UNIVERSIDADE DE SÃO PAULO

INSTITUTO DE MATEMÁTICA E ESTATÍSTICA

TRABALHO DE CONCLUSÃO DE CURSO

Análise de sentimentos em comentários da plataforma Colab

Aluno: Cesar Cano de Oliveira

Orientador: Marco Dimas Gubitoso

Resumo

O presente trabalho de conclusão de curso teve como objetivo o desenvolvimento de um classificador da emoção predominante em comentários da plataforma Colab.

Para obter um classificador de alto desempenho, foram estudados e implementados dois modelos baseados em aprendizado de máquina, o *naive Bayes* e a regressão logística. Também foram estudados vários métodos de pré-processamento, com destaque para a implementação do método de seleção baseado em informação mútua.

A biblioteca scikit-learn, em Python, foi utilizada para fins de comparação e possível aplicação final. Diversos experimentos foram feitos para decidir os melhores métodos e classificadores, utilizando validação de Monte Carlo e métricas como acurácia e F1 score.

Por fim, obteve-se um classificador de ótimo desempenho, com 81% de acurácia nos testes, que será utilizado em futuras aplicações reais na plataforma Colab, automatizando análises qualitativas, por exemplo.

Conteúdo

1 Introdução **5**

2 Contextualização **6**

2.1 Análise de sentimentos 6

2.2 Aprendizado de máquina 7

2.3 Colab 9

3 Conceitos **11**

3.1 Naive Bayes 11

3.2 Regressão Logística 15

3.3 Pré-processamento 20

3.3.1 Limpeza de dados 20

3.3.2 Extração de *features* 21

3.3.3 Seleção de *features* 22

3.3.4 Informação Mútua 22

3.4 Métricas 24

3.4.1 Acurácia 24

3.4.2 Precisão e revocação 24

3.4.3 F1 Score 26

4 Metodologia **27**

4.1 Tecnologias utilizadas 27

4.2 Detalhes dos dados 27

4.2.1 Obtenção e números 27

4.2.2 Problemas e suposições 28

4.2.3 Definições de polaridade 30

4.3 Experimentos 32

5 Resultados **37**

5.1 Pré-processamento 37

5.2 Implementações próprias 39

5.2.1 Naive Bayes 39

5.2.2 Regressão logística 39

5.3 Scikit-learn 41
5.4 Comparação final 43

6 Conclusão **44**
6.1 Aplicações 44
6.2 Possíveis melhorias 44

7 Agradecimentos **45**

1 Introdução

Ter conhecimento sobre a emoção expressa em um texto sobre determinado evento é de grande utilidade em diversas análises de dados, podendo ser utilizado para melhorar serviços de forma mais inteligente, direcionando os esforços aos assuntos mais relacionados a emoções negativas, por exemplo.

Infelizmente, analisar o sentimento predominante em um texto é difícil até mesmo para seres humanos, dada a natureza subjetiva da tarefa e possíveis divergências entre diferentes pessoas treinadas para o trabalho. Por isso, o uso de algoritmos e modelos computacionais é cada vez mais difundido para esse tipo de problema.

O objetivo do presente trabalho é o desenvolvimento de um classificador de textos em relação à polaridade do sentimento predominante, considerando uma possível aplicação real dentro da plataforma Colab.

Para atingir esse objetivo, foram estudados e implementados modelos comuns para esse tipo de aplicação, e comparados com modelos já implementados de uma biblioteca amplamente utilizada.

Além disso, também foram estudados e, empiricamente testados, métodos de pré-processamento, com a finalidade de melhorar o desempenho, tanto dos classificadores próprios quanto dos já implementados.

O trabalho será descrito conforme a seguinte estrutura:

Primeiro, é feita uma contextualização sobre a área de estudo envolvida, uma breve explicação da categoria dos algoritmos utilizados, além de explicar o domínio da aplicação para a qual o trabalho foi desenvolvido.

Em seguida, são explicados os conceitos, implementações, algoritmos (pseudocódigo), etapas de pré-processamento e respectivos métodos envolvidos no desenvolvimento do trabalho.

Após essa etapa, são descritos a metodologia e os materiais, isto é, tecnologias utilizadas, detalhes sobre obtenção e características dos dados, suposições feitas para simplificar e/ou lidar com problemas tanto de implementação quanto de outras naturezas. Dentro da metodologia, também será descrita a maneira com a qual os experimentos foram conduzidos.

Por fim, são apresentados os resultados, da forma mais clara e visual pos-

sível, seguida de uma breve conclusão.

2 Contextualização

2.1 Análise de sentimentos

Análise de sentimentos é um campo de estudos que tem por objetivo extrair, identificar e/ou classificar o conteúdo de sentimentos a partir da linguagem. Vários problemas são considerados parte da área de análise de sentimentos e não há apenas uma única técnica para cada um deles. Podemos ter aplicações simples, como querer identificar se um determinado texto expressa ou não qualquer sentimento que seja, até os casos mais complexos onde gostaríamos de identificar a emoção predominante no texto, se o autor estava sentindo tristeza, felicidade, raiva, etc.

O presente trabalho focará na tarefa de, dado um texto, classificar a polaridade da emoção predominante, se é positiva, negativa ou neutra (emoção ausente/apolar). É importante ressaltar que, nessa área, as definições do que é positivo, negativo e neutro não são universais, cada domínio linguístico tem suas particularidades e o que pode ser positivo em um pode ser negativo em outro. Por isso, o domínio no qual o trabalho foi desenvolvido será explicado e detalhado mais a frente, bem como suas definições de polaridade.

Existem diversos métodos que podem ser utilizados ao desenvolver algoritmos para análise de sentimentos, especificamente para o problema de classificação de polaridade: sistemas de regras, algoritmos baseados em aprendizado de máquina, ou até mesmo abordagens mistas.

Algoritmos que usam sistemas de regras são o mais próximo da programação convencional, onde especificamos um conjunto de regras para o processamento dos textos e assim, ao fim do processamento determinamos a polaridade do texto. Por exemplo, poderíamos determinar um conjunto de palavras que serão consideradas importantes e atribuírmos uma pontuação associada a cada ocorrência daquele termo no texto, ou apenas sua presença ou não, o tipo de abordagem depende da aplicaçao e costuma ser testado empiricamente. Assim, na frase "**Adorei** o produto, **satisfez** todas minhas necessidades", poderíamos

considerar que as palavras em negrito possuem pontuação positiva enquanto todas as outras não explicitam polaridade nenhuma e têm pontuação nula. Então, essa frase possuiria uma pontuação total maior do que zero e seria classificada como expressando um sentimento positivo. Esse método pode ser muito bom em alguns casos mas, quando as pontuações não são sempre as mesmas e dependem do contexto, ele apresenta resultados bem insatisfatórios, além de exigir o conhecimento prévio de pontuações para cada palavra possível, o que nem sempre é viável.

Outra abordagem que é bastante utilizada na área, e a que foi utilizada no trabalho desenvolvido, é a baseada em técnicas de aprendizado de máquina, que vem ganhando cada vez mais importância, considerando que vivemos num mundo com uma quantidade enorme de dados, que é uma necessidade do aprendizado de máquina e, muitas vezes, sua maior vantagem sobre os outros métodos.

2.2 Aprendizado de máquina

O aprendizado de máquina é um ramo da área de inteligência artificial, onde são estudados e desenvolvidos algoritmos nos quais o que é implementado, de fato, é a maneira com a qual o algoritmo melhora seu desempenho para executar determinada tarefa, contrastando assim com os algoritmos "tradicionais"em relação ao que de fato é tarefa do programador.

Como podemos ver na figura 1, nos algoritmos tradicionais, explicitamos as regras com as quais os dados serão processados para obtermos as respostas. Já em aprendizado de máquina (figuras 2 e 3), escolhemos um modelo que possui suas próprias regras com as quais conseguirá aperfeiçoar suas capacidades de obter a resposta após uma fase de treino.

Existem dois tipos principais de algoritmos de aprendizado de máquina: os supervisionados, que necessitam tanto dos dados quanto de suas respectivas respostas, e os algoritmos não supervisionados, que só necessitam dos dados para criar regras que os façam modelar alguma função relevante em relação aos dados.

Ambos precisam de uma etapa de treino, que é quando os dados (e res-

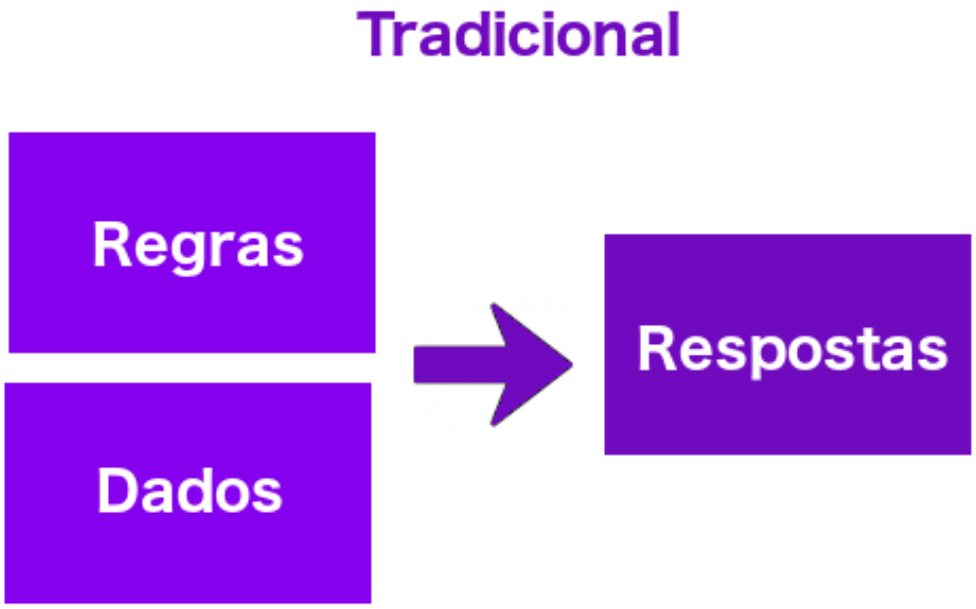

Figura 1: Algoritmos "tradicionais".

