Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Análise Comparativa Entre Aprendizado Supervisionado e Aprendizado por Transferência Aplicados a Análise de Sentimentos em Textos

Rodrigo Demetrio Palma

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof. Dr. Thiago de Paulo Faleiros

Brasília
2020

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Análise Comparativa Entre Aprendizado Supervisionado e Aprendizado por Transferência Aplicados a Análise de Sentimentos em Textos

Rodrigo Demetrio Palma

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof. Dr. Thiago de Paulo Faleiros (Orientador)
CIC/UnB

Profa. Dra. Roberta Barbosa Oliveira
UnB

Prof. Dr. Luís Paulo Faina Garcia
UnB

Prof. Dr. João José Costa Gondim
Coordenador do Curso de Engenharia da Computação

Brasília, 18 de Dezembro de 2020

Dedicatória

Dedico este trabalho à minha família.

Agradecimentos

À Deus pois sem ele nada disso faria sentido.

Aos meus pais Márcia Demetrio Palma e José Palma Júnior por nunca terem medido esforços para me proporcionar um ensino de qualidade durante todo o meu período escolar.

À minha vó Helena, por ter um coração gigante. À minha irmã Bruna, pelo companheirismo. À Dimitria, por acreditar na minha capacidade e por estar ao meu lado.

Ao meu orientador Prof. Dr. Thiago de Paulo Faleiros por me guiar e me ajudar nas decisões deste trabalho. À todos os professores que fizeram parte da minha trajetória até hoje.

À UnB pela infra-estrutura oferecida. À CAPES, por meio do Acesso ao Portal de Periódicos.

Resumo

Análise de sentimentos em textos é uma tarefa muito estudada no meio acadêmico e cada vez mais novos trabalhos estão sendo publicados. Empresas que pretendem adotar este tipo de tecnologia para seu negócio precisam conhecer e decidir qual a melhor estratégia de implementação de acordo com o cenário específico em que ela se encontra. Assim, este trabalho mostra que técnicas mais clássicas e simples podem ser a melhor escolha, principalmente quando recursos computacionais são limitados ou pequenos ganhos em performance não são tão críticos. Para isso, é conduzida uma análise comparativa de duas abordagens para o problema de análise de sentimentos: a abordagem clássica, onde utiliza-se algoritmos supervisionados para classificação (positivo ou negativo), como *SVM e Naïve Bayes*, e a abordagem moderna, onde utiliza-se métodos de aprendizado por transferência. Experimentos foram realizados para validar a ideia proposta, onde foram aplicados um conjunto de algoritmos clássicos e um conjunto de técnicas de aprendizado por transferência em quatro bases de dados focadas na tarefa de análise de sentimentos. A análise comparativa obteve resultados relevantes, a qual mostrou que os algoritmos clássicos são competitivos mesmo obtendo acurácia mais baixa que os modernos e que de fato podem ser uma boa alternativa para cenários limitados.

Palavras-chave: Análise de sentimentos, mineração de textos, processamento de linguagem natural, aprendizado de máquina, aprendizado por transferência

Abstract

Sentiment analysis is widely studied and it has many papers published nowadays. Companies that seek to use this kind of technology in their business have to understand and decide which approach is the best for their specific scenario. Here, we show that classic approaches can be the best choice when limited hardware resource is available or small performance improvements are not critical. To show these issues, we conduct a comparative analysis between two sentiment analysis approaches. The classic approach, where supervised learning algorithms are used to classify the sentiment (positive or negative) as Support Vector Machine and Naive Bayes, and the new approaches, where the state-of-the-art transfer learning method is used. We perform a series of experiments to validate the proposed idea, and we create two groups of algorithms with four different data sets focused on sentiment analysis tasks, one with supervised-learning-based algorithms and another with transfer-learning-based models. The analysis achieved relevant results, showing that classic algorithms are competitive even with lower accuracy than the new algorithms and that they can be a suitable alternative in limited scenarios.

Keywords: Sentiment analysis, text mining, natural language processing, machine learning, transfer learning

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

Amazon R. Amazon Reviews.

CNN Redes Neurais Convolucionais.

DT Decision Tree.

GPU Unidades de processamento gráfico.

IMDb Internet Movie Database.

KNN K-nearest Neighbors.

NB Naïve Bayes.

NLP Processamento de linguagem natural.

RF Random Forest.

RNN Redes Neurais Recorrentes.

SGDC SGD Classifier.

SVM Support Vector Machines.

TF-IDF Term Frequency–Inverse Document Frequency.

TPBG Transductive Propagation in Bipartite Graph.

TPU Unidades de processamento de tensores.

Yelp R. Yelp Reviews.

Capítulo 1

Introdução

Este capítulo introduz e contextualiza o leitor ao tema abordado neste trabalho, define e caracteriza o problema proposto, aborda questões de pesquisa e por fim resume a estrutura geral do trabalho.

1.1 Contextualização

A produção de conteúdos textuais por meio da internet tem se intensificado cada vez mais com a vinda das redes sociais, das lojas virtuais que permitem discussões e comentários sobre produtos, de fóruns de pesquisa, etc. É esperado que empresas detentoras desses dados tenham interesse em analisá-los para entender melhor o que seu público pensa sobre determinado produto ou assunto.

Pensando nisso, estudos sobre processamento de linguagem natural, isto é, fazer com que o computador processe e entenda a linguagem humana, foram iniciados. Uma das tarefas que chamaram a atenção neste campo foi a análise de sentimentos, isto é, identificar o sentimento relacionado a um trecho textual.

Esta tarefa foi utilizada para vários fins, como por exemplo em [5], onde são extraídos sentimentos de textos da rede social *Twitter* [1], para prever o comportamento das ações na bolsa de valores dos Estados Unidos, ou em [6] que também utiliza da rede social *Twitter* para prever resultados de eleições na Alemanha. Na área de saúde, [7] tenta encontrar *tweets* relacionados ao vírus da influenza e compara com dados estatísticos da *Centers for Disease Control and Prevention*[2].

Uma das técnicas mais tradicionais e eficazes para analisar o sentimento em textos é tratar este problema como um problema de classificação, de forma que um texto possa ter um sentimento positivo ou negativo e então aplicar um dos algoritmos supervisiona-

[1] `https://twitter.com`
[2] `https://www.cdc.gov/`

dos tradicionais como *Support Vector Machines (SVM), Naïve Bayes (NB) e K-nearest Neighbors (KNN)* [8].

Em seguida, arquiteturas robustas e complexas começaram a ser desenvolvidas para atingir uma maior acurácia, utilizando, por exemplo, Redes Neurais Recorrentes (RNN) e Redes Neurais Convolucionais (CNN) [9].

De uns anos para cá, pesquisadores e empresas têm investido muito nesta área a fim de desenvolver técnicas mais avançadas para resolver os mesmos problemas com mais acurácia. Um dos métodos que mais vem ganhando destaque ultimamente é o chamado aprendizado por transferência, onde utiliza-se como base um modelo já pré-treinado em uma base de dados grande e sem domínio específico, e então o aperfeiçoam com novos dados mais específicos do domínio estudado, de forma que o aprendizado é *transferido* de um modelo antigo para o modelo novo. Esse método tem atingido estado-da-arte em várias tarefas de processamento de linguagem natural [4], [10] e [11], como análise de sentimentos, perguntas e respostas, etc. Além disso, ganha-se a vantagem de não ser mais necessário treinar modelos grandes do zero.

Novos trabalhos sobre aprendizado por transferência são publicados cada vez mais e com técnicas mais avançadas, onde tentam melhorar o processo de pré-treinamento ou de aperfeiçoamento, como em [4], [10] e [11]. Porém, estas técnicas são muito dependentes de recursos computacionais como Unidades de processamento gráfico (GPU) ou Unidades de processamento de tensores (TPU) e ainda assim demandam um tempo consideravelmente alto de treinamento quando comparadas aos métodos clássicos.

1.2 Definição do Problema

A escolha de qual técnica é mais adequada para a execução de análise de sentimentos em textos é uma decisão importante e muitas vezes feita considerando simplesmente a acurácia que certo algoritmo ou técnica alcança.

Consequentemente, os algoritmos mais complexos são escolhidos visto que atingem uma maior acurácia. Porém, estes muitas vezes demandam de recursos computacionais, como GPU's ou ainda TPU's, podem demorar horas para realizar o treinamento e o resultado as vezes possui pouca diferença em acurácia comparado a algoritmos mais leves.

1.3 Objetivo do Projeto

O objetivo deste projeto é mostrar ao leitor que os algoritmos clássicos ainda podem ser úteis em tarefas de processamento de linguagem natural e análise de sentimentos em textos e que nem sempre algoritmos estado-da-arte são a melhor opção. Para isto, é realizada

uma análise comparativa da performance e do tempo de treinamento e aperfeiçoamento de alguns dos principais algoritmos e modelos de linguagem estado-da-arte e algoritmos tradicionais de classificação.

1.4 Questões de pesquisa

Para instigar a curiosidade do leitor, foram inseridas algumas questões de pesquisa que este trabalho tenta responder:

- Algoritmos modernos e com melhor desempenho são sempre a melhor escolha para a tarefa de análise de sentimentos?
- O desempenho em relação ao tempo de processamento é significativamente grande entre abordagens tradicionais e modernas?
- Como os resultados deste trabalho podem auxiliar na decisão de qual abordagem deve ser utilizada?

1.5 Apresentação do Manuscrito

No Capítulo 3 são abordados trabalhos relacionados que foram utilizados para inspirar este trabalho. O Capítulo 2 introduz o leitor aos conceitos e algoritmos utilizados neste trabalho. O Capítulo 4 explica a metodologia utilizada para realização dos experimentos. O Capítulo 5 mostra ao leitor os resultados encontrados e faz uma análise comparativa e por fim. O Capítulo 6 faz uma conclusão, mostra algumas limitações e cita trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta ao leitor os conceitos teóricos necessários para o entendimento geral deste trabalho. Definições de mineração de texto e aprendizado de máquina são os principais tópicos.

2.1 Mineração de textos

A maioria dos dados produzidos na internet na forma de texto estão desorganizados e sem nenhum valor atrelado. Para que estes dados fiquem organizados e possam ser analisados e assim gerarem algum valor, é feita uma limpeza e estruturação dos dados, chamada de pré-processamento. Em seguida, aplica-se técnicas de aprendizado de máquina e/ou de Processamento de linguagem natural (NLP) para extrair conhecimento e padrões desses dados. Por fim, métodos de avaliações são utilizados para medir o desempenho do modelo utilizado. A combinação destas tarefas é a chamada mineração de textos. Cada tarefa é explicada detalhadamente a seguir.

2.1.1 Pré-processamento

Resumidamente, pré-processamento de dados é a conversão de dados não estruturados em dados estruturados [12]. Para isso, é feita uma limpeza nos dados coletados inicialmente, podendo ser feita de forma diferente para cada natureza do dado ou para cada tipo de aplicação. Após esta limpeza, estrutura-se os dados de forma que os algoritmos possam entendê-los.

Limpeza dos dados

Existem várias técnicas de pré-processamento para limpeza dos dados [12]. Para dados textuais coletados na rede social Twitter, por exemplo, é comum a remoção de citações

com o carácter @ ou com a cerquilha, remoção de acentos, remoção de múltiplos espaços em branco, etc.

Além disso, durante o pré-processamento são realizadas tarefas como *Toquenização, Remoção de Stop Words e Stemming*. Cada tarefa é explicada a seguir:

- *Toquenização*: nada mais é do que decompor o documento em termos de interesse. No Twitter, é feita a decomposição de cada *Tweet* em palavras, que normalmente são separadas por um espaço em branco, além disso, pode-se também traduzir cada palavra para um número único, a fim de ficar mais fácil para cada algoritmo processar um conjunto de números ao invés de *strings*;
- Remoção de *StopWords*: StopWords são palavras que não acrescentam tanto valor a sentença e por isso são removidas. Normalmente são representadas por preposições, artigos, números, etc.
- *Stemming*: Aqui basicamente reduz-se as palavras ao seu radical de origem, a fim de diminuir a dimensionalidade e simplificar o conjunto de palavras. Após o *stemming*, palavras com um mesmo radical, como por exemplo trabalho e trabalhador, são agregadas em um único termo: *trabalh*;

Representação dos dados

Para que a análise textual seja feita, deve-se antes converter os dados textuais em algo que os algoritmos entendam, que normalmente são vetores numéricos. Para isso, além da *tokenização*, também é feito um processo de *contagem* da ocorrência dos *tokens* em cada documento e uma *normalização*, onde *tokens* que aparecem em muitos documentos perdem a importância. Esse processo de transformar o texto em vetores é chamado de *Vetorização*. Existem diversas implementações desta vetorização, as mais conhecidas são:

- *Count Vectorizer*: é a técnica mais simples, onde é feita a *tokenização* e contagem;
- *Term Frequency-Inverse Document Frequency Vectorizer (TF-IDF Vectorizer)*: Nesta técnica, leva-se em consideração a importância de uma palavra em um documento em relação a uma coleção de documentos. Para isso, palavras que ocorrem com muita frequência em todos os documentos recebem um peso baixo.

2.1.2 Técnicas de mineração de textos

Existem diversas técnicas para se aplicar em mineração de textos, as quais são definidas de acordo com o objetivo que se deseja alcançar. Alguns exemplos dessas técnicas são: extração de informação, recuperação de informação, sumarização e clusterização [13].

