UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

JOÃO RICARDO BARP NETO

UTILIZANDO TÉCNICAS DE APRENDIZADO DE MÁQUINA PARA CLASSIFICAR O TEMPO DE RESPOSTA DE PERGUNTAS NO STACK OVERFLOW

CHAPECÓ
2021

JOÃO RICARDO BARP NETO

UTILIZANDO TÉCNICAS DE APRENDIZADO DE MÁQUINA PARA CLASSIFICAR O TEMPO DE RESPOSTA DE PERGUNTAS NO STACK OVERFLOW

Trabalho de conclusão de curso de graduação apresentado como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.
Orientador: Prof. Dr. Denio Duarte

CHAPECÓ
2021

Barp Neto, João Ricardo

Utilizando Técnicas De Aprendizado De Máquina para classificar o tempo de resposta de perguntas no Stack overflow / por João Ricardo Barp Neto. – 2021.

55 f.: il.; 30 cm.

Orientador: Denio Duarte

Monografia (Graduação) - Universidade Federal da Fronteira Sul, Ciência da Computação, Curso de Ciência da Computação, RS, 2021.

I. Duarte, Denio. II. Título.

E-mail: barp.joao@gmail.com

JOÃO RICARDO BARP NETO

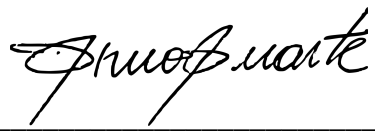
UTILIZANDO TÉCNICAS DE APRENDIZADO DE MÁQUINA PARA CLASSIFICAR O TEMPO DE RESPOSTA DE PERGUNTAS NO STACK OVERFLOW

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.

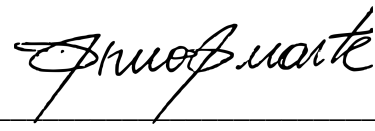
Orientador: Prof. Dr. Denio Duarte

Aprovado em: 16\11\2021

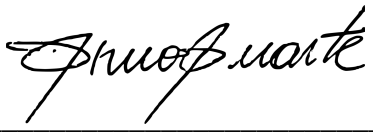
BANCA EXAMINADORA:



Dr. Denio Duarte - UFFS



Ma. Andressa Sebben - UFFS



Dr. Guilherme Dal Bianco - UFFS

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Cleonice Magrin e Idalecio Barp, por me apoiarem do inicio ao fim desta jornada, não medindo esforços para me ajudar a concluir a graduação.

Agradeço ao meu tio Claudecir, por me acolher em sua casa para que pudesse frequentar as aulas. Agradeço também minha avó Maria por sempre me motivar e me apoiar nessa etapa da minha vida.

Agradeço a todos os professores que contribuíram compartilhando seu conhecimento, em especial ao meu orientador Professor. Dr. Denio Duarte por todo o esforço e tempo dedicado para me guiar nesse trabalho de conclusão de curso. Agradeço também a banca, professora Ma. Andressa Sebben e professor Dr. Guilherme Dal Bianco por suas contribuições para a melhoria deste trabalho.

RESUMO

A internet mudou a forma como se compartilha conhecimento. Hoje em dia tornou-se comum pessoas pesquisarem suas dúvidas em mecanismos de buscas, encontrando assim uma vasta gama de páginas, associadas ao assunto buscado. Sites de perguntas e respostas surgem como uma forma rápida e eficaz de compartilhamento de conhecimento, Nestes sites, usuários publicam suas dúvidas e esperam que outros usuários forneçam uma solução para sua questão. Um dos problemas de se ter perguntas respondidas por outros usuários, é que às vezes a resposta pode demorar a ser fornecida. Este trabalho tem como objetivo extrair *features* das perguntas postadas no *Stack Overflow* e treinar um modelo capaz de predizer se uma pergunta vai ter uma resposta em até um dia. Para o treinamento do modelo foi utilizado o algoritmo *extra tree classifier*, foi aplicado sobre o conjunto de dados a técnica *SMOTE* para reamostragem do dados, Os resultados obtidos mostram que o modelo obteve uma precisão de 72%.

Palavras-chave: Stack Overflow; Aprendizado de máquina; Extração de features.

ABSTRACT

The internet has changed the way knowledge is shared. Nowadays it has become common for people to search their queries in search engines, thus finding a wide range of pages, associated with the searched subject. Question and answer sites appear as a quick and effective way of sharing knowledge. On these sites, users post their queries and expect other users to provide a solution to their question. One of the problems with having questions answered by other users is that sometimes the answer can take a while to be provided. This work aims to extract *features* from the questions posted on *Stack Overflow* and train a model capable of predicting whether a question will have an answer within a day. To train the model, the *extra tree classifier* algorithm was used, the *SMOTE* technique was applied to the dataset for resampling the data. The results obtained show that the model had a precision of 72% .

Keywords: Stack Overflow; Machine learning; Feature extraction.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

MLP *Artificial Neural Network*

SUMÁRIO

1 INTRODUÇÃO

Em sites de perguntas e respostas, os usuários publicam perguntas e esperam que outros membros da comunidade forneçam uma solução adequada às suas dúvidas. Os possíveis respondentes, examinam a lista de perguntas existentes, e decidem se contribuem ou não para as discussões em andamento. Tais decisões são influenciadas por vários fatores, como dificuldade da pergunta, sua disponibilidade ou o próprio conhecimento do usuário (YANG et al., 2014).

No entanto, um dos principais problemas de ter perguntas respondidas por outros usuários é que às vezes ninguém fornece uma resposta ou ela pode demorar. Por exemplo, no site de perguntas e respostas *Yahoo! Answers*, uma a cada oito perguntas não são respondidas (YANG et al., 2011).

Este problema se estende a outro site, o *Stack Overflow*, como pode ser observado na Tabela 1.1, que detalha o aumento do número de questões sem resposta ao longo dos anos. A coluna AQ representa o número de questões com respostas e UQ o número de questões sem respostas. A coluna [%] mostra o aumento da porcentagem de questões sem resposta a cada ano, foi de 0.16% em 2008 para 16.67% em 2012.

Year	AQ	UQ	[%]
2008	61.480	100	0.16
2009	350.510	2.799	0.79
2010	698.386	17.481	2.44
2011	1.176.422	102.207	7.99
2012	866.980	173.413	16.67

Tabela 1.1 – Proporção de perguntas não respondidas por ano no *Stack Overflow* (SAHA; SAHA; PERRY, 2013)

Logo, os usuários de sites de perguntas e respostas precisam entender quais fatores fazem uma pergunta ser mais atraente para ser respondida. Uma pergunta bem formulada aumenta a probabilidade dos usuários responderem a questão, e consequentemente aumenta a chance da resposta ser satisfatória (BALTADZHIEVA; CHRUPAŁA, 2015).

A internet mudou a forma como se compartilha conhecimento. Nos dias atuais tornou-se comum pessoas pesquisarem suas dúvidas em mecanismos de buscas, encontrando assim uma vasta gama de páginas potencialmente capazes de fornecer respostas e/ou o conhecimento buscado. Entretanto, nem sempre as respostas ou o conhecimento buscado são os mais satisfatórios, ou em alguns casos, nem são encontradas (BALTADZHIEVA; CHRUPAŁA, 2015).

Sites de perguntas e respostas surgem como uma forma de compartilhamento de conhecimento, segundo (BALTADZHIEVA; CHRUPAŁA, 2015), mais rápido e eficaz, substituindo outras formas, como documentos e banco de dados. Diferente de inúmeros sites de perguntas e respostas que cobrem uma gama grande de assuntos, o *Stack Overflow* surgiu com o objetivo de conquistar um público alvo mais específico: os programadores e os engenheiros de software (ASADUZZAMAN et al., 2013).

Segundo (BALTADZHIEVA; CHRUPAŁA, 2015), não receber uma resposta, ou apenas um número limitado de respostas, não é apenas decepcionante, mas também pode acarretar desvantagem educacional ou profissional. Segundo (ASADUZZAMAN et al., 2013), para gerir o crescimento das perguntas sem respostas ou com respostas demoradas, pode ser importante que o *Stack Overflow* introduza um mecanismo que ajude a diminuir o número de questões não respondidas.

Deste modo, este trabalho apresenta todas as etapas para a criação do modelo que propõe classificar o tempo de resposta das pergunta feitas no *Stack Overflow*. A partir da base de dados obtida do *Stack Overflow*, foram extraídas *features* de cada pergunta postada no período de 2008 à 2019, junto com e o tempo em minutos que a pergunta ficou sem resposta. O tempo em minutos é importante para a definição das classes propostas, por exemplo a classe de perguntas com respostas de até um dia. Devido ao desbalanceamento das classes, foi aplicada a técnica *SMOTE* sobre a base de dados. Essa técnica tem como objetivo balancear as classes minoritárias criando novos exemplos sintéticos (CHAWLA et al., 2002). Após experimentos utilizando alguns algoritmos classificadores, foi definido que o modelo seria produzido utilizando o algoritmo *Extra Trees Classifier*.

Foram produzidos dois modelos com duas e quatro classes. O modelo com duas classes mostrou-se eficiente para predizer se uma pergunta vai ter resposta em um dia com 72% de precisão, obtendo também 71% de precisão para perguntas com respostas mais demoradas ou não fornecidas. Os resultados demonstram que o modelo pode ser útil na previsão desses dois tipos de situação quando uma pergunta é postada no *Stack Overflow*. O segundo modelo, com quatro classes, obteve uma precisão de 63% para perguntas respondidas em até um dia. Resultados similares foram encontrados para as outras três classes.

Este trabalho está estruturado em 6 capítulos. No capítulo 1 foi apresentada a introdução. No capítulo 2 será apresentado o site *Stack Overflow*. No capítulo 3 será apresentada uma introdução ao aprendizado de máquina, com técnicas de extração de *features*, seleção de *featu-*

res, categorização, método de reamostragem de dados e métodos para avaliação dos modelos gerados. No capítulo 4 serão destacados alguns trabalhos relacionados. No capítulo 5 são apresentados os experimentos realizados, bem como os resultados finais. O capítulo 6 apresenta a conclusão.

2 STACK OVERFLOW

O *Stack Overflow* é um site de perguntas e respostas voltado para os engenheiros de software e programadores. Surgiu em 2008 com Jeff Atwood e Joel Spolsky, trazendo membros de suas próprias comunidades para o *Stack Overflow*. O site surgiu como um dos maiores da área. Nele inúmeros membros respondem ou participam de discussões, a fim de buscar soluções para diversos problemas (ASADUZZAMAN et al., 2013). Desde sua fundação, o número de usuários vem crescendo: no primeiro ano de existência eram cerca de 53 mil membros, já em 2012 passou para 1.2 milhões (ASADUZZAMAN et al., 2013).

O estímulo para o uso do site se dá através de bonificações, como por exemplo, o conceito de reputação, o qual indica o quanto um determinado membro do site participa em perguntas e respostas. Quando usuários atingem uma reputação alta, podem ser concedidos alguns privilégios, por exemplo, ter privilégios para o controle do site, como os moderadores oficiais. A reputação aumenta de acordo com a iteração do usuário no site, de modo que boas publicações rendem *up* de outros usuários que é a forma de mostrar que aquela publicação contribuiu com o site, fazendo assim a reputação do usuário subir. Em contrapartida, receber *down* nas publicações, indica publicações ruins ou irrelevantes para o site, fazendo a reputação do autor diminuir.

Figura 2.1 – Exemplo de pergunta e resposta no *Stack Overflow* (STACKOVERFLOW, 2018)

Como mostra a Figura 2.1, perguntas e respostas contam com uma série de dados. A

área destacada em (1) representa a pergunta do usuário, e (2) o seu detalhamento. A área (3) agrupa as Tags utilizadas para indicar um possível assunto para a pergunta. Em (4) são mostradas as datas da postagem, existentes tanto na pergunta como nas respostas. A área (5) destaca os autores da pergunta e da resposta. Em (6) é mostrada a postagem escolhida como a melhor resposta, complementada pelo indicador de melhor resposta em (8). A área (7) mostra o número de votos da pergunta, com o mecanismo de votos *up* ou *down* para perguntas e respostas. Esse sistema de votos é uma forma de avaliação de perguntas e respostas e mede o quanto uma pergunta ou resposta foi útil para a comunidade.