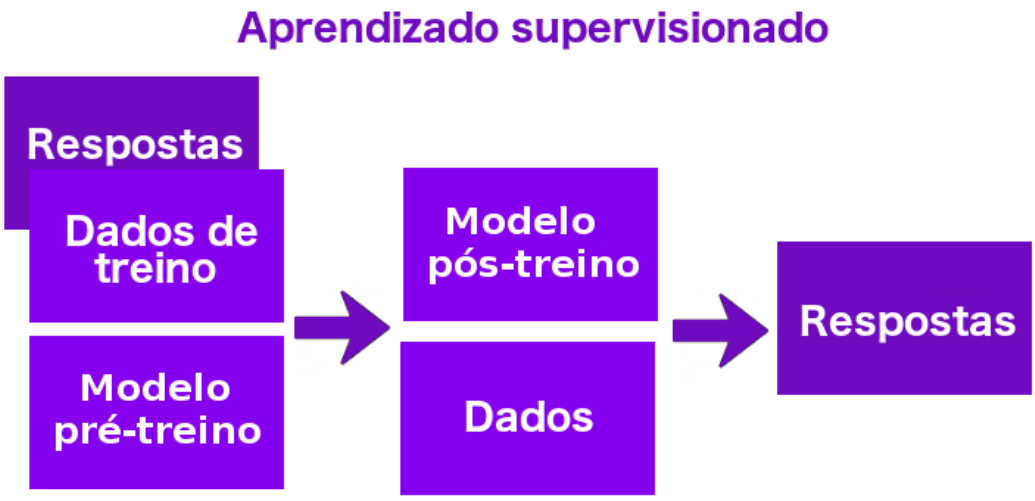

Figura 2: Algoritmos de aprendizado supervisionado.

postas, quando supervisionado) são fornecidos ao algoritmo para que ele, de acordo com cada algoritmo, consiga obter uma função que, após o treino, consiga predizer a resposta correta para dados não vistos.

No presente trabalho só trataremos de algoritmos de aprendizado de máquina supervisionado, portanto, sempre teremos em mente uma etapa de treino com dados previamente classificados, isto é, com uma resposta já conhecida associada a cada instância.

Figura 3: Algoritmos de aprendizado não supervisionado.

2.3 Colab

Em problemas de classificação de texto em geral, conhecer as características do domínio da aplicação é muito importante. É ele que determina quais palavras têm mais importância, e palavras podem mudar de significado e/ou polaridade se considerarmos domínios diferentes.

O Colab é uma rede social para cidadania, onde os cidadãos acessam um aplicativo, tanto *web* quanto *mobile*. Nela é possível reportar problemas de sua cidade, publicando fotos com alguma das categorias permitidas e a localização do problema.

Assim, a prefeitura, quando inscrita no aplicativo, responde e soluciona os problemas publicados. As publicações têm um conjunto de comentários associados, que podem ser tanto comentários da prefeitura quanto de cidadãos que estão acompanhando o andamento daquela fiscalização.

Dentro da plataforma Colab, também existe o lado administrativo, onde as prefeituras podem visualizar informações sobre as fiscalizações publicadas, administrar as mensagens e soluções para os problemas.

Os classificadores desenvolvidos ao longo do presente trabalho foram feitos com o objetivo de determinar a polaridade da emoção predominante em cada um desses comentários.

O perfil do usuário médio, o intuito da plataforma e a própria natureza da linguagem entre cidadão e prefeitura devem ser levados em consideração

na hora de analisarmos o domínio e entender suas características de forma a maximizar o desempenho das técnicas aplicadas. Um exemplo disso é observar que o uso de alguns emojis essencialmente positivos, como o sinal de positivo com o dedão, que muitas vezes é usado ironicamente por pessoas mais jovens em redes sociais, raramente é usado dessa forma no Colab e, assim, podemos considerar ele como um indicativo de emoção positiva associada ao comentário em que ele ocorre.

3 Conceitos

Para evitar uma possível confusão, usaremos o termo *feature* como sinônimo das palavras de um determinado texto a ser analisado pelos classificadores e também para características inerentes a estrura do texto, mas que não são palavras. *Features* são todas características que são levadas em consideração pelos algoritmos. A palavra característica só não é usada ao longo do texto para se referir às *features* pois poderia haver confusão com outros usos da palavra.

3.1 Naive Bayes

Um dos modelos implementados durante o desenvolvimento do trabalho foi o modelo de classificador *naive Bayes*. O *naive Bayes* é um dos métodos mais simples utilizados na área de classificação de textos mas, apesar de sua aparente "ingenuidade", consequência de suas fortes suposições sobre os dados, ele funciona muito bem e obtém resultados competitivos em relação a outros métodos mais sofisticados.

A principal suposição que caracteriza o *naive Bayes* como *naive* (ingênuo) é a hipótese de independência das *features*, ou seja, ele considera que a ocorrência de todas palavras são independentes umas das outras dado o contexto da categoria. Outra suposição forte é a do modelo chamado de *bag of words* ou saco de palavras, onde supõe-se que a ordem das palavras dentro de um documento não tem importância, o que é claramente falso na prática, mas não causa complicações nos resultados e é uma simplificação muito cômoda de se fazer do ponto de vista de implementação.

Ele funciona de forma probabilística, fazendo uso direto do teorema de Bayes, para probabilidades condicionais, ao classificar os textos. Como é um método de aprendizado de máquina supervisionado, ele possui uma etapa de treino, onde são calculados estimadores ótimos para os parâmetros do modelo. Existem diversas variações desse modelo, mas focaremos na implementada no presente trabalho, o *naive Bayes* multinomial, onde leva-se em consideração o número de ocorrências de cada palavra no texto, e não apenas se ela ocorre ou

não.

Primeiro, vamos definir alguns termos que serão utilizados durante a explicação do funcionamento do método.

Temos um conjunto D de textos, que chamaremos de documentos, cada um associado a uma categoria $c \in C$. Cada documento é composto por palavras $p \in V$, onde V é o nosso vocabulário formado por todas as palavras que ocorrem nos documentos utilizados na etapa de treino do modelo.

O que gostaríamos de saber é a probabilidade de um documento pertencer a determinada categoria dadas as *features* que ele possui. Escreveremos isso como uma probabilidade condicional $P(c|d)$, onde d é a representação do documento como um conjunto de *features*. Usa-se então o teorema de Bayes

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

que, dentro do nosso contexto, torna-se

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$

Assim, temos uma equação para obter a probabilidade desejada se possuirmos estimativas para o lado direito da equação. Queremos calcular esse valor para cada uma das possíveis categorias, e consideraremos o documento pertencente a classe com maior probabilidade condicional. Vale observar que, como o denominador é diferente de zero e é o mesmo para todas categorias, podemos ignorar esse termo no momento da decisão.

Então, a tarefa da etapa de treino é estimar os valores de $P(c_j)$ para todo $c_j \in C$ e $P(d_i|c_j)$ para todo par $d_i \in D$ e $c_j \in C$. Para estimar cada $P(c_j)$ simplesmente contamos, para cada categoria, entre todos os documentos de treino, quantos documentos pertencem a ela e temos a frequência relativa da categoria:

$$P(c_j) = \frac{|\{d \in D|cat(d) = c_j\}|}{|D|} \tag{1}$$

Já para calcularmos cada $P(d_i|c_j)$, primeiro lembremos que d é só o conjunto de *features* do documento d, e como nossa hipótese é de que essas *features* são independentes dada a classe, podemos escrever essa probabilidade como um

produto da probabilidade de cada *feature* dado c_j.

$$P(d_i|c_j) = P(x_1, x_2, ..., x_n|c_j)$$

onde n é o número de *features* no documento d_i. Sendo assim, nosso problema agora é saber, para cada par *feature* x_i e c_j, a probabilidade $P(x_i|c_j)$ e assim conseguiremos obter $P(d_i|c_j)$ para quaisquer pares de d_i e c_j. Esses valores também são obtidos através de contagem, para no fim termos a frequência relativa de uma *feature* em cada uma das classes. Assim, conta-se quantas vezes a *feature* x ocorre na classe c_j.

$$P(x_i|c_j) = \frac{\sum\limits_{d \in c_j} |x_i \in d|}{\sum\limits_{d \in c_j} \sum\limits_{x_k \in d} |x_k \in d|}$$

Temos, então, para um $d_i = (x_1, x_2, ..., x_n)$ e uma c_j:

$$P(d_i|c_j) = P(x_1, x_2, ..., x_n|c_j) = P(x_1|c_j) \cdot P(x_2|c_j)...P(x_n|c_j) \tag{2}$$

Agora, conseguimos calcular a probabilidade de um documento pertencer a cada uma das classes, e assim atribuir a classe de maior probabilidade ao documento.

E se, na fase pós treino, um documento possuir uma *feature* que nunca ocorreu em alguma das categorias? Teríamos na equação anterior um termo igual a zero, e isso faria com que toda a probabilidade $P(d_i|c_j)$ para a categoria onde essa *feature* nunca foi encontrada durante a fase de treino seja zero, algo que não gostaríamos afinal é como se tivéssemos ignorado todas as outras *features* apenas por esse motivo. Nesse caso, uma técnica chamada suavização de Laplace (*Laplace smooth*, não confundir com *Laplacian smoothing*), é utilizada a fim de evitar esse problema. Ela consiste em considerar que cada *feature* ocorre pelo menos uma vez em cada uma das categorias. Essa modificação resolve tanto o problema apresentado quanto outros casos extremos que outras soluções comuns não cobririam. Em muitas implementações é comum lidar com esses termos sem ocorrência apenas ignorando-os, como se não existissem no documento, mas para o caso extremo no qual nenhuma *feature* do documento a ser classificado é presente nos documentos de treino, o modelo não teria nenhuma informação para calcular os valores das probabilidades, já

a suavização de Laplace cuidaria desse problema, apesar de, claramente, não podermos considerar a classificação desse caso extremo como algo confiável. Então a equação modificada para calcular $P(x_i|c_j)$ é

$$P(x_i|c_j) = \frac{1 + \sum\limits_{d_k \in c_j} |x_i \in d_k|}{(\sum\limits_{d_k \in c_j} \sum\limits_{x \in d_k} |x \in d_k|) + |V|} \tag{3}$$

Por fim, tendo calculado os estimadores para os parâmetros do modelo a partir dos documentos de treino, podemos classificar qualquer documento a partir das *features* presentes nele, usando as equações (1) e (2) para cada uma das categorias, atribuindo a categoria de maior probabilidade ao documento.