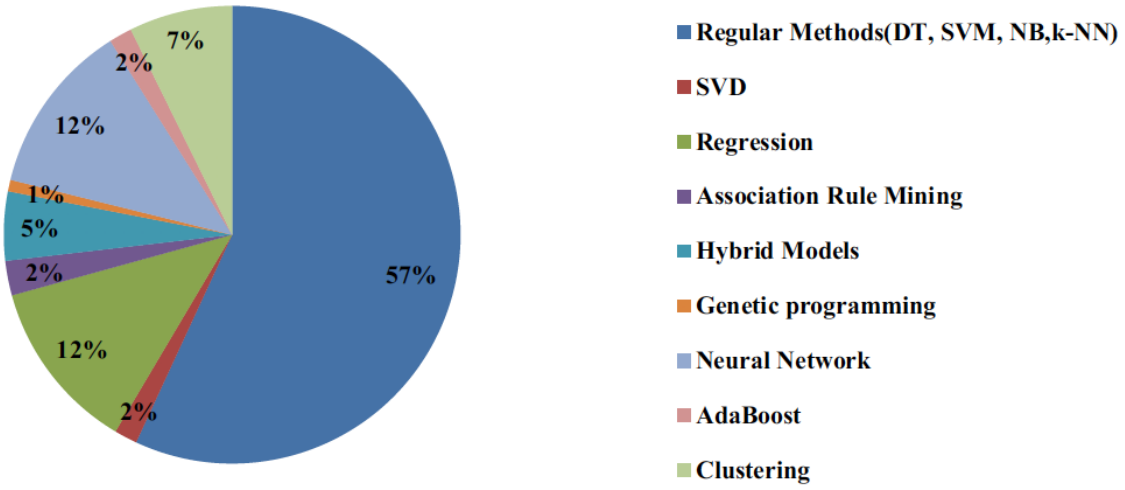

Figura 2.1: Lista de métodos aplicados a mineração de textos em diversos artigos de 2000 a 2016. Os métodos mais utilizados foram os métodos clássicos (*SVM, DT, NB e KNN*) com 57%. Fonte: [1]

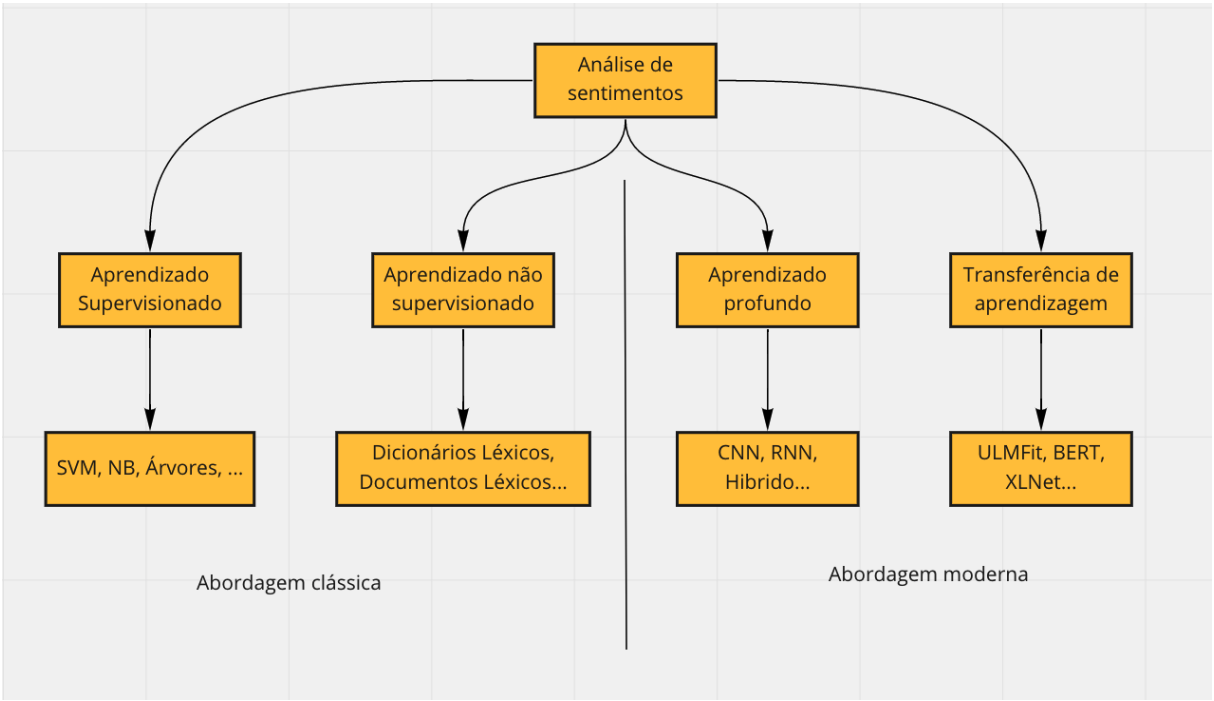

Figura 2.2: Principais abordagens mostradas em [2] para realizar análise de sentimentos em textos. Fonte: Própria

Um gráfico mais abrangente dos métodos utilizados é mostrado na Figura 2.1. Para a aplicação em análise de sentimentos, o esquemático na Figura 2.2 mostra quais as abordagens mais utilizadas na literatura. Este trabalho irá focar nas tarefas de aprendizado supervisionado e aprendizado por transferência. As comparações com aprendizado não-supervisionado e aprendizado profundo podem ser utilizados como trabalhos futuros.

2.2 Aprendizado de Máquina

O termo aprendizado de máquina, utilizado pela primeira vez em 1959 por Arthur L. Samuel em [14], nada mais é do que ensinar padrões ao computador por meio de uma coleção de dados e deixar que a máquina tome decisões por você. Este conceito foi melhor explicado em 1997 por Tom M. Mitchell em [15], onde é explicado como o aprendizado é feito de fato, os tipos de algoritmos de aprendizado de máquina e como avaliar sua eficácia. O aprendizado de máquina é classificado na literatura em 2 principais tipos: aprendizado supervisionado e não supervisionado. Um novo tipo de aprendizado também é utilizado neste trabalho e será explicado em breve, o chamado aprendizado por transferência.

2.2.1 Aprendizado supervisionado

Neste tipo de aprendizado, tem-se as possíveis categorias (ou rótulos) que os dados devem estar relacionados. Portanto, a tarefa que o computador deve fazer é aprender o padrão associativo entre o dado e a categoria e, após isso, prever a qual categoria um novo dado pertence [16, 17]. Este tipo de aprendizado é separado em dois tipos de problemas:

- Classificação: Quando os dados pertencem a duas ou mais classes e o aprendizado é feito a partir de uma base rotulada para depois prever dados não classificados. Um exemplo deste tipo de aprendizado seria o reconhecimento de dígito escrito à mão.
- Regressão: Quando o resultado almejado consiste em uma ou mais variáveis contínuas. Um exemplo deste tipo de aprendizado seria prever o comprimento de um peixe em função de sua idade e peso.

2.2.2 Aprendizado não-supervisionado

Aprendizados não supervisionados possuem dados sem nenhuma categoria associada, portanto a tarefa aqui é encontrar elementos semelhantes e agrupá-los [18], conhecido na literatura como *clusterização*. Além disso, na área de análise de sentimentos, a chamada abordagem léxica (*lexicon-approach*) é uma abordagem não supervisionada bastante utilizada para classificar sentimentos positivos e negativos.

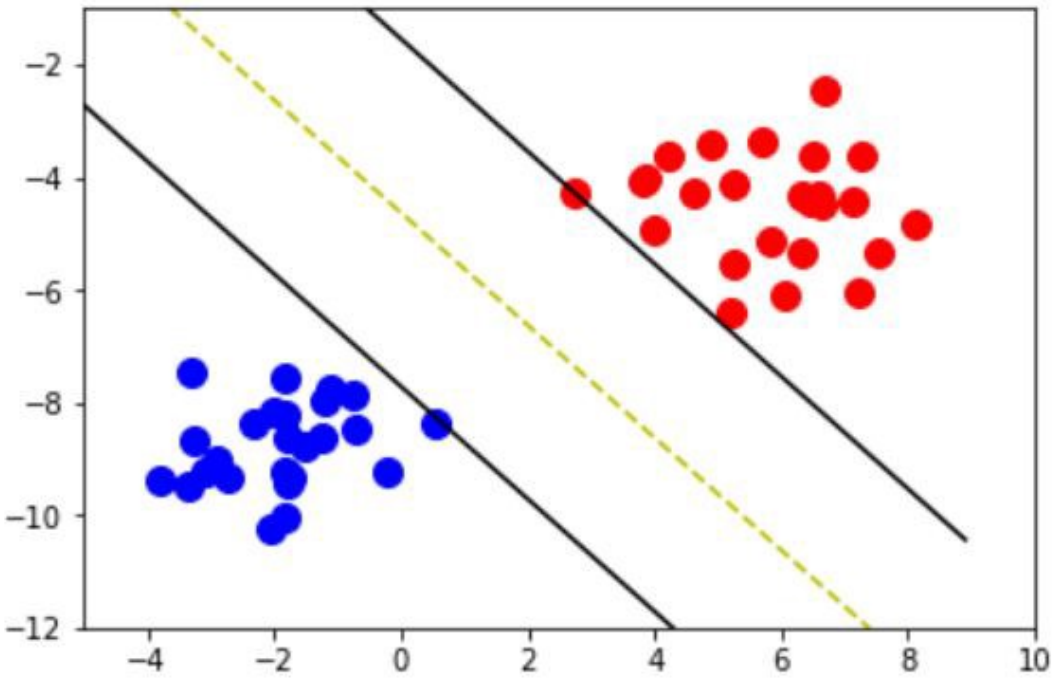

Figura 2.3: SVM Linear representado graficamente, onde existem duas classes, azul e vermelha, e o algoritmo tenta separá-las com um hiperplano. Fonte: Própria.

2.2.3 Aprendizado supervisionado para análise de sentimentos

A tarefa de classificar entidades é algo comum em muitas áreas. Em Biologia por exemplo, os ecologistas muitas vezes precisam classificar um pássaro selvagem da amazônia, ou então um paleontólogo deve saber classificar um tipo específico de fóssil.

Na ciência da computação, a classificação é um tipo de aprendizado de máquina supervisionado, em que ensinamos o computador a como classificar dados novos, dando a ele uma série de dados rotulados. Dentre as classes de algoritmos que performam a tarefa de classificar, os mais comuns para a tarefa de análise de sentimentos [17, 18, 19] são:

- *Support Vector Machines (SVM)*;
- *Naïve Bayes (NB)*;
- *Decision Tree (DT)*;

Além destes algoritmos, alguns outros algoritmos foram adicionados a este trabalho afim de acrescentar mais comparações, são eles:

- *Random Forest (RF)*;
- *K-nearest Neighbors (KNN)*;

- *SGD Classifier (SGDC)*;
- *Ridge Classifier*;
- *Transductive Propagation in Bipartite Graph (TPBG)*;

A seguir é explicado o funcionamento básico de cada um desses algoritmos, os quais podem ser encontrados com mais detalhes no livro [20].

SVM cite é um método de aprendizado de máquinas utilizado para resolver problemas de classificação e tem um desempenho muito bom como analisador de sentimentos [18]. A ideia do SVM é representar cada amostra no espaço e separá-las com o chamado *hiperplano* de modo a maximizar suas margens (Figura 2.3). Cada nova amostra é classificada em uma classe de acordo com sua posição no espaço. O SVM é considerado um dos melhores métodos supervisionados para classificar textos [17]. Existem diversos *kernels* possíveis para se utilizar com o SVM, que basicamente dita como esta separação é feita. Um *kernel* linear separa as classes com uma reta, enquanto um *kernel* polinomial utiliza uma função polinomial para separá-las. Existe ainda um parâmetro C referente ao grau de otimização, ou seja, o quanto se deseja evitar o erro. Quanto maior o C, menos tolerante a erros é o modelo.

Naïve Bayes é um método de classificação que utiliza da fórmula de Bayes e suposições de independência entre cada par de atributos [18]. Para entender, vamos ilustrar com um exemplo abaixo, inspirado em [20].

Imagine que gostaríamos de classificar uma fruta ω como sendo maçã ($\omega = \omega_1$) ou laranja ($\omega = \omega_2$). Para classificar uma fruta como sendo uma maçã, precisamos de uma cor vermelha, uma forma redonda e um diâmetro de 3 centímetros. Mesmo estas características sendo dependentes uma das outras, para o modelo *Naïve Bayes* cada característica contribui independentemente para que o classificador classifique a fruta como sendo uma maçã. Matematicamente, podemos formular esta ideia da seguinte maneira:

Inicialmente, a probabilidade da fruta ser uma maçã ou ser uma laranja é igual. Dizemos então que a probabilidade *priori* $P(\omega_1) = P(\omega_2)$. Se pudermos assumir que não existem outros tipos de frutas na região, então $P(\omega_1) + P(\omega_2) = 1$. Essas probabilidades são basicamente nosso conhecimento prévio da probabilidade de uma fruta ser maçã ou laranja.

Porém, somente com a probabilidade *priori*, nossa regra de decisão ficaria muito limitada: Escolha ω_1 se $P(\omega_1) > P(\omega_2)$. Além disso, podemos também utilizar a cor da fruta e sua textura. Podemos considerar então x sendo uma variável aleatória contínua, a qual depende do tipo de fruta e pode ser expressa como $p(x|\omega_1)$. Isto é a chamada *função de densidade de probabilidade condicional*. Assim, $p(x|\omega_1)$ e $p(x|\omega_2)$ denotam a diferença em cor entre maçã e laranja. Com isso, podemos escrever a probabilidade de

encontrar um padrão que está na categoria ω_j *e* tem uma característica x da seguinte forma: $p(\omega_j, x) = P(\omega_j|x)p(x) = p(x|\omega_j)P(\omega_j)$. Rearranjando os termos, encontramos a chamada fórmula de Bayes:

$$P(\omega_j|x) = \frac{p(x|\omega_j)P(\omega_j)}{p(x)} \tag{2.1}$$

Que informalmente pode ser traduzida para:

$$Posterior = \frac{Likelihood \times prior}{evidência} \tag{2.2}$$

A fórmula de Bayes mostra que observando um valor x podemos converter a probabilidade *prior* $P(\omega_j)$ na *posterior* $P(\omega_j|x)$, ou seja, na probabilidade de um elemento ser ω_j dado que a característica x foi observada. O fator *likelihood* é basicamente um termo para indicar que a categoria ω_j a qual produz um $p(x|\omega_j)$ grande é mais provável de ser a categoria correta. O termo do denominador chamado de evidência pode ser visto como um fator de escala.

A partir disso, temos um conjunto de características encontradas em um elemento representado por $p(\omega_j, x_1, \ldots, x_n)$, o qual pode ser aberto em um produtório e finalmente temos o modelo probabilístico:

$$p(\omega_j \mid x_1, \ldots, x_n) = \frac{1}{evidence} p(\omega_j) \prod_{i=1}^{n} p(x_i \mid \omega_j) \tag{2.3}$$

Com esse modelos, pode-se criar uma regra de classificação, como a máximo a *posteriori* (MAP), onde a hipótese mais provável é escolhida. O classificador *bayes* utiliza a seguinte regra:

$$\hat{y} = \underset{j \in \{1,\ldots,J\}}{\operatorname{argmax}} \; p(\omega_j) \prod_{i=1}^{n} p(x_i \mid \omega_j) \tag{2.4}$$

Existem alguns tipos de classificadores *Naïve Bayes*, mas eles diferem basicamente em como se assume a natureza do termo $p(x_i \mid \omega_j)$ (Gaussiano, Bernoulli ou Multinomial).

Árvores de Decisão são um tipo de fluxograma usado para tomar decisões. No aprendizado de máquina, cada nó desta árvore representa uma decisão de acordo com um atributo, como por exemplo na Figura 2.4 onde existem os atributos calor e vento e dependendo da existência delas ou não, o classificador tomará uma decisão final. São simples de entender e de utilizar, e funcionam bem para análise de sentimentos binária em textos onde cada palavra conta para montar a árvore de decisão.