Este capítulo apresentou, brevemente, a organização do *Stack Overflow* e como as postagens são estruturadas. Este conhecimento é importante para o processo de extração das *features* necessárias para este trabalho. A obtenção do melhor conjunto de *features* possibilitará um melhor desempenho dos algoritmos de classificação.

3 APRENDIZADO DE MÁQUINA

Aprendizado de máquina é um campo da computação que visa a construção de algoritmos que possam aprender utilizando um conjunto de dados como entrada. Ou seja, através de treinamentos e conhecimentos passados o algoritmo passa a aprender determinada tarefa, ou melhorar seu desempenho nela. Essa área cresceu nos últimos anos, hoje em dia são construídos modelos cada vez mais complexos, devido ao aumento do poder computacional (SENDERS et al., 2018).

Os algoritmos de aprendizado de máquina são utilizados geralmente para problemas onde há uma grande complexidade para a programação, incluindo atividades realizadas por seres humanos, como direção de automóveis ou reconhecimentos de fala e imagens. Outros problemas são as tarefas muito difíceis para seres humanos, como previsões do tempo, ou análise de grandes quantidades de dados. E por fim problemas onde os algoritmos precisam de uma adaptação ao conjunto de dados, como programas para detecção de *spam* por exemplo (SHALEV-SHWARTZ; BEN-DAVID, 2014).

Os algoritmos de aprendizado de máquina podem ser classificados, de forma genérica, de duas formas: aprendizado supervisionado e não supervisionado.

Os algoritmos de aprendizagem supervisionada recebem dados rotulados, ou seja, a entrada e a saída esperada são conhecidas. Um exemplo é a detecção de *spams*, onde cada e-mail do conjunto de treinamento receberá um rótulo de *spam* ou não *spam*. A partir disso o algoritmo deve aprender com os dados recebidos no conjunto de treinamento, passando a ser capaz de identificar o rótulo dos dados que não estão rotulados (SHALEV-SHWARTZ; BEN-DAVID, 2014).

Os algoritmos supervisionados podem ser divididos de acordo com o tipo de seu rótulo: classificadores, para rótulos com valores discretos, e algoritmos de regressão, para rótulos com valores contínuos (LORENA; CARVALHO, 2007) .

Já os algoritmos de aprendizagem não supervisionada trabalham com dados não rotulados. Sua principal função é encontrar semelhanças entre os dados ou agrupá-los (SHALEV-SHWARTZ; BEN-DAVID, 2014).

3.1 Extração de *features*

Um sistema de aprendizado de máquina é um algoritmo que toma decisões baseado em experiências acumuladas. Algoritmos de aprendizagem de máquina recebem um conjunto de *features*[1] junto com suas respectivas classes.

Classificadores buscam moldar a estrutura de classificação para um problema específico, encontrando uma forma genérica para relatar um padrão. Assim, para um bom desempenho dos algoritmos é necessário um conjunto de *features* relevantes para a tarefa em questão (LEE, 2000; KOZAREVA; MONTOYO, 2006).

A Tabela 3.1 apresenta um exemplo esquemático de como é formado um conjunto de exemplos para treinamento. Cada exemplo E_i consiste de n *features* $X_1, X_2, \cdots, X_n$ e uma classe Y associada.

	Features				
Exemplos	X_1	X_2	$\cdots$	X_n	$Classe(Y)$
E_1	x_{11}	x_{12}	$\cdots$	x_{1n}	y_1
E_2	x_{21}	x_{22}	$\cdots$	x_{2n}	y_2
E_3	x_{31}	x_{32}	$\cdots$	x_{3n}	y_3
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
E_m	x_{m1}	x_{m2}	$\cdots$	x_{mn}	y_m

Tabela 3.1 – Formato de um conjunto de exemplos (LEE, 2000)

Alguns trabalhos que propõem abordagens para caracterizar perguntas e respostas no *Stack Overflow* (KNOCHENHAUER; DORNELES; WIVES, 2018; ASADUZZAMAN et al., 2013; SAHA; SAHA; PERRY, 2013) já propuseram uma série de *features* que auxiliam nesse processo.

3.2 Seleção de Features

Após a extração do conjunto de *features* é, necessário a utilização de métodos de seleção de *features* para obter as aquelas de maior relevância para a tarefa de classificação (KNOCHENHAUER; DORNELES; WIVES, 2018). *Features* irrelevantes geralmente prejudicam a precisão do modelo de classificação, além de serem uma fonte de ineficiência computacional. Portanto, o objetivo dos algoritmos de seleção de *features* é selecionar os recursos mais informativos em relação ao rótulo da classe (AGGARWAL, 2015). Dentre as técnicas utilizadas,

[1] Características

o ganho de informação e o índice de GINI são as mais utilizadas. Ambas são apresentadas, brevemente, a seguir.

3.2.1 Ganho de informação

Para definir o ganho de informação, primeiro é necessário o realizar o cálculo da entropia, uma medida que caracteriza a impureza de uma coleção arbitrária de exemplos. A fórmula da entropia para uma coleção *S* de exemplos, contendo exemplos positivos e negativo é mostrada na Equação 3.1 (MITCHELL, 1997).

$$\textit{Entropia}(S) = -P_{\oplus} \log 2(P_{\oplus}) - P_{\ominus} \log 2(P_{\ominus}) \tag{3.1}$$

Onde:

- S é o conjunto de exemplos;
- $P\oplus$ é a proporção de exemplos positivos;
- $P\ominus$ é a proporção de exemplos negativos.

A Equação 3.1 descreve uma classificação booleana, mas nem sempre é assim, geralmente o atributo podem assumir N diferentes valores, então é necessária a equação a seguir:

$$\textit{Entropia}(S) = \sum_{i=1}^{N} -P_i \log 2(P_i) \tag{3.2}$$

Onde:

- S é o conjunto de exemplos;
- Pi é a proporção de exemplos da classe *i*.

Após o cálculo da entropia, pode-se então calcular o ganho de informação, que é a medida de eficácia de um atributo na classificação dos dados de treinamento. O ganho de informação, $Gain(S, A)$ de um atributo A, relativo a uma coleção de exemplos S, é definido como:

$$\textit{Gain(S,A)} = \textit{Entropia}(S) - \sum_{v \in Valores(A)} \frac{|S_v|}{|S|} Entropia(S_v) \tag{3.3}$$

- $Valores(A)$ é o conjunto de todos os valores possíveis para o atributo A.

- S_v é o subconjunto de S para o qual o atributo A possui valor v
  (isto é, S_v, = s ∈ S | A(s) = v))

3.2.2 Índice de GINI

O índice de Gini é uma medida de avaliação da pureza para um conjunto. A utilidade da *feature* para a classificação pode ser medida pela pureza do recurso (ZHU; LIN, 2013), ou seja, quanto mais próximo de 0 possível.

O índice de Gini mede a impureza de D, um conjunto de tuplas de treinamento, como

$$\textit{Gini(D)} = 1 - \sum_{j=1}^{m} P^2 i \tag{3.4}$$

onde Pi é a probabilidade de que uma tupla em D pertença à classe Ci e seja estimada por $|Ci, D|/|D|$. A soma é calculada sobre m classes. O índice de GINI é similar à entropia, porém não usa o log no cálculo, tornando-o assim mais eficiente computacionalmente.

3.3 Categorização

Para o treinamento do conjunto de dados, será necessário, a criação de classes a partir do intervalo de tempo calculado entre a postagem da pergunta e a primeira resposta. Os valores são contínuos e como no trabalho de (ASADUZZAMAN et al., 2013) vide Capítulo 4, esses valores serão discretizados. A seguir um método é apresentado.

3.3.1 Quantis

Os quantis são medidas de tendência central que permitem dividir uma amostra ou distribuição em partes iguais. Um caso particular dos quantis é a mediana, que permite dividir a distribuição em duas partes iguais. Os quantis mais conhecidos são: os quartis que dividem a distribuição em quatro partes iguais; os decis que dividem a distribuição em dez partes iguais; e os percentis que dividem a distribuição em cem partes iguais (OLIVEIRA; OLIVEIRA, 2011).

O cálculo da mediana é descrito na formula 3.5, onde n é o número de elementos da série. A série precisa estar ordenada em ordem crescente para o cálculo da mediana.

$$f(n) = \begin{cases} \frac{n+1}{2}, & \text{se } n \text{ é par} \\ \frac{\frac{n}{2}+\frac{n+1}{2}}{2}, & \text{se } n \text{ é ímpar} \end{cases} \tag{3.5}$$

Um exemplo para o cálculo da mediana é a partir da série { 1, 3, 5, 6, 9, 15, 25}, onde n=7, a mediana para a série será $\frac{7+1}{2} = 4$, ou seja, o quarto elemento da série, que é o número 6.

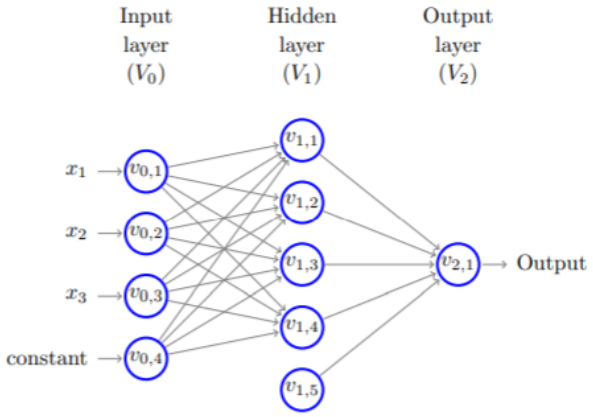
3.4 Técnicas supervisionadas de aprendizado de máquina

Este trabalho se apoia em algoritmos supervisionados. A seguir, algumas técnicas supervisionadas são apresentadas.

3.4.1 *Artificial Neural Network*

Uma rede neural artificial é um modelo de computação inspirado na estrutura de redes neurais no cérebro. Em modelos simplificados do cérebro, consiste em um grande número de dispositivos básicos que estão conectados a cada outro em uma rede de comunicação complexa, através da qual o cérebro é capaz de realizar cálculos altamente complexos. Uma rede neural pode ser descrita como um grafo direcionado cujos nós correspondem aos neurônios e as bordas correspondem às ligações entre eles. Cada neurônio recebe, como entrada, uma soma ponderada das saídas dos neurônios conectados à sua entrada arestas.

Figura 3.1 – Exemplo de *Artificial Neural Network* (SHALEV-SHWARTZ; BEN-DAVID, 2014)



Na Figura 3.1 é exemplificada a estrutura básica de uma *Neural Network*, onde *x1, x2, x3* e *constant* são entradas da rede. *V0* é a camada de entrada, responsável por conter neurônios com proporção equivalente à dimensão da entrada. *V1*, denominada camada oculta da rede (*hidden layer*), pode conter um número variável de neurônios (neste caso cinco). E por fim, *V2* responsável pelo resultado processado na rede. Vale ressaltar que todas as camada que estão entre a camada de entrada e saída são denominadas camadas escondidas

3.4.2 Decision Trees

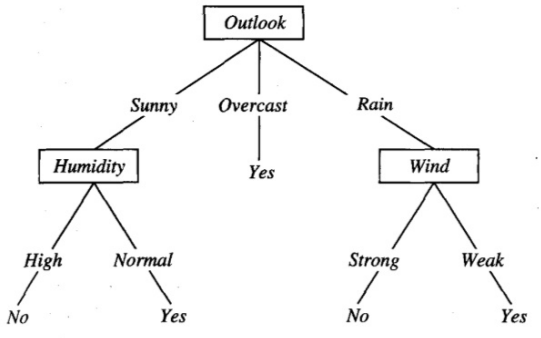
Uma árvore de decisão é uma técnica de aprendizado de máquina que prevê um rótulo associado a uma instância passando pelo nó raiz até o nó folha de uma árvore (SHALEV-SHWARTZ; BEN-DAVID, 2014). Cada nó na árvore específica o teste de algum atributo da instância e cada ramificação descendente desse nó corresponde a um dos valores possíveis para esse atributo (MITCHELL, 1997).

