Universidade de São Paulo

Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Luciana dos Santos Kayo

Paulo Mei

Implementação de Kernel Customizado Aplicado à Análise de Sentimentos em Resenhas de Filmes

São Paulo, Janeiro de 2016

Implementação de Kernel Customizado Aplicado à Análise de Sentimentos em Resenhas de Filmes

Monografia final da disciplina
MAC0499 – Trabalho de Formatura Supervisionado.

Supervisor: Prof. Dr. Marco Dimas Gubitoso

São Paulo, Janeiro de 2016

Resumo

Nos últimos tempos, com o crescimento das redes sociais e a quantidade massiva de dados produzida diariamente pelos usuários dessas redes, houve também um aumento de interesse em obter informações úteis a partir desses dados. Nesse contexto surgiu o termo análise de sentimentos referindo-se à aplicação de algoritmos capazes de extrair conteúdo subjetivo de amostras de dados. O tipo mais comum de análise de sentimentos é a classificação de textos pela sua polaridade, separando os textos que expressam opiniões positivas dos que expressam opiniões negativas. Essa separação é feita por uma função chamada classificador. Este trabalho se propõe a resolver o problema de classificação de textos, mais especificamente de resenha de filmes, proposto pelo site *Kaggle* na competição *Bag of Words Meets Bags of Popcorn*. A solução proposta envolve a criação de um *string kernel* customizado capaz de fazer análise de sentimentos com SVM e linguagem Python.

Palavras-chave: classificador, SVM, kernel customizado, análise de sentimentos, python, resenha de filmes.

Abstract

Recently with the social network's growth and the massive amount of data produced daily by those network's users, there was also a rise of interest on obtaining useful information from this data. In this context, the term sentiment analysis has appeared, referring to the application of algorithms capable of extracting subjective content from data samples. The most common type of sentiment analysis is the classification of texts by their polarity, separating the texts that express positive opinions from the ones that express negative opinions. This separation is done by a function called classifier. This work is about solving the text classification problem specifically involving movie reviews, proposed by *Kaggle's* website in the *Bag of Words Meets Bags Of Popcorn* contest. The proposed solution consists of a new custom string *kernel* capable of doing sentiment analysis with *SVM* and *Python* programming language.
Keywords: classifier, SVM, custom kernel, sentiment analysis, python, movie review.

Sumário

Capítulo 1

Introdução

A análise de sentimentos tem como objetivo extrair opiniões e informações subjetivas de conjuntos de dados. Também chamada de mineração de opinião, é um ramo relativamente novo no mundo computacional que ganhou importância junto com o crescimento das redes sociais nos últimos anos.

Diariamente, usuários de redes sociais e de sites em geral fornecem uma enorme quantidade de dados, como pensamentos, opiniões e resenhas sobre assuntos que vão desde eventos sociais a avaliações de produtos Gonçalves *et al.* (2013), gerando conteúdo relevante sobre as preferências ou críticas do público sobre um produto ou serviço oferecido por uma empresa.

A análise de sentimentos consiste, basicamente, em interpretar o sentimento ou a impressão do autor no momento em que escreveu determinado texto. A forma mais simplificada desse estudo caracteriza-se pela definição da polaridade do sentimento existente no texto, classificando-o como positivo, negativo ou neutro.

Como Sharma *et al.* (2014) afirma, opiniões influenciam no comportamento humano. A leitura de uma resenha sobre um produto pode influenciar na decisão do comprador de adquiri-lo, se as avaliações forem boas, ou de tomar a decisão oposta, se as resenhas forem ruins. Da mesma forma, a leitura de boas avaliações sobre um filme pode levar mais espectadores ao cinema, assim como avaliações ruins causariam o efeito contrário.

Para um pessoa lendo um comentário, é fácil decidir se ele é bom ou ruim, identificando também ironias e ambiguidades. Porém, para uma máquina, esse processo não é natural.

1.1 Os problemas da classificação de textos

O uso de algoritmos computacionais para reconhecer padrões em linguagem escrita ou falada é chamado de processamento de linguagem natural. Um algoritmo pode aprender a reconhecer palavras de conotação positiva ou negativa, seja por métodos de treino ou pelo uso de um dicionário pré-definido.

Porém, a extração de informações subjetivas de textos apresenta alguns desafios que

podem afetar os resultados obtidos pelos classificadores. Um classificador tende a associar a palavra *"bom"* a um sentimento positivo, o que levaria uma resenha como *"Esse filme é tão bom que eu preferiria esquecê-lo"* a ser avaliada como boa, quando na verdade a palavra *"bom"* é usada para demonstrar ironia, pois o espectador não gostou do filme.

Outro ponto a ser considerado para a análise de sentimentos em textos é que nem todas as palavras expressam algum tipo de subjetividade e não necessariamente precisariam ser analisadas durante o processo. Pronomes, por exemplo, não tem conotação nem positiva e nem negativa, e costumam aparecer com frequência nas frases, de forma que o uso da frequência de palavras na classificação subjetiva deve ser feito com cuidado.

Além disso, pode-se optar por fazer uma análise mais profunda dos textos por meio do estudo das relações semânticas entre as palavras, considerando cada termo e seus vizinhos na frase, em vez de simplesmente estudar os termos separadamente. O valor semântico de uma palavra pode ser mais valioso para o resultado final, mas também não é tão simples de ser analisado e requer o uso de algoritmos mais robustos e custosos.

