square error*
RMSLE	*Root Mean Squared Logarithmic Error*
XGBM	*eXtreme Gradient Boosting - xgboost*

SUMÁRIO

1 INTRODUÇÃO

No mundo atual, o tempo é algo essencial no cotidiano. Cumprir horário, seguir agenda, evitar atrasos são atividades normais na rotina de alguém. Em muitos casos para evitar o imprevisto e conseguir melhorar o uso do tempo, pessoas aderem a meios de transportes com o intuito de conseguir uma melhor mobilidade urbana, agilizando assim o tempo de chegada no trabalho ou a um evento. No entanto, não são todos que conseguem adquirir um meio de transporte. Essas pessoas ficam dependentes de um serviço público como rotas de ônibus, ou então usar serviços privados de empresas, que devido ao alto crescimento populacional e a expansão das cidades está se tornando um dos serviços mais usados, por oferecer um certo conforto e praticidade.

Com o avanço da tecnologia, o mundo se tornou mais conectado e muitas pessoas na atualidade possuem um *smartphone*. Com a tecnologia acessível a maior parte da população, os serviços de locomoção se tornaram mais suscetíveis a utilização de aplicações *online* para o fornecimento dos mesmos. Com isso foi possível criar aplicativos como por exemplo de despachos de táxi, indicadores de rotas, busca de locais e entre outros. Existem várias empresas que oferecem esses tipos de serviço. Assim eles facilitam a vida dos usuários desse meio, tornando assim mais fácil a vida em uma cidade muito movimentada.

O que esses serviços têm em comum é uma base para predição de tempo, muito usado pelos serviços de táxi para identificar o tempo de viagem das corridas, também usado como indicador para seleção de melhor rota a ser tomada. Esse tempo é um fator importante para utilização desses serviços e para eficiência deles. Um exemplo seria uma pessoa precisar pegar um táxi para pegar um voo que parte no dia seguinte. Como ela não está dirigindo, uma rota específica é menos importante e a única preocupação é saber quanto tempo vai demorar o deslocamento para decidir qual o melhor horário de partida (POLO, 2017). Outro exemplo, é relacionado a medidas de eficiência do tráfego da cidade. Uma métrica utilizada por pesquisadores da área é o tempo médio que as pessoas gastam todos dias para ir e voltar do trabalho, chamado de tempo de migração pendular (LEE; MCDONALD, 2003).

Em setembro de 2015 foi proposto pela plataforma *Kaggle*, juntamente com a Conferência Européia sobre Aprendizado de Máquina e Princípios e Prática da Descoberta de Conhecimento em Bancos de Dados (*The European Conference on MachineLearning and Principles and Practice of Knowledge Discovery in Databases* (ECML-PKDD)) uma competição que enfoca os serviços de despacho de táxis da cidade de Porto em Portugal. A proposta dessa

competição é criar um preditor capaz de realizar de forma precisa o cálculo do tempo de viagens de táxi na cidade de Porto, com base nas trajetórias anteriores de motoristas de táxis da cidade.

O trabalho aqui apresentado tem como objetivo demonstrar como realizar predições de tempo de viagem usando técnicas de aprendizado de máquina supervisionado, e assim criar um preditor para tentar chegar a uma aproximação precisa do valor de tempo de uma dada rota de automóveis, para realizar o mesmo iremos utilizar a competição anteriormente citada como base de dados, e avaliação do nosso preditor. Obtendo uma abordagem eficaz que possa ser empregada de forma geral para predição de tempo de viagem.

Este trabalho está dividido em 7 Capítulos, da qual o Capítulo 1 é a introdução. No Capítulo 2 os objetivos gerais e específicos, no Capítulo 3, fundamentação teórica do trabalho, no Capítulo 4 a os trabalhos relacionados, no Capítulo 5, os procedimentos metodológicos que foram realizados ao longo do trabalho, no Capítulo 6 são apresentados os resultados e no Capítulo 7 as conclusões.

2 OBJETIVOS

Este capítulo apresenta os objetivos gerais e específicos deste trabalho.

2.1 Objetivo Geral

O principal objetivo deste trabalho é selecionar um modelo preditivo existente capaz de calcular de forma precisa, o tempo de rota em viagens de veículos, com base nos dados de viagens anteriores realizadas.

2.2 Objetivos específicos

- Identificar técnicas eficientes para predizer o tempo de locomoção de veículos.
- Comparar e avaliar diferentes modelos de regressão para obter predições.
- Selecionar um modelo preditor eficaz para a predição de tempo de viagem.

3 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos usados e tem como objetivo explicar esses conceitos para a compreensão do trabalho. Desse modo, apresenta os conceitos de aprendizado de máquina, aprendizado supervisionado, engenharia de *features* e as métricas de avaliação de modelos preditivos.

3.1 *Feature*

Uma *feature* ou característica é um atributo ou propriedade compartilhada por todas as unidades independentes nas quais a análise ou predição deve ser feita. Qualquer atributo pode ser uma *feature*, desde que seja útil para o modelo.

O propósito de uma *feature*, além de ser um atributo, seria seria descrever o contexto de um problema e apresentar suas características. Uma *feature* é uma característica que pode ajudar na solução do problema (MASTERY, 2015)

3.2 Engenharia de Features

Engenharia de *Features* (*Feature engineering*) é o processo de usar o conhecimento de domínio dos dados para criar *features* ou escolher as que melhor representam o problema para aumentar a precisão dos algoritmos de aprendizado de máquina. A engenharia de *feature* é fundamental para a aplicação do aprendizado de máquina.

A engenharia de *feature* é considerado essencial no aprendizado de máquina aplicado. Criar features é difícil, consome tempo e requer conhecimento especializado. "Aprendizado de máquina aplicado"é basicamente engenharia de *feature*.

Dependendo da *feature*, ela pode ser fortemente relevante (ou seja, a *feature* tem informações que não existem em nenhuma outra *feature*), relevante, pouco relevante (algumas informações que outras *features* incluem) ou irrelevante (BOTTOU, 2010). É importante criar muitas *features*. Mesmo que algumas delas sejam irrelevantes, você não pode perder o resto. Depois disso, a seleção de *features* pode ser usada para evitar *overfitting* (SARKAR *et al.*, 2018)

.

3.3 Aprendizado de Máquina

Segundo (BAŞTANLAR; ÖZUYSAL, 2014) o campo de aprendizado de máquina pode ser resumidamente definido como uma forma de permitir que os computadores façam predições bem-sucedidas usando experiências passadas. Aprendizagem de máquina obteve desenvolvimento impressionante recentemente com a ajuda do rápido aumento na capacidade de armazenamento e no poder de processamento dos computadores.

Aprendizagem de máquina é um método de análise de dados que automatiza o desenvolvimento de modelos analíticos (MOURA, 2017). É possível utilizar algoritmos para coletar dados e aprender com os dados, levando em consideração todo o histórico, para então fazer uma determinação, ou predição sobre alguma coisa ou situação do mundo real.

Existem alguns tipos de problemas de aprendizagem de máquina, como por exemplo regressão e classificação. Um problema de regressão busca estimar os resultados em uma saída contínua, o que significa que as variáveis de entrada são mapeadas para alguma função contínua. Um problema de classificação, consiste em prever os resultados em uma saída discreta. Em outras palavras, mapeia se variáveis de entrada em categorias distintas.

Existem dois tipos principais de aprendizado da máquina: a aprendizagem supervisionada, onde apresenta-se um conjunto de entradas, e cada padrão de entrada possui sua saída e a mesma corresponde às classes desejadas, e a aprendizagem não supervisionada onde apenas as entradas são fornecidas. O padrão de saída deve ser deduzido pelo próprio sistema de aprendizado.

3.3.1 Aprendizado supervisionado

Segundo (RUSSELL; NORVIG, 2016) em aprendizado supervisionado, o agente observa alguns pares de entrada-saída de exemplo e aprende uma função que mapeia de entrada para saída. Nesse caso, o valor de saída está disponível diretamente das percepções do agente (após o fato).

(MONARD; BARANAUSKAS, 2003) fala que no aprendizado supervisionado é fornecido ao algoritmo de aprendizado, ou indutor, um conjunto de exemplos de treinamento para os quais o rótulo da classe associada é conhecido e todo exemplo possui um atributo especial, o rótulo ou classe, que descreve o fenômeno de interesse, isto é, a meta que se deseja aprender e poder fazer previsões a respeito. Como já mencionado, os rótulos são tipicamente pertencentes a

um conjunto discreto (nominal) de classes $C_1, C_2, ..., C_k$ no caso de classificação ou de valores reais no caso de regressão, por exemplo.

3.3.1.1 Classificação

Os algoritmos de classificação são usados quando o resultado desejado é um rótulo discreto. (JAMES *et al.*,) fala que as variáveis podem ser caracterizadas como quantitativas ou qualitativas (conhecidas como categóricas). Variáveis quantitativas assumem valores numéricos. Os exemplos incluem a idade, a altura ou a renda de uma pessoa, o valor de uma casa, o valor categórico e o preço de uma ação. Por outro lado, as variáveis qualitativas assumem valores em uma das K classes ou categorias diferentes. Exemplos de variáveis de classe qualitativas incluem o gênero de uma pessoa (homem ou mulher), a marca do produto comprado (marca A, B ou C), se uma pessoa não tem uma dívida (sim ou não) ou um diagnóstico de câncer (Mielogênese Aguda, Leucemia, Leucemia Linfoblástica Aguda ou Sem Leucemia). Tende-se a se referir a problemas com uma resposta quantitativa como problemas de regressão, enquanto aqueles que envolvem predição de uma resposta qualitativa são frequentemente chamados de problemas de classificação.