Um dos passos mais importantes para a construção de uma árvore é identificar como os atributos serão distribuídos na árvore, desde o nó raiz até os nós descendentes da raiz.

Para a identificar as posições dos atributos na árvore é utilizado uma propriedade estatística chamada ganho de informação (MITCHELL, 1997), brevemente apresentada anteriormente.

A Figura 3.2 mostra uma árvore de decisão para a atividade de jogar tênis em ambiente aberto. A árvore conta com três nós que correspondem à três *features*: *Outlook, Humidity* e *Wind*. A classificação começa a partir da *feature Outlook* que é o nó raiz: caso a perspectiva seja de sol, a classificação desce ao nó *Humidity*, nesse nó é feito a classificação de jogar ou não tênis, se a umidade for alta, não é recomendada a atividade de jogar tênis, caso a umidade seja normal, a atividade é recomendada.

Figura 3.2 – Exemplo de árvore de decisão (MITCHELL, 1997)



3.4.3 *Randon forest*

Uma floresta aleatória é um classificador que consiste em uma coleção de árvores de decisão, onde cada árvore é construída aplicando um algoritmo A no conjunto de treinamento S e um vetor aleatório adicional θ, onde θ é amostrado por variáveis aleatórias de alguma distribuição. A previsão da *Randon forest* é obtida por maioria de votos sobre as previsões das

árvores individuais (SHALEV-SHWARTZ; BEN-DAVID, 2014).

3.4.4 *Extra Trees Classifier*

Árvores extras ou árvores extremamente aleatórias foram introduzidas por (GEURTS; ERNST; WEHENKEL, 2006) e adicionam outra camada de aleatoriedade às florestas de decisão. A etapa de randomização adicional é introduzida no nó durante o treinamento da árvore: Em vez de procurar o ponto de corte ideal, ou seja, limite τ, um valor de limite aleatório é selecionado para cada característica. Posteriormente, o espaço de busca é reduzido, levando a um treinamento mais rápido. No lado negativo, o tamanho / profundidade da floresta é aumentado devido à introdução de cortes abaixo do ideal (MAIER et al., 2015).

3.4.5 *Regressão logística*

Este algoritmo é usado para classificar os indivíduos nas categorias com base na função logística. Existem muitos casos quando não se obtém um gráfico perfeito que se encaixe em todos os pontos de dados. Por exemplo, podemos encontrar problemas como o gráfico mencionado na Figura 3.4.5(a).

O gráfico na Figura 3.4.5(a) mostra como a ação varia com respeito à idade. Então, este gráfico não é apropriado e não se ajusta a todos os pontos de dados. A solução é implementar um algoritmo de regressão logística. Quando aplicamos este algoritmo para este conjunto de pontos de dados, obtemos o gráfico conforme desenhado na Figura 3.4.5(b). Este gráfico é muito apropriado, pois ele se encaixa perfeitamente a todos os pontos de dados. Este gráfico ou curva podem ser adequadamente visualizados conforme desenhado na Figura 3.4.

Esta é a especialidade por trás do uso de regressão logística. A curva da Figura 3.4 é obtida devido ao uso da função sigmóide na regressão logística. A função sigmóide é uma função matemática que é responsável por esta curva em forma de *S*. É um caso especial de regressão logística. Para entender a versão matemática, começamos com uma fórmula de regressão linear simples:

$$y = b0 + b1 * x \tag{3.6}$$

Então, agora a função sigmóide é aplicada nele, e é dado pela fórmula

$$p = \frac{1}{1 + e^{-y}} \tag{3.7}$$

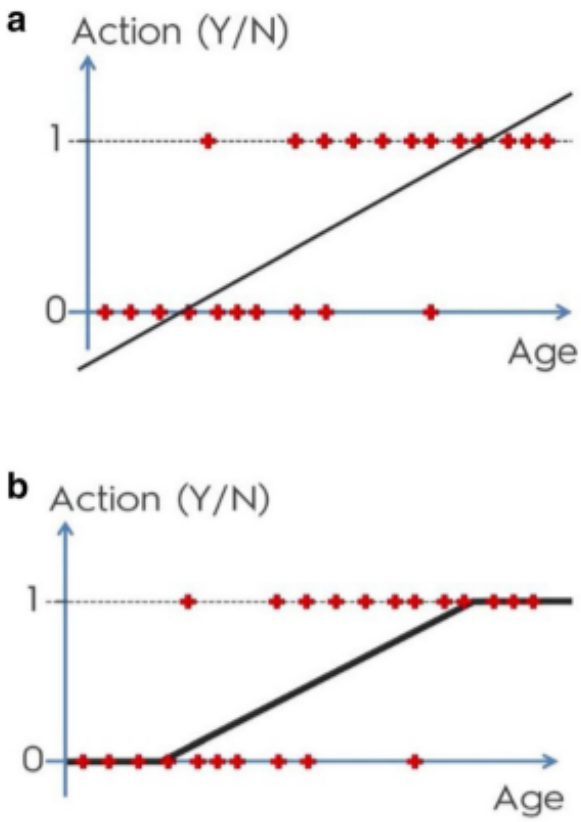


Figura 3.3 – Exemplo de classificação com Regressão logística (SHAH et al., 2020)

Agora, o valor de y é calculado substituindo uma fórmula pela outra; obtemos nossa fórmula de regressão logística como:

$$\ln\left(\frac{p}{1-p}\right) = b0 + b1 * x \tag{3.8}$$

ou

$$logit(S) = b0 + b1M1 + b2M2 + b3M3...bkMk... \tag{3.9}$$

onde, S é a probabilidade da presença de características de interesse. $M1, M2, M3...Mk$ são o valor preditor e $b0, b1, b2, b3...bk$ são a interceptação do modelo.

3.4.6 Máquinas de vetores suporte (SVMs)

Máquinas de vetores suporte são uma técnica de aprendizado de máquina que usa um mapeamento não linear para transformar os dados de treinamento originais em uma dimensão mais alta. Com essa nova dimensão, ele procura pelo hiperplano de separação ideal linear, ou seja, um "limite de decisão" que separa as tuplas de uma classe da outra. Com um mapeamento

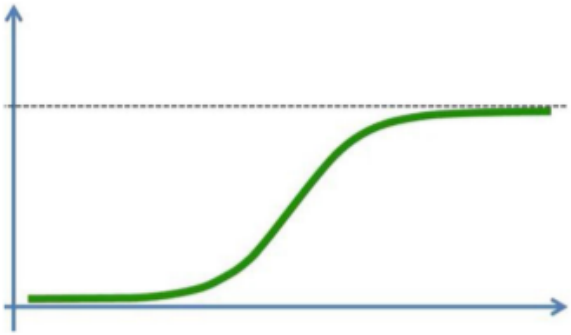
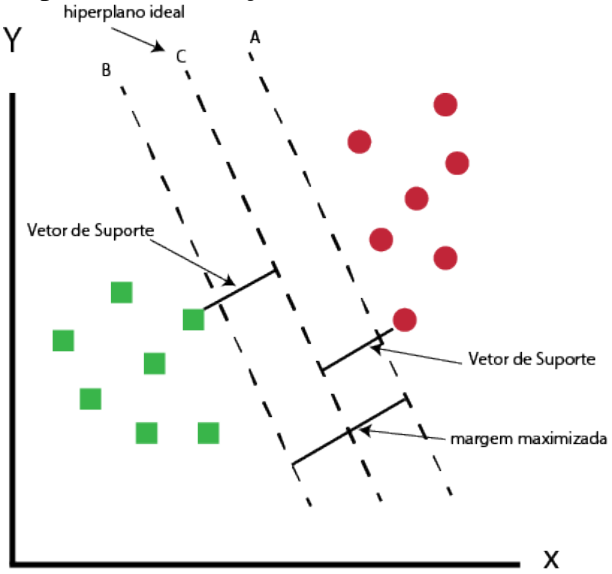


Figura 3.4 – curva sigmóide (SHAH et al., 2020)

não linear apropriado para uma dimensão suficientemente alta, os dados de duas classes sempre podem ser separados por um hiperplano. O SVMs encontra este hiperplano usando vetores de suporte que são tuplas de treinamento "essenciais" e margens definidas pelos vetores de suporte (HAN; PEI; KAMBER, 2011).

As SVMs são originalmente utilizadas para classificação dos dados em duas classes distintas. Contudo, muitas aplicações envolvem o agrupamento em mais de duas classes. Este fato não inviabiliza o uso das SVMs. Diversas técnicas são propostas para estendê-las a problemas multi classes. Um desses métodos é a decomposição "um-contra-todos", que consiste na geração de k SVMs, onde k é o número de classes. Na criação de cada uma dessas máquinas, uma classe é fixada como positiva e as restantes como negativas (LORENA; CARVALHO, 2003).

Figura 3.5 – Exemplo de classificação com SVMs (CODIGOFLUENTE, 2019)



A Figura 3.5 apresenta a classificação linear de duas classes (círculos e quadrados) com Máquinas de vetores suporte, as retas A e B representam as margens da classificação que são

definidas pelos vetores de suporte (tuplas de treinamento) e a reta C que representa o hiperplano de separação ideal das classes.

3.5 Métricas de avaliação

Nesta seção serão apresentadas algumas métricas para avaliação de algoritmos classificadores.

Para o cálculo das métricas propostas, primeiro é necessário a formação da matriz de confusão, que possui quatro conceitos importantes:

- *True Positive (TP)* : exemplo positivo classificado de forma correta;
- *False Positive (FP)*: exemplo positivo classificado de forma incorreta;
- *False Negative (FN)*: exemplo negativo classificado de forma incorreta;
- *True Negative (TN)* : exemplo negativo classificado de forma correta.

A Matriz de confusão é uma matriz de tamanho M, onde M é o número de classes. As linhas correspondem à classe atual e colunas às classes preditoras (HAN; PEI; KAMBER, 2011). A Tabela 3.5 apresenta um esquema para casos de classificação binária, ou seja, M=2.

	Classe prevista		
Classe atual		Sim	Não
	Sim	TP	FN
	Não	FP	TN

Tabela 3.2 – *Confusion matrix* (HAN; PEI; KAMBER, 2011)

3.5.1 Acurácia

Acurácia é a taxa de amostras positivas e negativas que foram classificadas corretamente (HAN; PEI; KAMBER, 2011). A acurácia é descrita na Fórmula 3.10. É composta pela taxa de verdadeiros positivos (TP) mais verdadeiros negativos (TN) dividido pelor total de tuplas positivas (P) mais total de tuplas negativas (N).

Acurácia é uma métrica bastante utilizada, porém apresenta um problema quando as classes são desbalanceadas. Por exemplo, se existem duas classes em um conjunto de 1000 exemplos e uma das classes corresponde a 900 exemplos, caso o modelo sempre prediga a

classe dominante, a acurácia será de 90%, que estaticamente parece bom, mas o modelo nunca prediz a outra classe. Nesses casos, a revocação e a precisão, descritas a seguir, são as mais indicadas.

$$\textit{Acurácia} = \frac{TP + TN}{P + N} \tag{3.10}$$

3.5.2 Revocação

Revocação é uma medida de completude que corresponde à taxa de exemplos positivos classificados corretamente. Uma pontuação de revocação perfeita *i. e.*, 1 para X significa que cada item da classe X foi rotulado como tal, mas não nos diz quantas outras tuplas foram incorretamente rotuladas como pertencentes à classe X (HAN; PEI; KAMBER, 2011). A revocação é descrita na Fórmula 3.11. É a taxa de verdadeiros positivos (TP) dividido por verdadeiros positivos (TP) mais falsos negativos (FN).

$$\textit{Revocação} = \frac{TP}{TP + FN} \tag{3.11}$$

3.5.3 Precisão

A precisão pode ser pensada como uma medida de exatidão: é a taxa de exemplos positivos previstos que são de fato positivos. Por exemplo, uma pontuação perfeita *i. e.*, 1 de precisão para uma classe X significa que toda tupla que o classificador classificou como pertencente à classe X de fato pertence à classe X. No entanto, ela não nos diz nada sobre o número de tuplas de classe X que o classificador atribuiu erroneamente (HAN; PEI; KAMBER, 2011). A fórmula da Precisão é descrita na Fórmula 3.12. É a taxa de verdadeiros positivos (TP) dividido por verdadeiros positivos (TP) mais falsos positivos (FP).

$$\textit{Precisão} = \frac{TP}{TP + FP} \tag{3.12}$$

3.5.4 *F-β Measure*

F-β Measure é a média harmônica entre precisão e revocação (HAN; PEI; KAMBER, 2011). O *F-β Measure* é utilizado quando a precisão e a revocação são importantes para o modelo. A medida F-β mais utilizada é o F-1. A fórmula do *F-β Measure* é descrita na fórmula

3.13. É 2 vezes a precisão vezes revocação dividido pela soma da precisão e da revocação.

$$F\text{-}\beta\ Measure = \frac{2 \times Precisão \times Revocação}{Precisão + Revocação} \tag{3.13}$$

3.6 *SMOTE*

Existem várias abordagens para lidar com o problema de desequilíbrio de classes. O problema pode ser resolvido de forma independente do classificador, equilibrando o conjunto de treinamento artificialmente antes da construção do modelo. Essa estratégia é conhecida como reamostragem de dados (MALDONADO; LÓPEZ; VAIRETTI, 2019).