1.2 Motivação e objetivos

Em dezembro de 2014, o site *Kaggle*[1] lançou a competição *Bag of Words Meets Bags of Popcorn*[2]. O objetivo da competição era criar um algoritmo capaz de fazer a análise de sentimentos em resenhas de filmes coletadas do *IMDb*[3], usando processamento de linguagem natural e aprendizado de máquina para determinar a classificação de cada resenha entre positiva ou negativa.

A proposta do site era utilizar a ferramenta *Word2Vec*[4], originalmente criada por um time de pesquisa da *Google*, que foca no estudo do significado das palavras e nas suas relações semânticas dentro do texto para fazer a análise de sentimentos.

O objetivo deste trabalho é resolver o problema de classificação proposto nessa competição a partir da criação de uma função de *kernel* customizada para tratamento textual, utilizando aprendizado supervisionado com SVM e processamento de linguagem natural, utilizando linguagem *Python* e ferramentas como o *Scikit-Learn*, em vez do *Word2Vec*.

O *kernel* customizado proposto deve ser capaz de resolver outros problemas de análise de sentimentos, no entanto neste texto será descrito o experimento feito exclusivamente com *reviews* de filmes. A escolha da amostra de dados se deu por dois motivos principais: o primeiro é que a competição estava aberta no período em que o tema para a monografia foi decidido; o segundo é que o tema interessa tanto aos membros do grupo quanto ao professor orientador.

Ao final da implementação, o algoritmo desenvolvido foi testado quanto a sua eficiência

[1] http://www.kaggle.com
[2] http://www.kaggle.com/c/word2vec-nlp-tutorial
[3] http://www.imdb.com
[4] http://code.google.com/archive/p/word2vec

e acurácia e comparado ao método de classificação linear. Os capítulos seguintes descreverão os conceitos teóricos, métodos computacionais utilizados, detalhes de implementação e os resultados obtidos.

1.3 Organização do Trabalho

O capítulo 2 apresenta os conceitos teóricos que foram fundamentais para o entendimento do problema e para o desenvolvimento do algoritmo.

O capítulo 3 apresenta a proposta de solução para o problema: um *kernel* customizado.

O capítulo 4 dá os detalhes da implementação do algoritmo e do processo, e apresenta os resultados obtidos, comparando-os com os resultados do algoritmo de classificação linear.

O capítulo 5 abrange as conclusões sobre os testes e as sugestões de melhorias para o algoritmo em trabalhos futuros.

Por fim, o capítulo 6 apresenta as impressões subjetivas dos integrantes do grupo durante a realização deste trabalho ao longo do último ano.

Capítulo 2

Conceitos

Este capítulo apresenta um resumo de toda a teoria que foi estudada para a implementação do *kernel*. É importante salientar que existem diferentes métodos e abordagens para a resolução do problema estudado, porém apenas as metodologias e técnicas que foram incorporadas à solução proposta serão descritas.

2.1 Aprendizado Supervisionado

Existem inúmeros tipos de dados, que vão desde informações publicadas em redes sociais a códigos genéticos, e existem também diversas formas de lidar com esses dados, examinando-os, sumarizando-os ou mostrando-os nas mais variadas formas de visualização. É possível, inclusive, usar esses dados como fontes de experiências para melhorar a performance de algoritmos computacionais Garreta e Moncecchi (2013).

O aprendizado de máquina é a ciência de fazer computadores agirem sem serem explicitamente programados. Em poucas palavras, o aprendizado de máquina, ou *machine learning* é o estudo de algoritmos de reconhecimento de padrões que permitam ao computador aprender informações relevantes sobre um conjunto de dados e prever o comportamento de novos dados baseado no que aprendeu.

Suponha, por exemplo, que se queira fazer a previsão do clima do dia de amanhã, determinando se ele será um dia ensolarado ou chuvoso, e que para isso, esteja disponível uma base de dados climáticos com os eventos que ocorreram em todos os dias dos últimos cinco anos. Analisando essa base observou-se que se dois dias seguidos eram ensolarados, o terceiro também era. Um algoritmo de aprendizado de máquina seria capaz de generalizar esses fatos, aprendendo com os dados disponíveis e se tornando capaz de prever que o dia de amanhã será ensolarado se os dois anteriores também foram Garreta e Moncecchi (2013).

Na última década, o aprendizado de máquina nos deu carros autônomos, prático reconhecimento de diálogos, pesquisas eficientes na *Web* e um vasto aumento no entendimento do genoma humano Ng (2015).

O aprendizado de máquina é uma área abrangente dentro da ciência da computação,

podendo ser separada em algumas categorias, sendo uma delas o aprendizado supervisionado.

Nessa categoria, os algoritmos de aprendizado supervisionado usam como base um conjunto de dados ou instâncias, em que cada instância é representada por uma série de características (*features*) com um atributo importante em particular que pode ser definido como o atributo-alvo. Os algoritmos tentam construir um modelo com esse conjunto de dados iniciais e, a partir daí, usá-lo para prever o atributo-alvo de novas instâncias na amostra, conhecendo apenas as suas *features* Garreta e Moncecchi (2013).

Quando o atributo-alvo é um conjunto de valores discretos, como uma lista de espécies de flores, diz-se que o problema tratado pelo aprendizado supervisionado é um problema de classificação. Quando é um conjunto contínuo, como os números reais, trata-se de um problema de regressão.

Um conhecido exemplo, geralmente utilizado no ensino de aprendizado supervisionado, é a classificação de flores da espécie Íris. A ideia é encontrar o atributo-alvo que define cada flor, que no caso é a sua subespécie, tendo em mãos apenas a largura e o comprimento de suas pétalas e sépalas[1].