3.3.1.2 Regressão

Regressão é útil para prever características contínuas. Isso significa que a resposta a sua pergunta é representada por um valor que pode ser determinado de forma flexível com base nas entradas do modelo, em vez de se limitar a um conjunto de rótulos possíveis. Os problemas de regressão com entradas ordenadas por tempo são chamados de problemas de previsão de séries temporais, que permite que os cientistas de dados expliquem padrões sazonais em vendas, avaliem o impacto de novas campanhas de marketing e muito mais.

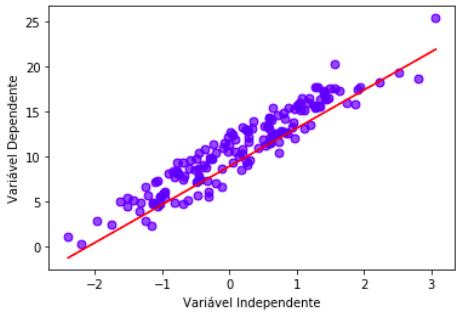
3.4 Algoritmos de Regressão

3.4.1 Regressão Linear

Segundo (CAMILO; SILVA, 2009), as regressões são chamadas de lineares quando a relação entre as variáveis preditoras e a resposta segue um comportamento linear. Neste caso, é possível criar um modelo no qual o valor de h_θ é uma função linear de x, por exemplo:

$h_\theta = \theta_0 + \theta_1 x$ onde o número θ_0 é chamado de coeficiente de h_θ e o número θ_1 é chamado termo constante. Pode-se utilizar o mesmo princípio para modelos com mais de uma variável preditora. Na Figura 1 tem-se um exemplo de uma regressão linear (*Linear Regression* (LR)).

Figura 1 – Regressão Linear



Fonte: Elaborado pelo Autor

Em aprendizado de máquina regressão linear é uma das técnicas mais utilizadas quando se quer ter um valor aproximado. É uma técnica utilizada quando se quer prever valores de algo futuro, o objetivo é criar uma reta que melhor represente a relação entre a variável dependente e a variável independente. A Equação 3.1 representa essa reta.

$$h_\theta = \theta_0 + \theta_1 x \tag{3.1}$$

Na qual h_θ é a variável a ser gerada, nossa hipótese; θ_0 é a ordenada de origem onde a reta corta em h_θ, e θ_1 é a inclinação da reta.

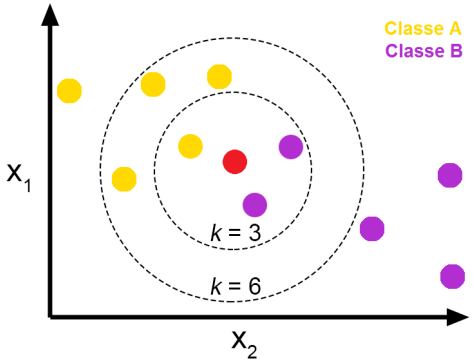
3.4.2 K-nearest neighbors

Segundo (ZHAO *et al.*, 2018) a ideia básica do algoritmo *K-nearest neighbors* (KNN) é definir uma quantidade k para pesquisar os pontos k mais próximos do estado atual na área de pesquisa e usar esses pontos para fazer previsões. Use todos esses pontos se houver menos de k pontos na área de busca e mantenha os k pontos mais próximos se houver mais de k pontos na área de busca. Valores k apropriados desempenham um papel fundamental nas previsões precisas.

A Figura 2[1] apresenta uma exemplificação do KNN, onde temos uma nova entrada e ela deve ser classificada em classe A ou classe B, para k igual a 3 temos ela classificada como classe B e para k igual a 6 temos ela pertencente a classe A.

[1] https://towardsdatascience.com/knn-k-nearest-neighbors-1-a4707b24bd1d

Figura 2 – *K-nearest neighbors*



Fonte: *Towards Data Science*

3.5 Técnicas de aprendizado supervisionado

3.5.1 Métodos de Ensemble

Segundo (ZHOU, 2012) os métodos *ensemble* treinam vários modelos para resolver o mesmo problema. Em contraste com as abordagens comuns de aprendizagem que tentam construir um aprendiz a partir dos dados de treinamento, os métodos de *ensemble* tentam construir um conjunto de modelos e combiná-los. Os métodos *ensemble* também são chamado de aprendizado baseado em comitê ou o aprendizado de sistemas de múltiplos classificadores ou regressores.

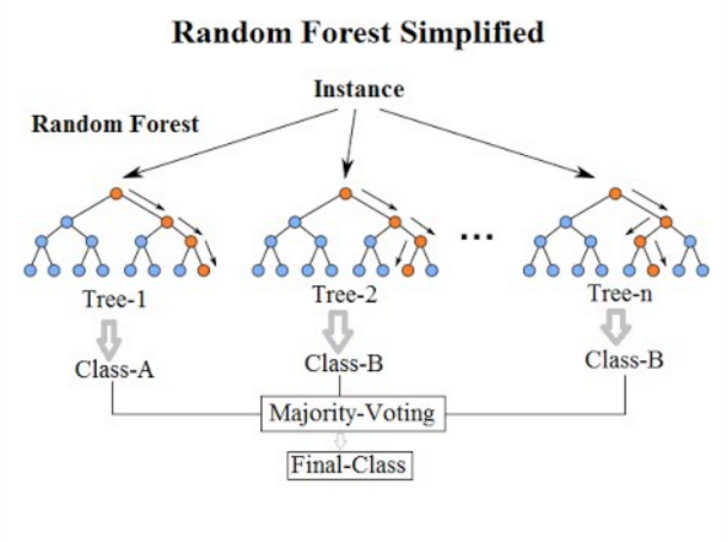
3.5.1.1 Random Forest

Segundo (BREIMAN, 2001) (*Random Forest* (RF)) são uma combinação de preditores de árvores, de tal forma que cada árvore depende dos valores de um vetor aleatório amostrado de forma independente e com a mesma distribuição para todas as árvores na floresta. O erro de generalização para florestas converge a.s. a um limite quando o número de árvores na floresta se torna grande. O erro de generalização de uma floresta de classificadores de árvores depende da força das árvores individuais na floresta e da correlação entre elas. Usar uma seleção aleatória de recursos para dividir cada nó gera taxas de erro que se comparam favoravelmente a *Adaboost*, mas são mais robustas em relação ao ruído. As estimativas internas monitoram o erro, a força e a correlação e são usadas para mostrar a resposta ao aumento do número de recursos usados na divisão. Estimativas internas também são usadas para medir a importância variável.

a Figura 3[2] apresenta como é a execução do algoritmo de *random forest*.

[2] https://towardsdatascience.com/predicting-success-in-online-education-2b5979fa7016

Figura 3 – *Random Forest*



Fonte: *Towards Data Science*

3.5.1.2 Gradient boosting machine

Segundo (GONG *et al.*, 2017) *Gradient boosting machine* é uma técnica poderosa para a construção de modelos preditivos. Ele seleciona a função objetivo e usa em um modelo de aprendizagem fraca - tipicamente árvores de regressão - para minimizar a perda. Os parâmetros das árvores adicionadas são ajustados por um algoritmo de gradiente descendente. Existem duas estruturas de *Gradient boosting machine* (GBM) que são amplamente usadas na comunidade de ciência de dados: XGBoost e *Lightgbm* (LGBM).

3.5.1.2.1 *eXtreme Gradient Boosting - xgboost*

Segundo (CHEN *et al.*, 2015) *eXtreme Gradient Boosting - xgboost* (XGBM). Trata-se de uma implementação eficiente e escalonável do *framework* de aumento de gradientes. O pacote inclui um solucionador de modelo linear eficiente e um algoritmo de aprendizado de árvores. Suporta várias funções objetivas, incluindo regressão e classificação. Características:

- Velocidade: O *xgboost* pode fazer automaticamente paralelismo. Geralmente é mais de 10 vezes mais rápido que o *gradient Boosting* Machine segundo (CHEN *et al.*, 2015).
- Tipo de Entrada: *xgboost* toma vários tipos de dados de entrada: Matriz Densa, Matriz esparsa, Arquivo de dados, xgb.DMatrix: a própria classe do *xgboost*.
- Esparsidade: o *xgboost* aceita entrada esparsa tanto para o *tree booster* quanto para o *linear booster*, e é otimizado para entrada esparsa.

- Personalização: *xgboost* suporta função de objetivo personalizada e função de avaliação.
- Desempenho: o *xgboost* tem melhor desempenho em vários conjuntos de dados diferentes segundo (CHEN *et al.*, 2015).

3.5.1.2.2 *Lightgbm*

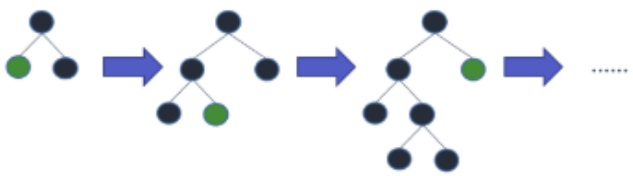
Segundo (WANG *et al.*, 2017) *LightGBM* é um algoritmo *Gradient Boosted Decision Trees* (GBDT) de código aberto da *Microsoft*. Ele usa um algoritmo baseado em histograma para acelerar o processo de treinamento, reduzir o consumo de memória e combinar comunicação de rede avançada para otimizar o aprendizado paralelo, chamado algoritmo de árvore de decisão de votação paralela. O *LigthGBM* divide os dados de treinamento em várias máquinas, a decisão de votação local para selecionar os atributos $top-k$ e a decisão de votação global para receber os atributos $top2k$ em cada iteração em que são executados. *LightGBM* usa a estratégia de *leaf-wise* para encontrar uma folha com maior ganho de divisor.