RF ou floresta aleatória é classificado como um método *ensemble*, que nada mais é que uma combinação de métodos estimadores para melhorar a generalidade e a robustez de um estimador. Neste algoritmo, cria-se várias árvores de decisão de forma aleatória

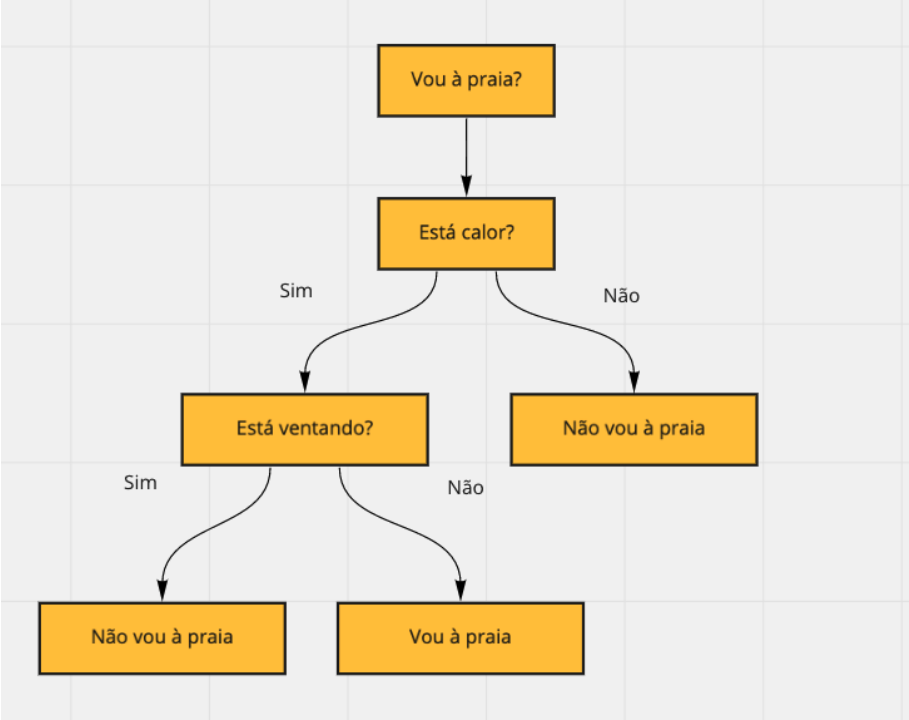

Figura 2.4: Árvore de decisão para ir à praia. Fonte: Própria

com os dados de treino. Em seguida, no processo de construção da árvore, encontra-se a melhor separação de nós dos dados de entrada ou de dados aleatórios.

O objetivo desta aleatoriedade inserida é diminuir a variância dos estimadores, visto que árvores de decisão individuais possuem uma alta variância. Com a média das predições de cada árvore, alguns erros podem se cancelar.

KNN é uma técnica clássica de classificação, que tem como princípio encontrar os k vizinhos mais próximos de um dado já conhecido. O número k pode ser definido pelo usuário, e a distância pode ser calculada de vários modos, como por exemplo a distância Euclidiana.

SGDC é um classificador linear, ou seja, assume que os dados se comportam linearmente. Este classificador utiliza da abordagem *Stochastic Gradient Descent* (SGC), que é uma técnica de otimização para treinar classificadores lineares e regressores. Quando utilizado com a função de perda *hinge* ele basicamente implementa o próprio *SVM*.

Ridge Classifier, ou também *Least-squares support-vector machine* é também utilizado para modelos lineares, porém tenta diminuir a soma dos quadrados entre os pontos observados na base de dados e os elementos já classificados pela aproximação linear. Para isso, acrescenta-se um fator de penalidade ao modelo. O *RidgeClassifier* tenta minimizar essa penalidade da soma de quadrados.

TPBG [21] é um algoritmo que estrutura os dados textuais na forma de grafos heterogêneos bipartidos, onde o grafo representa o documento, as palavras são os vértices

e as coocorrências de palavras são as arestas. A ideia principal deste algoritmo é a propagação das classes entre os vértices vizinhos. Esta propagação pode ocorrer localmente, ou seja, pela vizinhança de um grafo, ou globalmente, onde as informações se propagam pela estrutura completa do grafo bipartido. Essas propagações são realizadas até que um número máximo de iterações seja alcançado ou que as informações de cada classe não se alterem em sucessivas iterações.

2.2.4 Aprendizado por transferência

Aprendizado por transferência é um método de aprendizado de máquina que tem sido muito utilizado atualmente, visto que apresenta um resultado estado-da-arte em processamento de linguagem natural, com modelos modernos como *ULMFit* [4], *BERT* [10] e *XLNet* [11].

O aprendizado por transferência se baseia na reutilização de um modelo de linguagem[1] pré-treinado em um novo problema. Isto é, reutilizar uma rede neural treinada em outro conjunto de dados geralmente maior, para resolver um novo problema (Figura 2.5).

Este método traz diversas vantagens, como evitar treinamento de modelos de redes neurais profundas do zero o que consequentemente evita gastos com GPU's.

Para que o leitor entenda melhor este conceito, será utilizada a definição e a explicação feita em [3]:

> **Definição 1** [Aprendizado por transferência] Dado um domínio origem $\mathcal{D}_\mathcal{S}$ e uma tarefa de aprendizado $\mathcal{T}_\mathcal{S}$, um domínio alvo $\mathcal{D}_\mathcal{T}$ e uma tarefa alvo $\mathcal{T}_\mathcal{T}$, *aprendizado por transferência* tem como objetivo melhorar o aprendizado da função de predição alvo $f_T(\cdot)$ em $\mathcal{D}_\mathcal{T}$ usando o conhecimento em $\mathcal{D}_\mathcal{S}$ e $\mathcal{T}_\mathcal{S}$, onde $\mathcal{D}_\mathcal{S} \neq \mathcal{D}_\mathcal{T}$, ou $\mathcal{T}_\mathcal{S} \neq \mathcal{T}_\mathcal{T}$.

Na Definição 1, um domínio $\mathcal{D}$ consiste de um espaço de atributos $\mathcal{X}$ e uma distribuição de probabilidade $P(X)$, onde $X = \{x_1, ..., x_n\} \in \mathcal{X}$. Dado um domínio específico, $\mathcal{D} = \{\mathcal{X}, P(X)\}$, uma tarefa consiste de dois componentes: um espaço de rótulos $\mathcal{Y}$ e uma função de predição $f : \mathcal{X} \rightarrow \mathcal{Y}$. A função f é usada para prever o rótulo y de uma nova instância x. Sob um ponto de vista probabilístico, $f(x)$ pode ser escrita como $P(Y|X)$. Assim, essa tarefa denotada por $\mathcal{T} = \{\mathcal{Y}, P(y|x)\}$, aprende a partir dos dados de treinamento, que consistem de pares $\{x_i, y_i\}$, onde $x_i \in X$ e $y_i \in \mathcal{Y}$.

Assim, a partir da Definição 1, como $\mathcal{D}_\mathcal{S} \neq \mathcal{D}_\mathcal{T}$, então ou $\mathcal{X}_\mathcal{S} \neq \mathcal{X}_\mathcal{T}$ ou $P(X_S) \neq P(X_T)$. Isto é, entre a base de dados que foi utilizada para pré-treinar o modelo e a base de dados que temos para treinar o modelo, ou os atributos são diferentes (e.g., línguas diferentes), ou a distribuição delas são diferentes.

[1] Uma rede neural que tem como objetivo prever a próxima palavra em uma sequência de palavras.

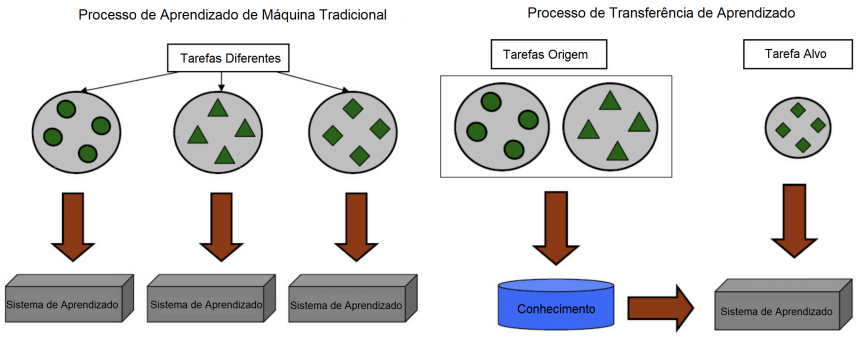

Figura 2.5: Diferenças entre o processo de aprendizagem tradicional e a transferência de aprendizado. Fonte: [3]

Similarmente, a condição $\mathcal{T}_\mathcal{S} \neq \mathcal{T}_\mathcal{T}$ implica que ou $\mathcal{Y}_\mathcal{S} \neq \mathcal{Y}_\mathcal{T}$ ou $P(Y_S|X_S) \neq P(Y_T|X_T)$. Quando os domínios origem e alvo são iguais, i.e. $\mathcal{D}_\mathcal{S} = \mathcal{D}_\mathcal{T}$ e suas tarefas de aprendizagem também são as mesmas, i.e., $\mathcal{T}_\mathcal{S} = \mathcal{T}_\mathcal{T}$, este problema se resume a um problema tradicional de aprendizado de máquinas. Quando os domínios são diferentes, então ou (1) o espaço de atributos entre os domínios são diferentes, isto é, $\mathcal{X}_\mathcal{S} \neq \mathcal{X}_\mathcal{T}$, ou (2) o espaço de atributos entre os domínios são iguais porém a distribuição de probabilidades entre os dados dos domínios são diferentes, isto é, $P(X_S) \neq P(X_T)$, onde $X_{S_i} \in \mathcal{X}_\mathcal{S}$ e $X_{T_i} \in \mathcal{X}_\mathcal{T}$.

Para exemplificar, o caso (1) acontece quando as duas bases de dados estão descritas em linguagens diferentes, e o caso (2) pode acontecer quando a base de dados de origem e a base de dados alvo focam em tópicos diferentes.

Dados os domínios específicos $\mathcal{D}_\mathcal{S}$ e $\mathcal{D}_\mathcal{T}$, quando as tarefas de aprendizado $\mathcal{T}_\mathcal{S}$ e $\mathcal{T}_\mathcal{T}$ são diferentes, então ou (1) o espaço de rótulos entre os domínios são diferentes, isto é, $\mathcal{Y}_\mathcal{S} \neq \mathcal{Y}_\mathcal{T}$, ou (2) as distribuições de probabilidade entre os domínios são diferentes, isto é, $P(Y_S|X_S) \neq P(Y_T|X_T)$, onde $Y_{S_i} \in \mathcal{Y}_\mathcal{S}$ e $Y_{T_i} \in \mathcal{Y}_\mathcal{T}$.

Para exemplificar, o caso (1) ocorre quando por exemplo a base de dados origem tem classes binárias enquanto o domínio alvo tem dez classes. O caso (2) ocorre quando os dados origem e alvo são desbalanceados em termos de classes.

Pode-se separar o aprendizado por transferência em 3 cenários:

- Aprendizado por transferência Indutivo;
- Aprendizado por transferência Transdutivo;
- Aprendizado por transferência Não-supervisionado;

Aprendizado por transferência Indutivo é o cenário em que a tarefa alvo é diferente da tarefa origem, independente dos domínios serem os mesmos ou não. Neste caso,

é necessária uma base rotulada para induzir um modelo preditivo. Neste cenário pode existir dois sub-cenários: (1.1) Existem muitos dados rotulados no domínio de origem, e (1.2) onde não existem dados rotulados no domínio de origem.

Aprendizado por transferência Transdutivo é o cenário em que as tarefas alvo e origem são as mesmas, enquanto os domínios são diferentes. Neste caso, a base não está rotulada no domínio alvo enquanto no domínio de origem possui base rotulada. Dois sub-cenários podem ocorrer neste cenário: (2.1) o espaço de atributos entre os domínios são diferentes (e.g. linguagens diferentes) ou (2.2) os espaços de atributos são iguais, porém a distribuição de probabilidade dos dados não (e.g. documentos com tópicos diferentes).

Aprendizado por transferência não supervisionado é o similar ao cenário indutivo onde a tarefa origem é igual a tarefa alvo, porém neste cenário o foco é resolver tarefas não supervisionadas no domínio alvo, como agrupamentos, redução de dimensionalidade, etc. Neste cenário, as bases não estão rotuladas em ambos domínios.

Este trabalho irá focar somente no cenário indutivo, deixando como trabalhos futuros os demais cenários.

O aprendizado por transferência como já foi dito, *transfere* conhecimento de um domínio para outro, porém, segundo [3], existem quatro tipos possíveis de conhecimento que podem ser transferidos:

- *Aprendizado por transferência de instâncias*, o qual assume que certas partes dos dados de origem podem ser reutilizados para o aprendizado no domínio alvo, fazendo um rebalanceamento das instâncias;
- *Aprendizado por transferência de atributos*, o qual aprende uma representação de atributos. Neste caso, o conhecimento a ser transferido é codificado dentro da representação do atributo;
- *Aprendizado por transferência de parâmetros*, o qual assume que as tarefas de origem e alvo compartilham alguns parâmetros ou algumas distribuições *priori*. O conhecimento a ser transferido é codificado dentro dos parâmetros compartilhados ou *priors*;
- *Aprendizado por transferência de conhecimento relacional*, que lida com domínios relacionados, ou seja, assume que existe uma relação entre os dados dos domínios origem e alvo. Portando, o conhecimento a ser transferido é a própria relação entre os dados;

A seguir são citados e explicados cada um dos métodos/algoritmos utilizados neste trabalho.

ULMFit

Universal Language Model Fine-tuning for Text Classification ou *ULMFit* [4] é um método criado para universalizar o processo de transferência do aprendizado indutivo para qualquer domínio alvo. Além disso, foram criadas novas técnicas para aperfeiçoar a transferência de aprendizado, chamadas de *discriminative fine-tuning*, *slanted triangular learning rates* e *gradual unfreezing*.

O método possui três fases:

1. Pré-treinamento de um modelo de linguagem em uma grande base de dados de domínio geral;
2. Aperfeiçoamento do modelo para uma tarefa alvo utilizando as técnicas novas;
3. Aperfeiçoamento do classificador para a tarefa alvo;

Os conceitos por trás das redes neurais não serão explicadas a fundo, visto que o foco principal deste trabalho é na metodologia do aprendizado por transferência. Entretanto, é sugerido como trabalhos futuros a implementação de outros modelos de redes neurais na fase de pré-treinamento para acrescentar aos experimentos.

O modelo de linguagem utilizado pelos autores para fazer o **pré-treinamento** foi o chamado *AWD-LSTM*, que é um tipo de rede neural recorrente (RNN), mais informações sobre este modelo podem ser encontradas em [22]. Este modelo é treinado com a base de dados chamada de *Wikitext-103* [23], que consiste de 28.6 mil artigos e 103 milhões de palavras da Wikipédia já pré-processadas.