2.2 Etapas do Aprendizado Supervisionado

O primeiro passo na aplicação do aprendizado supervisionado é a divisão da amostra de dados original em dois conjuntos: um para treino e outro para avaliação (ou validação). O primeiro conjunto, como o próprio nome diz, servirá para treinar o algoritmo. A partir dele o programa irá construir o modelo de previsão, associando uma certa quantidade de características de uma instância a seu atributo-alvo ou classe. O segundo conjunto será usado para validar o modelo construído.

A seguir, vem a preparação dos dados que serão utilizados. Os dados são colhidos em sua forma bruta e raramente estão organizados como uma lista de *features*, prontas para serem lidas pelos algoritmos.

Existe então a necessidade de aplicar a esses dados métodos para transformar as instâncias brutas em vetores de *features* que sirvam de entrada para os programas. Essa transformação dos dados originais em um vetor de características é chamado de pré-processamento.

Além do pré-processamento, é importante selecionar as características relevantes ao estudo que se deseja fazer. Nem todas as características encontradas em determinada instância são úteis, e o tamanho do vetor de características é um fator que precisa ser considerado com cuidado. Um espaço inicial de *features* muito vasto pode tornar o desempenho do algoritmo custoso, podendo também dificultar a construção do modelo de previsão. Assim, é importante selecionar o vetor de *features* com atenção, preservando as caraterísticas que sirvam para ajudar na classificação, e dispensando as inúteis, diminuindo assim o tamanho da entrada. Esse processo pode ser definido como redução de dimensionalidade.

[1] http://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html

O próximo passo é usar os dados pré-processados do conjunto de treino para alimentar o programa, que irá então determinar o modelo que servirá para mapear características em classes.

Normalmente o conjunto de valores de entrada para o algoritmo é um conjunto de valores numéricos. Dessa forma, durante o pré-processamento, costuma-se transformar as instâncias originais, que podem ser textos, imagens etc, em listas de características numéricas.

Uma vez construído o modelo, avalia-se os resultados obtidos, geralmente calculando-se o número de acertos de classificação para uma dada amostra. Para validar o classificador, usa-se o segundo conjunto: a amostra de validação.

Nesse passo, basicamente, aplica-se o classificador encontrado ao conjunto de validação, e os resultados obtidos são confrontados com a classificação original dessa parcela da amostra que foi separada. Essa fase é chamada de validação cruzada ou *cross validation*.

A validação cruzada é importante para evitar o fenômeno chamado de *overfitting*, que pode ser explicado como um viés em relação à amostra de treino. Em outras palavras, se o classificador for testado somente na amostra de treino, ele pode separá-la com uma taxa de acerto alta, porém pode não conseguir fazer o mesmo quando submetido a uma amostra diferente.

O capítulo 4 mostrará como essas etapas foram desenvolvidas neste trabalho, usando o exemplo real da classificação de *reviews* de filmes.

2.3 Pré-processamento de Dados

Conforme explicado anteriormente, o pré-processamento é a etapa de preparação dos dados que servirão de entrada para os classificadores, e vai desde limpeza a definição do vetor de *features*. Os dados brutos, em sua maioria, não estão estruturados e precisam passar por um tratamento inicial para serem padronizados para um formato que uma função matemática ou algoritmo possa entender. Essa padronização geralmente consiste na transformação do conteúdo textual em uma entrada numérica que seja aceita pelo classificador.

O pré-processamento também é útil para reduzir o tamanho das amostras que serão utilizadas pelos algoritmos, removendo o que é chamado de ruído nos textos, ou seja, as partes que são irrelevantes para a análise, sejam elas pontuações, *tags* HTML ou até mesmo classes gramaticais de palavras que não ajudam a rotular o texto, resultando assim em um dado de entrada mais enxuto e direcionado ao tipo de resultado que se espera extrair dele.

A seguir, estão listadas as diferentes técnicas de pré-processamento que foram combinadas e adotadas para o desenvolvimento deste trabalho.

2.3.1 Tokenização e POS Tagging

Tokenização é o processo de fragmentar uma sequência textual em palavras ou elementos significativos, denominados *tokens*, para uso posterior em análise e mineração de dados.

Uma forma de aplicar a tokenização é por meio do método de *POS Tagging (Part-of-Speech Tagging)*. A técnica consiste em determinar se um termo é um adjetivo, verbo, substantivo, advérbio etc, considerando a gramática e o contexto da palavra no texto, analisando também as palavras adjacentes e outros termos a ela relacionados.

As versões mais simples de *POS Tagging* apenas separam as palavras segundo sua classificação gramatical. Versões aprimoradas consideram a semântica. Vale ressaltar que uma mesma palavra pode ser gramaticalmente classificada em um ou outro tipo, de acordo com a função dela na frase como um todo. Por exemplo, na frase *"O jantar está pronto"*, a palavra *"jantar"* é gramaticalmente definida como substantivo, enquanto que na frase *"Eu vou jantar em casa hoje"*, a mesma palavra assume a função de verbo.

Aplicando-se o método de *POS-Tagging* à segunda frase usada como exemplo no parágrafo anterior, obteria-se uma classificação que pode ser demonstrada na forma de um vetor chave: valor, onde cada palavra da sentença seria uma chave e cada classe gramatical seria seu respectivo valor. A classificação seria como no exemplo a seguir:

["Eu": pronome, "vou": verbo, "jantar": verbo, "em": preposição, "casa": substantivo, "hoje": advérbio]

2.3.2 Stemização

A stemização consiste na redução de palavras que sofreram derivação, ou que foram flexionadas, à suas respectivas raízes (do inglês, *stem*, ou tronco).