Algoritmo 1 Etapa de treino - naive Bayes multinomial

$V \leftarrow$ lista de palavras nos comentários classificados
$C \leftarrow$ lista de categorias
$docs \leftarrow$ comentários classificados
for $c_j \in C$ **do**
  $docs_j \leftarrow$ comentários da categoria c_j
  $P(c_j) \leftarrow \frac{|docs_j|}{|docs|}$
end for
for $c_j \in C$ **do**
  for $x_i \in V$ **do**
    $P(x_i|c_j) = \frac{1+count(x_i,c_j)}{\sum\limits_{x \in V} count(x,c_j)+|V|}$
  end for
end for

Vale observar um detalhe no cálculo da probabilidade $prob_j(doc)$ de um comentário *doc* pertencer a uma determinada categoria. Esse cálculo envolve, na maioria das vezes, um produtório de muitas probabilidades próximas de zero e, isso pode causar um *underflow*, isto é, eventualmente o resultado é tão próximo de zero que é arredondado para zero e perde-se toda informação para aquele comentário. A solução é, então, utilizar os logaritmos das probabilidades, assim $prob_j(doc)$ vira um somatório, evitando qualquer risco de *underflow*.

Algoritmo 2 Etapa de classificação - naive Bayes multinomial

$docs \leftarrow$ comentários a serem classificados

for $doc \in docs$ **do**

$F \leftarrow$ lista de features em doc

for $c_j \in C$ **do**

$prob_j(doc) \leftarrow -log(P(c_j) + \sum_{f \in F} -log(P(f|c_j))$

end for

$class(doc) \leftarrow \min_{c_j \in C}(prob_j)$

end for

3.2 Regressão Logística

A regressão logística é uma das técnicas de aprendizado de máquina mais populares para resolver problemas de classificação binária, isto é, quando precisamos classificar os dados em uma de duas categorias discretas. Nessa seção, será explicado o algoritmo para o caso binário mas, como nosso problema envolve 3 categorias de classificação, também mostraremos como o modelo binário foi adaptado para o caso multiclasse. Serão apresentados o pseudocódigo da etapa de treino e da etapa de classificação para, por fim, discutir as limitações da implementação desenvolvida.

Consideremos então, o problema de classificar os comentários em positivos e não positivos (neutros e negativos em uma mesma categoria). Para qualquer problema de classificação binária, a regressão logística funciona aplicando a função sigmóide (também chamada de função logística) ao produto interno do vetor de *features* com um vetor θ de pesos, obtendo assim um valor no intervalo de 0 a 1 que representa, no caso exemplo aqui apresentado, a probabilidade do comentário representado por aquele vetor x de *features* ser classificado como positivo.

A nossa hipótese para um determinado comentário x então será dada pela equação:

$$h_\theta(x) = g(z)$$

onde:

$$z = \theta^T x$$

e g é a função sigmóide:

$$g(z) = \frac{1}{1 + e^{-z}}$$

A partir de um valor escolhido, normalmente 0.5, diremos que o comentário é positivo, se o valor da função logística for menor que isso, diremos que é não-positivo.

$$h_\theta(x) \geq 0.5 \rightarrow y = pos$$

$$h_\theta(x) < 0.5 \rightarrow y = \overline{pos}$$

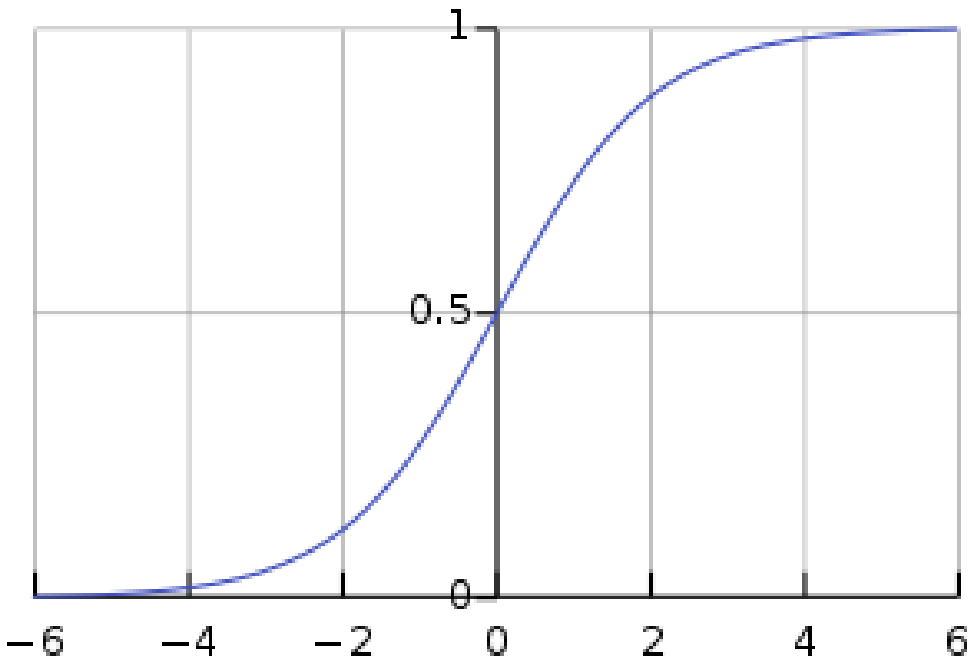

Figura 4: Gráfico da função sigmóide.

Precisamos estimar o vetor θ de pesos, é essa a tarefa da etapa de treino, na regressão logística. Definimos, então, uma função custo que, para cada comentário classificado, dirá o quão longe nossa hipótese atual chegou da classificação correta. Temos então um custo total (em função dos pesos estimados)

$J(\theta)$, que é a média aritmética do custo de cada comentário da etapa de treino. É essa função $J(\theta)$ que gostaríamos de minimizar, afinal, assim teríamos valores de θ que preveem com o menor erro possível as classificações reais dos comentários.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Custo(h_\theta(x_i), y_i)$$

Para resolver esse problema de minimização, utilizaremos o método do gradiente, que é um algoritmo de otimização iterativo que consegue achar o mínimo de uma função e, se a função for convexa, é garantido que esse mínimo é global. O método do gradiente funciona fazendo uso da propriedade de que se nos movermos um passo em direção contrária ao gradiente de dada função em um determinado ponto, estaremos sempre indo em direção ao mínimo dessa função, o tamanho desse passo é dado pelo parâmetro α, que chamamos também de taxa de aprendizado. Logo, precisamos ter certeza de que nossa função de custo total definida anteriormente seja convexa.

Lembremos que $J(\theta)$ é dado em função de cada par $h_\theta(x_i, y_i)$, onde x_i é o vetor de *features* que representa o comentário i e y_i é a categoria a qual ele pertence (estamos tratando com dados já classificados portanto possuímos a resposta de cada um deles). Ainda não definimos a função $Custo(h_\theta(x_i), y_i)$ e, não arbitrariamente, iremos escolher uma que garanta que $J(\theta)$ seja uma função convexa e, assim, podemos utilizar o método do gradiente com a certeza de acharmos valores ótimos de θ para nosso modelo.

$$Custo(h_\theta(x), y) = -log(h_\theta(x))$$

$$Custo(h_\theta(x), y) = -log(1 - h_\theta(x))$$

$$Custo(h_\theta(x), y) = -ylog(h_\theta(x)) - (1 - y)log(1 - h_\theta(x))$$

Definindo a função custo dessa maneira, garantimos que ela é convexa, e podemos fazer uso do resultado do método do gradiente.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}(y_i log(h_\theta(x_i)) + (1 - y_i)log(1 - h_\theta(x_i)))$$

Para fins de implementação, preferimos simplificar as contas fazendo as operações vetoriais possíveis. Dessa forma, a função $J(\theta)$ é definida por:

$$J(\theta) = \frac{1}{m} \cdot (-y^T log(h) - (1-y)^T log(1-h))$$

onde:

$$h = g(X\theta)$$

Sabendo que a função de custo é convexa, e tendo uma maneira de chegar nos valores de θ em que ela tem valor mínimo global, conseguimos obter parâmetros para um modelo que classifique da melhor forma possível, tendo nossos dados de treino.

$$\theta := \theta - \frac{\alpha}{m} X^T (g(X\theta) - \vec{y})$$

É válido notar que o método do gradiente pode demorar muitas iterações para convergir ao valor mínimo, então é definido um ϵ para o qual se a diferença do valor da função de custo de uma iteração para outra do método do gradiente for menor que esse ϵ, o algoritmo assume que chegou num valor aceitável e suficiente para o parâmetro θ e encerra a otimização.

Assim, partimos para a etapa de avaliação da qualidade do modelo produzido a partir dos dados de treino. Para cada comentário, obtemos o respectivo vetor de features, aplicamos a ele a função logística e obtemos assim a predição de nosso classificador.