A reamostragem de dados pode ser feita reduzindo o tamanho da classe predominante por meio do descarte de instâncias, conhecida como subamostragem. Também, pode-se adicionar novas amostras à classe minoritária, que é conhecido como sobreamostragem (MALDONADO; LÓPEZ; VAIRETTI, 2019). Neste trabalho será utilizada a sobreamostragem.

A sobreamostragem pode ser realizada simplesmente replicando os elementos existentes da classe minoritária no conjunto de treinamento. Esta estratégia, no entanto, é conhecido por ser propensa a *overfitting*, problema que consiste em bom desempenho durante o treino do modelo e desempenho ruim sobre o conjunto de teste. Para evitar esse risco, as novas amostras podem ser criadas artificialmente, respeitando a distribuição da classe minoritária (MALDONADO; LÓPEZ; VAIRETTI, 2019).

Uma dessas abordagens é a Técnica de Sobreamostragem de Minoria Sintética (*SMOTE - Synthetic Minority Oversampling TEchnique)* (CHAWLA et al., 2002), que tem duas etapas principais: primeiro, uma vizinhança é definida para cada elemento da classe minoritária, identificando os k vizinhos mais próximos. Em seguida, $N < k$ elementos da vizinhança são aleatoriamente selecionados e usados para construir novas amostras via interpolação (CHAWLA et al., 2002).

Este método tem as vantagens de ser rápido para calcular e bem-sucedido em fornecer desempenho de classificação equilibrado e preciso. Uma vez que SMOTE é independente do classificador, ele pode ser usado com qualquer técnica de classificação (MALDONADO; LÓPEZ; VAIRETTI, 2019).

A Figura 3.6 mostra a plotagem de uma base de dados antes e depois da aplicação do *SMOTE*, à esquerda a base de dados original, com predominância da classe 0, à direita a base de dados após a aplicação do *SMOTE*.

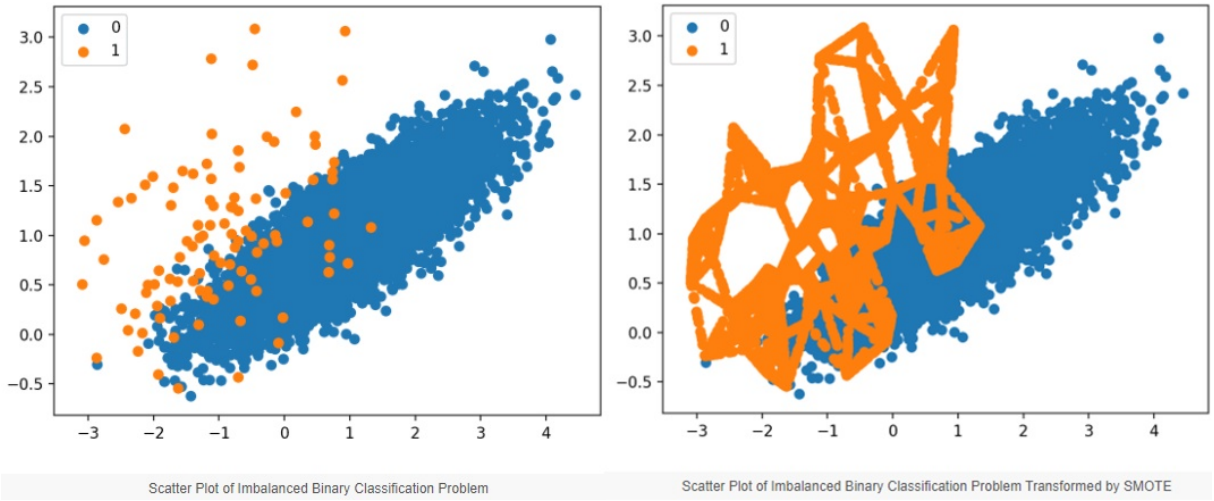


Figura 3.6 – Conjunto desbalanceado antes e após técnica do *SMOTE* (BROWNLEE, 2020)

Neste capítulo foram abordados alguns métodos utilizados na mineração de dados que serão importantes para o desenvolvimento do trabalho. Esses métodos têm como objetivos a extração das *features* de perguntas e usuários do *Stack Overflow*, bem como seleção das *features* mais relevantes para o processo de classificação através de métodos como o índice de GINI ou o ganho de informação. Classificação da faixa de tempo que uma pergunta do *Stack Overflow* demora para receber uma resposta, com a utilização de algoritmos supervisionados, como SVMs e árvores de decisão, e então a avaliação do modelo gerado utilizando algumas métricas, como precisão e revocação. Também apresentou a técnica de sobreamostragem que será útil neste trabalho para balancear as classes propostas.

4 TRABALHOS RELACIONADOS

Este capítulo apresenta alguns dos trabalhos relacionados encontrados na literatura.

4.1 Answering questions about unanswered questions of stack overflow

A abordagem proposta em ASADUZZAMAN et al. (2013) realiza um estudo qualitativo para identificar características presentes nas questões que ficaram por pelo menos um mês sem respostas no *Stack Overflow*. Após identificar as características que fazem uma pergunta ficar sem resposta, os autores categorizaram 400 perguntas em cada uma das características encontradas.

Essas características são descritas abaixo:

- Muito curto, pouco claro, vago ou difícil de seguir;
- Programa específico, sem um trecho de programa;
- Muito difícil, muito específico ou muito demorado;
- Tecnologia proprietária;
- Membros impacientes, irregulares ou imprudentes;
- Não é uma pergunta para *StackOverflow*;
- Respondendo às suas próprias perguntas;
- Não tem resposta;
- Uma pergunta duplicada;
- A resposta não é mais relevante ou necessária;
- Falha em atrair um membro especialista;
- Funciona para mim;
- Projeto de curso ou questão de lição de casa.

Após a categorização das 400 perguntas, ASADUZZAMAN et al. (2013) identificou que as 3 principais características que fazem uma pergunta ficar por pelo menos um mês sem respostas são: falha em atrair um membro especialista; muito curto, pouco claro, vago ou difícil de seguir; e uma pergunta duplicada.

Em seguida, foi conduzido um experimento para prever quanto tempo uma pergunta permaneceria sem resposta. Para a construção do modelo de previsão, foram selecionadas questões que receberam respostas após um dia. Foi escolhido um conjunto de *features* para o treinamento e as questões foram categorizadas com um esquema de categorização de frequência igual em três classes: $C1 : 1440m < t \leq 5570m, C2 : 5570m < t \leq 39285m$ e $C3 : t > 39285m$, onde *t* representa a faixa de tempo e *m* os minutos. Não foi categorizado além de $39285m$, pois o autor acredita que esse tempo de resposta é indesejável para um usuário e como 89,6% das perguntas receberam suas primeiras respostas em um dia, ASADUZZAMAN et al. (2013) concentrou sua atenção em questões que permanecem sem resposta por mais de um dia. Então foi construído o modelo de previsão utilizando algoritmos classificadores.

Os resultados de (ASADUZZAMAN et al., 2013) não foram muito satisfatórios, obtiveram as seguintes precisões para as classes: $C1 = 0.361,\ C2 = 0.349,\ C3 = 0.378$ e as seguintes revocações: $C1 = 0.351,\ C2 = 0.295,\ C3 = 0.447$, porém as características analisadas servem de ponto de partida para a extração das *features* propostas nesse trabalho.

4.2 Toward understanding the causes of unanswered questions in software information sites: a case study of stack overflow

Os autores em (SAHA; SAHA; PERRY, 2013) realizaram uma investigação para entender porque perguntas ficam sem respostas no *Stack Overflow*. As questões do *Stack Overflow* foram codificadas em um vetor de *Features* dividido em três classes: atributos estruturais, atributos de qualidade e atributos do usuário, respectivamente. Estes atributos estão descritos na abaixo:

Estruturais:

- Número de tags;
- Tamanho das perguntas;
- Presença de Código;

- Link externo.

Qualidade:

- Número de visualizações;
- *Score*;
- Número de favoritos;
- Número de comentários.

Usuário:

- Reputação;
- Número de perguntas respondidas no passado;
- Número de perguntas não respondidas no passado;
- Porcentagens de perguntas que obtiveram respostas no passado.

Após a criação do vetor de *Features*, foi utilizada a técnica de ganho de informação para selecionar as melhores *Features* que caracterizam questões com e sem respostas.

Os resultados apontam que *Features* como número de visualizações, *Score* e reputação dos usuários são os atributos mais relevantes na classificação se uma questão terá ou não resposta.

Para validar a eficácia da seleção de *Features* foram criados modelos de predição usando as *Features* propostas. Os algoritmos usados foram *Decision Tree*, KNN e *Naive Bayes*, *Random Forest*. Os resultados obtidos estão descritos na Tabela 4.1.

Classificador	Com resposta		Sem resposta	
	Precisão	Revocação	Precisão	Revocação
Árvore de decisão	0.92	0.89	0.88	0.91
KNN	0.78	0.91	0.87	0.81
Naive Bayes	0.98	0.56	0.65	0.98
Random Forest	0.96	0.77	0.78	0.96

Tabela 4.1 – Resultados (SAHA; SAHA; PERRY, 2013)

4.3 WANQA: uma Abordagem para Identificar Novas Questões Não Respondíveis em Comunidades de Perguntas e Respostas.

O trabalho de (KNOCHENHAUER; DORNELES; WIVES, 2018) se propõe à classificação de novas questões em respondíveis ou não respondíveis no momento da postagem, cuja abordagem possa ser aplicada a um conjunto mais genérico de comunidades de perguntas e respostas. Foram utilizadas as comunidades do *StackOverflow*, *Mathematics* e *Cross Validated*, nas categorias Java, Álgebra linear e Regressão, respectivamente. O principal objetivo é criar uma abordagem que use características mais comumente presentes nas comunidades e obtenha melhor acurácia na classificação das novas questões como respondíveis ou não respondíveis. As características estão disponíveis no momento da postagem e são provenientes das questões presentes nas discussões e dos usuários que criam as questões. O processo de classificação envolveu três passos: extração de *Features*, seleção de *Features* e treinamento do modelo.

Na etapa de extração de *Features* foram extraídas 20 *Features*, sendo 4 referentes ao usuário que foram: número de dias como membro, número de questões postadas pelo usuário, número de respostas postadas pelo usuário e proporção de respostas em relação ao número de perguntas postadas. As outras 16 *features* são referentes à questão, que são: Tamanho do Título, Tamanho da Descrição, Título inicia com palavra WH, Erros de Linguagem, Há Código na Descrição, Legibilidade, Subjetividade, Polidez, Seno e Cosseno do Dia, Seno e Cosseno da Hora, Contagem de tags, Contagem de URL, Contagem de palavras WH no título, Contagem de palavras WH na descrição.

Após a extração das características, fez-se um pré-processamento no texto removendo *stop words*, reduz-se os termos à sua raiz (*stemming*). Cria-se um vetor TF-IDF com todos os termos encontrados nas questões e seus respectivos pesos. Esse vetor contém uma combinação de todos os termos presentes na coleção de questões recebidas como entrada. Todos os valores dos termos são computados para todas as questões. A determinação do valor TF-IDF visa identificar os termos que são relevantes para a categoria das questões (Futebol, Javascript, etc.) que estão sendo usadas para a criação do modelo classificador.