A base da palavra não necessariamente corresponde à sua raiz morfológica, e pode até não ser uma palavra válida quando olhada fora do contexto da stemização, porém basta que ela seja uma raiz comum à várias palavras que a ela podem ser mapeadas.

Por exemplo, se forem consideradas as palavras *"natural"*, *"naturalização"* e *"natureza"*, durante o processo de stemização, elas seriam reduzidas à raiz comum *"natur"*, que não é uma raiz gramatical válida.

Ao longo do capítulo de implementação serão mostrados exemplos da aplicação das técnicas descritas acima fazendo uso das próprias resenhas que foram utilizadas como base de dados para o trabalho, especificando também quais bibliotecas e funções dentre as tecnologias adotadas permitiram a aplicação dessas técnicas aos textos.

2.3.3 Bag of Words

O *Bag of Words* é um modelo que representa um texto na forma de um saco de palavras, ou seja, levando em conta apenas a frequência das palavras, sem se levar em consideração a ordem relativa entre elas ou a gramática. O modelo é frequentemente utilizado na classificação de documentos.

O exemplo a seguir foi retirado da página da Wikipédia (2) e mostra, de maneira simples, o resultado do algoritmo. Consideram-se duas frases distintas:

1. John likes to watch movies. Mary likes movies too.

2. John also likes to watch football games.

Com base nas frases, monta-se o vetor de características, que é basicamente uma lista de todas as palavras contidas em ambos os textos, sem repetição:

["John", "likes", "to", "watch", "movies", "also", "football", "games", "Mary", "too"]

Usando os índices da lista, cada frase é reescrita como um vetor de dez posições:

1. **[1, 2, 1, 1, 2, 0, 0, 0, 1, 1]**
2. **[1, 1, 1, 1, 0, 1, 1, 1, 0, 0]**

Cada entrada do vetor refere-se à quantidade de vezes que aquela palavra, da lista de características, aparece na frase em questão, formando um histograma. Por exemplo, no primeiro vetor, que corresponde ao primeiro texto, a palavra *"likes"* aparece duas vezes, portanto sua entrada tem o valor 2, enquanto que no segundo vetor ela aparece apenas uma vez, e sua entrada guarda o valor 1.

2.3.4 Tf-idf

Tf-idf, do inglês *Term frequency-inverse document frequency*, é uma medida para calcular a importância de uma palavra, também chamada de termo, em um documento, baseada na frequência em que ela aparece em todos os documentos do conjunto.

Se uma palavra aparece várias vezes no documento, ela é considerada importante, ganhando uma nota alta. Porém, se ela aparece em muitos documentos, ela não deve ser usada como identificador, recebendo uma nota baixa.

O cálculo do *Tf-idf* é feito em duas fases: a primeira define a frequência do termo *(Tf)* e a segunda, o inverso da frequência nos documentos *(idf)*. A frequência do termo é dada pela quantidade de vezes que o termo se repete no texto, dividido pelo tamanho do texto, normalizando o valor, de forma a considerar que alguns documentos são mais longos que outros.

$$TF(t) = \frac{Número\ de\ ocorrências\ do\ termo\ t\ no\ texto}{Total\ do\ número\ de\ termos\ no\ texto}$$

O inverso da frequência nos documentos é dado pelo logaritmo do total de textos no conjunto dividido pelo número de textos em que o termo ocorre. O peso de um termo é calculado então pela multiplicação desses dois valores.

$$IDF(t) = \ln\left(\frac{Total\ de\ textos}{Quantidade\ de\ textos\ em\ que\ o\ termo\ t\ aparece}\right)$$

O exemplo[2] a seguir mostra como o peso do tf-idf é definido:

Considere um documento contendo 100 palavras onde a palavra *"gato"* apareça três vezes. A frequência do termo *(TF)* é dada por $3 \div 100 = 0,03$. Agora, assuma que temos 10 milhões de documentos e que a palavra gato aparece em 1000 desses documentos. O inverso da frequência nos documentos *(IDF)* é calculado como $\ln(10000000 \div 1000) = 4$. Logo, o peso *Tf-idf* é o produto desses valores: $0,03 \times 4 = 0,12$.

2.4 Máquinas de Vetores de Suporte

Se cada instância da amostra de dados for considerada como um ponto em um espaço multidimensional, pode-se dizer que o modelo construído por meio do classificador é uma superfície, ou hiperplano, que separa as instâncias (pontos) do resto do grupo.

As máquinas de vetores de suporte *(SVM - Support Vector Machines)* é um método de aprendizado supervisionado que busca encontrar hiperplanos de separação de modo otimizado, selecionando aqueles que passam pelos maiores espaços (*gaps*) entre as instâncias de classes diferentes. Cada nova instância que for estudada será classificada de acordo com o lado do hiperplano a qual ela pertence Garreta e Moncecchi (2013).

O hiperplano pode ser definido como uma função de mapeamento $y = f(x)$, entre um vetor de *features* multidimensional x e um valor (ou conjunto de valores) y. Para efetuar o mapeamento, a única informação disponível é o conjunto de treino, dado por $D = (x_i, y_i) \in X \times Y$, $i = 1, ..., l$, com l sendo o número de pares (x_i, y_i) da amostra e d_i é o valor almejado, ou atributo-alvo Kecman (2005).