Com isso, podemos resolver problemas de classificação binária utilizando o método da regressão logística. Mas nosso problema envolve três categorias, como adaptar o algoritmo desenvolvido para esse caso mais geral? Iremos adotar uma solução simples, que consiste em treinar um classificador binário para cada uma das nossas categorias, ou seja, teremos 3 classificadores onde cada um determinará a probabilidade daquele comentário pertencer, ou não, à sua respectiva categoria. Essa abordagem recebe o nome de *one versus rest* (um contra o resto), porque é treinado um classificador binário para identificar se o texto pertence à uma categoria ou às outras duas, essas duas vistas como uma única categoria (o *rest* de *one versus rest*). A partir daí, consideraremos como a categoria predita a categoria da qual o classificador binário correspondente responder com maior probabilidade/certeza.

Algoritmo 3 Etapa de treino - regressão logística binária

$X \leftarrow$ comentários classificados como vetores de features

$y \leftarrow$ categorias dos comentários classificados

$\theta \leftarrow \vec{0}$

$m \leftarrow |X|$

for $J(\theta) - J(\theta_{prev}) > \epsilon$ **do**

$\theta_{prev} \leftarrow \theta$

$\theta \leftarrow \theta - \frac{\alpha}{m} X^T(g(X\theta) - \vec{y}$

end for

Algoritmo 4 Etapa de classificação - regressão logística one vs rest

$docs \leftarrow$ comentários a serem classificados

for $doc \in docs$ **do**

$X \leftarrow$ lista de features em doc

for $c_j \in C$ **do**

$prob_j(doc) \leftarrow g(\theta^T X)$

end for

$class(doc) \leftarrow \max_{c_j \in C}(prob_j)$

end for

3.3 Pré-processamento

O pré-processamento dos dados é o processo de limpar e preparar o texto para a classificação. Nessse processo são executadas diversas etapas, e o conjunto delas é o que chamaremos aqui de *pipeline* de pré-processamento. Os benefícios de uma boa *pipeline* de pré-processamento podem ir de simplesmente diminuir o tempo de treinamento dos classificadores com os dados de treino até aumentar o desempenho das classificações de forma significativa.

Existem inúmeros métodos de pré-processamento mas, infelizmente, a tarefa de classificação de texto sofre por falta de robustez, ou seja, há pouquíssimas técnicas que, independentemente da aplicação, mostram bons resultados quanto a melhoria de desempenho dos classificadores. Por esse motivo, foram testados diversos métodos, com o objetivo de desenvolver uma *pipeline* de pré-processamento que melhore significativamente cada um dos classificadores testados.

Os métodos aqui apresentados serão divididos em três categorias para melhor compreensão mas, ao mostrar os resultados de se aplicar cada um deles, serão tratados dentro de um único conjunto, já que compartilham o mesmo objetivo, o de melhorar o desempenho dos classificadores desenvolvidos por mudanças nos dados antes deles serem utilizados nas etapas de treino dos classificadores.

3.3.1 Limpeza de dados

Primeiramente, temos a categoria de métodos que visam remover o ruído dos dados através de transformações no texto. Dentro dessa categoria estão os métodos que também são bastante utilizados em análise de dados em geral, envolvem operações tais como: remoção de pontuação e/ou caracteres especiais, remoção de *whitespaces* (espaços em branco, tabulações, etc) em excesso e fazer com que todas palavras fiquem apenas em letras minúscula. Nela, temos também métodos que transformam o texto de forma a agrupar palavras que podem ser tratadas como uma só, podendo acontecer de diversas formas. Poderíamos corrigir todos erros de digitação, ortografia, e assim as palavras escritas incorretamente seriam todas agrupadas em uma única forma correta.

Pode-se também reduzir palavras semelhantes à um mesmo radical, o que é chamado de *stemming*, onde, por exemplo, as palavras "resolvido", "resolvida", "resolver"todas seriam reduzidas para "resolv". De todos métodos citados, o único que não foi testado durante o desenvolvimento do trabalho foi o agrupamento de erros de digitação, ortografia e variações de uma mesma palavra, por não possuirmos um bom dicionário para relacionar palavras que podem ser agrupadas dentro do domínio do problema.

3.3.2 Extração de *features*

A segunda categoria de métodos envolve a extração de *features* a partir do texto, ou seja, transformar o resultado da busca de características específicas do texto em informação útil a ser utilizada pelos algoritmos. Entre as categorias de pré-processamento, essa é a mais dependente do domínio do problema, afinal ela tira proveito das características inerentes aos textos. Um exemplo clássico envolve a classificação de mensagens de e-mail em spam e não-spam. Nesse exemplo, a ocorrência de muitas palavras em letra maiúscula no campo assunto do e-mail é um forte indicador de spam. Então, se considerarmos que se todas as palavras do campo assunto forem maiúsculas quer dizer que ele possui a *feature* assuntoEmMaiúscula, podemos aumentar significativamente a eficiência do classificador. Nesse caso, a correlação entre a *feature* e ser spam é fácil de se observar. Já em outros casos, achar as melhores *features* para se extrair é um trabalho fortemente empírico. Nessa categoria, as *features* levadas em consideração foram: a presença de emojis positivos ou negativos, determinados por um conjunto estabelecido manualmente; a presença de reticências, sendo considerado reticências a ocorrência de dois ou mais pontos consecutivos no texto; a presença de ponto de exclamação; a presença de ponto de interrogação; a presença de palavras com todas letras maiúsculas; a presença de asterisco; a presença de múltiplos pontos de exclamação; a presença de múltiplos pontos de interrogação.

3.3.3 Seleção de *features*

A terceira e última categoria é a única que não envolve transformações nos dados, ela engloba os métodos de seleção de *features*, isto é, são métodos que filtram palavras que não são significativas ou até mesmo prejudicam a eficiências dos classificadores. O critério utilizado para essa filtragem depende do método utilizado, podendo ser algo simples como remover todas palavras que ocorrem menos do que um número escolhido de vezes em todos os comentários ou algo mais sofisticado, envolvendo métodos estatísticos para maximizar a correlação entre as palavras que serão consideradas *features* e a polaridade dos documentos onde elas aparecem mais vezes. Nessa categoria, os métodos testados foram: remover palavras com comprimento menor ou igual a algum n; remover palavras com ocorrência menor ou igual a algum n dentre todos os comentários utilizados para treino; método de seleção baseado em informação mútua. O último desses foi implementado como parte do estudo e desenvolvimento do trabalho e é descrito a seguir.

3.3.4 Informação Mútua

Um método de seleção de *features* muito comum é utilizar a função que computa a informação mútua entre uma *feature* e uma categoria. Essa função mede com quanta informação a presença/ausência de determinada *feature* contribui em uma classificação correta para a categoria c no universo de comentários classificados.

Seja U uma variável aleatória que tem o valor $e_f = 1$ quando um comentário possui a *feature* f e $e_f = 0$ quando não contém. Seja D uma variável aleatória que tem o valor $e_c = 1$ quando um comentário pertence a categoria c e $e_c = 0$ quando não pertence. Temos a equação para a informação mútua entre f e c dada por:

$$A(f,c) = \sum_{e_t \in 0,1} \sum_{e_c \in 0,1} P(U = e_t, D = e_c)\ log_2\ \frac{P(U = e_t, D = e_c)}{P(U = e_t)P(D = e_c)}$$

Simplificaremos o cálculo usando os termos N_{ab} onde o índice $a \in 0, 1, *$ diz se estamos considerando, para os valores de 0, 1 e *, documentos com

ausência da *feature* f, presença da *feature* f ou todos, respectivamente. O índice $b \in 0, 1, *$ diz se estamos considerando, para os valores de 0, 1 e *, documentos que pertencem a categoria c, não pertencem a categoria c, ou todos.

Assim, simplificamos a equação acima para:

$$A(f,c) = \sum_{i \in 0,1} \sum_{j \in 0,1} \frac{N_{ij}}{N_{**}} \; log_2 \; \frac{N_{**} N_{ij}}{N_{i*} N_{*j}}$$

Valores utilizados para cálculo de informação mútua entre a *feature* "obrigado"e a categoria "neutra"

	categoria(c) = neu	categoria(c) != neu
obrigado $\in$ c	$N_{11} = 147$	$N_{10} = 222$
obrigado $\notin$ c	$N_{01} = 1650$	$N_{00} = 2781$

Escolhe-se então uma porcentagem das *features* a ser utilizada e, após obter a informação mútua para todas *features*, filtra-se as com menor valor até atingir a porcentagem desejada.

Algoritmo 5 Seleção de *feature* - informação mútua

$V \leftarrow$ conjunto de features
$p \leftarrow$ porcentagem de melhores features a utilizar
for $c \in C$ **do**
  for $f \in V$ **do**
    $A_c(f) \leftarrow \sum_{i \in 0,1} \sum_{j \in 0,1} \frac{N_{ij}}{N_{**}} \; log_2 \; \frac{N_{**} N_{ij}}{N_{i*} N_{*j}}$
  end for
end for
$melhoresFeatures \leftarrow \emptyset$
for $c \in C$ **do**
  $melhoresFeatures_c \leftarrow sorted(A_c, decrescente)[0:p]$
  melhoresFeatures.add($melhoresFeatures_c$)
end for
return $melhoresFeatures$

3.4 Métricas

3.4.1 Acurácia

A acurácia é uma das mais métricas mais comuns para avaliar o desempenho de um classificador, por ser intuitiva e simples. Ela é definida pela porcentagem de comentários que, na etapa de teste, o modelo classificou corretamente. Para muitos casos, essa métrica é suficiente para termos uma boa estimativa da qualidade do classificador, porém, ela pode nos levar a uma conclusão positiva quando, na verdade, o classificador em questão é muito ruim.

Um exemplo clássico dessa situação é quando temos duas categorias desbalanceadas, isto é, a grande maioria dos dados pertencem a uma delas, e pouquíssimos dados pertencem a outra. Se 80% dos dados pertencem a categoria predominante, e obtivermos uma acurácia de 80%, nada nos garante que o nosso classificador não é a simples função constante que diz, para qualquer instância dos dados, que ele pertence a essa categoria. Dessa maneira, atingiríamos uma acurácia de 80%, quando obviamente nosso classificador é ruim, dado que ele nunca classificaria uma instância da categoria menos frequente de forma correta.