Características:

- Velocidade de treinamento mais rápida e maior eficiência
- Menor uso de memória
- Melhor precisão
- Aprendizado paralelo e GPU suportado
- Capaz de lidar com dados em grande escala

Como o *LightGBM* é baseado em algoritmos de árvore de decisão, ele divide a folha de árvore com o melhor ajuste, assim como apresentado na Figura 4, enquanto outros algoritmos de divisão dividem a profundidade da árvore em nível, apresentado na Figura 5, em vez de foliar.

Figura 4 – *leaf-wise*



Fonte: *Medium*

Figura 5 – Em Nível

Fonte: *Medium*

3.6 Geolocalização

Geolocalização utiliza a identificação de localizações geográficas de um usuário para coletar a localização de um dispositivo de computação por meio de uma variedade de mecanismos de coleta de dados. Normalmente, a maioria dos serviços de geolocalização usa endereços de roteamento de rede ou dispositivos GPS, interno para determinar este local. (FELIPE; DIAS,).

3.6.1 Fórmula de Haversine

Segundo (CHOPDE; NICHAT, 2013) a fórmula de Haversine é uma equação importante na navegação, dando grandes distâncias entre dois pontos em um círculo.

Estes nomes decorrem do fato de que eles são costumeiramente escritos em termos da função *haversine*, dada por $Haversin(\theta) = sin^2(\frac{0}{2})$. A fórmula de *haversine* é usada para calcular a distância entre dois pontos na Terra, onde os pontos são especificados como longitude e latitude. A fórmula de *haversine* é mostrada na equação 3.2

$$d = 2r\sin^{-1}(\sqrt{\sin^2(\frac{\theta_2-\theta_1}{2}) + \cos(\theta_1)\cos(\theta_2)\sin^2(\frac{\omega_2-\omega_1}{2})}) \tag{3.2}$$

na equação 3.2 d é a distância entre dois pontos com longitude e latitude (ω, θ) e r é o raio da Terra.

3.7 Métricas de Avaliação

As métricas de avaliação são aplicadas em classificadores e modelos de predição para determinar as características de cada um dos modelos. Cada métrica é uma medida de desempenho ou uma razão entre observações. Elas possuem características distintas e informam o estado atual do modelo, indicando se é um bom preditor ou não. É importante ressaltar esse aspecto devido ao fato de que cada modelo é único e a qualidade pode ser relativa.

Para calcular algumas métricas como acurácia, precisão e revocação (*accuracy,*

precision e recall), é utilizado a matriz de confusão, gerada a partir dos valores preditos do modelo de classificação.

Na regressão esse cálculo se torna diferente, pois não se tem um valor categórico como rótulo (verdadeiro ou falso, como detectar cachorro ou gato). Buscar-se prever um valor numérico, como, por exemplo, as vendas de uma empresa para o próximo mês. O valor passa a ser uma aproximação do valor real. Para avaliar um modelo preditivo nessa situação calcula-se em termos estatísticos, a média dos erros individuais entre os pontos. Para calcular esses erros utilizam-se de funções. Nesta parte estão as funções mais comuns utilizadas para avaliar o desempenho de modelos de regressão, que são: erro absoluto médio (*Mean Absolute Error* (MAE)), erro quadrático médio (*Mean Squared Error* (MSE)), raiz do erro quadrático médio (*root mean square error* (RMSE)), erro de porcentagem absoluta média (*Mean Absolute Percentage Error* (MAPE)), raiz do erro logarítmico quadrático médio (*Root Mean Squared Logarithmic Error* (RMSLE)) e entre outros.

Nos exemplos a seguir considere y o valor real do conjunto de dados e $\bar{y}$ o valor predito pelo modelo, e N como a quantidade de valores preditos pelo modelo.

3.7.1 Erro quadrático médio

Uma das técnicas mais utilizadas, esta função calcula a média dos erros do modelo ao quadrado. Ou seja, diferenças menores têm menos importância, enquanto diferenças maiores recebem maior peso. O MSE é definido na Equação 3.3.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{y}_i)^2 \tag{3.3}$$

3.7.2 Raiz do erro logarítmico quadrático médio

O RMSLE adiciona um a ambos, tanto no valor real quanto no predito, antes de tomar o logaritmo natural para evitar tomar o log natural de zero. Como resultado, a função pode ser usada se o valor real ou predito tiver elementos com valor zero. Mas esta função não é apropriada se ambos forem de valor negativo. O RMSLE é definido na Equação 3.4.

$$RMSLE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\log(y_i + 1) - \log(\bar{y}_i + 1))^2} \tag{3.4}$$

3.7.3 Raiz do erro quadrático médio

Em adição ao MSE, a raiz quadrada de MSE, ou Raiz do Erro Quadrático Médio, é comumente usada para expressar a acurácia dos resultados numéricos com a vantagem de que RMSE apresenta valores do erro nas mesmas dimensões da variável analisada. O RMSE é definido na Equação 3.5.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{y}_i)^2} \tag{3.5}$$

3.7.4 Erro absoluto médio

Bastante parecido com MSE, em vez de elevar a diferença entre a previsão do modelo, e o valor real, ao quadrado, ele toma o valor absoluto. Neste caso, em vez de atribuir um peso de acordo com a magnitude da diferença, ele atribui o mesmo peso a todas as diferenças, de maneira linear. O MAE é definido na Equação 3.6.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}(|y_i - \bar{y}_i|) \tag{3.6}$$

3.7.5 Erro de porcentagem absoluta média

Este erro calcula a média percentual do desvio absoluto entre as previsões e a realidade. É utilizado para avaliar sistemas de previsões de vendas e outros sistemas nos quais a diferença percentual seja mais interpretável, ou mais importante, do que os valores absolutos. O MAPE é definido na Equação 3.7.

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}(\frac{|y_i - \bar{y}_i|}{y_i}) \tag{3.7}$$

4 TRABALHOS RELACIONADOS

Neste capítulo serão descritos os trabalhos relacionados, que descrevem abordagens para predição de tempo de viagens em diferentes cenários. A seguir, apresentada uma breve descrição de como foi abordada a solução em cada trabalho.

Este trabalho se propõe a elaborar um preditor capaz de calcular o tempo de viagem de um automóvel para uma dada rota que ele pretende realizar. O conjunto de dados utilizado neste trabalho são dados coletados no período de um ano completo (de 01/07/2013 a 30/06/2014) das trajetórias para todos os 442 táxis que circulam na cidade do Porto, em Portugal, e está disponível na plataforma *Kaggle*. O objetivo deste trabalho é realizar predições de tempo de viagem usando técnicas de aprendizado de máquina supervisionado para obter uma aproximação precisa do valor de tempo de uma dada rota de táxi e dessa forma criar um preditor capaz de predizer esse valor de tempo de forma geral.

4.1 *Highway Travel Time Prediction Using Sparse Tensor Completion Tactics and-Nearest Neighbor Pattern Matching Method*

(ZHAO *et al.*, 2018) relatam uma nova tecnologia de um sensor de micro-ondas de transporte remoto (RTMS), que é um pequeno radar instalado na estrada, operando na banda de microondas. Simultaneamente, o sensor fornece a presença de pistas, assim como informações de volume, ocupação, velocidade e classificação em até 12 zonas de detecção definidas pelo usuário. As informações de saída são fornecidas aos controladores existentes por meio do fechamento de contato, e a outros sistemas de computação por porta de comunicação serial ou TCP / IP. Um único radar pode substituir vários detectores de loop indutivo, que são dispositivos eletrônicos capazes de medir a proximidade de objetos metálicos que entram em seu campo magnético e a quantidade de vezes que esse objeto entra em seu campo. Esta tecnologia sendo promovida para uso em rodovias na China. A distância é de cerca de 2 a 5 km entre RTMSs, o que leva à falta de dados e problemas de dispersão de dados. Esses dois problemas restringe seriamente a precisão da previsão do tempo de viagem. Visando o problema de falta de dados, baseado em características multimodo de tráfego, um método de complementação tensorial – *Tensor Completion* – é proposto para recuperar os dados perdidos de velocidade e volume RTMS. Visando o problema de dispersão de dados, nós sensores virtuais são configurados entre nós RTMS reais, e o método bidimensional de interpolação linear e por partes é aplicado para estimar

o tempo médio de viagem entre dois nós. O método vizinho mais próximo (kNN -*k nearest neighbors*) é proposto para a predição do tempo de viagem. O método de validação cruzada é usado para calibrar e melhorar a adaptabilidade do algoritmo kNN com base nos dados coletados da rodovia *Jinggangao*. Os resultados mostram que o método proposto em (ZHAO *et al.*, 2018) pode melhorar a qualidade dos dados e a precisão de predição do tempo de viagem.

O trabalho apresentado propõe calcular o tempo de viagem dos carros que trafegam nessa rodovia, o nosso trabalho avalia a utilização de um preditor capaz de calcular o tempo de viagem nesse tipo de ambiente e em outros.

4.2 *Taxi destination and trip time prediction from partial trajectories*

O trabalho de (LAM *et al.*, 2015) relata a solução proposta por uma equipe que obteve as 3ª e 7ª posições do ranking final para a competição de *ECML/PKDD 15: Taxi Trip Time Prediction* e *ECML/PKDD 15: Taxi Trajectory Prediction* . (LAM *et al.*, 2015) apresenta uma abordagem baseada em combinação de viagem e aprendizado conjunto, na qual os padrões são observados em um conjunto de dados de aproximadamente 1,7 milhões de viagens de táxi para prever o destino final correspondente, bem como o tempo de viagens de táxis em andamento. Os resultados mostram que essa abordagem é eficaz e robusta.

O trabalho apresentado propõe uma solução para a competição do Kaggle em que o trabalho proposto pelo artigo utiliza como base de dados para treinar e testar os modelos.