Em outras palavras, denota-se o conjunto de treino D como uma amostra rotulada - cada instância da amostra tem um determinado número de características (vetor x) e um rótulo, ou classe, associado (valor y). O modelo construído pelo classificador deve ser capaz de dar um rótulo às instâncias de uma amostra que não foi pré-rotulada.

A figura abaixo ilustra a definição de um hiperplano ótimo para classificar um conjunto de dados. A amostra está num espaço bidimensional, apresenta apenas duas características X_1 e X_2 e pode ser dividida em duas classes (preta e branca):

É possível ver que o hiperplano H_1 não consegue separar corretamente as duas classes, levando a erros de classificação. O hiperplano H_2 separa as classes de forma correta, assim

[2] http://www.tfidf.com

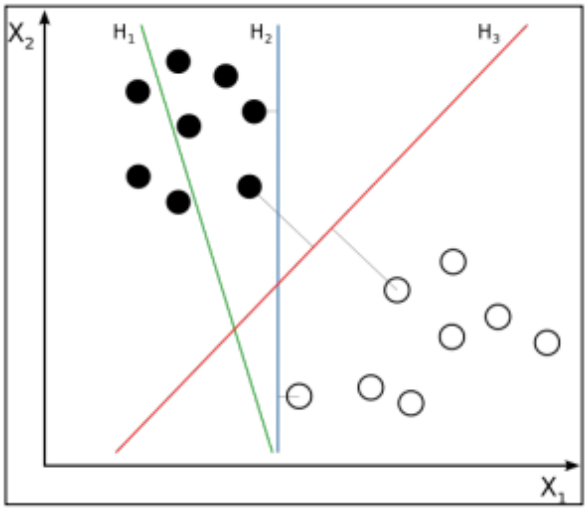

Figura 2.1: *Hiperplano de maior margem*

como H_3, porém o hiperplano H_3 apresenta a maior margem contada a partir da instância que está mais próxima do classificador.

A margem é definida pelos vetores perpendiculares que podem ser vistos saindo do hiperplano H_3 em direção às instâncias mais próximas. Esses vetores são os chamados vetores de suporte e dão nome ao método.

A escolha do hiperplano H_3 como classificador diminui a probabilidade de erros de generalização por classificações errôneas, evitando o problema de *overfitting*.

O exemplo acima demonstra o uso do SVM num espaço bidimensional, no entanto ele pode ser aplicado a espaços de várias ou até infinitas dimensões, quando considerados conjuntos que apresentam um grande número de *features* para serem avaliadas. O SVM também pode ser usado em casos onde as características formam um espaço grande, porém esparso. Por sua adaptabilidade a espaços de diferentes dimensões e por ser um método eficiente, o SVM tem sido aplicado em inúmeras áreas de estudo.

Por exemplo, em Cai *et al.* (2004) o SVM foi usado na área de biologia, para fazer previsões sobre o tipo de membranas proteicas. Na área de medicina pode-se ver a aplicação de SVM no diagnóstico de câncer, também por meio da análise de proteínas, como descrito em Chu *et al.* (2005). Já em Baccarini *et al.* (2011) é mostrado o uso do aprendizado supervisionado com SVM para fazer o diagnóstico de falhas mecânicas em motores e máquinas e, como citado em Osuna *et al.* (1997), o método também pode ser útil para programas de reconhecimento facial. Outra aplicação interessante de SVM é dirigida à área de análise de sentimentos, que será abordada nessa monografia.

Capítulo 3

Kernel

Neste capítulo será apresentada uma proposta de solução para o problema de classificação subjetiva de resenhas de filmes na forma de um *string kernel* customizado. O *kernel* desenvolvido foi idealizado pelo professor Marco Dimas Gubitoso durante uma das reuniões realizadas ao longo do ano.

3.1 Definição de Kernel

Kernel é uma função de similaridade que é utilizada em algoritmos de aprendizado de máquina para descrever uma regra entre as entradas. Ela recebe duas entradas e retorna um valor que representa o quão similar elas são[1].

Kernels funcionam devido a dois motivos:

1. Muitos algoritmos de aprendizagem computacional podem ser expressos inteiramente em termos de produtos internos.

2. Em condições especifícas, toda função de *kernel* pode ser expressa como um produto interno de algum espaço[2]

Funções de *kernel* são particularmente úteis quando os dados não estão no formato aceito pelo classificador, geralmente numérico. Nesses casos existem duas possibilidades: Representar os dados no formato numérico (nem sempre é trivial); Utilizar uma *matriz de kernel*, também chamada de *matriz de Gramian*, para que ela sirva de produto interno para o algoritmo de classificação.

Além disso, funções de *kernel* permitem que os algoritmos de classificação trabalhem com um espaço de *features* de dimensão muito alta de forma implícita, sem a necessidade de calcular as coordenadas de cada registro nesse espaço.

A figura abaixo ilustra o uso de *kernels* em *SVM*. A função de *kernel*, dada por $K(x, z) = < \varphi(x), \varphi(z) >$, calcula produtos internos para cada par de instâncias (x, z) do espaço de

[1] http://www.quora.com/What-are-Kernels-in-Machine-Learning-and-SVM
[2] http://en.wikipedia.org/wiki/Mercer's_theorem

features e armazena-os na matriz de *kernel*. A matriz é então passada como entrada do algoritmo de classificação do *SVM* que analisa os padrões e cria a função classificadora.