Para o **aperfeiçoamento do modelo de linguagem**, algumas novas técnicas foram criadas pelos autores para melhorar a transferência do aprendizado:

- *Discriminative fine-tuning*: Como cada camada do modelo captura diferentes tipos de informação, elas devem ser aperfeiçoadas de modo diferente entre si. Para isso, os autores utilizam taxas de aprendizado diferentes para cada camada;
- *Slanted triangular learning rates*: Para que o modelo adapte seus parâmetros a diferentes tipos de tarefas, os autores resolvem utilizar uma nova taxa de aprendizado diferente da linear. Este novo tipo primeiramente cresce rapidamente de forma linear e depois decai lentamente também de forma linear (Figura 2.6), trazendo assim uma melhor performance.

Para o *aperfeiçoamento do classificador da tarefa alvo* os autores acrescentam dois blocos lineares ao modelo de linguagem pré-treinado, cada um com normalização *batch* e *dropout*, com a função de ativação $ReLU$ para o intermediário e *softmax* para a saída. Assim, somente essas camadas irão aprender do zero. Para que informações não sejam perdidas, os autores utilizam a técnica chamada de *Concat pooling*.

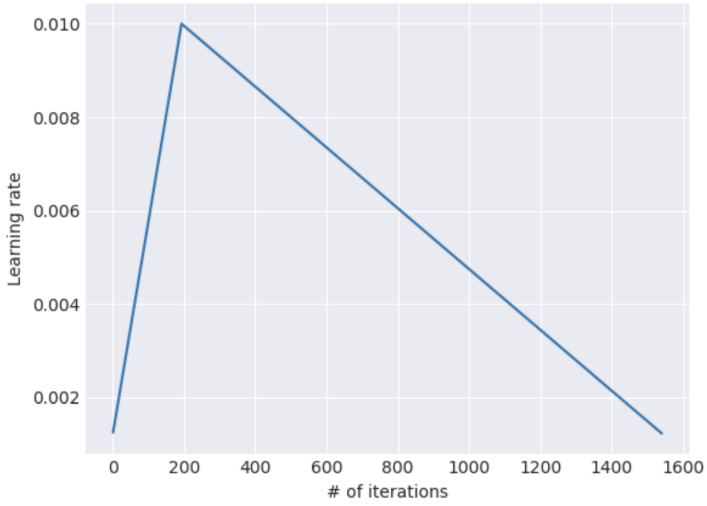

Figura 2.6: *Slanted triagular learning rate.* Fonte: [4]

Além disso, os autores propõem uma nova técnica para aperfeiçoar o classificador, chamada de *Gradual unfrezing*. Ao invés de aperfeiçoar todas as camadas de uma vez só, o que pode gerar o chamado *esquecimento catastrófico*, fazendo com que o modelo perca abruptamente informações aprendidas nas etapas anteriores, os autores propõem um descongelamento gradual começando da última camada. Ou seja, todas as camadas começam congeladas e a última camada é descongelada e aperfeiçoada. Em seguida, a próxima camada é descongelada e todas as camadas descongeladas são aperfeiçoadas. Isso se repete até que todas as camadas fiquem descongeladas.

Os autores propõem também um método de *backpropagation* para classificação de texto, chamado por eles de *BPT3C*, o qual é similar à técnica de *variable length backpropagation sequences* [22].

Além disso, o modelo de linguagem não é limitado a unidirecionalidade, ou seja, o modelo pode aprender em ambas as direções do texto, porém separadamente, ou seja, deve ser criado um modelo para cada direção e em seguida incorporados de alguma forma.

Foram feitos experimentos em bases de dados conhecidas, como *IMDb* para análise de sentimentos, *TREC* para perguntas e respostas, *AG News* e *DBpedia* para classificação de tópicos e conseguiram superar o estado-da-arte em 6 tarefas de classificação.

BERT

Bidirectional Encoder Representations from Transformer ou BERT [10] é um modelo de representação de linguagem desenhado para pré-treinar textos sem rótulos de maneira bidirecional graças a técnica chamada de *modelo de linguagem mascarado*. Para entendermos o BERT, devemos primeiramente entender dois novos conceitos que são utilizados por ele: *Attention* e *Transformers*.

Attention definido em [24] e [25], é um mecanismo que faz com que o modelo de linguagem dê maior peso a outros elementos da sequência de entrada a fim de ter alguma pista que ajude a entender o contexto.

Por exemplo, na sentença "A pessoa não atravessou a rua pois ela ...", a palavra "ela" poderia se referir tanto a "pessoa" quando a "rua", porém por estar próxima de "rua", modelos tradicionais não teriam a associação de "ela" com "pessoa". A técnica *Attention* permite que cada palavra possua um vetor de chave-valores que possam ser traduzidos a porcentagens de associação a cada uma das outras palavras na sentença (ver Figura 2.7).

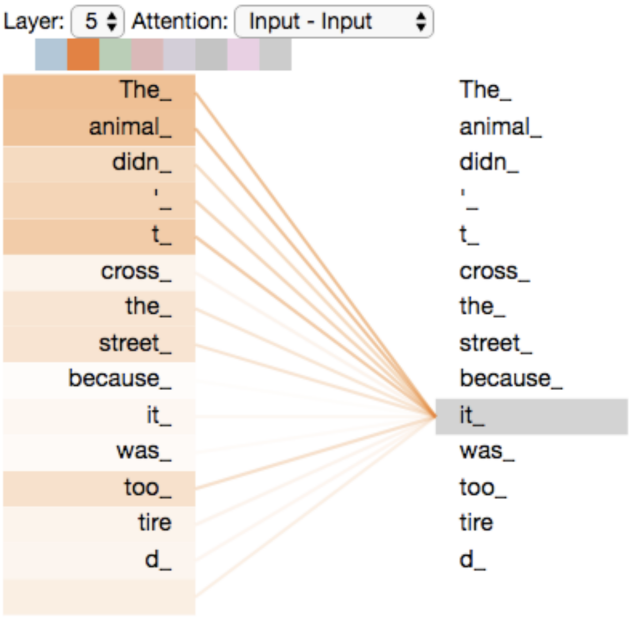

Figura 2.7: Enquanto estamos codificando a palavra "*it*", parte do mecanismo de *attention* estava prestando atenção no trecho "*The animal*". Fonte: Tensor2Tensor notebook

Transformer [26] é uma nova arquitetura de redes baseada unicamente em mecanismos de *attention*, dispensando recorrências e convoluções. Sua arquitetura se baseia em uma pilha de 6 codificadores e 6 decodificadores, onde cada codificador possui uma camada de *attention* e uma camada de *Rede Neural Feed Forward* e cada decodificador possui as mesmas camadas, porém no meio delas possui uma outra camada de *attention* que ajuda o decodificador a focar em partes relevantes dos dados de entrada.

Finalmente, o modelo de linguagem *BERT* utiliza da arquitetura de *Transformers*. Mais especificamente, sua arquitetura consiste de 12 camadas (modelo base) ou 24 camadas (modelo grande).

Utiliza-se do algoritmo *WordPiece* [27] com um vocabulário de 30 mil *tokens* para representar as palavras.

Existem dois passos para a criação do modelo: um **pré-treinamento** e um **aperfeiçoamento**.

Para o **pré-treinamento**, o modelo não utiliza do método tradicional da esquerda para a direita ou direita para esquerda, chamado de *auto-regressive*. Ao invés disso, BERT é pré-treinado bidirecionalmente usando duas tarefas não supervisionadas:

Tarefa #1 - Modelo de linguagem mascarado: Para que o modelo seja bidirecional, os autores utilizam máscaras em uma porcentagem de *tokens* aleatoriamente e então tentam prever estes *tokens* mascarados.

Por exemplo, na frase `Meu cachorro é [MASK] legal`, o objetivo desta tarefa é descobrir qual a melhor palavra para esse cenário, e.g. `Meu cachorro é muito legal`

Tarefa #2 - Predição da próxima sentença: O algoritmo escolhe 2 sentenças A e B durante o pré-treinamento, onde 50% das vezes B é a sentença que procede A, e os outros 50% das vezes é uma sentença aleatória do texto. Exemplo:

Entrada = `O homem foi para [MASK] loja`
`ele comprou um galão [MASK] leite`
Saída = `IsNext`
Entrada = `O homem foi para [MASK] loja`
`pinguim são aves [MASK] neve`
Saída = `NotNext`

As bases de dados utilizadas para pré-treino foram o *BookCorpus* [28] (800 milhões de palavras) e Wikipédia (2,5 milhões de palavras).

Para o **aperfeiçoamento**, simplesmente são inseridas as entradas específicas da tarefa alvo e a saída é enviada para o BERT, que aperfeiçoa todos os parâmetros.

São feitos experimentos em 11 tarefas de processamento de linguagem natural e os resultados são comparados com modelos similares.

XLNet

O *XLNet* [11] é uma nova técnica de pré-treinamento que tem como principal objetivo resolver os problemas encontrados no *BERT*, os quais serão explicados mais a frente. Para entendermos como ele faz isso, primeiramente vamos conceituar os dois principais tipos de modelos de linguagens abaixo.

- Modelo de linguagem *auto-regressive*, é um modelo de linguagem que utiliza somente dos *tokens* anteriores para prever o próximo *token* da sentença. Podem percorrer a sentença da esquerda para a direita ou da direita para a esquerda. A grande limitação deste modelo é ter acesso somente aos *tokens* anteriores, perdendo assim informações de contexto que poderiam ser utilizadas. Chamamos este problema de limitação da ordem de fatorização da sentença.

- Modelo de linguagem *auto-encoding*, é um modelo de linguagem que utiliza todos os *tokens* da sentença menos os mascarados, portanto, tem acesso aos dois lados da sentença. Um exemplo de uso deste modelo é o próprio *BERT* [10]. A principal limitação deste modelo é que ele negligencia a dependência entre *tokens* mascarados. Isto é, suponha que o algoritmo *BERT* está sendo pré-treinado na seguinte sequência: `New York is a city.` Para isso, o algoritmo mascara a sequência da seguinte maneira: `[MASK] [MASK] is a city.` Como *BERT* prevê todas as palavras de uma só vez, o fato da primeira palavra ser *New* não influencia a segunda palavra, fazendo com que o resultado *New Franscisco* e *San York* sejam dois resultados perfeitamente válidos para o algoritmo, o que na prática não deveria ser (ver Equações 2.5 e 2.6).

$$P_{BERT} = log\ p(New|is\ a\ city) + log\ p(York|is\ a\ city), \tag{2.5}$$

$$P_{XLNet} = log\ p(New|is\ a\ city) + log\ p(York|New, is\ a\ city). \tag{2.6}$$

O *XLNet* resolve estes problemas combinando ambos os conceitos de *autoregressive* e *autoenconding*. Para isso, eles propõem a utilização do método *autoregressive* utilizando todas as possíveis permutações dos termos da sentença, fazendo com que o modelo não seja limitado a permutação original. Com todas as permutações possíveis, o modelo consegue aprender o contexto bidirecionalmente, visto que cada *token* terá passado por todas as posições possíveis.

Além disso, o *XLNet* não utiliza do método de máscaras, o que atrapalhava a fase de aperfeiçoamento dos modelos, uma vez que o modelo foi pré-treinado em dados com máscaras enquanto os dados alvos não as possuem.

Além disso, os autores propõem a utilização de uma outra arquitetura de *transformers*, a chamada *Transformer-XL* [29] que empiricamente melhora a performance das tarefas envolvendo sequências longas de texto, porém para utilizar esta arquitetura, os autores propõem uma nova parametrização para ela, para que a arquitetura se adapte à técnica de permutação.

O modelo empiricamente supera o *BERT* em 20 tarefas, como por exemplo na análise de sentimentos, pergunta e resposta e inferência de linguagem natural [11].

Capítulo 3

Revisão Bibliográfica

O processamento de linguagem natural teve sua primeira aparição em 1950, quando Alan Turing publicou o artigo chamado *Computing Machinery and Intelligence* [30] e vem crescendo rapidamente, com técnicas melhores e que resolvem a "simples" tarefa de prever a próxima palavra em uma frase. Técnicas de mineração de texto são continuamente aplicadas a indústrias, academia, aplicações web, internet e outros campos [31].

Para que seja possível processar dados, precisa-se primeiramente de alguma base de dados com vários textos. Para isso, podemos construir nosso próprio conjunto de dados, porém, deveríamos também rotular os dados de acordo com seu sentimento, que é um processo demorado e normalmente requer mais de uma pessoa para decidir o rótulo certo (principalmente para julgar se uma frase tem sentimento positivo ou negativo, que é algo muitas vezes relativo). Para evitar este trabalho, artigos como [9], [32] e [33] foram fundamentais, pois possibilitaram a utilização de bases de dados como *Internet Movie Database (IMDb)*, *Yelp Reviews (Yelp R.)*, *Amazon Reviews (Amazon R.)* e *Sentiment 140*, que são bases de dados já rotuladas.

Além disso, muitas vezes os textos possuem ruídos, como caracteres especiais, muitos espaços, acentos, etc. Para isso, técnicas de pré-processamento de dados como *stop words, stemming e tokenização* devem ser aplicadas, e o trabalho de [34] foi muito importante para entender quais técnicas tem melhores resultados em textos e em algoritmos específicos, onde foi concluído que o algoritmo *SVM* pode ter uma grande melhora utilizando *stemming e stop words*.

Após o pré-processamento, os dados devem ser estruturados de alguma maneira, normalmente utiliza-se a forma de vetores. O trabalho de [35] foi utilizado como base para a escolha do *Term Frequency–Inverse Document Frequency (TF-IDF)*, onde o autor cita sua simplicidade e eficácia, apesar de possuir limitações para cenários de *big data*.

O problema tratado neste trabalho, onde é argumentado que os algoritmos estado-da-arte para modelagem de linguagem são complexos e custosos e que muitas vezes trazem

pouca melhoria na acurácia do modelo, foi baseada no trabalho de [36], onde é feita uma comparação entre os algoritmos *SVM* e *ULMFit* [4]. A fim de acrescentar mais comparações, principalmente com novos algoritmos modernos, realizou-se uma pesquisa com o objetivo de encontrar quais são os atuais modelos de linguagem estado-da-arte para a tarefa de análise de sentimentos.

Foi utilizado o repositório *NLP-Progress* [1], o qual possui bases de dados e os trabalhos mais atuais feitos na área de processamento de linguagem natural.

Assim, os algoritmos estado-da-arte, ou chamados aqui de modernos, que foram selecionados para comparação foram o próprio *ULMFit* [4], o modelo de linguagem da Google, *BERT* [10], e o atual estado-da-arte também da Google juntamente com a Carnegie Mellon University, *XLNet* [11]. Foi decidido focar no método de aprendizado por transferência uma vez que todos os trabalhos estado-da-arte citados anteriormente o utilizam.