Para que a métrica utilizada nos dê confiança em relação a esse tipo de cenário, além de também nos fornecer mais informações quanto aos tipos de erro que o classificador comete, é necessário adotarmos alternativas, sem abandonar inteiramente o uso da acurácia, tanto por sua simplicidade quanto pelo fato de que, como veremos a seguir, no caso específico desse trabalho, ela possui uma forte correlação com as outras métricas mais sofisticadas.

3.4.2 Precisão e revocação

Qual é a falha de usarmos apenas a acurácia? O que acontece é que só computamos a quantidade de acertos em relação ao total mas não levamos em consideração detalhes sobre cada uma das instâncias que levaram a essa métrica, deixando passar assim possíveis indicações de um classificador enviesado. Queremos, então, entender detalhes sobre cada acerto/erro do nosso método para, assim, obtermos métricas mais confiáveis. Uma maneira de fazer isso é

observando o que chamamos de matriz de confusão.

Com a matriz de confusão, é possível saber o número de verdadeiros positivos, falsos positivos, verdadeiros negativos e falsos negativos para cada categoria.

A partir dessas informações, podemos calcular diversas métricas de avaliação comumente utilizadas em estatística. Nesse trabalho, só levaremos em consideração duas delas, precisão e revocação.

A precisão de um classificador em relação a uma determinada categoria é uma métrica dada a partir da razão entre os verdadeiros positivos e os totais preditos para aquela categoria. Ou seja, dos preditos como positivos, quantos realmente eram positivos?

$$Prec(c) = \frac{VerdadeirosPositivos_c}{Preditos_c}$$

Já sua revocação em relação a determinada categoria é dada pela razão entre os verdadeiros positivos e os totais reais daquela categoria, a porcentagem do número real de comentários pertencentes àquela categoria que o classificador predisse corretamente.

$$Revoc(c) = \frac{VerdadeirosPositivos_c}{Totais_c}$$

Essas métricas são calculadas sempre em relação a apenas uma categoria contra as demais. Como nosso problema envolve 3 categorias, precisamos considerar uma matriz de confusão desse tipo para cada categoria. Com isso, calcularemos o valor da precisão e da revocação do classificador em relacão a cada uma delas.

Assim, temos 6 valores representando a qualidade de nosso classificador, mas gostaríamos de ter um único número como métrica, a fim de facilitar a visualização e comparação de diferentes classificadores e/ou mudanças de desempenho devido a alterações de parâmetros em um mesmo classificador.

3.4.3 F1 Score

Para combinar os valores de precisão e revocação para cada categoria, foi utilizada a métrica conhecida como F Score. O F score é uma média harmônica, com valor de 0 a 1. Existem diversas variações do F Score, utilizamos a F1, que serve para nossa suposição de que a precisão e a revocação são igualmente importantes, logo, na equação que define essa métrica, esses dois valores possuem mesmo peso para o valor final.

$$F_1(c) = 2 \cdot \frac{Prec_c \cdot Revoc_c}{Prec_c + Revoc_c}$$

Agora, dos 6 valores, fomos para 3 F1 scores, um para cada par de precisão e revocação. É necessário ainda decidir como juntarmos esses valores em uma única métrica que representará, finalmente, a qualidade do classificador. F1 Scores combinados para problemas de várias categorias podem ser calculados de diversas formas. Foi decidido por uma média aritmética, que poderia ser alterada se quiséssemos, por exemplo, valorizar um melhor desempenho em classificar corretamente os comentários positivos do que os outros. Temos, por fim, uma métrica mais sofisticada do que a simples acurácia e, ainda assim, representada por apenas um número no intervalo real de 0 a 1, como desejávamos.

$$F_1Score = \frac{\sum_{c \in C} F_1(c)}{|C|}$$

4 Metodologia

4.1 Tecnologias utilizadas

Todas implementações próprias foram feitas em Python 2.7. Os algoritmos já implementados fazem parte de uma biblioteca de aprendizado de máquina em Python chamada scikit-learn. A scikit-learn é uma biblioteca de aprendizado de máquina *open source*, com contribuidores ao redor do mundo. Ela possui diversas implementações prontas e amplamente utilizadas em diversas aplicações reais, inclusive para os classificadores estudados e implementados nesse trabalho. Por esse motivo, foi utilizada para servir de comparação e possivelmente escolha final dentre os classificadores desenvolvidos.

4.2 Detalhes dos dados

4.2.1 Obtenção e números

Para o treino e avaliação de todos algoritmos, tanto os implementados durante o desenvolvimento do trabalho quanto os utilizados para fim de comparação e/ou uso final, é necessário que se tenha dados classificados, ou seja, comentários da plataforma Colab com respectivas polaridades de sentimento associadas. As classificações para esses dados foram feitas manualmente, seguindo definições subjetivas para as polaridades, a fim de auxiliar o processo manual.

Os comentários foram obtidos a partir do banco de dados do Colab, considerando os seguintes critérios:

Foram extraídos comentários apenas de usuários comuns, evitando assim comentários de representantes de prefeituras. Além disso, os comentários também precisavam ser referentes a publicações com um certo nível de interação com a prefeitura, com o objetivo de ter dados de interações que, de fato, aconteceram e os sentimentos presentes são devido a ações/não-ações das prefeituras envolvidas.

Ao todo, foram utilizados 4800 comentários classificados para as etapas de treino e avaliação dos algoritmos, nos quais estão representadas 2651 publicações de 63 cidades diferentes, datadas de janeiro de 2015 até julho de 2017.

4.2.2 Problemas e suposições

Dados desbalanceados A maioria dos algoritmos mais populares para classificação de dados assume uma distribuição balanceada entre os dados classificados. Infelizmente, nossos dados não apresentam esse tipo de distribuição uniforme e, então, eram esperadas algumas complicações em relação a essa característica. Para os classificadores onde isso se mostrou um problema, a decisão foi de pesquisar alguns métodos simples para atacar esse problema. Os métodos escolhidos envolvem técnicas de *oversampling* e *undersampling*.

Oversampling é um tipo de técnica para aumentar a quantidade de amostras de uma determinada categoria. Mais especificamente, foi utilizado um *oversampling* aleatório que consiste em duplicar comentários aleatoriamente selecionados da categoria menos frequente (sentimento positivo). Já o *undersampling* tem a função de diminuir a amostragem de categorias muito frequentes, e foi feito aleatoriamente removendo dados classificados das categorias neutra e negativa, até obtermos uma quantidade suficiente para balancear os dados classificados.

Ironia/Sarcasmo Os métodos utilizados não conseguem lidar bem com comentários irônicos/sarcásticos, já que é uma dificuldade até dos classificadores mais robustos atualmente. Então, a decisão foi de retirar manualmente dos dados classificados que serão usados para treinar os classificadores. Essa remoção se deve ao fato de que queremos maximizar o desempenho dos classificadores em comentários não irônicos, acrescentar esse tipo de comentário só adicionaria um ruído nos dados de treino que prejudicaria o desempenho em geral.

Comentários repetidos Comentários repetidos por conta de problemas antigos na plataforma (que faziam com que as pessoas enviassem o mesmo comentários múltiplas vezes) foram removidos dos dados classificados manualmente. O objetivo dessa remoção é de evitar que os classificadores acabem enviesados para as *features* contidas nesses comentários duplicados já que eles seriam utilizados múltiplas vezes durante a etapa de treino.

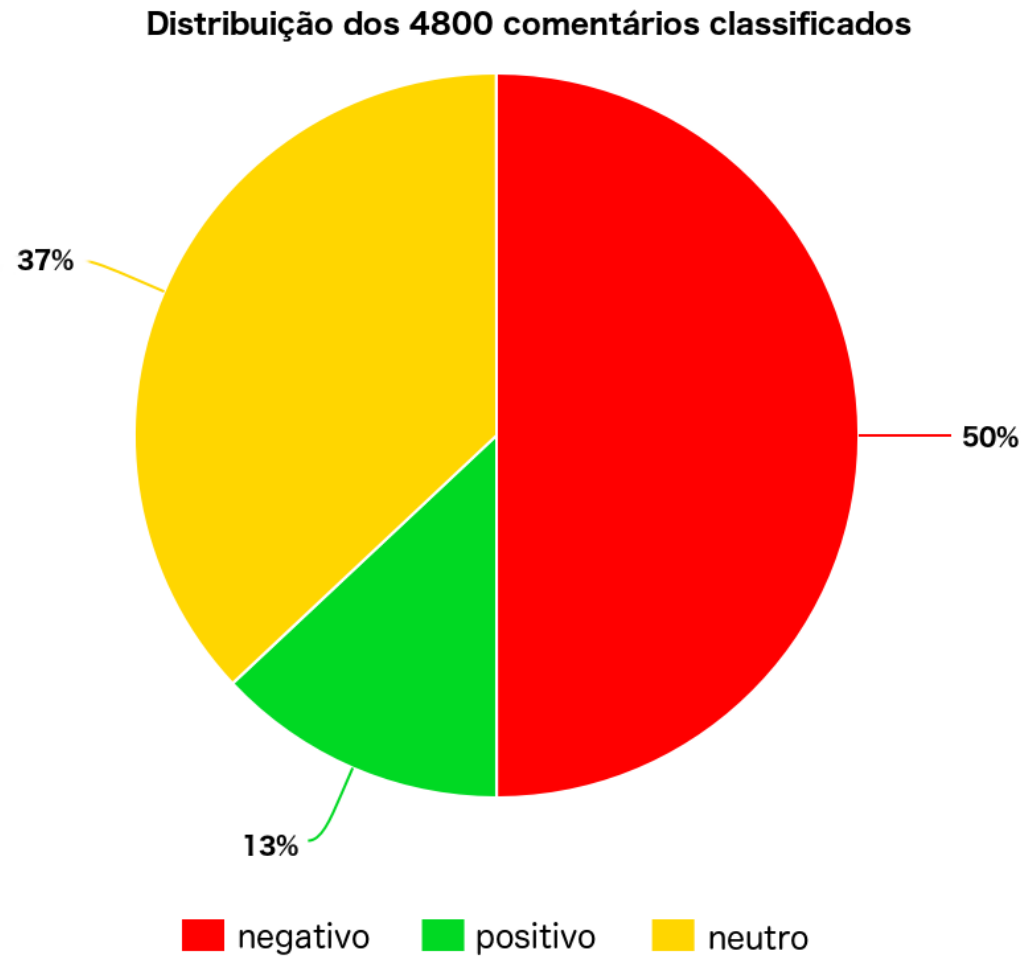

Figura 5: Gráfico de distribuição dos comentários entre as categorias

Erros ortográficos/de digitação Se nada for feito, palavras com erros de digitação ou ortografia são vistas como palavras diferentes de suas formas corretas e, pelo fato de aparecerem raramente e com diversas variações, acabam por não acrescentar nada no desempenho dos classificadores, ou pior, prejudicam o desempenho por adicionar ruído aos dados classificados. Como foram encontradas grandes dificuldades quanto a utilização de corretores ortográficos automáticos ou outros métodos que possivelmente trariam alguma melhora quanto a isso, esse problema não foi tratado, e esse ruído está presente nos dados classificados.