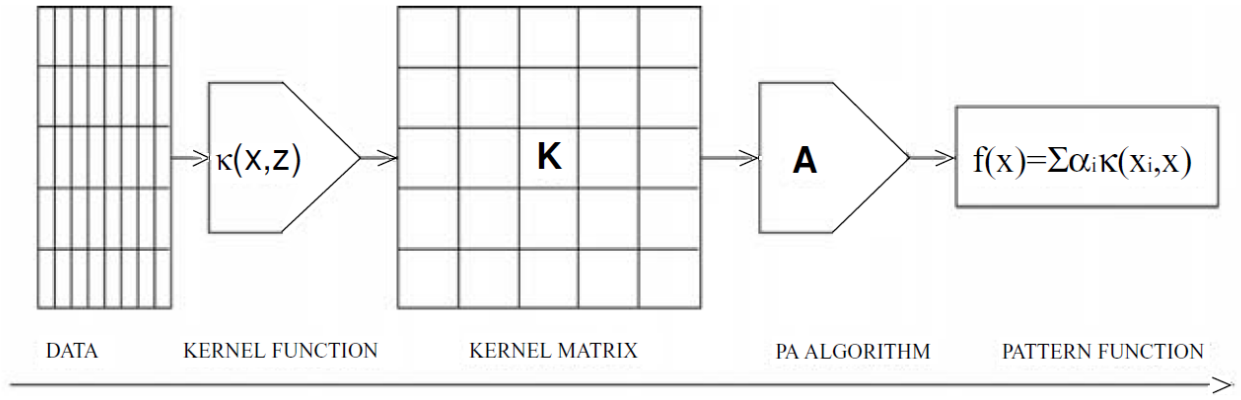

Figura 3.1: *Estágios na Aplicação de Métodos de Kernel; fonte: Shawe-Taylor e Cristianini (2004)*

3.2 Kernel Proposto

O *kernel* customizado foi idealizado inicialmente com uma ideia abstrata, cuja premissa era considerar os *reviews* não apenas estudando as palavras separadamente, mas também os termos vizinhos, com o intuito de misturar as técnicas de clusterização e cálculo de distâncias entre palavras em um mesmo texto.

Para isso, monta-se uma matriz para cada par de *reviews* (x_1, x_2), cujas linhas representam as palavras de x_1, e as colunas representam as palavras de x_2. A ordem das palavras de cada *review* é mantida para garantir que expressões e sentenças comuns aos *reviews* sejam consideradas.

A matriz serve para o cálculo de um peso, um grau de similaridade entre o par de *reviews*. Palavras que aparecem seguidas em ambas as instâncias ganham um peso maior, significando que existe uma expressão que é repetida nas duas frases. Palavras que aparecem em ambos os *reviews*, porém sem nenhuma palavra vizinha (à direita ou à esquerda) em comum, recebem apenas peso 1. Palavras que aparecem em apenas uma das instâncias recebem peso 0.

Antes de montar a matriz, o vetor de *features* originais é redimensionado. Como mencionado anteriormente, reduzir o número de termos a serem estudados ao considerar apenas os que têm relevância para a análise pode ser um abordagem útil para diminuir o custo computacional e direcionar a amostra para o tipo de resultado que se espera obter. Dessa forma, os termos mais interessantes de cada *review* seriam aqueles que expressam alguma opinião ou sentimento, e a montagem da matriz seria feita utilizando-se apenas os adjetivos de cada instância.

Porém, como mostra Benamara *et al.* (2007), são obtidos melhores resultados de classificação quando considerados adjetivos juntamente com advérbios, uma vez que os advérbios

servem como intensificadores dos adjetivos. Assim, considerando o exemplo do seguinte par de *reviews*:

1. ***"The movie is interesting and the plot is very good. The characters are funny and very adorable."***
2. ***"The story is different and the movie is very good and funny. There are great and adorable characters acting."***

Seriam filtrados adjetivos e advérbios, sempre mantendo a ordem em que eles aparecem nas frases, preservando expressões como *"very good"* e a intensificação que o advérbio proporciona ao adjetivo:

1. ***"interesting very good funny very adorable"***
2. ***"different very good funy great adorable"***

A tabela abaixo mostra um rascunho de como, idealmente, ficaria a matriz calculada para esse par de *reviews*:

$K_{ij}(x_1(i), x_2(j))$	*interesting*	*very*	*good*	*funny*	*very*	*adorable*
different	0	0	0	0	0	0
very	0	8	8	0	0	0
good	0	8	10	8	0	0
funny	0	0	8	8	0	0
great	0	0	0	0	0	0
adorable	0	0	0	0	0	1

Tabela 3.1: *Rascunho da matriz da função kernel não normalizada*

A tabela acima é preenchida de forma que cada posição da matriz $K(x_1, x_2)$ representa o peso de uma palavra dentro das duas resenhas. Cada posição (i, j) da matriz, quando não zerada, indica que a palavra pertence a uma das duas classes:

1. A palavra $x_1(i)$ ocorre na posição i de x_1 e na posição j de x_2, ou seja $x_1(i) = x_2(j)$.
2. A palavra $x_1(i)$ ocorre na posição i de x_1 e em uma posição vizinha a j ($j-1$ ou $j+1$).

O peso de cada posição (i, j) aumenta cada vez que uma palavra vizinha é descoberta, criando um *cluster* em volta da palavra observada.