Em seguida (KNOCHENHAUER; DORNELES; WIVES, 2018) seleciona o subconjunto de características que mais influenciam na definição das classes das questões. A relevância de uma característica é dada pela atribuição de pesos que relacionam os valores das características com as classes. Quanto maior o peso atribuído, maior a relevância da característica na classificação. Foram utilizados o *Information gain*, Correlação de *Person* e GINI *Gain*.

A criação do modelo foi efetuada com a utilização do classificadores: SVM, *Naive Bayes*, Regrssão Logística e *Hyperpipes*.

A Tabela 4.2 apresenta o melhor resultado obtido com a base de dados do *Stack Overflow* na categoria *Java*. Foram efetuados diversos experimentos, com diferentes algoritmos, diferentes métodos de definição de importância de *feature* e também com bases de dados com e sem a adição do vetor TF-IDF. O melhor resultado foi utilizando o classificador *Naive Bayes* e o método de seleção de características foi Correlação de *Person* e a base de dados com a adição do vetor TF-IDF.

Classificador	Com resposta		Sem resposta	
	Precição	Revocação	Precição	Revocação
Naive Bayes	83,14	91,79	92,73	81,20

Tabela 4.2 – Resultados (KNOCHENHAUER; DORNELES; WIVES, 2018)

4.4 Considerações importantes

Os trabalhos relacionados nas seções anteriores, foram muito importantes na execução deste projeto. O entendimento das *features* que auxiliaram neste projeto, informando as características que podem fazer uma pergunta ter ou não uma resposta rápida foram de grande importância. Os algoritmos de classificação utilizados e as medidas avaliativas abordadas nos trabalhos relacionados foram pilares para este projeto.

5 EXPERIMENTOS E RESULTADOS

Este capítulo apresenta o projeto dos experimentos para a criação de um modelo para predizer o comportamento temporal da resposta de uma pergunta no *Stack Overflow*. O projeto é realizado inicialmente pela obtenção do conjunto de dados, extração e criação de atributos, definição das classes de tempos de resposta. Em seguinda, testes dos modelos candidatos, definição dos melhores hiperparâmetros para o modelo escolhido e aplicação de técnica de reamostragem de dados. Finalmente, são feitos dois experimentos com os modelos criados: um com duas classes e outro com quatro classes.

5.1 Obtenção dos dados

O inicio do projeto de experimentos se deu através da obtenção da base de dados do *Stack Overflow* pelo site https://archive.org/download/stackexchange. Após o download, algumas etapas foram efetuadas até a obtenção do conjunto de dados para análise e treinamento.

1. Verificação estrutural da base de dados do *StackOverflow*;
2. Extração das informações relevantes; e
3. Criação do conjunto de dados;

Na Etapa 1, foi analisado o conjunto de dados para identificar como seria implementado o algoritmo para extração das *features*. Os dados do *StackOverflow* são disponibilizados em formato XML sob a Licença *Creative Commons license*. O conjunto de dados é dividido em `badges.xml`, `comments.xml`, `posts.xml`, `users.xml` e `votes.xml` (BARUA; THOMAS; HASSAN, 2014). Para o experimento foi utilizado o arquivo *post.xml*, que contém as pergunta e respostas publicadas pelos usuários. Foram utilizadas postagens de 2008 a 2019, contendo cerca de 18 milhões de perguntas e 28 milhões de respostas.

A Figura 5.1 apresenta uma pergunta no site *Stackoverflow*, já na Figura 5.2 a mesma pergunta salva no arquivo `posts.xml`. Pode-se observar na Figura 5.2 as informações sobre a postagem, como: *CreationDate* que representa a data de criação da postagem, *Title* que representa o título quando a postagem é uma pergunta, *Tags* que representam marcações sobre o assunto da postagem, *OwnerUserId* é a identificação do usuário que fez a postagem, *Body* é onde está contido o texto da postagem, entre outros. Por exemplo, nesse extrato, pode-se

Figura 5.1 – Postagem no *Stackoverflow*

observar que o *id* da resposta aceita para a pergunta foi o 516 de acordo com a TAG *AcceptedAnswerId* e a pergunta foi postada em 01/08/2008 às 8:33 como mostra a *TAG CreationDate*.

Figura 5.2 – Exemplo de pergunta salva no arquivo XML (STACKEXCHANGE, 2018)

```
<?xml version="1.0"?>
<row FavoriteCount="195" CommentCount="6" AnswerCount="29" Tags="<sql-server><database><svn><version-
    control>" Title="Versioning SQL Server database" LastActivityDate="2017-08-21T08:55:26.907" LastEditDate="2017-
    08-21T08:55:26.907" LastEditorDisplayName="user5857081" LastEditorUserId="1836618" OwnerUserId="83"
    Body="<p>I want to get my databases under version control. Does anyone have any advice or recommended
    articles to get me started?</p> <p>I'll always want to have at least <em>some</em> data in there (as <a
    href="https://stackoverflow.com/users/80/alumb">alumb</a> mentions: user types and administrators). I'll
    also often want a large collection of generated test data for performance measurements.</p> "
    ViewCount="78634" Score="279" CreationDate="2008-08-01T18:33:08.333" AcceptedAnswerId="516" PostTypeId="1"
    Id="173"/>
```

Após compreender a estrutura da base de dados iniciou-se a Etapa 2. Foi percorrida toda a base de dados do *Stack Overflow* e de cada pergunta foram extraídas 14 *features* descritas abaixo:

Tamanho do corpo da pergunta: representa o número de tokens existentes no corpo da pergunta, incluindo palavras e símbolos

Número de *tags*: quantidade de *tags* que aparecem no atributo *tags* do documento xml;

Tamanho do código: quantidade de caracteres existentes no trecho de programa contido na pergunta. Caso não exista código, esse atributo recebe 0;

Interrogação no título: atributo binário (0 ou 1) que representa se existe ou não algum ponto de interrogação na pergunta;

Legibilidade do texto: legibilidade do corpo da pergunta, calculada através da biblioteca *textstat*[2] utilizando a métrica *flesch reading ease*. Esta métrica retorna um valor real, sendo que valores negativos representam textos confusos de serem lidos (FLESCH, 1948). A tabela 5.2 apresenta uma possível interpretação dos valores retornado [3];

Título inicia inicia com WH: verificado se o título inicia com WH, também chamado de *WH Questions*: *who, where, why, what, when, how*. O valor é um inteiro, 1 em caso afirmativo para inicial com WH e 0 em caso negativo;

Número de perguntas feitas pelo usuário: número inteiro que indica a contagem de perguntas fornecidas pelo usuário até a data da pergunta;

Tamanho do título: número inteiro que indica a contagem de palavras e símbolos no título da questão;

Subjetividade: para o cálculo da subjetividade do corpo da pergunta foi utilizada a biblioteca *TextBlob*, valores *Float* vão de um intervalo de [0,1.0], onde 0 é muito objetivo e 1.0 é muito subjetivo;

Número de respostas feitas pelo usuário: número inteiro que representa a contagem de respostas fornecidas pelo usuário até a data da pergunta;

Polaridade do texto: para o cálculo da polaridade do corpo da pergunta foi utilizada a biblioteca *TextBlob*. A pontuação de polaridade é um número *Float* dentro do intervalo [-1,0, 1,0], -1 define um sentimento negativo e 1 define um sentimento positivo;

Número de frases no corpo da pergunta: número inteiro que representa a quantidade de frases presentes no corpo da pergunta;

Média de caracteres das frases: tamanho médio de caracteres das frases presentes no corpo da pergunta;

Tempo que a pergunta ficou sem resposta: cada resposta encontrada é salva junto com seu *ParentId* que representa o *Id* da pergunta, essas datas são ordenadas e então é feito a subtração da data da postagem da pergunta com a data da primeira resposta. Caso não seja encontrada uma resposta, é atribuído o valor -1 ao tempo.

A Tabela 5.1 apresenta as *features* propostas para este projeto. Por exemplo, as *features*, número de questões e respostas postadas no site são características que medem a interação do

[2] pypi.org/project/textstat/
[3] en.wikipedia.org/wiki/Flesch–Kincaid_readability_tests.

usuário no site. O tamanho do título da pergunta é importante, pois é o resumo do questionamento, um título vago pode não explicar o real problema. Título iniciado com "WH" *(what, why, when, who,which, how, whose, whom)* tendem a serem mais claros porque especificam o que deve ser respondido. A presença ou não destas palavras pode ser uma maneira de medir a objetividade ou subjetividade da pergunta. A presença de código fonte é importante, pois geralmente questões postadas no site envolvem programação. A legibilidade do texto indica o quão difícil é ler e entender a questão. Questões difíceis de ler e entender podem ficar mais tempo sem respostas (legibilidade de texto pode ser medidas através da métrica Flesch reading ease (ASADUZZAMAN et al., 2013)).

Features	**Fonte**
Tamanho do corpo da pergunta	(ASADUZZAMAN et al., 2013))
Número de tags	(SAHA; SAHA; PERRY, 2013)
Tamanho do código	(KNOCHENHAUER; DORNELES; WIVES, 2018)
Legibilidade do texto	(KNOCHENHAUER; DORNELES; WIVES, 2018)
Título inicia inicia com WH	(KNOCHENHAUER; DORNELES; WIVES, 2018)
Número de perguntas feitas pelo usuário	(ASADUZZAMAN et al., 2013))
Tamanho do título	(ASADUZZAMAN et al., 2013))
Subjetividade	(KNOCHENHAUER; DORNELES; WIVES, 2018)
Número de respostas fornecidas pelo usuário	(ASADUZZAMAN et al., 2013))
Polaridade do texto	(KNOCHENHAUER; DORNELES; WIVES, 2018)
Número frases no corpo da pergunta	Proposta
Média de caracteres das frases	Proposta
Interrogação no título	Proposta

Tabela 5.1 – *Features*

Uma estrutura de dados do tipo dicionário (python) foi utilizada para armazenar os atributos extraídos. Ao final da extração das *features* se deu a Etapa 3, em que foi gerada a base de dados, transformando os dicionários em um arquivo do tipo *.CSV*.

A Figura 5.3 apresenta um extrato do conjunto de dados extraídos e a linha destacada representa a postagem presente nas Figuras 5.1 e 5.2. É possível verificar que que a *feature* tamanho do título (*Tam Titulo*) obteve o valor 4, o mesmo valor para o número de tags (*N de tags*) e a legibilidade do corpo da pergunta (*flesch*) alcançou o valor de 10.24.

Tabela 5.2 – Discretização dos valores retornados pelo método *flesch_reading_ease*.

Score	Difficulty
–90.1	Muito fácil de ler.
90.0–80.1	Moderadamente fácil.
80.0–70.1	Fácil de ler.
70.0–60.1	Texto claro.
60.0–50.1	Dificuldade normal. Leitores com ensino médio
50.0–30.1	Difícil de ler. Entendido apenas por leitores com nível superior.
30.0–10.1	Muito Difícil de ler. Entendido apenas por leitores com nível superior.
10.0–0.00	Extremamente difícil de ler. Entendido apenas por leitores com nível superior.