4.3 *Gradient boosting method to improve travel time prediction*

No trabalho de (ZHANG; HAGHANI, 2015) é falado que os métodos de conjunto baseados em árvore alcançaram um status de celebridade no campo de previsão. Ao combinar árvores de regressão simples com desempenho "ruim", elas geralmente produzem juntas alta precisão de previsão. Os métodos de conjunto baseados em árvore fornecem resultados interpretáveis, embora exijam pouco pré-processamento de dados, são capazes de lidar com diferentes tipos de variáveis de previsão e podem se ajustar a relacionamentos não lineares complexos. Estas propriedades tornam os métodos de conjunto baseados em árvores bons candidatos para resolver problemas de previsão de tempo de viagem. No trabalho de ZHANG; HAGHANI é empregado o método de árvore de regressão GBM – *gradient boosting machine* – para analisar e modelar o tempo de viagem da autoestrada em *Maryland* localizada nos Estados Unidos, para

melhorar a precisão da predição e a interpretabilidade do modelo.

O trabalho apresentado propõe um estudo sobre a relevância do GBM para a predição de tempo, no trabalho proposto aplicamos técnicas de GBM.

4.4 Comparação entre os trabalhos relacionados e o proposto

Na Quadro 1 são apresentadas as semelhanças e diferenças entre os trabalhos relacionados e este trabalho, com relação às suas principais características.

Quadro 1 – Comparação entre os trabalhos relacionados e o proposto

	ZHAO *et al.* (2018)	**LAM *et al.* (2015)**	**ZHANG; HAGHANI (2015)**	**Trabalho Proposto**
Base de Dados	N.A.	Kaggle	INRIX	Kaggle
Dados Abertos	Não	Sim	Sim	Sim
Objetivos	tempo de viagem	melhor trajetória e tempo de viagem	tempo de viagem	tempo de viagem
Uso de *ensemble* de classificadores	Não	Sim	Sim	Sim
Algoritmos utilizados	K-nearest neighbors e Tensor Completion	Random Forest	Gradient Boosting	K-nearest neighbors, Random Forest, lightGBM, Linear Regression e XGboost

Fonte: Elaborado pelo autor.

5 METODOLOGIA

No decorrer deste trabalho irão ocorrer 10 etapas, na qual cada etapa está descrita neste Capítulo: na Seção 5.1, é explicado como um estudo das técnicas de aprendizado de máquina que utilizadas para resolver problemas de regressão e pesquisas de trabalhos relacionados a este; na Seção 5.2, é explicado a escolha dos dados a serem usados neste trabalho; na Seção 5.3 é explicada a análise dos dados, na Seção 5.4 é explicado o pré-processamento e a engenharia de *features*; na Seção 5.5, são explicados a Preparação dos dados para geração e avaliação de modelos; na Seção 5.6, são explicados como é realizado o estudo dos principais algoritmos de regressão e a implementação do modelos preditivos; na Seção 5.7, se realiza o treinamento do modelos propostos a partir dos dados de treino; na Seção 5.8, é realizado uma análise do desempenho dos modelos propostos; e por último na seção 5.9, é executada a avaliação dos modelos propostos para escolha do modelo com melhor resultado e explicado o modelo preditivo final.

Na Figura 6 é apresentado um fluxograma com os passos metodológicos a serem executados neste trabalho.

5.1 Revisão bibliográfica

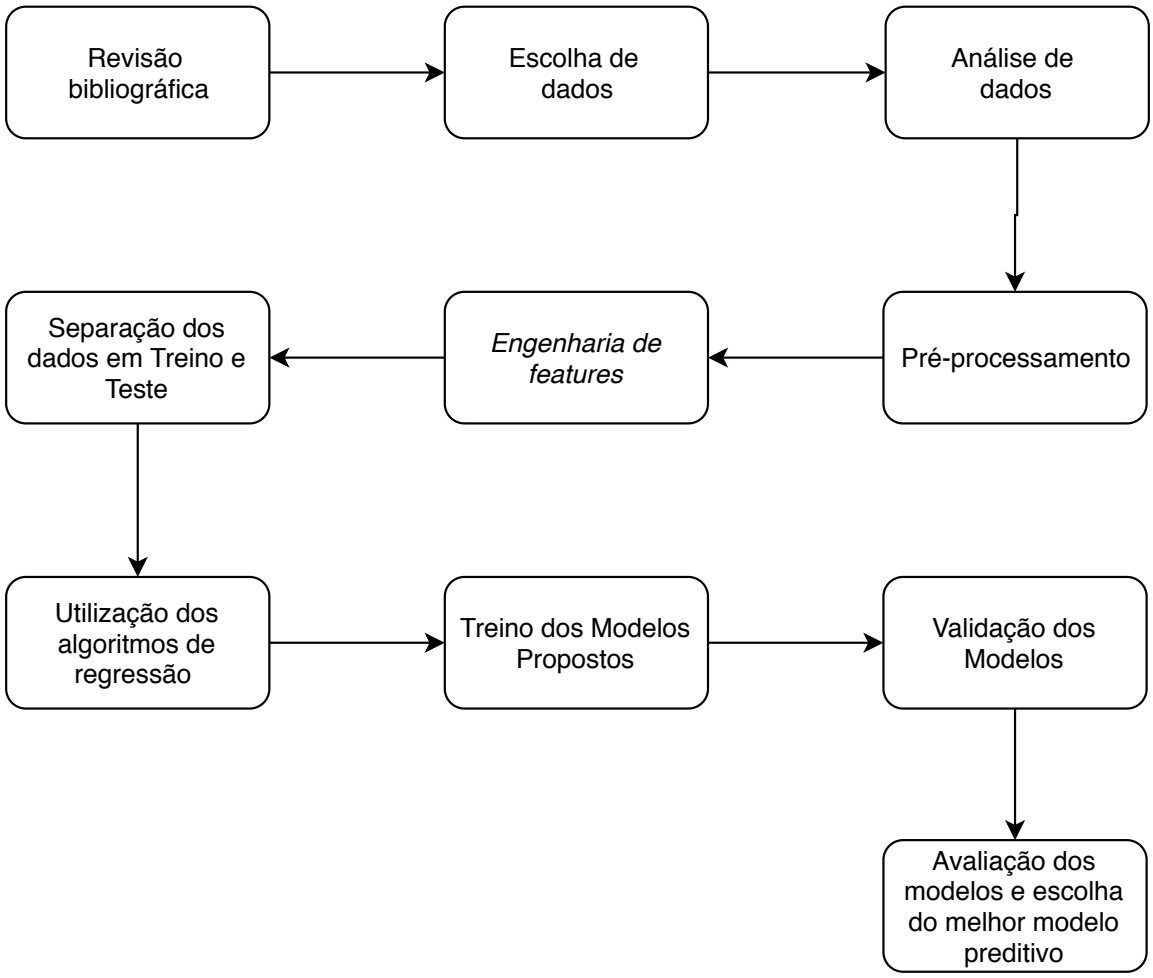
A primeira etapa deste trabalho é realizar uma revisão bibliográfica onde se é feito um estudo de técnicas utilizadas por trabalhos relacionados a este para resolver o problema de regressão, que é predizer o tempo de locomoção de um veículo.

5.2 Escolha dos dados

Dado que o objetivo deste trabalho é propor um preditor que possa informar o tempo de viagem de um automóvel, foi selecionado um problema proposto pela plataforma *Kaggle* em 2015, onde os dados são fornecidos pela própria plataforma, que se trata em predizer o tempo de viagem de táxis.

O conjunto de dados utilizado neste trabalho são dados coletados no período de um ano completo (de 01/07/2013 a 30/06/2014) das trajetórias para todos os 442 táxis que circulam na cidade do Porto, em Portugal. Cada táxi possui um sistema telemático instalado, adquire a posição atual do GPS e alguns metadados adicionais, Cada amostra de dados corresponde a uma viagem concluída. Ele contém um total de 9 (nove) *features*: TIPO DE CHAMADA, que

Figura 6 – Passos Metodológicos



Fonte: Elaborado pelo Autor

identifica a forma como o serviço de táxi é exigido. CHAMADA DE ORIGEM: Único id para identificar o chamador do serviço. ORIGIN STAND: ID exclusivo para identificar o ponto de táxi. ID DO TÁXI: ID exclusivo do taxista. TIME STAMP: hora de início da viagem. TIPO DE DIA: Identificador para o tipo de dia do início da viagem. FALTA DE DADOS: Indica se dados parciais da viagem estão faltando. Uma detalhada descrição do sistema de despacho e do processo de aquisição de dados pode ser encontrado em (MOREIRA-MATIAS *et al.*, 2013).

A plataforma disponibiliza um conjunto de treino (um arquivo CSV chamado "train.csv"), onde este conjunto fornece cada viagem categorizada em três formas: A) *taxi central based*, B) *stand-based* ou C) *non-taxi central based*. Para a primeira categoria, é fornecido um ID anonimizado, quando tal informação está disponível a partir da chamada telefônica. As duas últimas categorias referem-se a serviços que foram exigidos diretamente aos taxistas em um ponto de táxi (B) ou em uma rua aleatória (C).

5.3 Análise de Dados

Nesta etapa os dados fornecidos nesta plataforma são estudados e analisados, para utilização dos dados da plataforma de uma forma a encontrar padrões, tendências e outras diferenças que podem sugerir, melhorando assim o modelo de predição.

5.4 Pré-processamento

A partir da etapa de Análise de dados, trabalhar com o conjunto de dados usado neste trabalho a fim de encontrar *features* com valores faltantes ou nulos e assim decidir uma abordagem para solucionar esses eventuais problemas.