Os algoritmos *BERT* e *XLNet* chamaram bastante atenção devido a sua arquitetura, que é composta por uma estrutura relativamente nova chamada de *Transformers* [26], a qual utiliza de uma técnica muito interessante chamada de *Attention* introduzida em [24] e [25].

Para escolher quais algoritmos clássicos seriam utilizados na comparação com os modernos, foi aproveitado o *benchmark* de [37], o qual utiliza vários algoritmos em diversas bases de dados voltadas a classificação e clusterização. Algoritmos como *SVM*, *NB* e *KNN* entre outros, são executados exaustivamente com diversos parâmetros, os quais também foram utilizados como base neste trabalho.

Além desses algoritmos clássicos, alguns outros foram adicionados por motivos empíricos, como o *SGDClassifier* e o *RidgeClassifier*. Finalmente, o algoritmo *Transductive Propagation in Bipartite Graph (TPBG)* [21], o qual foi desenvolvido na tese de doutorado do orientador deste trabalho, o Prof. Dr. Thiago de Paulo Faleiros, para aprendizado semi-supervisionado, foi adaptado para o aprendizado supervisionado e, assim, inserido neste trabalho para acrescentar em comparações. O TPBG utiliza uma técnica diferente dos outros, onde armazena os documentos textuais em grafos bipartidos.

[1] http://nlpprogress.com/english/sentiment_analysis.html

Capítulo 4

Metodologia

Este capítulo apresenta ao leitor a metodologia utilizada para a implementação deste trabalho. A Seção 4.1 descreve as bases de dados utilizadas, a Seção 4.2 descreve os pré-processamentos aplicados, a Seção 4.3 trata dos algoritmos de aprendizado utilizados e, por fim, a Seção 4.4 se refere às métricas de avaliação utilizadas.

4.1 Conjunto de dados

Foram utilizadas 4 bases de dados neste trabalho, sendo todas amplamente conhecidas e bastante utilizadas na tarefa de análise de sentimentos. A tabela 4.1 descreve cada base. A seguir é detalhada cada base de dados.

4.1.1 IMDb

Esta base de dados [32] amplamente utilizada contém 50 mil revisões sobre títulos de filmes em inglês, sendo separado em 25 mil para treino e 25 mil para teste. Esta base de dados separa os sentimentos em positivo e negativo, portando a análise de sentimentos é binária. Para este trabalho, foi utilizado todo a base de dados.

4.1.2 Yelp Reviews

Esta base de dados [38] é um sub-grupo de uma base maior criada na *Yelp Dataset Challenge*, que foi uma oportunidade para estudantes fazerem pesquisas e análises em cima dos dados da empresa e compartilharem os resultados. Ela possui 700 mil revisões, porém, para este trabalho, foi utilizado um sub-grupo com 50 mil revisões. A base possui 5 classificações (1-5), para *binarizá-la* foi feito o seguinte mapeamento: classificações 1 e 2 ficaram como negativas (0) e classificações 4 ou 5 ficaram positivas (1). A classificação 3 foi removida da base de dados, sendo julgada como um sentimento neutro.

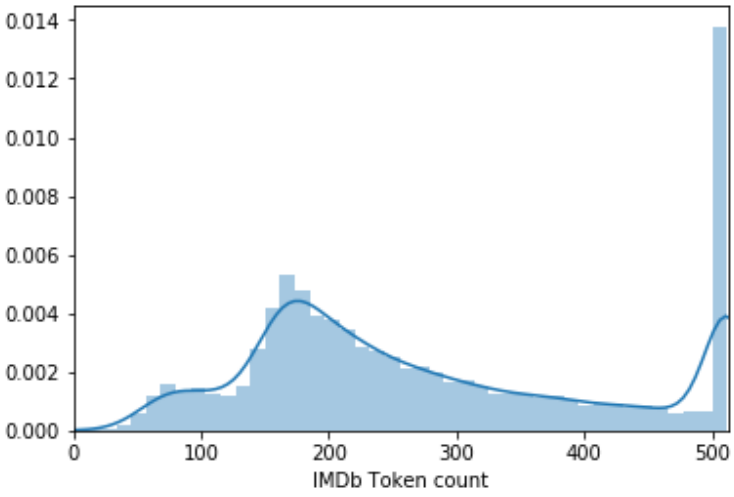

Figura 4.1: Quantidade de tokens por sentença na base de dados IMDb

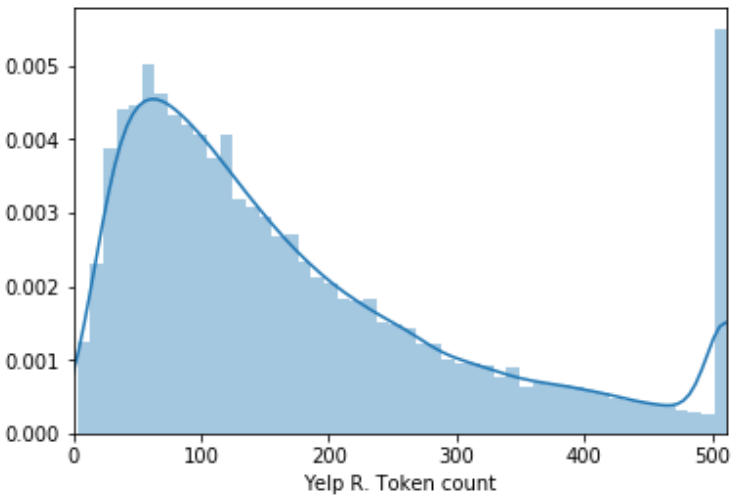

Figura 4.2: Quantidade de tokens por sentença na base de dados Yelp R.

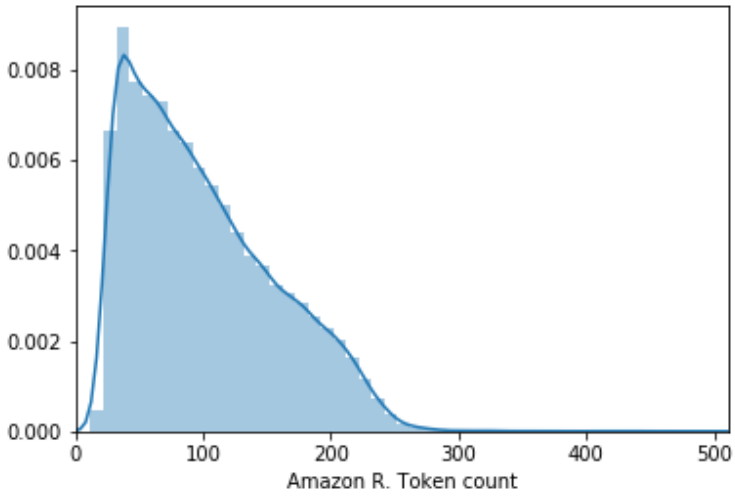

Figura 4.3: Quantidade de *tokens* por sentença na base de dados Amazon R.

	# Classes	Documentos	# médio de tokens
IMDb	2	50,000	279
Yelp R.	5	700,000	176
Amazon R.	5	3,600,000	101
Sentiment140	2	1,600,000	24

Tabela 4.1: Características das bases de dados utilizadas

4.1.3 Amazon Reviews

Esta base de dados também é um subgrupo de uma base maior [38] que consiste de milhões de revisões de produtos da *Amazon*. Para este trabalho foram utilizadas 50 mil revisões. O mesmo mapeamento feito na base da dados Yelp foi feita aqui, a fim de *binarizar* a base.

4.1.4 Sentiment 140

Esta base de dados possui 1,6 milhões de *tweets* extraídos do Twitter [33], cada um associado a sentimentos positivo, negativo ou neutro. Para este trabalho foram utilizados 50 mil *tweets*. Os dados com sentimento neutro foram removidos.

4.2 Pré-processamentos

O pré-processamento consiste em remover os ruídos julgados desnecessários para o domínio da aplicação. Neste trabalho, onde o sentimento é analisado a partir de textos, pouca informação é de fato desnecessária, abaixo são explicadas quais técnicas de pré-processamento foram adotadas.

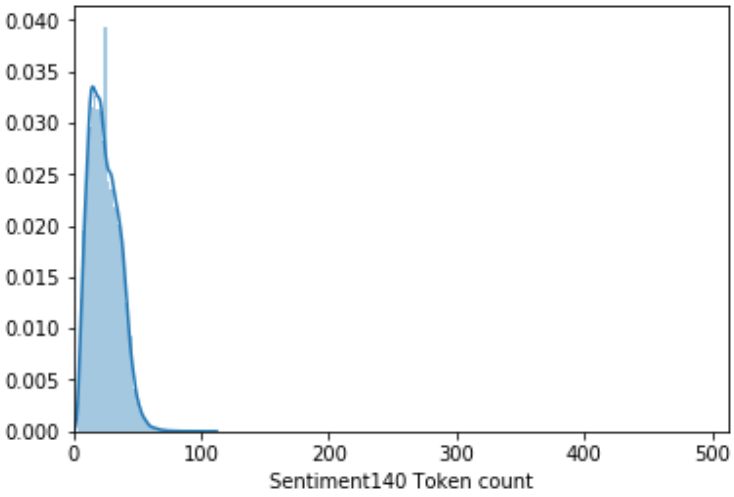

Figura 4.4: Quantidade de tokens por sentença na base de dados Sentiment140. Fonte: própria.

4.2.1 Algoritmos modernos

A seguir é explicado o pré-processamento utilizado para cada algoritmo moderno, isto é, os que utilizam a técnica de aprendizado por transferência.

ULMFit

Como foi utilizada a biblioteca da fastai[1] para implementar o *ULMFit*, foi então utilizado o método de *tokenização* implementado por eles, o qual faz os seguintes pré-processamentos:

- Troca caracteres HTML por algo mais representativo;
- Troca caracteres e palavras repetidas por um símbolo específico;
- Adiciona espaço entre / e #;
- Remove espaços duplicados;
- Faz *lower case* de palavras e letras e adiciona um símbolo específico para identificar;

BERT

Para o algoritmo BERT, foi utilizada a biblioteca da HuggingFace[2], a qual utiliza um método de tokenização baseado em *WordPiece* [39] e [27], além de fazer os seguintes pré-processamentos:

[1]https://www.fast.ai/
[2]https://huggingface.co/

- Transforma o texto para *lower case*;
- Adiciona *tokens* especiais: [UNK] para palavras desconhecidas, [SEP] para separar sentenças, [PAD] para deixar todas as sentenças do mesmo tamanho, [CLS] para o inicio da sentença;
- Transforma caracteres chineses em *tokens*;
- Remove acentos.

XLNet

Para o algoritmo do *XLNet*, também foi utilizada a biblioteca da *HuggingFace*, porém aqui é utilizado um método de *tokenização* baseado no *SentencePiece* [40]. Além disso, os passos de pré-processamento feitos para o *XLNet* são semelhantes aos feitos no *BERT*, com pequenas alterações.

4.2.2 Algoritmos clássicos

Para os algoritmos clássicos, também transformou-se as bases de dados em binários e aqui optou-se pela utilização da técnica de *Term frequency–inverse document frequency* para representação dos dados (Tf-idf). Técnicas como *Tokenização, StopWord e Stemming* também foram utilizadas.

4.3 Algoritmos de aprendizado utilizados

Este trabalho utiliza diversos algoritmos para comparações, onde alguns são considerados modernos e outros são chamados de algoritmos clássicos, pois são utilizados há bastante tempo e bem conhecidos. Esta seção cita quais foram esses algoritmos e como foram aplicados, isto é, os parâmetros escolhidos.

4.3.1 Algoritmos modernos

ULMFit

O ULMFit [4] se baseia no modelo de aprendizagem por transferência, que nada mais é que reutilizar um modelo já treinado em uma base de dados normalmente grande e sem um domínio específico (Wikipédia por exemplo), e aperfeiçoá-lo com bases específicas de nosso domínio (IMDb por exemplo). O ULMFit se baseia em 3 fases:

- Modelo de linguagem pré-treinado;

	Camada 1	Camada 2
IMDb	9.12E-03	9.12E-03
Yelp R.	3.98E-02	2.09E-03
Amazon R.	9.12E-03	4.37E-03
Sentiment 140	2.75E-02	9.12E-07

Tabela 4.2: Taxas de aprendizado utilizadas na etapa de aperfeiçoamento do modelo do ULMFit para cada base.

- Aperfeiçoamento do modelo com dados do domínio específico;
- Aperfeiçoamento do classificador no domínio específico;

O modelo utilizado foi pré-treinado na base Wikitext-103 [23], a qual possui 28.6 mil artigos da Wikipédia já pré processados e 103 milhões de palavras. A arquitetura do modelo utilizada foi a *ASGD Weight-Dropped LSTM* (AWD-LSTM) [22], com *embeddings* de tamanho 400, 3 camadas, 1150 *hidden activations* por camada.

Em seguida, o modelo foi aperfeiçoado em cima de cada base de dados utilizada neste trabalho, resultando em 4 modelos. Assim como em [4], foi aplicada a técnica de *discriminative fine-tuning*, onde para cada camada foi aplicada uma taxa de aprendizado otimizada. Os valores de cada taxa de aprendizado utilizados podem ser visualizados na Tabela 4.2. A biblioteca da *fastai*[3] nos ajuda a encontrar esses valores.

Para aperfeiçoar o classificador, 2 blocos lineares extras foram inseridos, ambos com *normalização batch*, *dropout* de 0.1 e *ReLu* como função de ativação.

BERT

O *BERT* [10] possui dois principais modelos pré-treinados para utilizar, o *BERT-base* e o *BERT-large*, onde o primeiro possui uma pilha de 12 *Transformers* [26] enquanto o segundo possui 24, ambos utilizaram como base de dados o *Toronto Book Corpus* (800 milhões de palavras) e a Wikipédia em inglês (2,500 milhões de palavras). Neste trabalho, foi utilizado o *BERT-base* (uma vez que o tempo de treinamento da versão *large* é muito grande) por meio da biblioteca *HuggingFace*. Este modelo possui um *hidden size* de 768 , 12 *attention heads* e 109 milhões de parâmetros.

Para o classificador, utilizou-se uma camada de *dropout* com probabilidade 0.3, uma camada de saída linear com uma função *softmax*.

Para o treinamento do modelo, foram utilizados os seguintes hiperparâmetros: um *batch size* de 16, a função de otimização *AdamW* com a taxa de aprendizado de $2 * 10^{(-5)}$, a função de perda *CrossEntropyLoss*, e um número de *EPOCHS* de 5. As escolhas dos parâmetros foram baseadas em [10] ou em testes empíricos. O tamanho máximo da

[3]https://www.fast.ai/

Base	# tokens
IMDb	512
Yelp R.	350
Amazon R.	250
Sentiment140	80

Tabela 4.3: Tamanho máximo da sentença para BERT e XLNet

sentença foi definido de acordo com a base de dados, visto que eles variam em tamanho de sentença, como visto na Tabela 4.1. Os valores utilizados para o tamanho máximo das sentenças pode ser visualizado na tabela 4.3. Os gráficos que representam a quantidade de palavras para cada base da dados podem ser visualizados nas Figuras 4.1, 4.2, 4.3 e 4.4.