Figura 5.3 – Trecho do conjunto de dados extraído de *Posts.xml*

Tam corpo	Tam Titulo	N frases corpo	flesch	Media Caracteres Frase	Tamanho Codigo	Interogacao	Inicia com WH	Subjetividade	Polaridade	N de tags	N perguntas Feitas	N respostas Feitas	Rotulo
92	12	5	46.1	86.2	66	0	0	0.2972	0.0722	3	0	0	2.0
217	6	11	42.79	91.0909	225	0	0	0.4732	0.0508	3	1	0	1.0
25	9	2	65.05	64.0	0	0	0	0.6321	0.3119	4	0	0	1.0
37	14	4	25.12	47.0	25	1	1	0.5	0.0	4	1	0	1.0
115	7	5	53.75	118.8	0	0	0	0.5148	0.0833	3	0	0	1.0
62	4	5	10.24	62.8	0	0	0	0.4143	-0.0429	4	2	0	1.0
137	11	10	-21.23	60.5	716	1	1	0.4746	0.0418	4	0	0	1.0
142	4	12	55.24	57.75	0	0	0	0.6	0.3929	2	0	1	6.0

5.2 Experimentos iniciais e definições das classes

Após a geração da base de dados, começou uma bateria de experimentos para análise das *features* e quais classes seriam criadas a partir da *feature* do tempo que a pergunta ficou sem resposta. Estes experimentos foram conduzidos com uma base reduzida de 20% devido ao tamanho total do conjunto, que contém cerca de 18 milhões de exemplos. A separação do suconjunto ocorreu de modo aleatório utilizando a biblioteca do *Python train_test_split*. O experimento foi conduzido seguindo uma estrutura conforme passos abaixo:

1. Escolha das classes;
2. Definição da importância das *features*;
3. Treinamento.

Inicialmente, no Passo 1, foram definidas quais classes seriam utilizadas no experimento. A Tabela 5.3 apresenta as definições das classes: o rótulo da classe, período de tempo que engloba cada classe, a proporção e o total de exemplos de cada classe. É possível verificar que a classe 1 é dominante, com 67% dos exemplos.

As classes foram definidas analisando a distribuição do tempo que as perguntas ficam sem respostas. O Gráfico 5.4 apresenta as distribuições em 5 faixas. É notável que a faixa de

Figura 5.4 – Distribuição de tempo

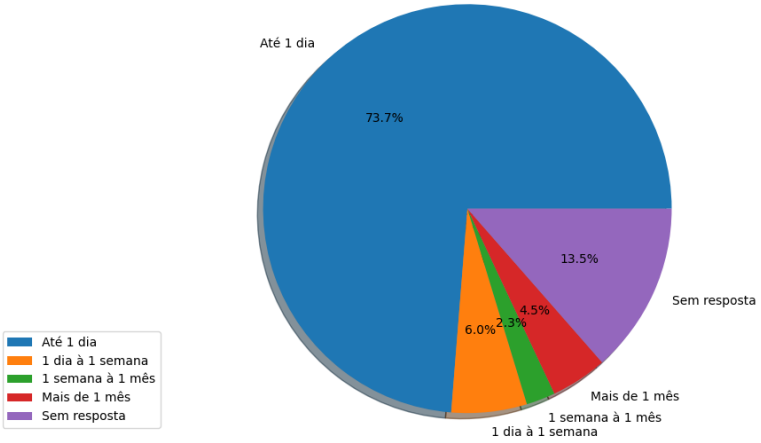


Tabela 5.3 – Classes e suas distribuições

Classe	Período	Proporção	Total
C1	$0m < t \leq 480m$	67,6%	12210202
C2	$480m < t \leq 960m$	3,4%	629399
C3	$960m < t \leq 1440m$	2,5%	464277
C4	$1440m < t \leq 2280m$	1,7%	311453
C5	$2280m < t \leq 5790m$	2,9%	523521
C6	$t > 5790m$	8,1%	1461567
C7	$t = -1$	13,5%	2443025

tempo "até 1 dia" é a maior com 73.7%. Com esse dado foi definido que a essa faixa de tempo seria dividida em 3 classes, que são as classe C1, C2 e C3.

Em seguida, no Passo 2, foi definida a importância de cada *feature* através da biblioteca do *python* chamada *selectkbest* [4]. A Tabela 5.4 apresenta o *ranking* de importância das *features*, número de palavras no corpo da pergunta e número de *tags* foram as duas mais importantes.

Após definir a importância das *features*, foram feitas diversas rodadas de treinamento (Etapa 3), iniciando com o conjunto das duas *features* mais importantes e incrementando uma nova *features* do *ranking* até o total de 13.

O algoritmo utilizado foi o *random forest* da biblioteca *sklearn* do *python*. Foram utilizados hiperparâmetros *default* para cada execução. Os hiperparâmetros são uma maneira de regular o algoritmo da melhor forma ao seu conjunto de dados.

[4] scikit-learn.org/stable/modules/generated/sklearn.feature$_s election.SelectKBest.html$

Posição	*feature*
1	Tamanho do corpo da pergunta
2	Número de tags
3	Número frases no corpo da pergunta
4	Tamanho do código
5	Legibilidade do texto
6	Interrogação no título
7	Título inicia inicia com WH
8	Número de perguntas feitas pelo usuário
9	Tamanho do título
10	Subjetividade
11	Número de respostas feitas pelo usuário
12	Média de caracteres nas frases
13	Polaridade

Tabela 5.4 – Importância das *features* no experimento 1

O conjunto foi dividido em 30% para teste e 70% para treinamento, divididos de forma aleatória utilizando a biblioteca *train_test_split* do *python*. Os métodos de avaliação foram a Precisão e o Revocação.

A Tabela 5.5 apresenta os resultados do experimento para cada execução do algoritmo. É verificado que a Classe C1 obteve os melhores resultados em relação às demais classes. Enquanto as Classes C2, C3, C4 e C5 não foram preditas. Classes C6 e C7 obtiveram uma baixa predição em relação a C1. Ao analisar a Tabela 5.3, é possível verificar que as classes com menor proporção não foram preditas. Já a Classe C1, que detém a maior proporção de exemplos obteve uma precisão de 0,68 e um revocação de 1, o que significa que maioria dos exemplos foi predita para Classe C1. Também é possível verificar que a partir da oitava execução os resultados se mantiveram similares.

Devido aos resultados não satisfatórios com sete classes, foi reduzido o número de classes e, assim, novos algoritmos de aprendizado foram testados. Primeiramente foi diminuído o número de classes da base de dados para quatro. As faixas temporais de cada classe são mostradas na Tabela 5.6. A diminuição do número de classes ocorreu com a intenção de aumentar a quantidade de exemplos de outras classes e também aumentar a faixa de tempo das classes, para que os algoritmos obtivessem melhor distinção em cada classe. Outros algoritmos também foram testados, como *Decision Tree, MLP Classifier* e *Logistic Regression*.

Os resultados se mantiveram similares, como mostra a Tabela 5.8. A Classe C1 se mantém dominante em relação à demais classes em todos os algoritmos. Neste momento foram

		Classes						
		C1	C2	C3	C4	C5	C6	C7
Execução 1	Precisão	0.67704	0.03846	0.04032	0.01694	0.05714	0.11434	0.17462
	Revocação	0.99885	0.00013	0.00017	0.00005	0.00025	0.00058	0.00094
Execução 2	Precisão	0.67704	0.03968	0.04032	0.01694	0.05714	0.11512	0.17396
	Revocação	0.99886	0.00013	0.00017	0.00005	0.00025	0.00058	0.00094
Execução 3	Precisão	0.67791	0.03316	0.02935	0.01734	0.02982	0.08282	0.14414
	Revocação	0.81192	0.01809	0.01599	0.00913	0.01608	0.04557	0.09272
Execução 4	Precisão	0.68188	0.04070	0.02973	0.02312	0.03784	0.10582	0.15018
	Revocação	0.91420	0.00947	0.00616	0.00491	0.00867	0.03253	0.04879
Execução 5	Precisão	0.68232	0.03997	0.02888	0.02117	0.03573	0.10759	0.15325
	Revocação	0.91705	0.00907	0.00591	0.00443	0.00791	0.03210	0.04869
Execução 6	Precisão	0.68204	0.04029	0.02555	0.02553	0.03604	0.11211	0.15395
	Revocação	0.93105	0.00753	0.00426	0.00438	0.00645	0.02873	0.04142
Execução 7	Precisão	0.67991	0.04653	0.03049	0.02300	0.03511	0.11916	0.17290
	Revocação	0.97268	0.00271	0.00157	0.00122	0.00193	0.01267	0.02211
Execução 8	Precisão	0.67846	0.04835	0.02259	0.00869	0.03703	0.12251	0.19406
	Revocação	0.98984	0.00066	0.00028	0.00010	0.00050	0.00486	0.01148
Execução 9	Precisão	0.67760	0.05263	0.02857	0.00000	0.04166	0.15974	0.21055
	Revocação	0.99639	0.00013	0.00007	0.00000	0.00012	0.00202	0.00553
Execução 10	Precisão	0.67740	0.03225	0.00000	0.00000	0.06451	0.15412	0.20251
	Revocação	0.99702	0.00005	0.00000	0.00000	0.00012	0.00145	0.00448
Execução 11	Precisão	0.67744	0.08333	0.04761	0.11111	0.08823	0.15797	0.22303
	Revocação	0.99782	0.00005	0.00003	0.00005	0.00009	0.00114	0.00413
Execução 12	Precisão	0.67723	0.00000	0.00000	0.00000	0.09090	0.17380	0.22715
	Revocação	0.99853	0.00000	0.00000	0.00000	0.00003	0.00078	0.00296

Tabela 5.5 – Resultados experimento 1

Tabela 5.6 – Quatro classes

Classe	Período	Total	Proporção
C1	$0m < t \leq 1440m$	13171169	73.7%
C2	$1440m < t \leq 10080m$	1067850	6.0%
C3	$10080m < t$	1205715	6.8%
C4	$t = -1$	2418457	13.5%

testadas precisão e o revocação durante os treinamentos, para verificar se a baixa predição das Classes C2, C3 e C4 se mantinham durante o treinamento.

		Classes			
		C1	C2	C3	C4
Randon Forest Treinamento	Precisão	1	1	1	1
	Revocação	1	1	1	1

Tabela 5.7 – Resultados treinamento *Randon Forest*

Foi constatado que o modelo estava sofrendo de *Overfitting*, problema que consiste em baixo erro durante o treinamento e alto índice de erro durante a validação. Esse resultado é mostrado na Tabela 5.7. É possível visualizar que durante o treinamento, o algoritmo *Randon Forest* obtêm valores 1 para precisão e revocação para todas as classes, porém para o conjunto

de teste, descritos na primeira linha da Tabela 5.8 os valores são baixos.

		Classes			
		C1	C2	C3	C4
Randon Forest	Precisão	0.74	0.32	0.19	0.25
	Revocação	1	0	0	0
Decision Tree	Precisão	0.75	0.07	0.08	0.15
	Revocação	0.72	0.08	0.09	0.16
MLP	Precisão	0.74	0	0.21	0.20
	Revocação	1	0	0	0
Logistic Regression	Precisão	0.74	0.07	0.15	0.17
	Revocação	0.99	0	0	0.01

Tabela 5.8 – Resultados quatro classes

Para reduzir os efeitos do *Overfitting* um novo algoritmo foi usado, o *Extra Trees Classifier*. Segundo (KUMAR et al., 2020), o *Extra Trees Classifier* ajuda a minimizar as chances de *Overfitting*. Porém o problema permaneceu mesmo com a utilização do *Extra Trees Classifier*, a predição durante o treinamento obtinha baixa taxa de erro, porém no conjunto de testes um alto índice de erro era obtido nas classes com menor proporção de exemplos.

Devido os resultados anteriores, foi determinada uma nova redução de classes que resultou em 2 classes: s $C1 : 0m < t \leq 1440m$ e $C2 : t > 1440m \mid t = -1$. Assim, as duas classes representam perguntas respondidas em até um dia, que seriam perguntas com uma resposta rápida e perguntas com respostas após 1 dia e sem respostas. Com as novas classes definidas, um novo experimento utilizando o algoritmo *Extra Trees Classifier* foi conduzido. Os resultados são apresentados na Tabela 5.9. Note que a precisão da Classe C1 foi de 0.74 e o Revocação foi de 0.99, esse alto valor de Revocação indica que o algoritmo está conseguindo predizer quase todos os exemplos da Classe C1. Porém a revocação no valor de 0.02 da Classe C2 mostra que muito exemplos da Classe C2 foram preditos pelo algoritmo como pertencendo à Classe C1.

Os resultados obtidos com a diminuição para duas classes ainda não foram considerados satisfatórios, com isso, foi feita uma análise na distribuição de valores das *features* para cada classe, a fim de observar a relação entre as *features* e cada classe. Alguns gráficos estão descritos na próxima seção.