5.4.1 Engenharia de Features

Esta etapa destina-se a extrair, selecionar e gerar, a partir de dados brutos, características que possam ser utilizadas de forma efetiva nos modelos preditivos. Como a qualidade das *features* influencia muito a qualidade dos resultados. Esse passo se torna crucial para a produção de um bom preditor e para a obtenção de bons resultados.

5.5 Preparação dos dados para geração e avaliação de modelos

Nesta etapa, o conjunto de dados será preparado para ser utilizado ao longo do trabalho e pelos algoritmos de predição. Os dados serão separados em conjuntos de dados para treino, validação e teste.

5.6 Utilização dos algoritmos de regressão para criação dos modelos preditivos

Eleger um algoritmo é essencial no processo de aprendizagem. Calcular o tempo de viagem de um táxi é o principal ponto a ser trabalhado. Para gerar esse cálculo é necessário um algoritmo de regressão que funcione de forma eficiente por isso é importante que realmente se adapte ao caso de uso do problema em questão.

Como já discutido e explicado na Seção 3.3.1.2, regressão é o método que será utilizado neste trabalho. Nesta etapa são utilizados e testados modelos (com diferentes algoritmos de regressão e diferentes configurações) com uma pequena parte do conjunto de dados, para analisar qual dos modelos se adapta melhor para o problema abordado.

O algoritmo com melhor resultado nos testes que serão executados na etapa descrita na Capítulo 3, será utilizado para a criação do modelo final.

Nesta etapa foi utilizado os seguintes algoritmos *Lightgbm, K-nearest neighbors, Random Forest, Linear Regression e XGBoost* para a criação dos modelos para predição dos valores referente ao tempo.

5.7 Treinamento dos modelos propostos

Nesta etapa realiza-se o treinamento modelo proposto, criado na etapa descrita na Seção 5.6, com todo o conjunto de dados de treino criados na etapa descrita na Seção 5.5 considerando as *features* geradas.

5.8 Validação dos modelos preditivos

Nesta etapa é executado a validação dos modelos criados, testando novos hiper-parâmetros para uma melhor escolha dos mesmos, melhorando assim a predição.

5.9 Avaliação dos modelos propostos para escolha do modelo com melhor resultado

Por último, utiliza-se o conjunto de teste criado na etapa descrita na seção 5.6, para testar o desempenho do modelo final, as métricas RMSE, RMSLE e MAE são usadas para avaliar os modelos criados.

6 EXPERIMENTOS E RESULTADOS

Neste capítulo é apresentado os experimentos realizados ao longo do trabalho, assim como os resultados obtidos. Os experimentos foram rodados em *kernels* da plataforma Kaggle com 4 núcleos de processamento e 16 *Gigabytes* de memória RAM. É apresentado a pré análise nos dados, a escolha das *features* no passo descrito como engenharia de *features* e a seleção modelo de predição

6.1 Pré análise

Nesta seção é explicado como foi realizado o processamento em nossa base de dados. Realizou-se um pré-processamento no conjunto de dados a ser utilizado neste trabalho. Os dados descrevem um ano completo (de 01/07/2013 a 30/06/2014) das trajetórias para todos os 442 táxis que circulam na cidade do Porto, em Portugal. O conjunto de dados contém 1.710.670 entradas de viagens com 9 colunas, onde cada coluna representa uma *features*, e são organizadas da seguinte forma:

TRIP_ID : os valores nesta *feature* estão salvos como números inteiros, está *feature* representa um identificador único para cada viagem de táxi. Onde uma viagem é o ato de partir de um lugar para outro, relativamente distante, o deslocamento que se faz para se chegar de um local a outro relativamente distante;

CALL_TYPE : os valores nesta *feature* estão salvos como caracteres. Esses caracteres identificam o caminho usado para demandar este serviço, ou seja, a forma utilizada para requisição do serviço. Pode conter um dos três valores possíveis: 'A' se esta viagem foi requisitada diretamente a central; 'B' se esta viagem foi requerida diretamente a um taxista em um ponto específico onde se encontra um grupo de táxis; 'C' de outra forma (ou seja, uma viagem requerida em uma rua aleatória).

ORIGIN_CALL : os valores nesta *feature* estão representados como números de ponto flutuante. Esse número é um identificador único para cada número de telefone que foi usado para solicitar, pelo menos, um serviço. Identifica o cliente da viagem se CALL_TYPE = 'A'. Caso contrário, assume um valor nulo;

ORIGIN_STAND : os valores nesta *feature* estão representados como números de ponto flutuante. Essa *feature* é um identificador exclusivo para o ponto de táxi. Identifica o ponto de partida da viagem se CALL_TYPE = 'B'. Caso contrário, assume um valor NULL;

TAXI_ID: os valores nesta *feature* estão salvos como números inteiros, e contém um identificador exclusivo para o taxista que realizou cada viagem;

TIMESTAMP: os valores nesta *feature* são armazenados como números inteiros, Unix Timestamp (em segundos). Identifica o início da viagem;

DAY_TYPE: os valores nesta *feature* são representados como caracteres. A *feature* identifica o tipo de dia do início da viagem. Ela assume um dos três valores possíveis: 'A' um dia que não faz parte de um feriado ou feriado prolongado; 'B' é um feriado ou um dia de um feriado prolongado; 'C' um dia anterior de um dia do tipo B;

MISSING_DATA : os valores desta *feature* são *booleanos*. O valor é *FALSE* quando o fluxo de dados GPS não está completo e *TRUE* sempre que uma (ou mais) localização está faltando.

POLYLINE : os valores nesta *feature* estão salvos como *String*, conjunto de caracteres que forma uma palavra, essa string contém uma lista de coordenadas GPS (isto é, formato WGS84) mapeadas como uma string. O início e o fim da string são identificados com colchetes. Cada par de coordenadas também é identificado pelos mesmos parênteses ([LONGITUDE, LATITUDE]). Esta lista contém um par de coordenadas para cada 15 segundos de viagem. O último item da lista corresponde ao destino da viagem, enquanto o primeiro representa seu início;

Primeiramente foi feita uma observação nos dados brutos, utilizando a linguagem Python e a biblioteca Pandas, para identificar a quantidade de dados, os tipos de dados utilizados e a quantidade de dados faltantes para cada *feature*. Os resultados são apresentados na Quadro 2.

Quadro 2 – Informação dos Dados

Feature	**Quantidade de dados válidos**	**Quantidade de dados nulos**	**Tipo de dados**
TRIP_ID	1710670	0	Numérico
CALL_TYPE	1710670	0	Categórico
ORIGIN_CALL	364770	1345900	Numérico
ORIGIN_STAND	806579	904091	Numérico
TAXI_ID	1710670	0	Numérico
TIMESTAMP	1710670	0	Numérico
DAY_TYPE	1710670	0	Numérico
MISSING_DATA	1710670	0	Booleano
POLYLINE	1710670	0	Categórico

Fonte: Elaborado pelo Autor

Também existe um segundo arquivo com dados que são informações que acrescem aos dados os pontos de táxi na cidade de Porto, e que têm como objetivo informar-nos sobre

eles para tornar mais fácil a sua organização, esse arquivo contém um id, uma Descrição que é o nome do bairro que se localiza o ponto e a Latitude e Longitude do local. Sobre esses dados foi possível identificar as características apresentadas na Quadro 3

Quadro 3 – Informação dos Dados sobre pontos de táxi na cidade

Feature	**Quantidade de dados válidos**	**Quantidade de dados nulos**	**Tipo de dados**
ID	63	0	Numérico
Descrição	63	0	Categórico
Latitude	63	0	Numérico
Longitude	62	1	Numérico

Fonte: Elaborado pelo Autor

Após essa análise iniciou-se a fase de engenharia de *features* onde trabalhamos em cima das *features* identificadas.

6.2 Engenharia de *features*

O primeiro passo foi definir quais *features* do *dataset* seriam usadas e quais *features* seriam criadas, isso demanda tempo e testes, foram realizados vários testes com essas *features* e observado o comportamento delas nos modelos com as *features* criadas.

Primeiramente foram removidos ORIGIN_CALL e ORIGIN_STAND os dados contidos nessas *features* não foram usados devido a quantidade de valores nulos contidos nelas.

Também foram removidos do modelo final as colunas TRIP_ID e TAXI_ID, como o preditor proposto pelo trabalho pretende atender de forma geral a proposta de predição de tempo de viagem, dessa forma essas colunas do *dataset* não se encaixam bem no tipo de informação que precisamos.

Outra *feature* que também foi removida foi a CALL_TYPE, ela contém informação importante quando o contexto e voltado para viagem de táxi, neste caso queremos deixar o contexto mais geral possível, então cada viagem no nosso *dataset* é tratada como partida de um ponto A a um ponto B, independente se o ponto de partida é a casa do usuário ou não.

Após algumas adaptações e testes com as *features* restantes foram criadas as seguinte *features*.

MONTH: esse valor é derivado do TIMESTAMP da base de dados que foi usada, essa *feature* guarda o mês em que ocorreu a viagem, com os valores de 1 a 12 representando seus respectivos meses.

DAY_OF_WEEK: esse valor também é derivado do TIMESTAMP da base de dados que foi usada, essa *feature* guarda o dia da semana em que ocorreu a viagem, com os valores de 1 a 7 representando seus respectivos dias da semana, onde o primeiro dia da semana é considerado como a segunda.

TIME_SECOND_START: esse valor também é derivado do TIMESTAMP da base de dados que foi usada, essa *feature* guarda o instante de hora em que começou a viagem, esse instante é salvo em segundos e organizado em trechos de 5 minutos.

LONGITUDE_INICIAL: esse valor deriva da *feature* POLYLINE e guarda o primeiro valor de Longitude capturado pelos sensores de GPS dos táxis.