Para ilustrar melhor a ideia do *cluster* pode-se observar a palavra *"good"*. A entrada $K_{ij}(\text{"}good\text{"}, \text{"}good\text{"})$ tem um peso maior do que a entrada $K_{ij}(\text{"}very\text{"}, \text{"}good\text{"})$ e que a entrada $K_{ij}(\text{"}good\text{"}, \text{"}funny\text{"})$. Esse fato se justifica por *"good"* estar entre os termos *"very"*

e *"funny"* em ambos os *reviews*, enquanto os outros dois termos tem apenas um vizinho comum *("good")*. Os valores usados para mostrar qual palavra tem maior peso dentro de cada expressão encontrada são apenas ilustrativos, o conceito importante é apenas que $10 > 8 > 1$.

A partir daí, o peso da matriz seria totalizado (uma ideia era somar todas as entradas da matriz) e depois o valor seria normalizado. O peso da similaridade seria calculado para todos os pares de instâncias da amostra e seus valores finais seriam armazenados em uma matriz de *kernel*, onde cada entrada representaria o grau de similaridade normalizado de um par de *reviews*.

É importante salientar que o modelo descrito acima é apenas um rascunho de como um *kernel* poderia ser construído tendo como base a presença de expressões comuns. Uma vez definida a idéia do *kernel*, pesquisou-se sobre estudos relacionados a *string kernels*.

String kernels, como o nome diz, são *kernels* especializados em calcular a similaridade a partir de entradas textuais. Um dos *kernels* já existentes que foi encontrado apresentou um princípio muito semelhante ao resultado final buscado pelo *kernel* customizado. Esse modelo foi usado como base da implementação do *kernel* idealizado.

3.3 Gap-Weighted Subsequences Kernel

O algoritmo aqui descrito, citado em Shawe-Taylor e Cristianini (2004) como *Gap-Weighted Subsequences Kernel*, e em Lodhi *et al.* (2002) como *String Subsequence Kernel*, é um *string kernel* com foco em determinar a distância que separa duas subsequência e calcular, a partir daí, um peso para elas.

Em outras palavras, o *kernel* considera que o grau da presença de uma subsequência é uma função dada por quantos espaços existem dentro delas. O tamanho das subsequências é considerado fixo para o algoritmo.

A ideia principal por trás do *Gap-Weighted Subsequences Kernel* é a comparação de sequências de palavras pelas subsequências que elas contêm - quanto mais subsequências em comum, mais similares elas são - mas em vez de pesar todas as ocorrências igualmente, o grau de contiguidade das subsequências em uma *string* de entrada s determina quanto ela irá contribuir para a comparação Shawe-Taylor e Cristianini (2004).

Como exemplo, pode-se considerar a subsequência *"gon"*. Ela aparece nas palavras *"gone"*, *"going"* e *"galleon"*, porém em *"gone"* o peso da subsequência analisada é maior por não haver espaços (ou outros caracteres) separando as três letras buscadas. Enquanto isso, *"galleon"* tem o menor peso dentre os exemplos, pois entre *"g"* e *"on"* existem outras quatro letras.

Por meio desse exemplo, dois pontos importantes podem ser visualizados:

1. O *kernel* considera como subsequências as chamadas *substrings*. Ele trabalha com subsequências de caracteres dentro de palavras ou frases completas e não com sequências

de palavras (o intuito do algoritmo não é definir uma subsequência suficientemente grande para abranger uma ou mais palavras)

2. Os espaços (*gaps*) considerados pelo *kernel* se referem à distância entre as letras da subsequência, aos outros caracteres que diferem dos pesquisados, e não necessariamente a espaços vagos em si.

Para lidar com os espaços entre as letras de uma subsequência, define-se um fator $\lambda \in (0, 1)$ (fator de decaimento), usado para balancear o peso da *feature* na sequência avaliada. Assim, considerando-se uma subsequência u de uma *string* s, e sendo i o índice da subsequência na *string* com $u = s(i)$, denota-se $l(i)$ como o tamanho da *string* em s que contém u. O peso da ocorrência de u é calculado por $\lambda^{l(i)}$.

O vetor de *features* associado ao *kernel* para subsequências de tamanho p é $I = \Sigma^p$, onde Σ é o alfabeto, e a função que calcula o peso de cada subsequência u em uma palavra s é dada por:

$$\phi_u^p(s) = \sum_{i:u=s(i)} \lambda^{l(i)}, u \in \Sigma^p$$

A função de *kernel*, para duas palavras s e t, é então dada por:

$$K_p(s,t) = \langle \phi^p(s), \phi^p(t) \rangle = \sum_{u \in \Sigma^p} \phi_u^p(s) \phi_u^p(t)$$

O exemplo a seguir mostra como ficaria um *kernel* não normalizado para as palavras *"cat"*, *"car"*, *"bat"* e *"bar"*, fixando o tamanho das subsequências em 2.

Tabela 3.2: *Matriz de uma função de kernel não normalizada*

ϕ	ca	ct	at	ba	bt	cr	ar	br
cat	λ^2	λ^3	λ^2	0	0	0	0	0
car	λ^2	0	0	0	0	λ^3	λ^2	0
bat	0	0	λ^2	λ^2	λ^3	0	0	0
bar	0	0	0	λ^2	0	0	λ^2	λ^3

Assim, o valor do *kernel* não normalizado para *"cat"* e *"car"* é $K(\text{"}cat\text{"}, \text{"}car\text{"}) = \lambda^4$.

O valor normalizado é $\hat{K}(\text{"}cat\text{"}, \text{"}car\text{"}) = \frac{\lambda^4}{(2\lambda^4+\lambda^6)} = (2+\lambda^2)^{-1}$, onde $K(\text{"}cat\text{"}, \text{"}cat\text{"}) = K(\text{"}car\text{"}, \text{"}car\text{"}) = 2\lambda^4 + \lambda^6$.