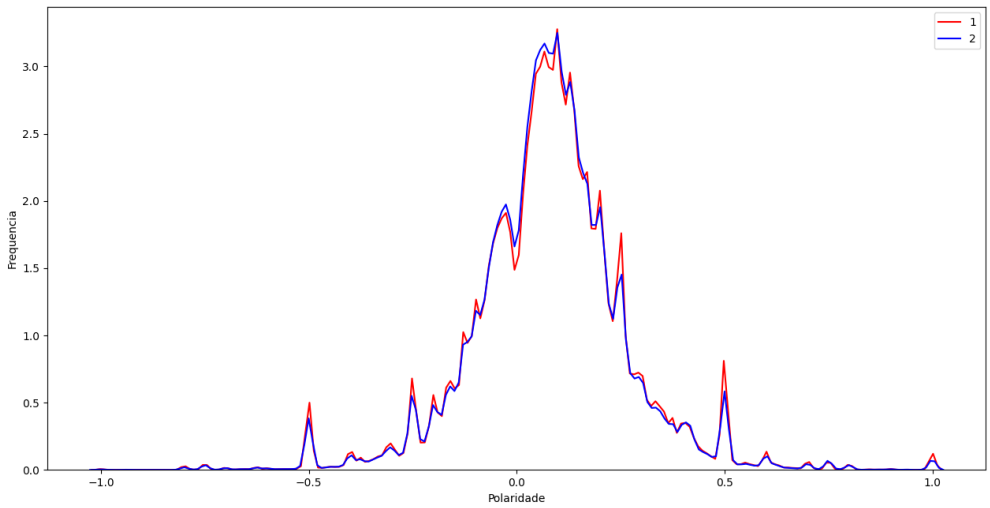
		Classes	
		1	2
Extra Trees Classifier	Precisão	0.74	0.42
	Revocação	0,99	0,02

Tabela 5.9 – Resultados *Extra Trees Classifier*

5.3 Análise das *Features*

Após a rodada de experimentos, alguns gráficos com a distribuição de valores das *features* em cada classe foram gerados. O objetivo é analisar se as *features* propostas neste trabalho seriam capazes de distinguir cada classe.

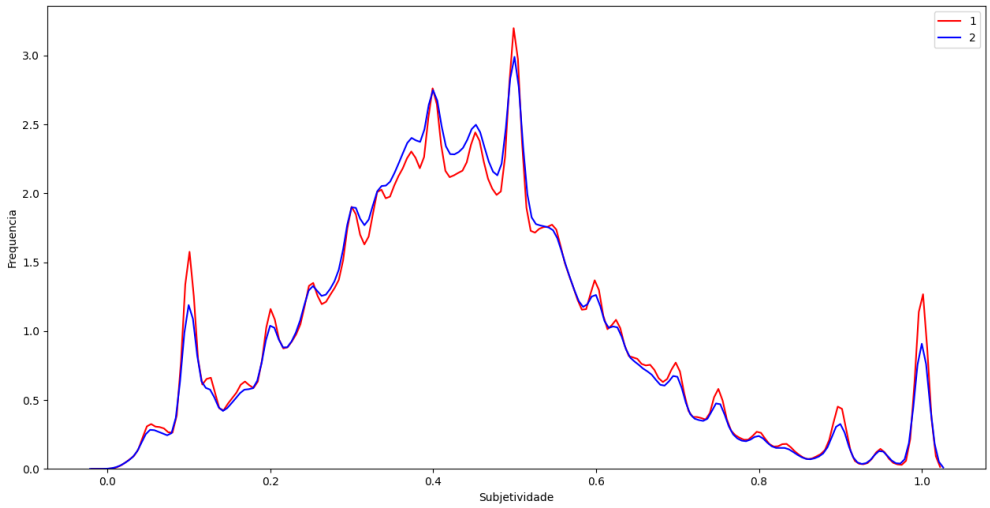
Figura 5.5 – Distribuição polaridade



A Figura 5.5 mostra a frequência da *feature* polaridade. É possível verificar que a distribuição da polaridade é bastante similar entre as classes, evidenciando que a polaridade é uma *feature* incapaz de auxiliar os algoritmos a predizer um novo exemplo. Já a Figura 5.6 apresenta a distribuição da *feature* subjetividade, e assim como a polaridade a distribuição é bastante similar entre as classes.

Com a geração dos gráficos de distribuição das *features* em relação à classes, foi possível verificar que as *features* propostas não são capazes de informar com precisão as características das classes. Os gráficos foram gerados para todas as classes, sendo que o comportamento se manteve similar para todas as *features*.

Figura 5.6 – Distribuição subjetividade



5.4 Definição e escolha dos hiperparâmetros

Após o período de experimentos foi definido que o experimento final seria conduzido com 2 classes. Para melhorar o desempenho de predição do algoritmo *Extra Trees Classifier* alguns hiperparâmetros foram selecionados, com o objetivo de obter a melhor configuração possível de valores. Os hiperparâmetros selecionados estão descritos na lista abaixo:

- ***n_estimator*** : número de árvores na floresta;
- ***criterion*** : função para medir a qualidade de uma divisão;
- ***max_features*** : o número de recursos a serem considerados ao procurar a melhor divisão;
- ***max_samples*** : número de amostras a serem retiradas de X para treinar cada estimador de base;
- ***min_samples_split*** : número mínimo de amostras necessárias para dividir um nó interno;
- ***min_samples_leaf*** : número mínimo de amostras necessárias para estar em um nó folha.

Após a definição dos hiperparâmetros, foram definidos valores para cada um deles e, em seguida, foi gerada a melhor combinação possível de valores. A Tabela 5.10 mostra

o conjunto de todos valores escolhidos para cada hiperparâmetro e em negrito é mostrado o valor selecionado. É possível observar que a melhor combinação de hiperparâmetros foi: *n_estimator*: 120, *criterion*: gini, *max_features*: 0.8, *max_samples*: 0.6, *min_samples_split*: 2 e *min_samples_leaf*: 4. A medida avaliativa para escolher os melhores hiperparâmetros foi a precisão.

hiperparâmetros	
n_estimator	[100,**120**, 150, 200, 250]
criterion	[entropy, **gini**]
max_features	[0.1, 0.4, 0.6, **0.8**,1]
max_samples	[**0.6**, 0.8, 0.9]
min_samples_split	[**2**, 4, 10, 20]
min_samples_leaf	**4**, 8, 10, 16]

Tabela 5.10 – hiperparametros

5.5 Experimentos finais

Após a definição dos hiperparâmetros, o experimento final foi conduzido. O modelo foi gerado utilizando o algoritmo *Extra Trees Classifier*. Foram feitas 12 execuções, iniciando com o conjunto das duas *features* mais importantes e incrementando uma nova *features* do ranking até o total de 13. O ranking de importância das *features* é mostrado na Figura 5.7, é possível verificar que o número de palavras no corpo da pergunta e o número de *tags* foram as 2 *features* mais importantes.

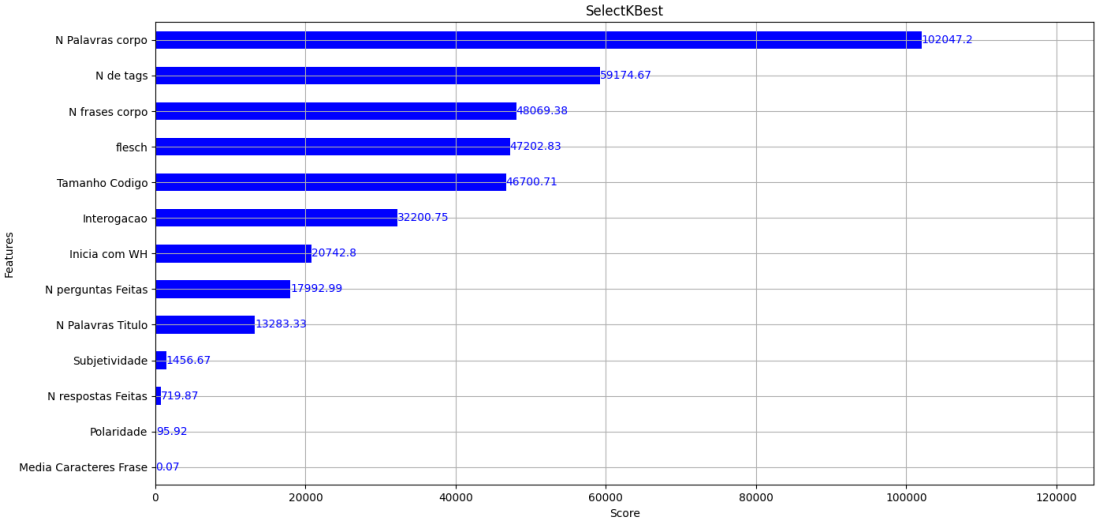
A base de dados contêm cerca de 18 milhões de exemplos, com 73.7% de exemplos com perguntas respondidas em até 1 dia e 26.3% de exemplos com perguntas sem respostas ou com respostas após 1 dia, ambas as classes estão descritas na Tabela 5.11. Destes quase 18 milhões de exemplos, 70% foram utilizados para treinamento do modelo, os 30% restantes foram utilizados para teste. Os métodos avaliativos do modelo foram a precisão, Revocação e *f1-score*.

Tabela 5.11 – Duas Classes

Classe	Período	Total	Proporção
C1	$0m < t \leq 1440m$	13171169	73.7%
C2	$1440m < t \mid t = -1$	4692022	26.3%

A Tabela 5.12 apresenta os resultados. É possível verificar que a revocação da Classe

Figura 5.7 – Importância das *features*



C1 com perguntas respondidas em até 1 dia ficou alta, se mantendo em 0.99 em quase todas as execuções. Isso significa que dos exemplos que realmente eram da Classe C1, 99% deles foram preditos de maneira correta. A precisão da classe C1 ficou em 74% em todas as execuções, com esse números de precisão e revocação da Classe C1 é possível afirmar que muitos exemplos que eram da Classe C2 foram preditos como exemplos da Classe C1 pelo modelo. Já o *F1-score* que representa a média harmônica entre precisão e revocação ficou em 0.85.

A classe com perguntas que ficaram por mais de 1 dia sem resposta ou não tiveram resposta obteve resultados inferiores, tanto na revocação quanto na precisão. A revocação ficou em apenas 0.01, o que significa que muitos exemplos que eram da Classe C2 foram preditos como da Classe C1. A precisão ficou em 0.48 na melhor execução, ou seja, de todos os exemplos preditos como da Classe C2, 48% foram preditos de forma correta. O *F1-score* ficou em 0.03.

Analisando a Tabela 5.11 que apresenta a proporção de exemplos para cada classe e observando os resultados finais, é notável que a predominância da classe com perguntas respondidas em até um dia interferiu no desempenho da predição. O algoritmo acabou classificando muitos exemplos da Classe C2 como sendo da Classe C1. A baixa capacidade das *features* descreverem as classes também foi determinante para a baixa predição, em especial a Classe C2.

O modelo criado mostrou-se efetivo para predizer se uma pergunta vai ter resposta em um dia, porém ineficiente para respostas mais demoradas ou não dadas. Tomando como base a

	Classe	Precição	Revocação	F1-score
Execução 1	C1	0.74	1	0.85
	C2	0.43	0	0
Execução 2	C1	0.74	0.99	0.85
	C2	0.41	0.01	0.02
Execução 3	C1	0.74	0.99	0.85
	C2	0.38	0.02	0.03
Execução 4	C1	0.74	0.99	0.85
	C2	0.40	0.02	0.04
Execução 5	C1	0.74	0.99	0.85
	C2	0.40	0.03	0.05
Execução 6	C1	0.74	0.98	0.84
	C2	0.40	0.03	0.05
Execução 7	C1	0.74	0.99	0.85
	C2	0.41	0.03	0.05
Execução 8	C1	0.74	0.99	0.85
	C2	0.44	0.02	0.04
Execução 9	C1	0.74	0.99	0.85
	C2	0.45	0.02	0.03
Execução 10	C1	0.74	0.99	0.85
	C2	0.45	0.02	0.03
Execução 11	C1	0.74	0.99	0.85
	C2	0.46	0.01	0.03
Execução 11	C1	0.74	0.99	0.85
	C2	0.48	0.02	0.03

Tabela 5.12 – Resultados finais

precisão, pode-se dizer que o modelo pode informar ao usuário se sua pergunta pode ter uma resposta em um dia com uma precisão de 74% e, em outros casos, com precisão de 48%.