4.2.3 Definições de polaridade

Com o objetivo de auxiliar a classificação manual dos comentários e também deixar documentado um pouco do que, através do treino, pode ser visto como justificativas das regras por trás de cada classificador, dado que eles se baseiam nos dados de treino para desenvolver suas próprias regras, foi criado esse guia com as definições de polaridade de sentimento dentro do domínio do trabalho. É importante que todas as pessoas que participaram na classificação manual dos dados, que envolveu alguns funcionários do Colab, tenham usado os mesmos critérios, para que as classificações sejam consistentes e não exista um ruído tão grande de opiniões subjetivas quanto aos sentimentos presentes nos dados classificados.

Para cada possível categoria (positivo, neutro e negativo), foram listados os principais critérios para pertencimento com exemplos da própria base de dados.

Polaridade positiva A emoção associada a esse tipo de comentário é predominantemente positiva.

- Parabenização - "Resolvido. Implantação de faixa de estacionamento proibido do lado que interferia no campo de visão, rua Tutóia e demarcação de toda sinalização horizontal do cruzamento. ótimo trabalho. parabéns. Grato."

- Agradecimento efusivo (elaborados e/ou com exclamações) - "O serviço foi executado no dia 02/06/2017. Agradeço a colaboração! Serviço bem feito!"

- Mensagens positivas a outros usuários - "A Poluição Sonora também agride a natureza, parabéns Jose Francisco pela postagem!"

- Elogios - "muito obrigado!!!! Estão dando show de atenção e efetividade de serviços!!!!!"

- Constatação de fatos onde pode-se inferir que a pessoa está satisfeita - "Realizaram o reparo em menos de dois dias"

- Esperança sem conotação negativa - "Hoje cheguei em casa e constatei movimentação nos trabalhos. Acho que agora vaii!!"
- Elogio e observação/crítica na mesma mensagem mas com o elogio se sobrepondo - "Quero deixar aqui meu agradecimento, pela agilidade e o pronto atendimento, mas também gostaria de pedir de volta meu cone que estava sinalizando o local que o SAAE deve ter levado por engano. Parabéns pelo trabalho."
- Agradecimento ou constatação de solução com exclamação e/ou emoji positivo - "Resolvido! Galeria limpa :)!"
- Aguardando com agradecimento/exclamação positiva - "Ok, fico no aguardo! Obrigada pela atenção."

Polaridade negativa A emoção associada a esse tipo de comentário é predominantemente negativa.

- Xingamentos a entidades ou usuários - "parem de reclamar Zé povinho"
- Insatisfação - "Serviço mal feito, muito mal executado, logo logo estarei postando novamente."
- Cobranças - "E aí pessoal, vamos trabalhar não?"
- Constatação de solução com modificador negativo - "A prefeitura resolveu de forma rápida, porém com qualidade meia boca"
- Aguardando solução com modificadores negativos - "Aguardando providências, prefeitura..."

Ausência de polaridade Não há emoção associada a esse tipo de comentário.

- Requisitando informação sem demonstração explícita de negatividade - "Apenas um dúvida: essa programação é pra qual mês?"
- Constatação de fatos - "A praça não tem nome."

- Conversa neutra entre usuários - "Sr. Hildemar, protocolo conforme Lei Federal"
- Correções de digitação - "zona acalmada...erro do corretor..."
- Complemento de informação - "A placa está na rua Caeté, logo após a esquina com a Av. Pereira Passos."
- Mensagens fora de contexto - "BOA NOITE"
- Agradecimento por formalidade/sem modificadores - "Bom dia e obrigado."
- Constatação de solução sem modificadores - "Resolvido."
- Aguardando solução sem modificadores - "aguardando"
- Apontamento de falha sem expressar sentimento - "essa solicitação não foi atendido , favor verificar"

4.3 Experimentos

As métricas apresentadas anteriormente são relativas ao desempenho de um classificador após sua etapa de treino, no que chamamos de etapa de teste. Mas o seu desempenho pode mudar de acordo com a divisão dos comentários classificados entre dados de treino e dados de teste. Por isso, para evitarmos que a métrica associada a determinado classificador seja muito dependente de um treino específico, decidimos utilizar o método de validação cruzada de Monte Carlo. A validação de Monte Carlo é feita através de n instâncias de treino e teste, onde a distribuição dos dados é aleatória (dada uma porcentagem fixa para treino e teste) para cada uma das instâncias e, depois, tira-se a média das métricas de cada uma delas, obtendo assim a métrica final daquele classificador com aqueles dados/parâmetros.

Em todos experimentos feitos, 70% dos dados classificados eram dados de treino e os 30% restantes foram utilizados na etapa de teste, distribuídos aleatoriamente entre a totalidade de dados classificados. A quantidade n de instâncias utilizadas na validação de Monte Carlo foi de 100 repetições de

treino seguido de teste, cada um com métricas associadas. A métrica final para o classificador em cada experimento, acurácia e/ou F1 Score, foi definida pela média aritmética dessas 100 instâncias de treino e teste.

Para garantir uma comparação justa entre diferentes classificadores, é determinada uma semente do gerador pseudoaleatório, garantindo que, apesar de aleatórias, as distribuições das instâncias de treino/teste são as mesmas para cada um dos classificadores comparados.

Os experimentos foram focados em três classificadores: a implementação própria de regressão logística, a implementação própria de naive Bayes e a implementação da scikit-learn do modelo chamado Support Vector Machine (SVM), que será brevemente explicado na seção 5.3, por não ter sido estudado assim como os outros modelos.

Para essas três implementações, foram geradas curvas de aprendizado. A curva de aprendizado é um gráfico que apresenta a acurácia do classificador treinado para os dados de teste e de treino em função do aumento da quantidade de dados classificados. O objetivo dela é mostrar a evolução do modelo com o aumento de dados de treino e pode ser usada para diagnosticar possíveis pontos de melhora.

Era esperado observar que, quando a quantidade de dados fosse pequena, a acurácia para os dados de teste fosse a menor possível e para os dados de treino a maior possível, pois os modelos não tem informação o suficiente para generalizarem para dados desconhecidos (dados de teste) mas são muito bons em classificar os dados com os quais foi treinado. Com o aumento da quantidade de dados classificados a acurácia de treino diminui pois o modelo, ao generalizar e aumentar sua acurácia de teste, perde acurácia em relação aos dados de treino.

A *pipeline* foi comum a todos classificadores, com pequenas exceções descritas mais adiante. A ordem dos métodos foi a seguinte:

1. Retirar pontuações e caracteres especiais
2. Extrair *feature* em relação à presença de emojis (positivos e/ou negativos)
3. Remover espaços em branco em geral

4. Passar comentário para letra minúscula.

5. Extrair *feature* em relação a presença de reticências.

6. Extrair *features* em relação a presença única ou múltipla de pontos de exclamação/interrogação

7. Remover palavras que ocorrem raramente (menos de 2 vezes em todos dados classificados)

8. Remover palavras muito curtas (menos de 3 caracteres)

9. Aplicar método de seleção baseado em informação mútua

A porcentagem de *features* filtradas pelo método de informação mútua descrito na seção 3.3.4 foi escolhida empiricamente, e tem valores diferentes para cada tipo de classificador (mas implementações equivalentes usam o mesmo valor).

Apenas para a implementação própria da regressão logística foram feitos experimentos para observar a diferença entre o classificador antes e depois de aplicar as técnicas de *oversampling e undersampling* descritas na seção 4.2.2.

A implementação da scikit-learn para o classificador naive Bayes foi utilizada com os parâmetros padrão da biblioteca para a classe *MultinomialNB*, que implementa a variação multinomial do naive Bayes, a mesma implementada durante o desenvolvimento do trabalho.

A implementação da scikit-learn para regressão logística foi utilizada com parâmetro **solver="lbfgs", multi_class="ovr"**. O parâmetro multi_class explicita o uso da mesma abordagem da implementação própria, o *one versus rest*, mas vale ressaltar que essa implementação possui sofisticações ausentes da implementação própria. Fora esses dois parâmetros, foi utilizado o padrão da classe LogisticRegression da biblioteca.

O classificador SVM da scikit-learn foi o único que não utilizou os passos 7 e 8 da *pipeline* de pré-processamento, pois se mostraram significativamente prejudiciais ao seu desempenho.

Apesar de dada ênfase à importância da métrica alternativa F1 Score na seção 3.4.3, a maioria dos experimentos usam como métrica apenas a acurácia.