LATITUDE_INICIAL: esse valor deriva da *feature* POLYLINE e guarda o primeiro valor de Latitude capturado pelos sensores de GPS dos táxis.

LONGITUDE_FINAL: esse valor deriva da *feature* POLYLINE e guarda o último valor de Longitude capturado pelos sensores de GPS dos táxis.

LATITUDE_FINAL: esse valor deriva da *feature* POLYLINE e guarda o último valor de Latitude capturado pelos sensores de GPS dos táxis.

As três *features* descritas a seguir são obtidas a partir do cálculo de distância entre 2 pontos de latitude e longitude usando a fórmula de *Haversine*. esses pontos são a latitude e longitude inicial e final de cada viagem.

DISTANCE_KM_CENTER_START: define o valor de distância entre o centro da cidade e o local onde começou a viagem.

DISTANCE_KM_CENTER_FINAL: define o valor de distância entre o centro da cidade e o local de destino da viagem.

DISTANCE_KM_TRAVEL: define a distância percorrida na viagem.

SPEED: o valor de *speed* é obtido a partir do dados de GPS capturados e salvos na *feature* POLYLINE

JOURNEY_TIME: essa é a nossa *label*, nessa **feature** contém o valor a ser predito, onde ela é obtida a partir da *feature* POLYLINE.

Essas *features* foram criadas usando a *DefineData* que foi definida e criada pelo autor, e está descrita no Apêndice A na página 48.

Nessa classe é realizado um pré processamento, onde são excluídos as *features* que não serão utilizadas e onde são feitas as definições das demais *features*.

A Figura 7 apresenta os dados após passar pelo processo e preparação das *features*.

Figura 7 – Dados Processados

	0	1	2	3	4
MES	7.000000	7.000000	7.000000	7.000000	7.000000
DAY_OF_WEEK	0.000000	0.000000	0.000000	0.000000	0.000000
TIME_SECOND_START	0.000000	300.000000	0.000000	0.000000	0.000000
LONGITUDE_INICIAL	-8.618643	-8.639847	-8.612964	-8.574678	-8.645994
LONGITUDE_FINAL	-8.630838	-8.665740	-8.615970	-8.607996	-8.687268
LATITUDE_INICIAL	41.141412	41.159826	41.140359	41.151951	41.180490
LATITUDE_FINAL	41.154489	41.170671	41.140530	41.142915	41.178087
DISTANCE_KM_CENTER_START	7512.000000	7515.000000	7512.000000	7509.000000	7517.000000
DISTANCE_KM_CENTER_FINAL	7514.000000	7518.000000	7512.000000	7511.000000	7521.000000
DISTANCE_KM_TRAVEL	2.000000	4.000000	1.000000	4.000000	5.000000
SPEED	0.000000	0.000000	0.000000	0.000000	0.000000
JOURNEY_TIME	330.000000	270.000000	960.000000	630.000000	420.000000

Fonte: Elaborado pelo Autor

É importante ressaltar que a escolha dos dados é baseada na *feature* MISSING_DATA, apenas os dados que possuem toda a sua viagem descrita, ou seja o valor Falso nessa *feature*, são utilizados. Qualquer viagem que tenha dados de posição faltando é descartado, pois eles apenas tornam o modelo impreciso. Outro ponto importante é que a classe DefineData foi criada para trabalhar em modelos em modelos de *features* igual ao da nossa base de dados.

6.3 Modelos de predição

Após a definição das *features* a serem utilizadas, iniciou-se a fase de treinamento dos modelos para predição, os algoritmos utilizados para o treinamento desses modelos foram *Lightgbm, K-nearest neighbors, Random Forest, Linear Regression e XGBoost*.

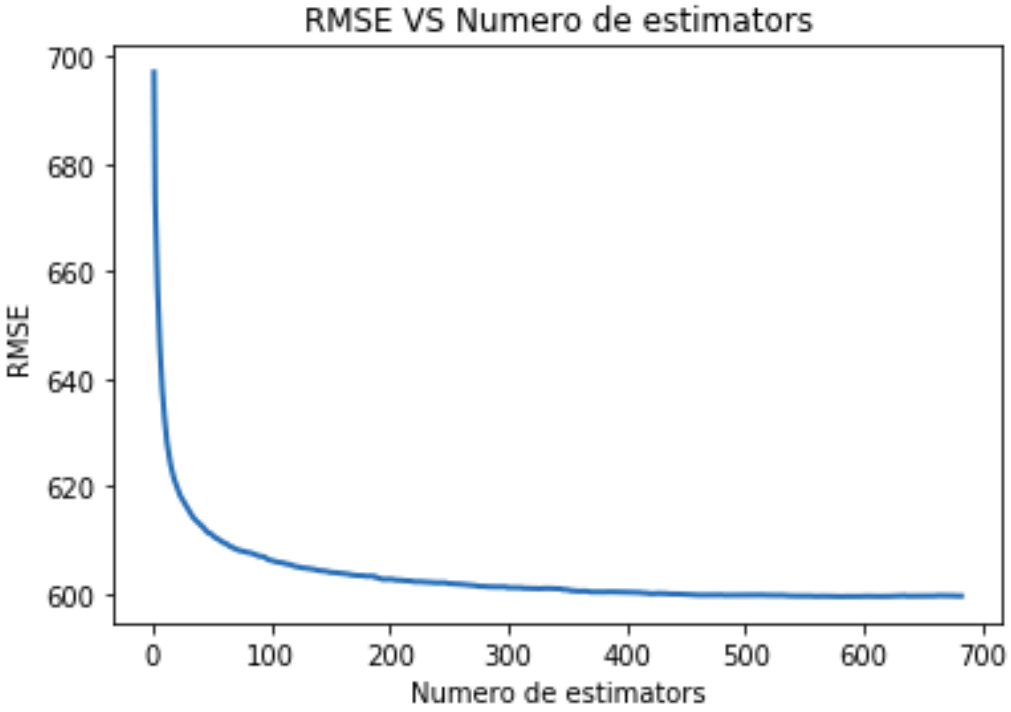
Os modelos foram treinados com 60% dados do originais do *dataset*, sendo que antes de ser recuperado esses dados do *dataset*, esses valores são randomizados, essa abordagem foi realizada pois os dados passados para o modelo estavam provocando *overfitting*, os outros 40% dos dados são usados para validação e testes no modelo.

Cada modelo foi testado usando o mesmo conjunto para testes e através da plataforma Kaggle, para cada um dos modelos é calculado o RMSE, RMSLE e o MAE.

6.3.1 Model 1 - Lightgbm

Este é um modelo baseado em árvores, foi treinado um modelo com o algoritmo Lightgbm usando a técnica de *early stopping* que consiste em aumentar gradualmente o número de árvores usadas pelo algoritmo, na Figura 8 é apresentado um gráfico que demonstra o resultado dos testes.

Figura 8 – Lightgbm RMSE resultados



Fonte: Elaborado pelo Autor

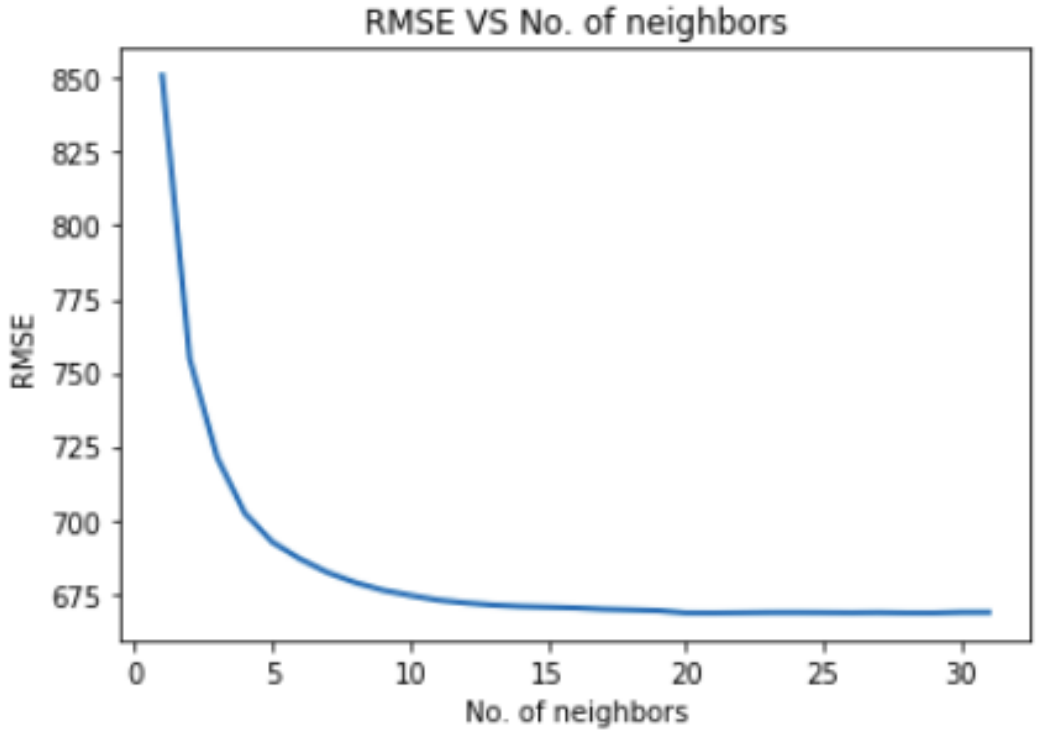
Para esse algoritmo foi definido um *early stopping* igual a 100 e o número máximo de árvores foi igual a 10000. Para esse algoritmo o melhor resultado nos testes de validação do modelo foi com o número de *estimators* igual a 583 com o RMSE igual a 599.42.