A revocação, que mede a quantidade de exemplos positivos que foram preditos em relação ao total de exemplos positivos existentes, foi de 0.02 para a Classe C2 , ou seja, 2% dos exemplos foram corretamente cobertos. Por outro lado, se for considerada a Classe C1, a revocação foi de 99%, porém a precisão de 74%. isso indica que o modelo está prevendo muitos falsos positivos, ou seja, o modelo prediz a Classe C2 como C1. Este comportamento explica a baixa revocação para a Classe C2.

As duas prováveis razões para o problema são: o desbalanceamento das classes (74% e 26%) e os atributos propostos não serem muito informativos. A Seção 5.3 discorre um pouco em relação a este problema, os valores dos atributos são muitos próximos nas classes consideradas.

Com o objetivo de melhorar o desempenho do modelo, a técnica *SMOTE* foi aplicada sobre os dados. O *SMOTE* é uma técnica de balanceamento dos dados descrita na Seção 3.6 e tem como objetivo balancear o número de exemplos das classes criando novos exemplos sintéticos paras as classes minoritárias.

As classes do experimento estão descritas na Tabela 5.11 com as respectivas quantidades de exemplos para cada classe. Com o uso da técnica de *SMOTE*, o total de exemplos da classe C2 passou de 4.692.022 exemplos para 13.171.169, com isso a proporção de exemplos ficou 50% para cada classe, e o total de exemplos do conjunto passou para 26.342.338.

Em seguida um novo treinamento utilizando o *Extra Trees Classifier* foi conduzido nos mesmos moldes do experimento anterior: 70% para treinamento (18.439637 exemplos) e 30% restantes para teste (7.902.701 exemplos).

	Classe	Precição	Revocação	F1-score
Extra Trees Classifier	C1	0.72	0.70	0.71
	C2	0.71	0.72	0.71

Tabela 5.13 – Resultados *SMOTE* duas classes

A Tabela 5.13 apresenta os novos resultados obtidos treinando o modelo com a nova base de dados gerada após o uso do *SMOTE*. Os resultados mostram uma melhora no desempenho do modelo para predição da Classe C2, tanto na precisão quanto na revocação em comparação com os resultados obtidos anteriormente. A precisão da classe C2 ficou em 0.71 um aumento de 0.23 em comparação aos resultados anteriores (vide Tabela 5.12). A revocação também obteve um aumento significativo, passando de 0.02 para 0.72. Isso significa que o mo-

delo diminuiu a predição de falsos negativos, ou seja exemplos da classe C2 que foram preditos como da Classe C1.

Os resultados para a classe C1 tiveram uma leve redução: a precisão passou de 0.74 para 0.72, indicando que não teve uma significativa diferença na predição do verdadeiros positivos, ou seja, exemplos da classe C1 que realmente foram preditos para a classe C1. A revocação da classe C1 obteve uma redução maior em relação à precisão, passou de 0.99 para 0.70, indicando que o algoritmo deixou de classificar inúmeros exemplos da classe C2 como da classe C1, problema existente antes do uso do *SMOTE*.

Como o desempenho de predição utilizando a técnica de sobreamostragem se mostrou positivo, um novo experimento com quatro classes também foi executado. As quatro classes são aquelas presentes na Tabela 5.6. A Tabela 5.14 apresenta o resultado do modelo de predição das quatro classes. Perceba que o desempenho foi acima daquele sem a sobreamostragem. As classes tiveram revocação e precisão similares, exceto a C4 que ficou um pouco abaixo.

	Classe	Precição	Revocação	F1-score
Extra Trees Classifier	C1	0.63	0.68	0.65
	C2	0.69	0.67	0.68
	C3	0.66	0.67	0.66
	C4	0.60	0.55	0.57

Tabela 5.14 – Resultados *SMOTE* quatro classes

Comparando os resultados obtidos antes e após a aplicação da técnica do *SMOTE* é possível verificar uma melhora geral no desempenho do modelo, tanto para o experimento com 2 classes, quanto para o com 4 classes. Os resultados mostraram-se satisfatórios para prever se uma pergunta vai ter uma resposta em até um dia com uma precisão de 72% e 63% para 2 e 4 classes, respectivamente.

6 CONCLUSÃO

Este trabalho de conclusão de curso teve por objetivo predizer se uma pergunta do *Stack Overflow* obterá uma resposta rápida da comunidade. Para este projeto, consideramos uma resposta rápida aquela fornecida em até 1 dia.

Foi efetuado o download da base de dados do *Stack Overflow* com perguntas e respostas de 2008 à 2019, contendo cerca de 18 milhões de perguntas e 28 milhões de respostas. Foram analisadas e extraídas 13 *features* de cada pergunta da base de dados do *Stack Overflow*, junto com o tempo em minutos que a pergunta ficou sem resposta. Com isso cada pergunta foi rotulada tendo uma resposta rápida (em até um dia) ou demorada para perguntas com respostas após 1 dia ou sem respostas. Após criação da base de dados foi utilizado o *SMOTE*, uma técnica de balanceamento de classes, para corrigir o problema de desbalanceamento.

Em seguida o modelo para predizer se uma pergunta teria ou não uma resposta rápida foi produzido utilizando o algoritmo *Extra Trees Classifier*. Os métodos de avaliação foram a precisão, revocação e F1-score.

Os resultados obtidos após o uso do *SMOTE* mostraram que o modelo foi capaz de predizer se uma pergunta teria uma resposta rápida com precisão de 72%. Para perguntas com respostas após um dia o modelo apresentou uma precisão de 71%. O modelo obteve uma melhora significativa em relação aos resultados obtidos sem o uso da técnica: precisão de 74% para perguntas com respostas rápidas e de 48% para perguntas com respostas após 1 dia ou sem respostas. Para o modelo com quatro classes, também foi obtido uma melhora no desempenho: a classe de perguntas respondidas rapidamente (C1) obteve 63% precisão, já a precisão para perguntas respondidas após um dia e até uma semana ficou com 69% (C2), para preguntas após uma semana a precisão foi de 66% (C3)e para perguntas sem respostas a precisão ficou em 60% (C4).

6.1 Trabalhos futuros

Como possíveis trabalhos futuros, pode-se apontar:

- Aplicar o processo engenharia de *features*, afim de buscar maneiras de criar novas *features*, seja pela combinação das atuais ou proposição de novas. Efetuar uma analise nas características que possam descrever com maior precisão as classes propostas, com isso

melhorar o desempenho de predição;

- Fazer o balanceamento das classes reduzindo o número de exemplos da classe dominante.
- Utilização do TF IDF, visando identificar os termos que são relevantes para a categoria das questões.

REFERÊNCIAS

AGGARWAL, C. C. **Data mining**: the textbook. [S.l.]: Springer, 2015.

ASADUZZAMAN, M. et al. Answering questions about unanswered questions of stack overflow. In: WORKING CONFERENCE ON MINING SOFTWARE REPOSITORIES, 10. **Proceedings...** [S.l.: s.n.], 2013. p.97–100.

BALTADZHIEVA, A.; CHRUPAŁA, G. Question quality in community question answering forums: a survey. **Acm Sigkdd Explorations Newsletter**, [S.l.], v.17, n.1, p.8–13, 2015.

BARUA, A.; THOMAS, S. W.; HASSAN, A. E. What are developers talking about? an analysis of topics and trends in stack overflow. **Empirical Software Engineering**, [S.l.], v.19, n.3, p.619–654, 2014.

BROWNLEE, J. **SMOTE for Imbalanced Classification with Python**. Accessed: 29/10/2021, https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/.

CHAWLA, N. V. et al. SMOTE: synthetic minority over-sampling technique. **Journal of artificial intelligence research**, [S.l.], v.16, p.321–357, 2002.

CODIGOFLUENTE. **Aula 08 scikit learn maquina de vetores de suporte**. Accessed: 17/06/2019, https://www.codigofluente.com.br/aula-08-scikit-learn-maquina-de-vetores-de-suporte/.

FLESCH, R. A new readability yardstick. **Journal of applied psychology**, [S.l.], v.32, n.3, p.221, 1948.

GEURTS, P.; ERNST, D.; WEHENKEL, L. Extremely randomized trees. **Machine learning**, [S.l.], v.63, n.1, p.3–42, 2006.

HAN, J.; PEI, J.; KAMBER, M. **Data mining**: concepts and techniques. [S.l.]: Elsevier, 2011.

KNOCHENHAUER, L. V.; DORNELES, C. F.; WIVES, L. K. WANQA: uma abordagem para identificar novas questões não respondíveis em comunidades de perguntas e respostas. In: SBBD. **Anais...** [S.l.: s.n.], 2018. p.1–12.

KOZAREVA, Z.; MONTOYO, A. Paraphrase identification on the basis of supervised machine learning techniques. In: **Advances in natural language processing**. [S.l.]: Springer, 2006. p.524–533.

KUMAR, V. et al. Ensembling classical machine learning and deep learning approaches for morbidity identification from clinical notes. **IEEE Access**, [S.l.], v.9, p.7107–7126, 2020.

LEE, H. D. **Seleção e construção de features relevantes para o aprendizado de máquina.** 2000. Tese (Doutorado em Ciência da Computação) — Universidade de São Paulo.

LORENA, A. C.; CARVALHO, A. C. de. Uma introdução às support vector machines. **Revista de Informática Teórica e Aplicada**, [S.l.], v.14, n.2, p.43–67, 2007.

LORENA, A. C.; CARVALHO, A. d. Introduçaoas máquinas de vetores suporte. **Relatório Técnico do Instituto de Ciências Matemáticas e de Computação (USP/Sao Carlos)**, [S.l.], v.192, 2003.

MAIER, O. et al. Extra tree forests for sub-acute ischemic stroke lesion segmentation in MR sequences. **Journal of neuroscience methods**, [S.l.], v.240, p.89–100, 2015.

MALDONADO, S.; LÓPEZ, J.; VAIRETTI, C. An alternative SMOTE oversampling strategy for high-dimensional datasets. **Applied Soft Computing**, [S.l.], v.76, p.380–389, 2019.

MITCHELL, T. M. **Machine Learning**. 1.ed. New York, NY, USA: McGraw-Hill, Inc., 1997.

OLIVEIRA, A.; OLIVEIRA, T. **Elementos de estatística descritiva**. [S.l.: s.n.], 2011.

SAHA, R. K.; SAHA, A. K.; PERRY, D. E. Toward understanding the causes of unanswered questions in software information sites: a case study of stack overflow. In: JOINT MEETING ON FOUNDATIONS OF SOFTWARE ENGINEERING, 2013. **Proceedings...** [S.l.: s.n.], 2013. p.663–666.

SENDERS, J. T. et al. An introduction and overview of machine learning in neurosurgical care. **Acta neurochirurgica**, [S.l.], v.160, n.1, p.29–38, 2018.

SHAH, K. et al. A comparative analysis of logistic regression, random forest and KNN models for the text classification. **Augmented Human Research**, [S.l.], v.5, n.1, p.1–16, 2020.

SHALEV-SHWARTZ, S.; BEN-DAVID, S. **Understanding machine learning**: from theory to algorithms. [S.l.]: Cambridge university press, 2014.

STACKEXCHANGE. **Stack Exchange. Stack Exchange Data Explorer**. Accessed: 15/06/2018, https://data.stackexchange.com.

STACKOVERFLOW. **Stack Overflow. How do I check if a list is empty?** Accessed: 15/06/2018, https://stackoverflow.com/questions/53513/how-do-i-check-if-a-list-is-empty.

YANG, J. et al. Asking the right question in collaborative q&a systems. In: ACM CONFERENCE ON HYPERTEXT AND SOCIAL MEDIA, 25. **Proceedings...** [S.l.: s.n.], 2014. p.179–189.

YANG, L. et al. Analyzing and Predicting Not-Answered Questions in Community-based Question Answering Services. In: OF THE 25TH . **Anais...** [S.l.: s.n.], 2011.

ZHU, W.; LIN, Y. Using GINI-index for feature weighting in text categorization. **Journal of Computational Information Systems**, [S.l.], v.9, n.14, p.5819–5826, 2013.