Isso se justifica por uma observação feita ao longo do desenvolvimento do trabalho. A acurácia e o F1 Score, nessa aplicação em particular, são altamente correlacionados e por esse motivo, deu-se preferência à acurácia, já que essa métrica tem um significado mais intuitivo para interpretação da qualidade do classificador. O F1 Score foi utilizado no experimento que compara todos classificadores, com o objetivo de deixar documentada com mais rigor a escolha do melhor classificador.

Figura 6: Fluxo de classificação pós treino.

5 Resultados

Os resultados de cada um dos classificadores implementados e dos classificadores da biblioteca scikit-learn utilizados serão apresentados nesta seção. Serão apresentados, também, os resultados de experimentos com foco nos métodos de pré-processamento e o ganho de desempenho associado.

5.1 Pré-processamento

Apesar dos resultados serem breves de se apresentar, muito tempo foi utilizado para escolha, desenvolvimento e experimentação envolvendo o maior número de métodos de pré-processamento possíveis. Durante o trabalho, foram observados 3 fatores determinantes no desempenho de um classificador baseado em aprendizado de máquina. São eles o modelo/algoritmo utilizado, a quantidade (e qualidade!) de dados classificados e, por último mas não menos importante, a *pipeline* de pré-processamento.

A tabela a seguir mostra o ganho em porcentagem na acurácia quando acrescentamos cada um dos métodos de pré-processamento na *pipeline*. É importante ressaltar que os experimentos foram feitos considerando uma *pipeline* 1, contendo todos métodos menos o da linha em questão, e uma *pipeline* 2, incluindo o da linha em questão. O valor apresentado é a diferença entre as acurácias da *pipeline* 2 e 1. Algo que não foi considerado e é uma possibilidade real, que foi observado em alguns testes durante o desenvolvimento do trabalho, é a diferença causada pela combinação de dois ou mais métodos, esse experimento só leva em consideração cada método isoladamente, mas alguns deles combinados podem modificar a acurácia de formas diferentes do que somente a soma de suas influências isoladas. Também não foram feitos experimentos levando em consideração a ordem do método na *pipeline*, quando essa ordem poderia fazer alguma diferença.

Diferença na acurácia do teste de
acordo com o uso de cada método de pré-processamento ou extração de *feature*

	scikit SVM	naive Bayes	regressão logística
emojis	+0.6%	+0.5%	+0.6%
único ? e !	+2.5%	+1.6%	+2.0%
multi ? e !	+0.2%	+0.4%	+ 0.2%
reticências	+0.5%	+0.5%	+0.6%
informação mútua	+2.2%	+1.7%	+2.0%
sem palavras raras/muito curtas	+0.3%	+3.9%	+0.6%
stemming	-0.5%	-2.3%	-0.7%
sem whitespace/pontuação	+0.4%	+0.6%	+0.4%

No gráfico a seguir, é possível ver a contribuição da *pipeline* de pré-processamento desenvolvida (dessa vez, completa) na acurácia dos classificadores.

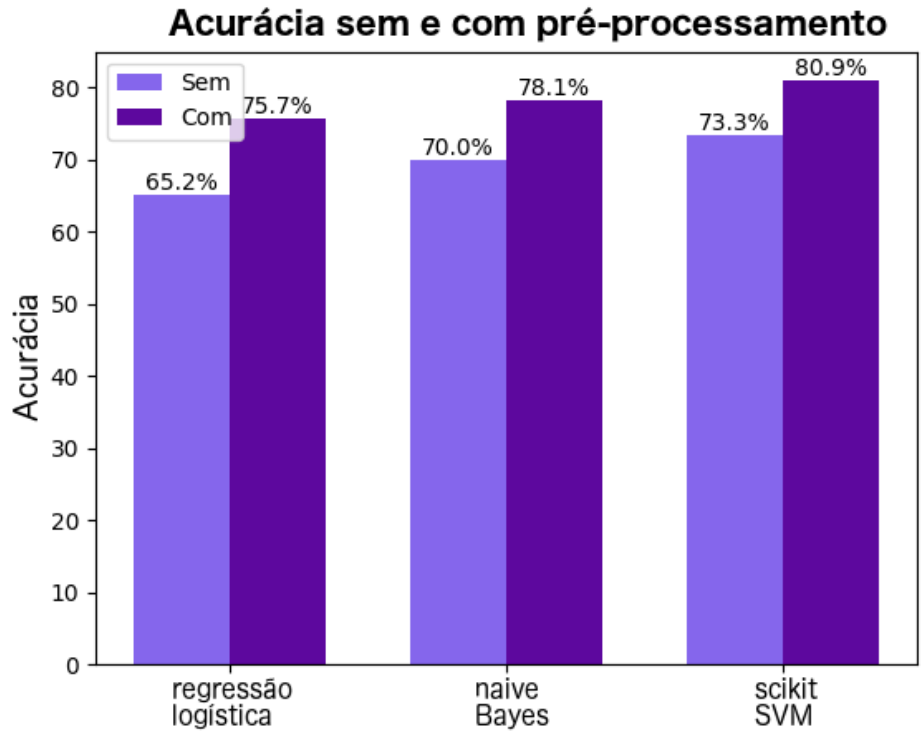

Figura 7: Diferença de acurácia para classificadores sem e com etapa de pré-processamento.

5.2 Implementações próprias

5.2.1 Naive Bayes

O naive Bayes foi o primeiro classificador a ser implementado durante o desenvolvimento do trabalho e, mesmo com poucos dados, já apresentou resultados promissores. Esse resultado foi visto com otimismo para o restante do trabalho mas também gerou certa frustração, dado que muitos dos métodos inicialmente testados não faziam diferença significativa no desempenho do naive Bayes. Foi possível melhorar seu desempenho através dos métodos de pré-processamento que foram se mostrando melhores ao longo do desenvolvimento do trabalho mas, em comparação com a implementação própria do classificador baseado em regressão logística, ele precisa de menos sofisticações para atingir um desempenho semelhante.

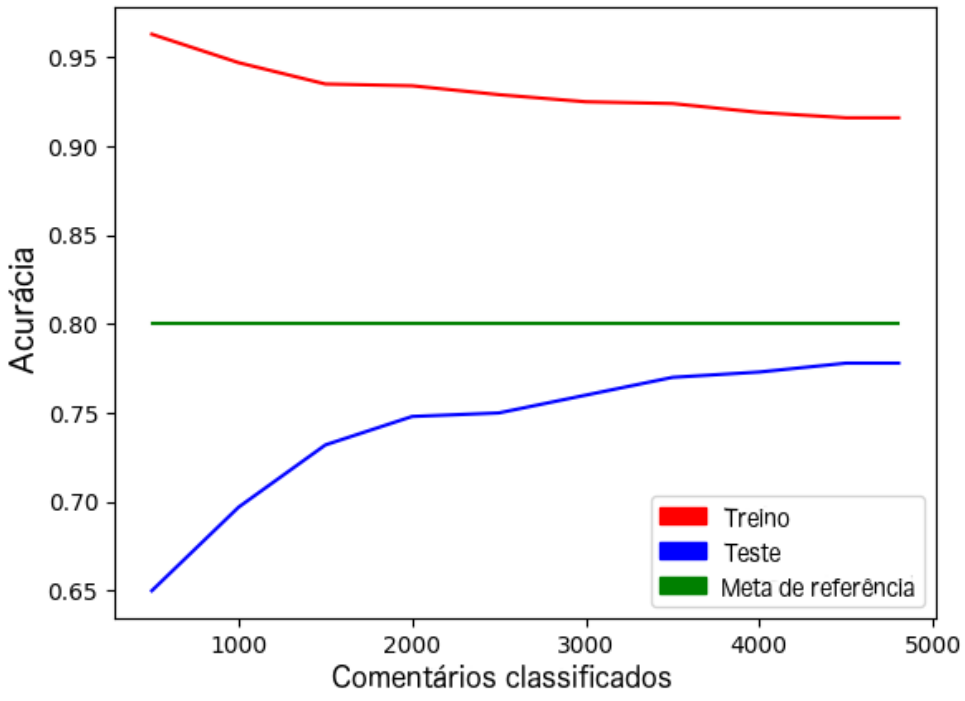

Figura 8: Curva de aprendizado para o naive Bayes próprio.

5.2.2 Regressão logística

A versão inicial do classificador com regressão logística teve péssimos resultados, algo que a princípio parecia ser relacionado exclusivamente ao fato de ter

sido implementado sem quase nenhuma sofisticação, usando a abordagem *one versus rest* ao invés de sua generalização sofisticada e com um algoritmo bem simples para convergência da função de custo. Após perceber que o problema das classes desbalanceadas era um dos principais problemas, foram aplicadas as técnicas de *oversampling* e *undersampling* descritas na metodologia. É possível ver, nos resultados a seguir, que essa medida, apesar de simples, mostrou-se extremamente eficaz para melhorar o desempenho do classificador.