XLNet

O *XLNet* [11] possui também dois modelos pré-treinados, o *XLNet-base* e o *XLNet-large*, treinados na Wikipedia, *BooksCorpus, Giga5, ClueWeb e Common Crawl*. Neste trabalho, optou-se pela versão *base* pelo mesmo motivo do algoritmo *BERT*, que também é disponibilizada pela biblioteca *HuggingFace*. Esta versão possui 12 camadas, com tamanho 768 (*hidden size*), 12 *attention heads* e 110 milhões de parâmetros.

Para o treinamento do modelo, foram utilizados os seguintes hiperparâmetros: um *batch size* de tamanho 4, número de *EPOCHS* de 3, uma função de otimização *AdamW* com a taxa de aprendizado de $3 * 10^{(-5)}$ e um peso de decaimento de 0.1. As escolhas dos parâmetros foram baseadas em [11] ou em testes empíricos. O tamanho máximo da sentença foi o mesmo definido para o algoritmo BERT, como visto na Tabela 4.3.

4.3.2 Algoritmos clássicos

Todos os algoritmos utilizados são da biblioteca *Scikit Learn* [41] implementados em *Python*. Para a escolha dos parâmetros, foi utilizado o trabalho de [37] como motivação. Foram feitos testes iniciais com diversos algoritmos para saber quais apresentariam uma performance competitiva assim como um tempo de treinamento razoável. Os algoritmos escolhidos para esta fase foram o *SVM* com o parâmetro C variando entre os valores [0.1, 1, 10], *Complement Naïve Bayes, Multinomial Naïve Bayes, Bernoulli Naïve Bayes* (ambos com *alpha* variando entre [0.1, 0.01, 0.001], *KNN* com K variando entre os valores [3, 13, 29, 89], *RandomForest* com o número de estimadores entre [10, 50 ,100], *DT* com os critérios *gini e entropy*, *SGDC* com o parâmetro *alpha* variando entre os valores [0.00001, 0.000001] e *penalty* entre [l2', 'elasticnet'] e *Ridge Classifier* com *alpha* entre [1,1.5,2].

Optou-se por não utilizar o algoritmo *SVM* com *kernel* polinomial ou *rbf* pois em um teste inicial apresentaram resultados muito abaixo do *kernel* linear e um elevado tempo de treinamento. Os algoritmos *RidgeClassifier* e *SGDClassifier* foram incluídos por motivos de alta acurácia e baixo tempo de treinamento.

4.4 Avaliação

Antes de explicar as métricas utilizadas neste trabalho, conceituaremos alguns termos importantes que serão utilizados abaixo:

- Verdadeiro Positivo (VP): é usado quando algo foi previsto como verdadeiro e realmente era verdadeiro;
- Verdadeiro Negativo (VN): é usado quando algo foi previsto como negativo e realmente era negativo;
- Falso Positivo (FP): é usado quando algo foi previsto como positivo porém seu real valor era negativo;
- Falso Negativo (FN): é usado quando algo foi previsto como negativo porém seu real valor era positivo.

Para realizar a análise comparativa entre os algoritmos citados anteriormente, utilizou-se da acurácia, *F1-score* e do tempo gasto para treinar os modelos com cada algoritmo. Abaixo são explicadas cada uma dessas métricas:

Acurácia nada mais é do que a taxa de acertos do modelo. Portanto, se em uma base de dados com 10 documentos, o modelo acertou 8, a acurácia deste modelo é 0.8, ou 80%. Em fórmula, temos que $acc = \frac{VP+VN}{VP+VN+FP+FN}$.

F1-Score é uma métrica que combina duas outras mais simples, a *Precisão* e a *Revocação*. Precisão responde a seguinte pergunta: Dos que foram classificados como positivos, quais realmente eram? Para deixar mais claro, a fórmula é a seguinte: $prec = \frac{TP}{TP+FP}$. Por outro lado, a revocação responde a seguinte pergunta: Quando realmente é da classe X, o quão frequente classifica-se de fato como X? Em fórmula, ficaria: $prec = \frac{TP}{TP+FN}$. Finalmente, a métrica *F1-Score* é a junção da precisão e da revocação em um número só, o qual indica o quão bom está seu modelo, até mesmo quando os dados estão desproporcionais, ou seja, tem mais dados positivos do que negativos ou vice-versa, sua fórmula é:

$$F1 = \frac{2 \times precisão \times revocação}{precisão + revocação} \tag{4.1}$$

O *Tempo de treinamento* nada mais é que a quantidade de segundos gastos para que um modelo seja treinado. Para os algoritmos clássicos, esta métrica é disponibilizada pela própria biblioteca *Scikit Learn*, enquanto para os algoritmos modernos foi feita a contagem através da biblioteca *tqdm*, a qual dentre várias funções, conta o tempo gasto em uma etapa do código.

Para separação do banco de dados, foi feita a seguinte estratégia: para os algoritmos modernos foi separado 25% de cada base de dados para teste e 25% para avaliar os resultados dos algoritmos modernos. Os outros 50% foram utilizados para treinamento em si. Por outro lado, para os algoritmos clássicos, foi utilizada a técnica de *cross-validation*, onde separou-se a base de dados em 5 grupos e a acurácia final utilizada foi a média entre as 5. Não foi possível utilizar *cross-validation* para os algoritmos modernos devido a seu alto tempo de treinamento. Na próxima Seção encontram-se os resultados obtidos em cada base de dados.

Capítulo 5

Resultados

No decorrer deste capítulo são apresentados os resultados obtidos por intermédio de experimentos. A Seção 5.1 mostra os resultados obtidos com os algoritmos clássicos de acordo com sua respectiva parametrização, a Seção 5.2 se refere aos resultados obtidos com os algoritmos modernos, a Seção 5.3 faz uma análise comparativa entre os algoritmos clássicos e modernos e, por fim, a Seção 5.4 descreve as limitações encontradas durante os experimentos.

5.1 Algoritmos Clássicos

Para realizar experimentos em vários algoritmos e variando os parâmetros interessados, utilizou-se a função *GridSearchCV* da biblioteca *Scikit Learn* [41], a qual treina exaustivamente um algoritmo a partir de uma lista de valores recebida como parâmetro. Além disso, o *GridSearchCV* também implementou a técnica de *Cross-Validations*. Os resultados da acurácia de cada algoritmo podem ser visualizados na Tabela 5.1, os tempos de treinamento estão na Tabela 5.2 e o *F1-score* está na Tabela 5.3.

A partir dos dados mostrados nas Tabelas 5.1, 5.2, e 5.3 é notável que diversos algoritmos clássicos ainda são competitivos para a tarefa de classificação de sentimentos em textos. As análises abaixo se referem à base de dados *IMDb* e podem ser estendidas para as outras bases. O algoritmo **SVM**, apesar de atingir acurácia e *F1-score* altos como 0.9066, apresentou um tempo de treinamento muito alto, chegando a até 38 minutos, o que deixa a desejar quando comparado com algoritmos mais complexos demandam a mesma faixa de tempo de treinamento. Os três algoritmos baseados em métodos **Naïve Bayes** apresentaram acurácia, tempo de treinamento e *F1-Score* muito similares, onde o *Complement NB* com $\alpha = 10^{(-2)}$ obteve a maior acurácia, 0.8617. O algoritmo **KNN** não obteve um resultado competitivo quando comparado aos outros algoritmos, pois teve uma acurácia inferior (maior acurácia de 0.7770) e um tempo de treinamento relativamente maior

Acurácia dos algoritmos clássicos				
Algoritmos	IMDb	Yelp R.	Amazon R.	Sentiment140
SVM (C=0.1)	0.8858	0.9122	0.8378	0.7654
SVM (C=1)	**0.9066**	0.9223	0.8496	**0.7773**
SVM (C=10)	0.8875	0.9061	0.8207	0.7477
Complement NB ($\alpha = 10^{(-3)}$)	0.8408	0.8454	0.7414	0.7103
Complement NB ($\alpha = 10^{(-2)}$)	0.8547	0.8599	0.7618	0.7135
Complement NB ($\alpha = 10^{(-1)}$)	0.8617	0.8740	0.7888	0.7304
Bernoulli NB ($\alpha = 10^{(-3)}$)	0.8415	0.7526	0.7551	0.7190
Bernoulli NB ($\alpha = 10^{(-2)}$)	0.8498	0.7536	0.7729	0.7238
Bernoulli NB ($\alpha = 10^{(-1)}$)	0.8560	0.7519	0.7908	0.7397
Multinomial NB ($\alpha = 10^{(-3)}$)	0.8408	0.8455	0.7414	0.7094
Multinomial NB ($\alpha = 10^{(-2)}$)	0.8547	0.8599	0.7614	0.7125
Multinomial NB ($\alpha = 10^{(-1)}$)	0.8617	0.8740	0.7888	0.7293
KNN (K=3)	0.7632	0.7127	0.6459	0.6679
KNN (K=13)	0.7770	0.7418	0.7016	0.7052
KNN (K=29)	0.7751	0.7493	0.7306	0.7240
KNN (K=89)	0.7590	0.7418	0.7594	0.7325
RandomForest (n=10)	0.7468	0.7916	0.7220	0.7181
RandomForest (n=50)	0.8258	0.8572	0.7973	0.7529
RandomForest (n=100)	0.8424	0.8703	0.8092	0.7598
DecisionTree (gini)	0.7152	0.7599	0.6822	0.6817
DecisionTree (entropy)	0.7116	0.7598	0.6789	0.6737
SGDClassifier ($\alpha = 10^{(-5)}$ e l2)	0.8997	0.9192	0.8438	0.7696
SGDClassifier ($\alpha = 10^{(-5)}$ e elasticnet)	0.8997	0.9194	0.8444	0.7722
SGDClassifier ($\alpha = 10^{(-6)}$ e l2)	0.8823	0.9034	0.8171	0.7405
SGDClassifier ($\alpha = 10^{(-6)}$ e elasticnet)	0.8820	0.9038	0.8154	0.7414
RidgeClassifier ($\alpha = 1$)	0.9007	0.9211	0.8463	0.7726
RidgeClassifier ($\alpha = 1.5$)	0.9022	0.9228	0.8495	0.7752
RidgeClassifier ($\alpha = 2$)	0.9028	**0.9234**	**0.8512**	0.7767
TPBG	0.8583	0.8075	0.8061	0.7387

Tabela 5.1: Resultados experimentais dos algoritmos clássicos - Acurácia

que os outros (45s). Os resultados deste algoritmo indicam que possivelmente a acurácia seria maior caso aumentasse o valor de k ainda mais, porém ainda assim o algoritmo perderia para os outros em tempo de treinamento. O mesmo ocorre com o algoritmo **RandomForest**, que apresenta resultados interessantes (acurácia de 0.8424) mas o tempo de treinamento é altamente dependente de n fazendo com que outros algoritmos sejam mais interessantes. O algoritmo **DecisionTree** apesar de ter um baixo tempo de treinamento (0.08s), sua acurácia e *F1-score* não foram competitivos (maior acurácia de 0.7152). O algoritmo **SGDClassifier** obteve uma acurácia alta (0.8997) e o menor tempo de treinamento dentre todos os algoritmos (0.01s). O algoritmo com os melhores resultados, **RidgeClassifier**, o qual obteve acurácias próximas ou maiores do que o *SVM* (0.9028), e o segundo tempo de treinamento mais baixo deste grupo de algoritmos (0.02s). E por fim, o algoritmo testado **TPBG**, o qual obteve um tempo de treinamento quase constante em todas as bases de dados, em torno de 1400 segundos e uma acurácia de 0.8583.

É possível reparar que na Figura 5.1 que os algoritmos *SVM, SGDClassifier e RidgeClassifier* obtiveram resultados muito similares, deixando o *SVM* clássico pouco competitivo devido ao seu alto tempo de treinamento.

5.2 Algoritmos Modernos

Nesta fase experimental, algumas limitações foram encontradas. A primeira limitação foi a falta de recursos de *hardware* para treinamento, uma vez que estes algoritmos modernos necessitam de GPU para que o tempo de treinamento fique viável. Para contornar esta limitação, foi utilizada a plataforma gratuita da empresa *Google* chamada de *Google Colab* [1], a qual disponibiliza um ambiente virtual no navegador com *Python* e com acesso a GPU, a qual varia de sessão para sessão.

Nesta fase, não foi possível variar parâmetros dos algoritmos, devido ao tempo de treino ser muito elevado e a plataforma estipular um limite de uso da GPU, bloqueando este recurso quando o limite é atingido.

Para implementar os algoritmos, foi utilizada a biblioteca *HuggingFace* [2], que disponibiliza diversas funções e modelos estado-da-arte de processamento de linguagem natural (NLP) na linguagem *Python*. Os resultados da acurácia de cada algoritmo podem ser visualizados na Tabela 5.4, os tempos de treinamento estão na Tabela 5.5 e o *F1-score* está na Tabela 5.6.