6.3.2 Model 2 - K-nearest neighbors

Este é um modelo baseado em vizinhança, foi treinado um modelo com o algoritmo K-nearest neighbors onde esse modelo iria trabalhar em cima das semelhanças das viagens, assim a partir da interpolação local dos alvos associados aos vizinhos mais próximos no conjunto de treinamento teríamos o nosso valor de tempo de viagem. Na Figura 9 é apresentado um gráfico que demonstra o resultado da predição baseado no número de vizinhos que foi utilizado para

cada teste no modelo.

Figura 9 – *K-nearest neighbors* resultados



Fonte: Elaborado pelo Autor

Para esse algoritmo o melhor resultado nos testes de validação do modelo foi utilizando o número de vizinhos igual a 21, obtendo assim um RMSE igual a 668.35.

6.3.3 Model 3 - Random Forest

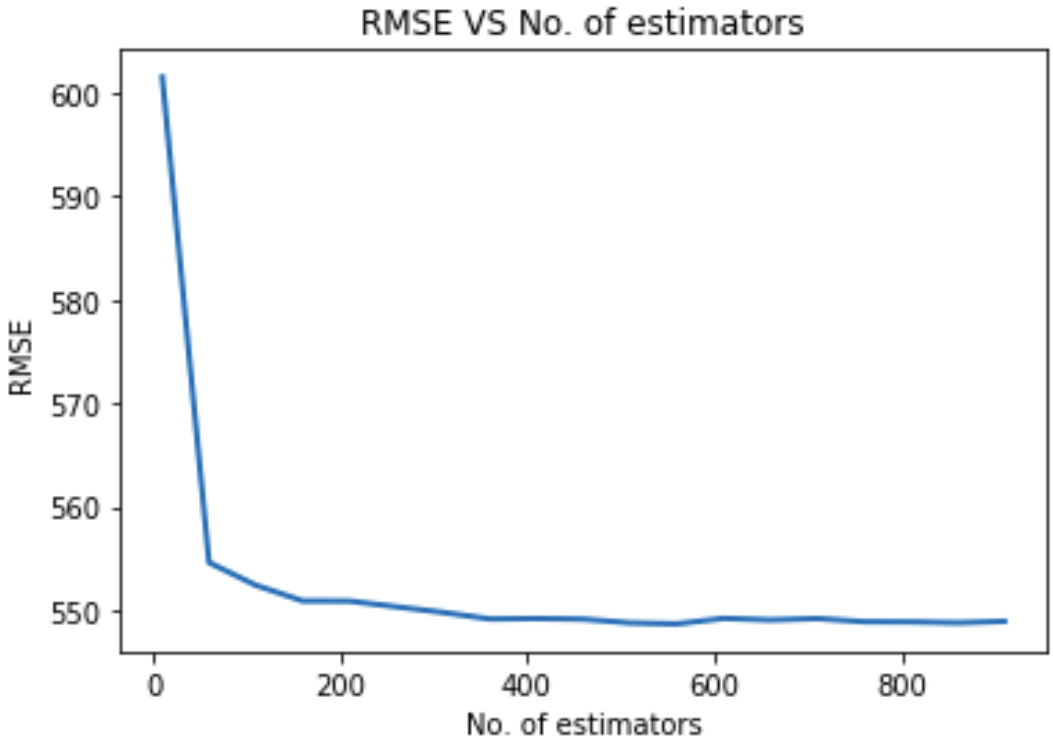
Este é um modelo baseado em árvores, foi treinado um modelo com o algoritmo *Random Forest* implementado pela biblioteca *sklearn* no qual foi utilizado a técnica de *early stopping* para aumentar a eficiência de escolha dos hiperparâmetros do algoritmo, na Figura 10 é apresentado um gráfico que demonstra o resultado após os testes no modelo.

Para esse algoritmo o melhor resultado nos testes de validação do modelo foi com o número de *stimators* igual a 560 com um RMSE igual a 548.67.

6.3.4 Model 4 - Linear Regression

Este é um modelo de regressão linear, foi treinado um modelo com o algoritmo *Linear Regression* onde esse modelo iria criar uma função que melhor representa os dados, assim

Figura 10 – *Random Forest* resultados



Fonte: Elaborado pelo Autor

a partir dessa função teríamos o nosso valor de tempo de viagem. Para esse algoritmo foi tentado aplicar Normalização dos dados, mas não houve melhora do mesmo.

Para esse algoritmo o melhor resultado nos testes de validação do modelo foi com um RMSE igual a 599.42.

6.3.5 Model 5 - XGBoost

Este modelo foi validado usando a biblioteca H2O, onde, o H2O é um software baseado em Java para modelagem de dados e computação geral. O software H2O é muitas coisas, mas o objetivo principal do H2O é como um mecanismo de processamento distribuído (muitas máquinas), paralelo (muitas CPUs), na memória (várias centenas de GBs Xmx). Usando a biblioteca foi possível testar vários hiperparâmetros para o modelo e avaliar qual se adequa melhor para o problema[1].

[1] Documentação H2O, http://h2o-release.s3.amazonaws.com/h2o/master/3574/docs-website/h2o-py/docs/intro.html

6.3.6 Comparação

Após a calibração dos hiperparâmetros dos modelos através do nosso conjunto de teste, testamos cada um dos modelos de forma individual na plataforma *Kaggle*, com o conjunto de dados fornecido pela plataforma, assim foi calculado o dois RMSLE para cada modelo, uma para os dados da competição para o público e o outro para o privado. A Quadro 4 apresenta esses valores para cada um dos modelos e na Quadro 5 apresentação os resultados dos testes com o conjunto de teste criado.

A pontuação final no *Kaggle* é baseada em um subconjunto privado da Quadro de classificação pública do qual se gera o RMSLE privado enquanto o RMSLE público é apenas uma indicação aproximada de sua pontuação final.

Para validar os modelos pela plataforma Kaggle, gerou-se um conjunto de dados das *features* que definimos neste trabalho com os dados de teste fornecidos pela plataforma, assim o resultado é gerando e submetido na plataforma.

Quadro 4 – Resultados dos testes realizados no Kaggle

Modelo	**RMSLE Privado (Kaggle)**	**RMSLE público (Kaggle)**
KNN	0.86704	0.83854
LGBM	0.80222	0.90338
LR	0.80614	0.81143
RF	0.81037	0.96318
XGBM	**0.79958**	**0.77431**

Fonte: Elaborado pelo Autor

Quadro 5 – Resultados dos testes realizados com os dados de teste

Modelo	**RMSE**	**RMSLE**	**MAE**
KNN	650.31	1.02	176.42
LGBM	**575.09**	0.67	201.74
LR	632.18	1.02	201.74
RF	581.84	**0.57**	**138.75**
XGBM	594.23	0.70	159.75

Fonte: Elaborado pelo Autor

6.4 Dificuldades encontradas

Definir as *features* do modelo não é fácil, demanda tempo e poder de processamento, foram executados várias abordagens para tentar melhor representar o problema abordado, muitas

delas descartadas devido a uma péssima predição.

O processo de definição de *features* requer um conhecimento no que se quer modelar, para assim obter uma boa definição, todo esse processo de definição de *features* foi sendo amadurecido com o decorrer do trabalho, essas definições foram ficando mais eficientes e robustas ao decorrer do trabalho.

Utilizamos de *kernels* no Kaggle para realizar os experimentos, esses *kernels* tem uma duração máxima de 9 horas com um poder de processamento de 4 núcleos de processamento e 16 *Gigabytes* de memória RAM.

Definir hiperparâmetros não era intuitivo, foram realizados muitos testes para se chegar a uma abordagem eficiente para a definição desses parâmetros, e foram realizados várias abordagens diferentes para a definição dos mesmos.

Os modelos com árvore demandam de tempo e poder de processamento quando estamos trabalhando com um número significativo de árvores, como tínhamos um tempo limite para utilização do *kernel*, era preciso limitar o número de entradas para trabalhar em cima desses algoritmos de árvores para assim se ter uma base para testar com o conjunto de dados de treino completo.

7 CONCLUSÕES E TRABALHOS FUTUROS

Com base nos valores obtidos o modelo de predição *XGBoost* que obteve um RMSLE de 0.79958 no privado na pontuação do *kaggle* e um RMSE de 594.23, RMSLE de 0.7 e MAE de 159.75 nos testes com o conjunto de dados gerado para os testes dos modelos, sendo o segundo melhor modelo com os dados de teste do trabalho proposto e o melhor nos testes com os dados do kaggle, foi o selecionado para definir o preditor final.

As configurações de melhor hiper parametro desse modelo são *nfolds = 5, keep cross validation models = False, keep cross validation predictions = True, fold assignment = 'Modulo', stopping metric = 'RMSLE', stopping tolerance = 0.005, seed = 1, ntrees = 56, max depth = 5, min rows = 3, learn rate = 0.05, sample rate = 0.8, col sample rate=0.8, col sample rate per tree = 0.8, score tree interval = 5.*

O código que gera o modelo para predição dos valores está descrito no Apêndice B a base de dados está disponível na competição do Kaggle[1]

Como trabalhos futuros, podemos testar hiper parâmetros melhores utilizando maquinas mais robustas, tentar abordagens diferentes, usar combinação de modelos, criar modelos específicos para determinadas situações ou horários, testar novas abordagens com *features* de forma a melhorar o resultado de forma geral não apenas para os dados da competição, testar com novos dados de cidades diferentes.

[1] https://kaggle.com/c/pkdd-15-taxi-trip-time-prediction-ii.

REFERÊNCIAS

BAŞTANLAR, Y.; ÖZUYSAL, M. Introduction to machine learning. In: **miRNomics:**: Microrna biology and computational analysis. [S.l.]: Springer, 2014. p. 105–128.