Acurácia do classificador baseado em regressão logística

	sem pré-processamento	com pré-processamento
sem under/over sampling	54.8%	65.2%
com under/over sampling	65.0%	75.7%

F1 Score do classificador baseado em regressão logística

	sem pré-processamento	com pré-processamento
sem under/over sampling	0.434	0.548
com under/over sampling	0.569	0.742

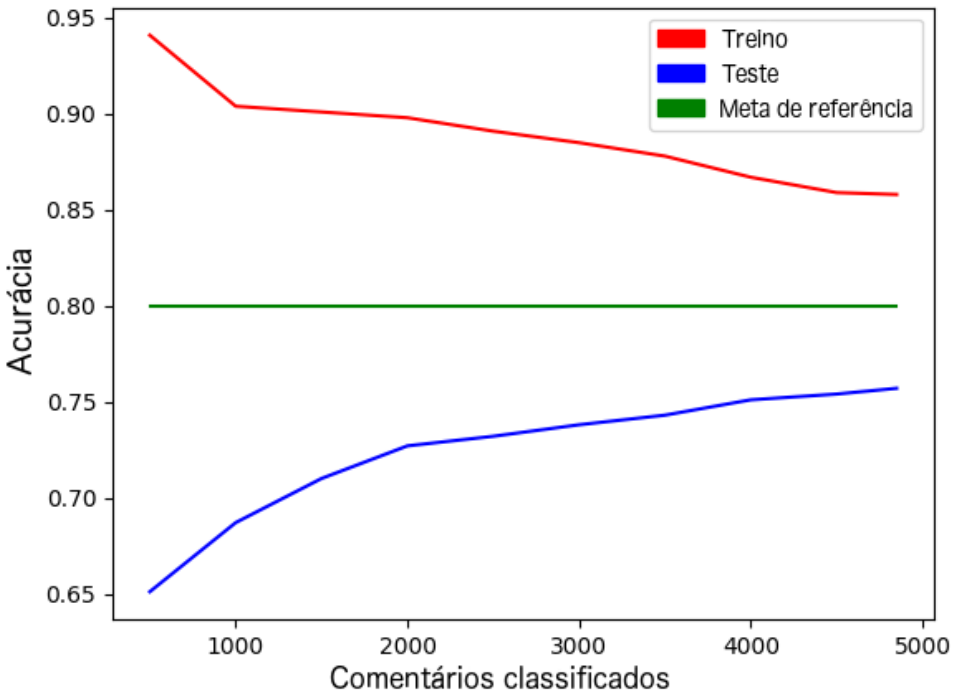

Figura 9: Curva de aprendizado para o logistic regression próprio.

5.3 Scikit-learn

A biblioteca scikit-learn foi utilizada com duas finalidades. A primeira era servir como comparação para validar os classificadores implementados durante o desenvolvimento do trabalho. A segunda era fazer experimentos com o maior número possível de classificadores que a biblioteca fornece uma implementação pronta e achar o melhor entre eles, dado que o foco do trabalho, por fim, era obter um classificador com bom desempenho para aplicações reais. Por falta de experimentos com os classificadores do scikit que não pareceram promissores em uma primeira investigação, focaremos nos resultados apresentados pelo classificador que melhor se adaptou ao problema e que foi escolhido para ser utilizado na aplicação real, a fim de classificar os comentários da plataforma Colab.

Support Vector Machine (SVM) SVM *(Support Vector Machine)* é um modelo de aprendizado supervisionado muito utilizado mas, como não foi es-

tudado a fundo e apenas sua implementação pronta foi utilizada para fins de comparação/possível aplicação final, os detalhes de seu funcionamento não constam nesse trabalho.

Simplificadamente, a ideia por trás do SVM é a de achar, por meio de um problema de otimização, uma separação geométrica entre as categorias dos dados classificados, com uma margem de tamanho máximo entre a separação e cada uma das instâncias de cada categoria.

Esse modelo foi o que apresentou melhores resultados dentre as implementações próprias, suas variantes do scikit-learn e alguns outros modelos da scikit-learn que foram testados com pouco rigor apenas para checar se existia algum potencial mas que, por falta de tempo, não foram testados a ponto de poderem ser incluídos no presente trabalho.

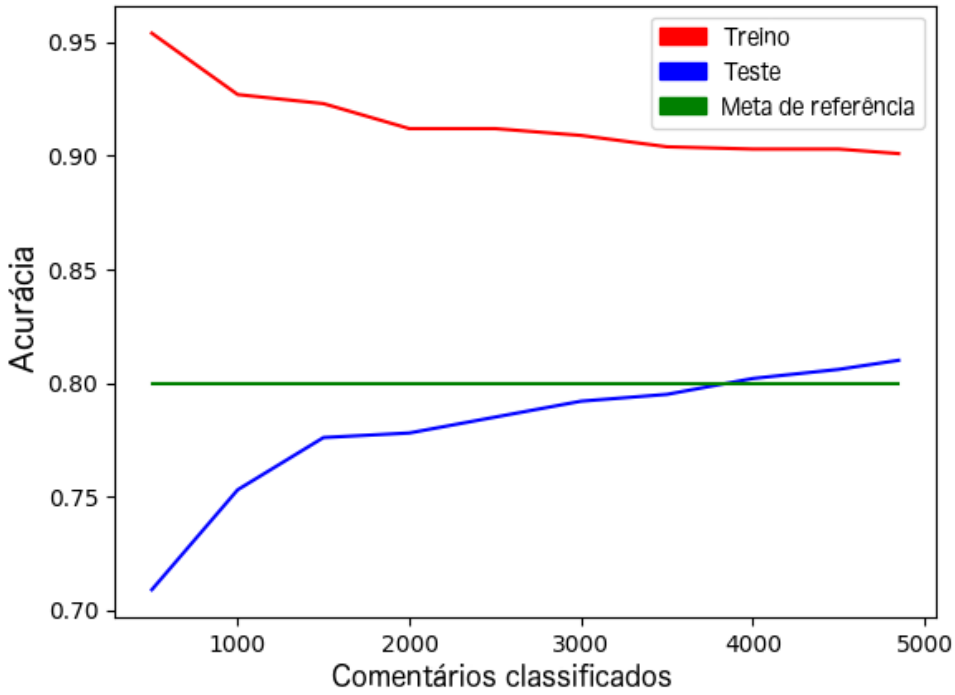

Figura 10: Curva de aprendizado para o scikit SVM.

Tendo em vista que esse modelo é o melhor candidato a ser utilizado na plataforma Colab, foi gerada a matriz de confusão para um de seus treinos, com o objetivo de observar os erros mais comuns do classificador, que costumam indicar um bom caminho para possíveis futuras melhorias.

Matriz de confusão para o scikit SVM

$\frac{real}{predito}$	positivo	neutro	negativo	total
positivo	118	52	10	180
neutro	23	433	93	549
negativo	9	88	615	712
total	150	573	718	

A diagonal principal da matriz, representa os acertos do classificador. Todas demais entradas são os erros, a coluna representa a classificação real do dado e linha representa a predição do classificador. Podemos ver que os erros mais comuns do modelo envolve classificar como negativo comentários neutros e vice-versa. O outro tipo de erro destacado na matriz é a predição de positivo quando o comentário é, na realidade, neutro.

5.4 Comparação final

Segue aqui, uma tabela com todos os classificadores para os quais experimentos foram feitos e métodos de melhoria de desempenho foram estudados e testados, para justificar a escolha do scikit SVM como classificador a ser usado em produção.

Acurácia e F1 Score para cada um dos classificadores

	Acurácia	F1 Score
scikit naive Bayes	75.4%	0.739
regressão logística	75.7%	0.742
naive Bayes	77.7%	0.762
scikit logistic regression	79.5%	0.784
scikit SVM	81.0%	0.790

Na tabela acima, pode-se observar que as implementações próprias chegaram a resultados muito semelhantes aos da scikit-learn, inclusive com a surpresa de que a implementação própria do *naive Bayes* superou a da scikit-learn. Também é possivel observar a correlação entre acurácia e F1 score. Por último, a justificativa para o uso do SVM da scikit como classificador final, com os 81% de acurácia alcançados.

6 Conclusão

6.1 Aplicações

Com o classificador escolhido, podemos armazenar o modelo (pós treino) e utilizá-lo em produção, na plataforma Colab.

Algumas aplicações foram discutidas e sugeridas pela equipe do Colab para auxiliar em processos que atualmente são feitos sem automação alguma. Um exemplo é a análise qualitativa do atendimento das prefeituras, que é um relatório que deve ser gerado periodicamente e envolve analisar as publicações em determinada cidade e, com alguns critérios subjetivos, determinar o grau de qualidade do atendimento. Nesse exemplo, o classificador pode servir para simplificar/automatizar o processo de avaliar o sentimento dos cidadãos em relação ao atendimento da prefeitura através dos comentários nas publicações do período analisado.

Outra possível aplicação discutida é a criação de gráficos na parte administrativa da plataforma (de acesso apenas para funcionários das prefeituras), com o intuito de fornecer visualizações do sentimento geral em publicações baseado em filtros como localização, categoria do problema, funcionário responsável, entre outros.

6.2 Possíveis melhorias

Existem diversas maneiras com as quais poderíamos obter classificadores com melhor desempenho do que os desenvolvidos durante o trabalho.

A implementação/classificador escolhido só pode nos ajudar até certo ponto desse processo. A partir de determinado momento, o que acaba sendo o diferencial é a quantidade de dados classificados, que faz com que o algoritmo consiga generalizar para um número cada vez maior de cenários, tornando-o mais eficiente. Mas, considerando os resultados dos experimentos desenvolvidos, pode-se dizer com uma certa confiança que ainda há possibilidade de melhoria em relação aos métodos de pré-processamento. Uma boa maneira de entender quais possíveis métodos de pré-processamento podem ser acrescentados ou modificados pode vir da análise da matriz de confusão, sabendo os

tipos de erro mais comum do classificador, podemos analisar os comentários onde os erros ocorrem em busca de suas características principais e, a partir delas, extrair *features* ou filtrar algum possível ruído em comum. Porém, deve-se observar que a acurácia atingida pelo SVM do scikit-learn se mostrou alta em relação a outros problemas de análise de sentimento e que é improvável que haja uma margem muito grande de melhorias sem tratarmos os problemas mais complexos de processamento de linguagem natural que foram ignorados nesse trabalho, como ironia e outros recursos de linguagem subjetivos, além de levar em consideração erros ortográficos, de digitação e abreviações.

7 Agradecimentos

Gostaria de agradecer a minha família e todos meus amigos, por me apoiarem durante a graduação. Dos gestos mais singelos até a companhia em momentos onde nada parecia dar certo, eu só tenho a agradecer.

Também gostaria de agradecer os colegas de curso, os professores e os funcionários do IME pelo apoio, ensinamentos, compreensão e paciência.

Agradeço também a todo mundo do Colab, pelo ótimo período de estágio que tive com todos de lá, os aprendizados, a oportunidade de poder desenvolver um trabalho desses para uma aplicação real, o dia a dia no escritório e a amizade.

"... I knew exactly what to do. But in a much more real sense, I had no idea what to do."