Os algoritmos modernos obtiveram acurácia e *F1-score* elevados, porém o tempo de treinamento também foi elevado. O algoritmo **ULMFit**, por exemplo, atingiu 0.9408 de

[1] `https://colab.research.google.com/`
[2] `https://huggingface.co/`

Tempo de treinamento dos algoritmos clássicos (s)				
Algoritmos	IMDb	Yelp R.	Amazon R.	Sentiment140
SVM (C=0.1)	2282.87	1057.71	1048.57	442.05
SVM (C=1)	1521.09	696.13	682.02	196.30
SVM (C=10)	750.92	315.25	347.13	113.60
Complement NB ($\alpha = 10^{(-3)}$)	0.02	0.02	0.02	0.01
Complement NB ($\alpha = 10^{(-2)}$)	0.02	0.02	0.02	0.01
Complement NB ($\alpha = 10^{(-1)}$)	0.02	0.02	0.02	0.01
Bernoulli NB ($\alpha = 10^{(-3)}$)	0.04	0.03	0.03	0.02
Bernoulli NB ($\alpha = 10^{(-2)}$)	0.04	0.03	0.03	0.02
Bernoulli NB ($\alpha = 10^{(-1)}$)	0.04	0.03	0.03	0.02
Multinomial NB ($\alpha = 10^{(-3)}$)	0.02	0.02	0.02	0.01
Multinomial NB ($\alpha = 10^{(-2)}$)	0.02	0.02	0.02	0.01
Multinomial NB ($\alpha = 10^{(-1)}$)	0.02	0.02	0.02	0.01
KNN (K=3)	45.41	31.74	24.96	7.66
KNN (K=13)	47.99	34.48	27.72	9.03
KNN (K=29)	47.90	34.58	27.76	9.00
KNN (K=89)	47.96	34.66	27.88	9.09
RandomForest (n=10)	0.16	0.12	0.10	0.12
RandomForest (n=50)	0.54	0.40	0.39	0.56
RandomForest (n=100)	0.99	0.75	0.73	1.09
DecisionTree (gini)	0.08	0.05	0.03	0.02
DecisionTree (entropy)	0.08	0.05	0.03	0.02
SGDClassifier ($\alpha = 10^{(-5)}$ e l2)	**0.01**	**0.01**	**0.01**	**0.01**
SGDClassifier ($\alpha = 10^{(-5)}$ e elasticnet)	**0.01**	**0.01**	**0.01**	**0.01**
SGDClassifier ($\alpha = 10^{(-6)}$ e l2)	**0.01**	**0.01**	**0.01**	**0.01**
SGDClassifier ($\alpha = 10^{(-6)}$ e elasticnet)	**0.01**	**0.01**	**0.01**	**0.01**
RidgeClassifier ($\alpha = 1$)	0.02	0.02	0.02	0.02
RidgeClassifier ($\alpha = 1.5$)	0.02	0.02	0.02	0.02
RidgeClassifier ($\alpha = 2$)	0.02	0.02	0.02	0.02
TPBG	1614	1366	1311	1384

Tabela 5.2: Resultados experimentais dos algoritmos clássicos - Tempo de treinamento médio (s)

F1-Score dos algoritmos clássicos				
Algoritmos	IMDb	Yelp R.	Amazon R.	Sentiment140
SVM (C=0.1)	0.8850	0.9122	0.8377	0.7653
SVM (C=1)	0.9018	0.9223	0.8496	**0.7773**
SVM (C=10)	0.8828	0.9061	0.8206	0.7476
Complement NB ($\alpha = 10^{(-3)}$)	0.8408	0.8453	0.7413	0.7099
Complement NB ($\alpha = 10^{(-2)}$)	0.8547	0.8599	0.7617	0.7131
Complement NB ($\alpha = 10^{(-1)}$)	0.8616	0.8740	0.7886	0.7298
Bernoulli NB ($\alpha = 10^{(-3)}$)	0.8415	0.7506	0.7551	0.7188
Bernoulli NB ($\alpha = 10^{(-2)}$)	0.8497	0.7513	0.7728	0.7236
Bernoulli NB ($\alpha = 10^{(-1)}$)	0.8560	0.7495	0.7908	0.7395
Multinomial NB ($\alpha = 10^{(-3)}$)	0.8407	0.8455	0.7413	0.7088
Multinomial NB ($\alpha = 10^{(-2)}$)	0.8547	0.8598	0.7613	0.7119
Multinomial NB ($\alpha = 10^{(-1)}$)	0.8616	0.8740	0.7886	0.7285
KNN (K=3)	0.7630	0.7103	0.6459	0.6672
KNN (K=13)	0.7769	0.7377	0.7015	0.7040
KNN (K=29)	0.7745	0.7437	0.7306	0.7221
KNN (K=89)	0.7554	0.7328	0.7594	0.7298
RandomForest (n=10)	0.7453	0.7904	0.7199	0.7171
RandomForest (n=50)	0.8258	0.8571	0.7969	0.7528
RandomForest (n=100)	0.8424	0.8701	0.8090	0.7598
DecisionTree (gini)	0.7152	0.7599	0.6822	0.6817
DecisionTree (entropy)	0.7116	0.7597	0.6789	0.6737
SGDClassifier ($\alpha = 10^{(-5)}$ e l2)	0.8997	0.9192	0.8438	0.7696
SGDClassifier ($\alpha = 10^{(-5)}$ e elasticnet)	0.8997	0.9193	0.8444	0.7722
SGDClassifier ($\alpha = 10^{(-6)}$ e l2)	0.8823	0.9034	0.8170	0.7405
SGDClassifier ($\alpha = 10^{(-6)}$ e elasticnet)	0.8819	0.9038	0.8153	0.7413
RidgeClassifier ($\alpha = 1$)	0.9006	0.9211	0.8463	0.7725
RidgeClassifier ($\alpha = 1.5$)	0.9022	0.9228	0.8494	0.7752
RidgeClassifier ($\alpha = 2$)	**0.9028**	**0.9234**	**0.8511**	0.7767
TPBG	0.8578	0.8067	0.8065	0.7378

Tabela 5.3: Resultados experimentais dos algoritmos clássicos - F1-Score

Acurácia dos algoritmos modernos				
Algoritmos	IMDb	Yelp R.	Amazon R.	Sentiment140
ULMFit	0.9408	0.9606	0.9183	0.6929
BERT	0.9319	0.9357	0.9228	0.8198
XLNet	**0.9553**	**0.9616**	**0.9231**	**0.8227**

Tabela 5.4: Resultados experimentais dos algoritmos modernos - Acurácia

Tempo de treinamento dos algoritmos modernos (s)				
Algoritmos	IMDb	Yelp R.	Amazon R.	Sentiment140
ULMFit	**11161**	**6777**	**3122**	**720**
BERT	12002	10018	8579	2342
XLNet	12603	10799	9181	2466

Tabela 5.5: Resultados experimentais dos algoritmos modernos - Tempo de treinamento médio (s)

acurácia na base IMDb e, para isso, levou cerca de três horas para atingir este resultado. O algoritmo **BERT** por sua vez, obteve o resultado de 0.9319 de acurácia na mesma base, e para isso, levou cerca de 3h e 20 minutos para treinar. Por fim, o algoritmo **XLNet** obteve a maior acurácia na mesma base de dados, de 0.9553, levando cerca de 3h e 30 minutos para treinar o modelo.

Podemos visualizar na Imagem 5.2 que para base de dados com sentenças pequenas como a Sentiment140, que possui um número médio de *tokens* (palavras após pré-processamento) de 24, o algoritmo *ULMFit* apresentou uma queda acentuada na acurácia, 0.6943, enquanto o *BERT* e o *XLNet* obtiveram na mesma base 0.8204 e 0.8231 respectivamente.

5.3 Comparações entre os algoritmos

Nesta seção, os algoritmos clássicos são comparados diretamente com os algoritmos modernos, através de análises qualitativas, quantitativas, tabelas e gráficos auxiliares.

A partir da Tabela 5.7 e da Figura 5.3, é possível perceber que, em geral, os algoritmos modernos atingiram uma acurácia superior aos algoritmos clássicos em quase todas

F1-Score dos algoritmos modernos				
Algoritmos	IMDb	Yelp R.	Amazon R.	Sentiment140
ULMFit	0.9413	0.9602	0.9154	0.6943
BERT	0.9311	0.9323	0.9231	0.8204
XLNet	**0.9551**	**0.9601**	**0.9233**	**0.8231**

Tabela 5.6: Resultados experimentais dos algoritmos modernos - F1-Score

	IMDb	**Yelp R.**	**Amazon R.**	**Sentiment140**
ULMFit	0.9408	0.9606	0.9183	0.6929
BERT	0.9319	0.9357	0.9228	0.8198
XLNet	**0.9553**	**0.9616**	**0.9231**	**0.8227**
SVM	0.9066	0.9223	0.8496	0.7773
SGDClassifier ($\alpha = 10^{(-5)}$ **e el.**)	0.8997	0.9194	0.8444	0.7772
RidgeClassifier ($\alpha = 2$)	0.9028	0.9234	0.8495	0.7767
TPBG	0.8583	0.8075	0.8061	0.7387

Tabela 5.7: Comparação da acurácia entre os algoritmos modernos e os principais algoritmos clássicos

	IMDb	**Yelp R.**	**Amazon R.**	**Sentiment140**
ULMFit	11161	6777	3122	720
BERT	12002	10018	8579	2342
XLNet	12603	10799	9181	2466
SVM (C=1)	1521.09	696.13	682.02	196.30
SGDClassifier ($\alpha = 10^{(-5)}$) **e el.)**	**0.01**	**0.01**	**0.01**	**0.01**
RidgeClassifier ($\alpha = 2$)	**0.02**	**0.02**	**0.02**	**0.02**
TPBG	1614	1366	1311	1384

Tabela 5.8: Comparação do tempo de treinamento em segundos entre algoritmos modernos e principais clássicos

as bases de dados, com uma diferença relativamente pequena. Quando comparamos algoritmos específicos, como por exemplo o *BERT* e o *RidgeClassifier* na base de dados Yelp R., pode-se perceber que os resultados são extremamente próximos, 0.9357 e 0.9234 respectivamente, deixando o *RidgeClassifier* competitivo em relação ao *BERT*. Além disso, para a base de dados Sentiment140, a qual é pequena quando comparada com as outras, o algoritmo *ULMFit* apresentou o pior resultado entre todos os algoritmos, podendo ser um indício de que este algoritmo não apresenta bons resultados para bases que possuem sentenças curtas ou que tenham origem de redes sociais.

Além disso, a partir da Tabela 5.8 pode-se perceber que os algoritmos clássicos em geral gastam pouco tempo para realizar o treinamento e obter valores de acurácia próximos aos algoritmos modernos, como por exemplo os algoritmos *SGDClassifier* e *RidgeClassifier* gastaram 0.01 e 0.02 segundos para realizar o treinamento em todas as bases, salvo o *SVM* que gastou cerca de 25 minutos na base IMDb. O algoritmo *XLNet*, o qual obteve as maiores taxas de acurácia, gastou cerca de 3 horas e 30 minutos para fazer o treinamento na base IMDb.

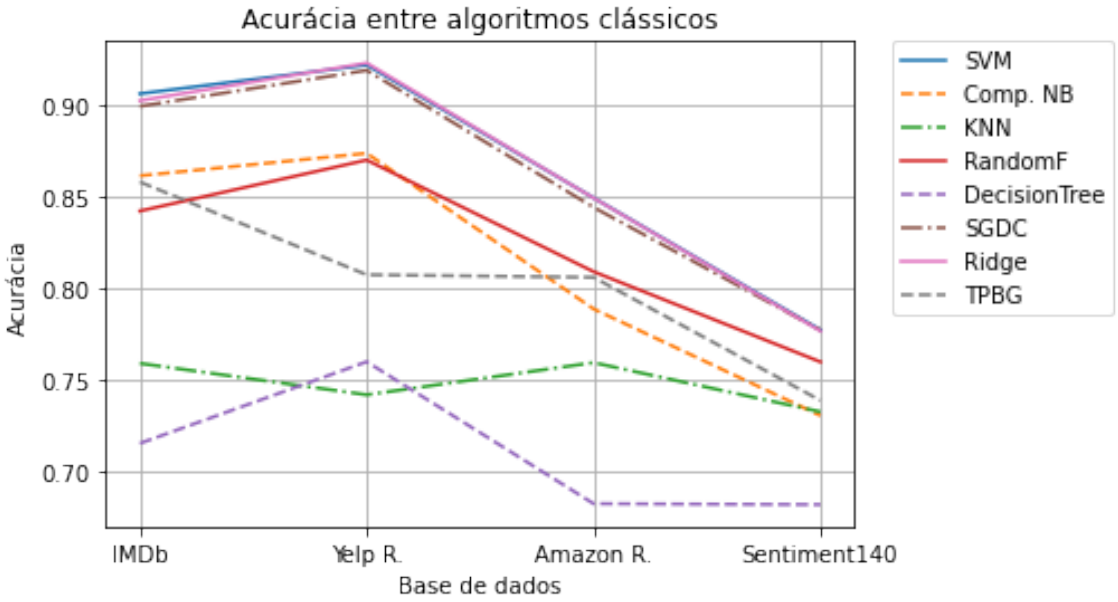

Figura 5.1: Acurácia dos algoritmos clássicos

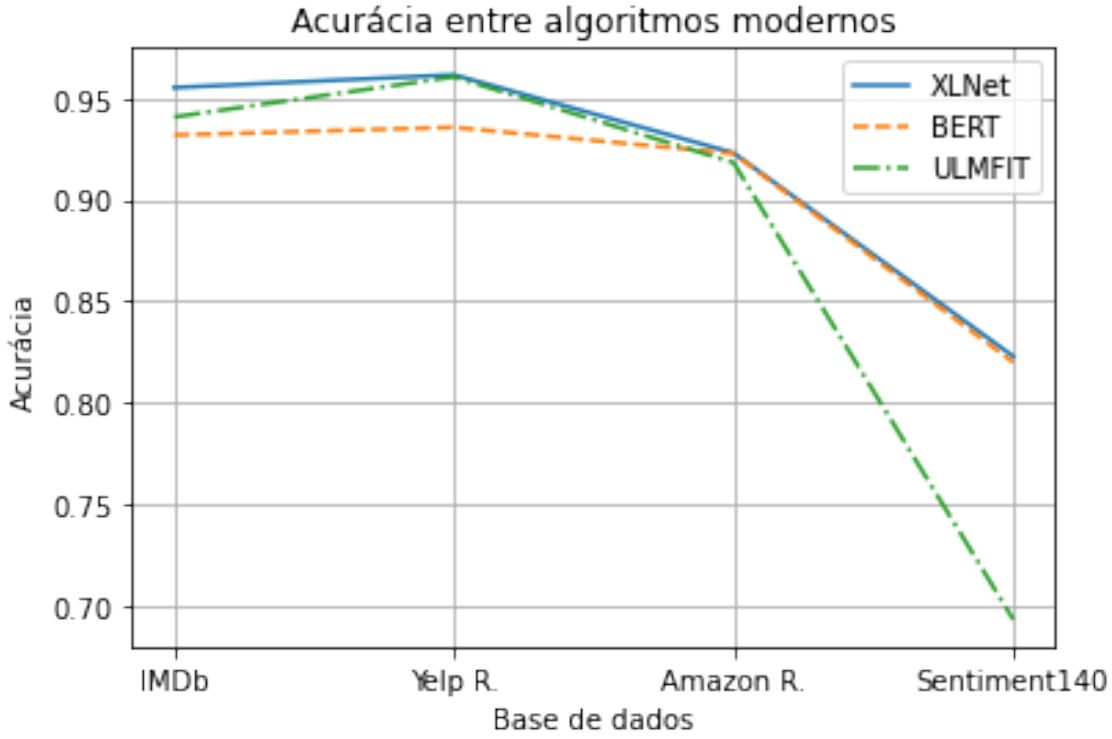

Figura 5.2: Acurácia dos algoritmos modernos

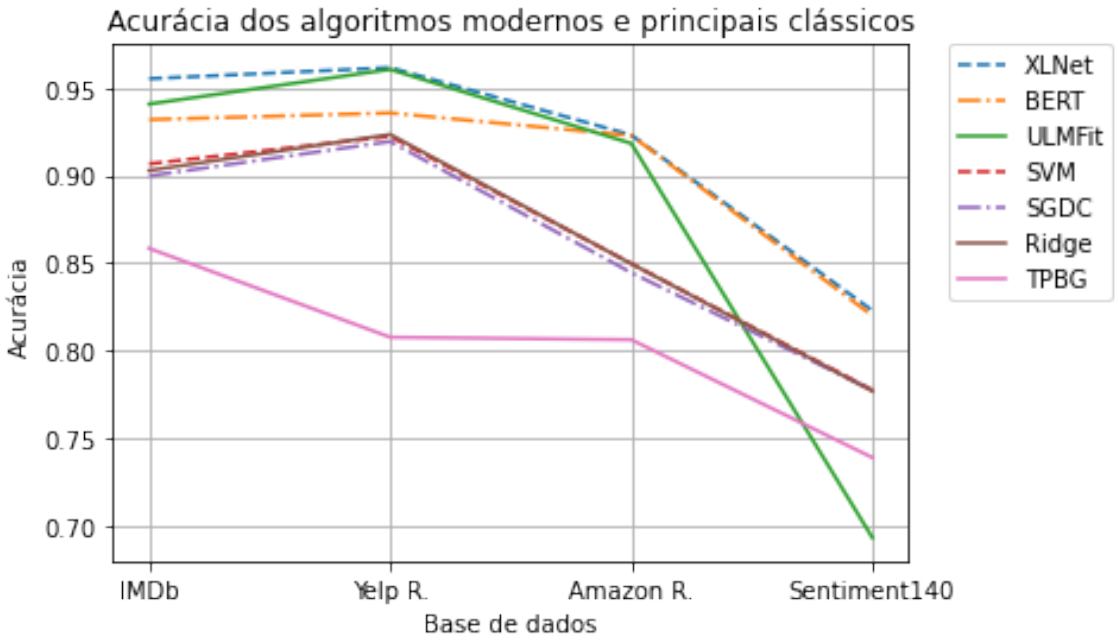

Figura 5.3: Comparação da acurácia entre os algoritmos modernos e os principais algoritmos clássicos

5.4 Limitações

Durante os experimentos, algumas limitações técnicas foram encontradas, as quais serão citadas e explicadas nesta seção.