BOTTOU, L. Feature engineering. **COS 424**, 2010.

BREIMAN, L. Random forests. **Machine learning**, Springer, v. 45, n. 1, p. 5–32, 2001.

CAMILO, C. O.; SILVA, J. C. d. **Mineração de dados:** conceitos, tarefas, métodos e ferramentas: Universidade federal de goiás (ufg). p. 1–29, 2009.

CHEN, T.; HE, T.; BENESTY, M.; KHOTILOVICH, V.; TANG, Y. Xgboost: extreme gradient boosting. **R package version 0.4-2**, p. 1–4, 2015.

CHOPDE, N. R.; NICHAT, M. Landmark based shortest path detection by using a* and haversine formula. **International Journal of Innovative Research in Computer and Communication Engineering**, v. 1, n. 2, p. 298–302, 2013.

FELIPE, H. L.; DIAS, J. W. **Aplicações Baseadas em Geolocalização**. Paranavaí: Universidade Paranaense (UNIPAR), 2017.

GONG, R.; FONSECA, E.; BOGDANOV, D.; SLIZOVSKAIA, O.; GOMEZ, E.; SERRA, X. Acoustic scene classification by fusing lightgbm and vgg-net multichannel predictions. **Proc. IEEE AASP Challenge Detection Classification Acoust.** scenes events. In: . [S.l.: s.n.], 2017. p. 1–4.

JAMES, G.; WITTEN, D.; HASTIE, T.; TIBSHIRANI, R. **An introduction to statistical learning**. [S.l.]: Springer. v. 112, 2013.

LAM, H. T.; DIAZ-AVILES, E.; PASCALE, A.; GKOUFAS, Y.; CHEN, B. (blue) taxi destination and trip time prediction from partial trajectories. **arXiv preprint arXiv:1509.05257**, 2015.

LEE, B. S.; MCDONALD, J. F. Determinants of commuting time and distance for seoul residents: The impact of family status on the commuting of women. **Urban Studies**, Sage Publications Sage UK: London, England, v. 40, n. 7, p. 1283–1302, 2003.

MASTERY, M. L. **Discover Feature Engineering, How to Engineer Features and How to Get Good at It**. 2015. Disponível em: https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/. Acesso em: 20 set 2018.

MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre aprendizado de máquina. **Sistemas Inteligentes-Fundamentos e Aplicações**, v. 1, n. 1, p. 32, 2003.

MOREIRA-MATIAS, L.; GAMA, J.; FERREIRA, M.; MENDES-MOREIRA, J.; DAMAS, L. Predicting taxi–passenger demand using streaming data. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, v. 14, n. 3, p. 1393–1402, 2013.

MOURA, C. **Aprendizado de Máquina:**: conceitos e práticas da área que está movendo o mundo. 2017. Disponível em: https://www.profissionaisti.com.br/2017/12/aprendizado-de-maquina-conceitos-e-praticas-da-area-que-esta-movendo-o-mundo/. Acesso em: 30 out 2018.

POLO, L. F. **Aplicação big data para predição de tempo de viagens de táxi**. Rio Grande do Sul: Universidade Federal do Rio Grande do Sul (UFRS), 2017.

RUSSELL, S. J.; NORVIG, P. **Artificial intelligence**: a modern approach. [S.l.]: Malaysia; Pearson Education Limited,, 2016.

SARKAR, D.; BALI, R.; SHARMA, T. Feature engineering and selection. In: **Practical Machine Learning with Python**. [S.l.]: Springer, 2018. p. 177–253.

WANG, D.; ZHANG, Y.; ZHAO, Y. Lightgbm: an effective mirna classification method in breast cancer patients. In: ACM. **Proceedings of the 2017 International Conference on Computational Biology and Bioinformatics**. [S.l.], 2017. p. 7–11.

ZHANG, Y.; HAGHANI, A. A gradient boosting method to improve travel time prediction. **Transportation Research Part C: Emerging Technologies**, Elsevier, v. 58, p. 308–324, 2015.

ZHAO, J.; GAO, Y.; TANG, J.; ZHU, L.; MA, J. Highway travel time prediction using sparse tensor completion tactics and-nearest neighbor pattern matching method. **Journal of Advanced Transportation**, Hindawi, v. 2018, 2018.

ZHOU, Z.-H. **Ensemble methods**: foundations and algorithms. [S.l.]: Chapman and Hall/CRC, 2012.

APÊNDICE A - CÓDIGO-FONTE UTILIZADO PARA DEFINIÇÃO DE *FEATURES* DO MODELO

Código-fonte 1 – Codigo para definição de *Features*

```
class DefineDate:
    def __init__(self, Data, CenterOfCityLatLon = (41.1622022,-8.657059),
        Testing=False):
        self.Testing = Testing
        if Testing:
            self.model_data = Data
        else:
            self.model_data = Data[Data.MISSING_DATA == False]
        self.CenterOfCityLatLon = CenterOfCityLatLon
        self.defineColumns()


    def get_distance(self, X):
        try:
            ini = (float(X["LONGITUDE_INICIAL"]), float(X["
                LATITUDE_INICIAL"]))
            fim = (float(X["LONGITUDE_FINAL"]), float(X["LATITUDE_FINAL"
                ]))
            return math.ceil(haversine(ini, fim))
        except:
            return None

    def get_distance_center_s(self, X):
        try:
            ini = (float(X["LONGITUDE_INICIAL"]), float(X["
                LATITUDE_INICIAL"]))
            return math.ceil(haversine(ini, self.CenterOfCityLatLon))
        except:
            return None

    def get_distance_center_f(self, X):
        try:
            fim = (float(X["LONGITUDE_FINAL"]), float(X["LATITUDE_FINAL"
                ]))
```

```
30              return math.ceil(haversine(self.CenterOfCityLatLon, fim))
31          except:
32              return None
33
34      def get_Speed(self, X):
35          try:
36              if (X["JOURNEY_TIME"] == 0):
37                  return 0
38              return (X["DISTANCE_KM"] / (X["JOURNEY_TIME"]/3600))
39          except:
40              return 0
41
42      def __Polyline_R(self, x):
43          try:
44              return float(x)
45          except:
46              return None
47
48      def defineColumns(self):
49          datetime = pd.to_datetime(self.model_data["TIMESTAMP"],unit="s")
50          Polyline = self.model_data.POLYLINE.apply(lambda X: X[2:-2].split
                ("],["))
51          self.model_data["MES"]  = datetime.dt.month.values
52          self.model_data["DAY_OF_WEEK"] = datetime.dt.dayofweek
53          self.model_data["TIME_SECOND_START"] = list(map(lambda x: ((x.
                hour * 60) + x.minute) * 60, datetime.dt.floor("5T")))
54          self.model_data["LONGITUDE_INICIAL"] = list(map(lambda X: self.
                __Polyline_R(X[0].split(",")[0]), Polyline))
55          self.model_data["LONGITUDE_FINAL"] = list(map(lambda X: self.
                __Polyline_R(X[-1].split(",")[0]), Polyline))
56          self.model_data["LATITUDE_INICIAL"] = list(map(lambda X: self.
                __Polyline_R(X[0].split(",")[-1]), Polyline))
57          self.model_data["LATITUDE_FINAL"] = list(map(lambda X: self.
                __Polyline_R(X[-1].split(",")[-1]), Polyline))
58          self.model_data["DISTANCE_KM_CENTER_START"] = self.model_data.
                apply(self.get_distance_center_s, axis=1)
59          self.model_data["DISTANCE_KM_CENTER_FINAL"] = self.model_data.
                apply(self.get_distance_center_f, axis=1)
```

```
60          self.model_data["DISTANCE_KM_TRAVEL"] = self.model_data.apply(
                self.get_distance, axis=1)
61          self.model_data["SPEED"] = 0
62          self.model_data["JOURNEY_TIME"] = list(map( lambda x: (len(x)-1)
                *15, Polyline))
63          self.model_data = self.model_data.drop(["CALL_TYPE","POLYLINE","
                ORIGIN_CALL","ORIGIN_STAND","TAXI_ID","MISSING_DATA", "
                DAY_TYPE", "TIMESTAMP"], axis=1)
64          if not self.Testing:
65              self.model_data = self.model_data.drop(["TRIP_ID"], axis=1)
66          self.model_data["SPEED"] = self.model_data.apply(self.get_Speed,
                axis=1)
67          self.model_data = self.model_data.dropna()
68          self.model_data["SPEED"] = list(map(math.ceil,self.model_data["
                SPEED"]))
```

APÊNDICE B - CÓDIGO-FONTE UTILIZADO PARA DEFINIÇÃO DO MODELO

Código-fonte 2 – Codigo para definição do modelo

```
1  import xgboost as xgb
2  import numpy as np
3  import pandas as pd
4  import math
5  import DefineDate
6
7  from haversine import haversine
8  from sklearn.model_selection import train_test_split
9
10
11 df = pd.read_csv("Train.csv")
12
13 modelo_df = DefineData(df)
14
15 X = modelo_df.model_data.values[:,:-1]
16 y = modelo_df.model_data.values[:,-1]
17
18 model = xgb.XGBRegressor(nfolds=5, keep_cross_validation_models=False,
                  keep_cross_validation_predictions=True, fold_assignment="
       Modulo", stopping_metric="RMSLE", stopping_tolerance=0.005, seed=1,
       ntrees=56, max_depth=5, min_rows=3, learn_rate=0.05, sample_rate=0.8,
       col_sample_rate=0.8, col_sample_rate_per_tree=0.8, score_tree_interval
       =5)
19
20 model.fit(X, y)
21 model.save_model("PredicTime.model")
```