Os algoritmos modernos - *ULMFit, BERT e XLNet*, são dependentes de unidades gráficas de processamento, e para isso, a ferramenta *Google Colab* foi utilizada a fim de ter acesso a uma GPU virtual, uma vez que não havia acesso a uma máquina física com tal *hardware*. O *Google Colab* é uma máquina virtual com um ambiente *Python* com o hardware necessário para rodar modelos de aprendizado de máquina. Porém, esta ferramenta tem limitações e não garante a disponibilidade destes recursos 100% do tempo (a não ser que contrate o plano pago). As principais limitações que afetaram este trabalho foram:

- Tempo de sessão limitada a 1 hora e meia, após esse tempo, se não houver atividade do usuário, a sessão é encerrada, fazendo com que os treinamentos longos necessitassem de constantes visitas à sessão para garantir que esta não fosse encerrada no meio do treinamento;
- Restrição do uso de GPU's paralelas, restrito a somente uma seção com uso de GPU por vez, fazendo com que os treinamentos não pudessem ser paralelizados;

- Restrição do uso excessivo da GPU, a empresa não cita qual é este limite, porém diz que usos muito prolongados de GPU podem bloquear temporariamente o recurso, o que fez com que alguns treinamentos fossem adiados ou cancelados;

Capítulo 6

Conclusão

Neste projeto, foram comparados e analisados algoritmos de aprendizado supervisionado e de aprendizado por transferência em bases textuais aplicadas a análise de sentimentos. A principal motivação se deve ao fato de muitos acharem que devem escolher os algoritmos mais complexos somente pelo fato de obterem os melhores resultados.

A Seção 6.1 descreve as considerações finais do estudo, juntamente com a validação das hipóteses e dos objetivos previamente propostos. A Seção 6.2 tem como finalidade indicar projeções futuras proporcionadas pelo desenvolvimento dessa pesquisa.

6.1 Considerações finais

Neste trabalho, comparou-se os principais algoritmos estado-da-arte em processamento de linguagem natural com algoritmos clássicos por meio da realização de experimentos em 4 bases de dados textuais voltadas para análise de sentimentos. Os experimentos consistiram da fase de pré-processamento dos dados, estruturação dos dados, treinamento e avaliação. Os algoritmos empregaram técnicas de classificação de acordo com a polaridade do sentimento.

Os resultados experimentais na classificação dos sentimentos de cada base de dados possibilitaram observar que de fato os algoritmos estado-da-arte utilizados, *ULMfit, BERT e XLNet*, conseguem alcançar uma acurácia maior que os algoritmos clássicos utilizados, *SVM, NB, KNN, RF, DT, SGDC, Ridge Classifier e TPBG*, mesmo que em alguns casos pequena. Por outro lado, o tempo gasto para aperfeiçoamento do modelo de linguagem é demorado, chegando a 3 horas e 30 minutos para bases relativamente pequenas como a IMDb, e requerem recursos computacionais caros como placas gráficas, enquanto os algoritmos clássicos não demandam muito tempo e geralmente não necessitam de recursos computacionais caros.

Finalmente, pode-se concluir que, mesmo com um aumento na acurácia, os algoritmos modernos podem não ser a melhor escolha, visto que utilizando uma base de dados relativamente pequena como o IMDb, o tempo de treinamento gasto com o algoritmo moderno com melhor desempenho (acurácia de 0.9553) foi de 3 horas e 30 minutos, enquanto o algoritmo clássico (acurácia de 0.9028) gastou 0.2 segundos. Além disso, a alta complexidade, a necessidade de ajustar muitos hiperparâmetros e a alta dependência de recursos computacionais como GPU's fazem dos algoritmos modernos uma opção mais difícil de ser implementada. A decisão de qual técnica deve ser usada deve ser analisada com cautela. Caso pequenas melhorias em acurácia sejam cruciais para o problema e recursos computacionais não são uma limitação, modelos de transferência de aprendizado podem ser a melhor opção. Por outro lado, se os resultados dos algoritmos clássicos são suficientes e/ou recursos computacionais são limitados, os algoritmos clássicos de aprendizado supervisionado são a melhor escolha.

É importante citar que os algoritmos que utilizaram a técnica de aprendizado por transferência são amplamente utilizados e este trabalho não descoraja o leitor a utilizá-los, principalmente em cenários que recursos computacionais não são uma limitação e pouca melhoria é algo crítico.

6.2 Trabalhos futuros

Durante a realização deste trabalho, foram observadas diversas possíveis melhorias ou adições que podem ser feitas no futuro. A variação de hiperparâmetros dos modelos modernos na fase de aperfeiçoamento, a utilização de mais modelos modernos além dos três já utilizados, a utilização da técnica de *n-gram* nos algoritmos clássicos e a utilização de mais de duas classes para classificar um sentimento podem fortalecer ainda mais o problema discutido.

Referências

[1] Kumar, B Shravan e Vadlamani Ravi: *A survey of the applications of text mining in financial domain*. Knowledge-Based Systems, 114:128–147, 2016. ix, 6

[2] Liu, R., Y. Shi, C. Ji e M. Jia: *A survey of sentiment analysis based on transfer learning*. IEEE Access, 7:85401–85412, 2019. ix, 6

[3] Pan, S. J. e Q. Yang: *A survey on transfer learning*. IEEE Transactions on Knowledge and Data Engineering, 22(10):1345–1359, 2010. ix, 12, 13, 14

[4] Howard, Jeremy e Sebastian Ruder: *Universal language model fine-tuning for text classification*, 2018. ix, 2, 12, 15, 16, 21, 26, 27

[5] Bollen, Johan, Huina Mao e Xiaojun Zeng: *Twitter mood predicts the stock market*. Journal of computational science, 2(1):1–8, 2011. 1

[6] Tumasjan, Andranik, Timm O Sprenger, Philipp G Sandner e Isabell M Welpe: *Predicting elections with twitter: What 140 characters reveal about political sentiment*. Em *Fourth international AAAI conference on weblogs and social media*, 2010. 1

[7] Culotta, Aron: *Towards detecting influenza epidemics by analyzing twitter messages*. Em *Proceedings of the first workshop on social media analytics*, páginas 115–122. acm, 2010. 1

[8] Kumar, Akshi e Arunima Jaiswal: *Systematic literature review of sentiment analysis on twitter using soft computing techniques*. Concurrency and Computation: Practice and Experience, 32(1):e5107, 2020. 2

[9] Zhang, Lei, Shuai Wang e Bing Liu: *Deep learning for sentiment analysis: A survey*. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 8(4):e1253, 2018. 2, 20

[10] Devlin, Jacob, Ming Wei Chang, Kenton Lee e Kristina Toutanova: *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2019. 2, 12, 16, 19, 21, 27

[11] Yang, Zhilin, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov e Quoc V. Le: *Xlnet: Generalized autoregressive pretraining for language understanding*, 2020. 2, 12, 18, 19, 21, 28

[12] Haddi, Emma, Xiaohui Liu e Yong Shi: *The role of text pre-processing in sentiment analysis*. Procedia Computer Science, 17:26–32, 2013. 4

[13] Tandel, Sayali Sunil, Abhishek Jamadar e Siddharth Dudugu: *A survey on text mining techniques*. Em *2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS)*, páginas 1022–1026. IEEE, 2019. 5

[14] Samuel, Arthur L.: *Some studies in machine learning using the game of checkers*. IBM JOURNAL OF RESEARCH AND DEVELOPMENT, páginas 71–105, 1959. 7

[15] Mitchell, Tom M: *Machine learning*, 1997. 7

[16] Kotsiantis, Sotiris B, I Zaharakis e P Pintelas: *Supervised machine learning: A review of classification techniques*. Emerging artificial intelligence applications in computer engineering, 160:3–24, 2007. 7

[17] Thakare, Santosh e Sandeep R Sirsat: *Review of market prediction using sentiment and opinion mining*. International Journal of Electronics, Communication and Soft Computing Science & Engineering (IJECSCSE), página 115, 2015. 7, 8, 9

[18] Mittal, Abhilash e Sanjay Patidar: *Sentiment analysis on twitter data: A survey*. Em *Proceedings of the 2019 7th International Conference on Computer and Communications Management*, ICCCM 2019, páginas 91–95, New York, NY, USA, 2019. ACM, ISBN 978-1-4503-7195-7. `http://doi.acm.org/10.1145/3348445.3348466`. 7, 8, 9

[19] Nassirtoussi, Arman Khadjeh, Saeed Aghabozorgi, Teh Ying Wah e David Chek Ling Ngo: *Text mining for market prediction: A systematic review*. Expert Systems with Applications, 41(16):7653–7670, 2014. 8

[20] Duda, Richard O, Peter E Hart *et al.*: *Pattern classification*. John Wiley & Sons, 2006. 9

[21] Paulo Faleiros, Thiago de: *Propagação em grafos bipartidos para extração de tópicos em fluxo de documentos textuais*. Tese de Doutoramento, Universidade de São Paulo, 2016. 11, 21

[22] Merity, Stephen, Nitish Shirish Keskar e Richard Socher: *Regularizing and optimizing lstm language models*, 2017. 15, 16, 27

[23] Merity, Stephen, Caiming Xiong, James Bradbury e Richard Socher: *Pointer sentinel mixture models*, 2016. 15, 27

[24] Bahdanau, Dzmitry, Kyunghyun Cho e Yoshua Bengio: *Neural machine translation by jointly learning to align and translate*, 2016. 17, 21

[25] Luong, Minh Thang, Hieu Pham e Christopher D. Manning: *Effective approaches to attention-based neural machine translation*, 2015. 17, 21

[26] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser e Illia Polosukhin: *Attention is all you need*, 2017. 17, 21, 27

[27] Wu, Yonghui, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes e Jeffrey Dean: *Google's neural machine translation system: Bridging the gap between human and machine translation*, 2016. 17, 25

[28] Zhu, Yukun, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba e Sanja Fidler: *Aligning books and movies: Towards story-like visual explanations by watching movies and reading books*, 2015. 18

[29] Dai, Zihang, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le e Ruslan Salakhutdinov: *Transformer-xl: Attentive language models beyond a fixed-length context*, 2019. 19

[30] Turing, Alan M: *Computing machinery and intelligence*. Em *Parsing the turing test*, páginas 23–65. Springer, 2009. 20

[31] Liao, Shu Hsien, Pei Hui Chu e Pei Yuan Hsiao: *Data mining techniques and applications–a decade review from 2000 to 2011*. Expert systems with applications, 39(12):11303–11311, 2012. 20

[32] Maas, Andrew L., Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng e Christopher Potts: *Learning word vectors for sentiment analysis*. Em *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, páginas 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. `http://www.aclweb.org/anthology/P11-1015`. 20, 22

[33] Go, Alec, Richa Bhayani e Lei Huang: *Twitter sentiment classification using distant supervision*. CS224N project report, Stanford, 1(12):2009, 2009. 20, 24

[34] Eler, Danilo Medeiros, Denilson Grosa, Ives Pola, Rogério Garcia, Ronaldo Correia e Jaqueline Teixeira: *Analysis of document pre-processing effects in text and opinion mining*. Information, 9(4):100, 2018. 20

[35] Qaiser, Shahzad e Ramsha Ali: *Text mining: use of tf-idf to examine the relevance of words to documents*. International Journal of Computer Applications, 181(1):25–29, 2018. 20

[36] Luz de Araujo, Pedro H., Teófilo E. de Campos e Marcelo Magalhaes Silva de Sousa: *Inferring the source official texts: can SVM beat ULMFiT?* Em *International Conference on the Computational Processing of Portuguese (PROPOR)*, Lecture Notes on Computer Science (LNCS), Evora, Portugal, March 2-4 2020. Springer. `https://propor.di.uevora.pt/`, Code and data available from `https://cic.unb.br/~teodecampos/KnEDLe/`. 21

[37] Rossi, Rafael Geraldeli, Ricardo Marcondes Marcacini, Solange Oliveira Rezende *et al.*: *Benchmarking text collections for classification and clustering tasks.* 2013. 21, 28

[38] Zhang, Xiang, Junbo Zhao e Yann LeCun: *Character-level convolutional networks for text classification.* Em *Advances in neural information processing systems*, páginas 649–657, 2015. 22, 24

[39] Schuster, M. e K. Nakajima: *Japanese and korean voice search.* Em *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, páginas 5149–5152, 2012. 25

[40] Sennrich, Rico, Barry Haddow e Alexandra Birch: *Neural machine translation of rare words with subword units.* Em *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, páginas 1715–1725, Berlin, Germany, agosto 2016. Association for Computational Linguistics. `https://www.aclweb.org/anthology/P16-1162`. 26

[41] Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot e E. Duchesnay: *Scikit-learn: Machine learning in Python.* Journal of Machine Learning Research, 12:2825–2830, 2011. 28, 31

Apêndice A

Fichamento de Artigo Científico

Anexo I

Documentação Original UnB-CIC (parcial)