3.3.1 Gap-Weighted Subsequences Kernel versão com Programação Dinâmica

Em Shawe-Taylor e Cristianini (2004) são apresentadas 3 formas de implementação do *Gap-Weighted Subsequences Kernel*, uma ingênua (bem ineficiente), uma utilizando programação dinâmica e outra com árvore de prefixos.

Para o propósito desse estudo foi escolhido como base a versão por programação dinâmica, por ser uma técnica vista durante o curso de graduação, mesmo sendo menos eficiente que a implementação com a árvore de prefixos. Assim detalhes sobre a implementação ingênua foram omitidos.

A recursão ingênua do *Gap-Weighted Subsequences Kernel* é dada por:

$$\begin{aligned} K_p^S(sa, tb) &= \sum_{(i,j)\in I_p^{|s|+1}\times I_p^{|t|+1}:sa(i)=tb(j)} \lambda^{l(i)+l(j)} \\ &= [a=b]\sum_{i=1}^{|s|}\sum_{j=1}^{|t|}\lambda^{2+|s|-i+|t|-j} \sum_{(i,j)\in I_{p-1}^{i}\times I_{p-1}^{j}:s(i)=t(j)} \lambda^{l(i)+l(j)} \\ &= [a=b]\sum_{i=1}^{|s|}\sum_{j=1}^{|t|}\lambda^{2+|s|-i+|t|-j}\ K_{p-1}^S(s(1:i), t(1:j)) \end{aligned}$$

A partir dessa recorrência é calculada uma matriz auxiliar DP_p onde os valores são dados por:

$$DP_p(k,l) = \sum_{i=1}^{k}\sum_{j=1}^{l}\lambda^{k-i+l-j}k_{p-1}^S(s(1:i), t(1:j))$$

Por essa matriz auxiliar o kernel pode então ser definido como:

$$K_p^S(sa, tb) = \begin{cases} \lambda^2 DP_p(|s|,|t|) & \text{se } a=b \\ 0 & \text{caso contrário} \end{cases}$$

O valor de $DP_p(k,l)$ pode então ser calculado recursivamente em termos de $DP_p(k-1,l)$, $DP_p(k,l-1)$ e $DP_p(k-1,l-1)$:

$$\begin{aligned} DP_p(k,l) &= \sum_{i=1}^{k}\sum_{j=1}^{l} \lambda^{k-i+l-j} k_{p-1}^{S}(s(1:i), t(1:j)) \\ &= k_{p-1}^{S}(s(1:i)t(1:j) + \lambda DP_p(k, l-1) \\ &+ \lambda DP_p(k-1, l) - \lambda^2 DP_p(k-1, l-1) \end{aligned}$$

3.3.2 Corretude da Recursão e Complexidade

A corretude da recursão é dada em vista que a soma da equação $DP_p(k,l)$ pode ser dividida em 4 grupos:

- Quando $(i,j) = (k,l)$, grupo 1 (primeiro termo);
- Quando $i = k$ e $j < l$, grupo 2 (segundo termo com o peso correto);
- Quando $j = l$ e $i < k$, grupo 3 (terceiro termo com o peso correto);
- Quando $i < k$ e $j < l$, grupo 4 (segundo, terceiro e quarto termo com o peso invertido)

Levando a correta inclusão na soma total.

A complexidade da computação necessária para calcular a matrix DP_p para um único valor de p é $O(|t||s|)$, assim como a complexidade de calcular K_p^S de DP_p, fazendo a complexidade todal de se calcular o kernel $K_p(s,t)$ igual a $O(p|t||s|)$.

3.4 Adaptação para o Kernel Customizado

O *kernel* customizado foi criado como uma variação do *Gap-Weighted Subsequences Kernel*. A grande diferença entre eles é que o *kernel* customizado procura por subsequências de palavras (expressões), enquanto o *Gap-Weighted Subsequences Kernel* procura por subsequências de caracteres.

Assim, foi necessária uma adaptação do algoritmo original para a versão customizada com as seguintes modificações:

1. Cada subsequência pesquisada pelo algoritmo seria formada por um conjunto de palavras, mantendo sua ordem de aparição no texto original, e não apenas por um conjunto de caracteres;
2. O tamanho p da subsequência seria usado para denotar o tamanho da expressão comum procurada, ou seja, a quantidade de palavras sequenciais no *review*, de forma a obter os *clusters* mostrados anteriormente;

3. Os espaços (*gaps*) representariam a quantidade de palavras diferentes presentes entre os termos da expressão procurada;

4. Somente seriam considerados adjetivos e advérbios para o vetor de *features*. No *Gap-Weighted Subsequences Kernel* não havia essa necessidade, pois ele busca padrões de similaridade mais generalizados, enquanto que o *kernel* customizado é voltado para a análise de sentimentos.

3.5 Kernel Linear

Por fim, a apresentação do *kernel* linear se faz necessária por ser ele o modelo usado na comparação de desempenho com o *kernel* customizado.

O *kernel* linear é a função de *kernel* mais simples, definida por um produto interno $< x, y >$ somado a uma constante de otimização [3], onde x e y são as instâncias a serem avaliadas pela similaridade

A função de *kernel* pode ser formalizada como:

$$K(x, y) = x^T y + c$$

Assim, quando aplicado ao *SVM*, o *kernel* linear cria um hiperplano de separação dos dados que é basicamente uma linha reta, como na figura a seguir.

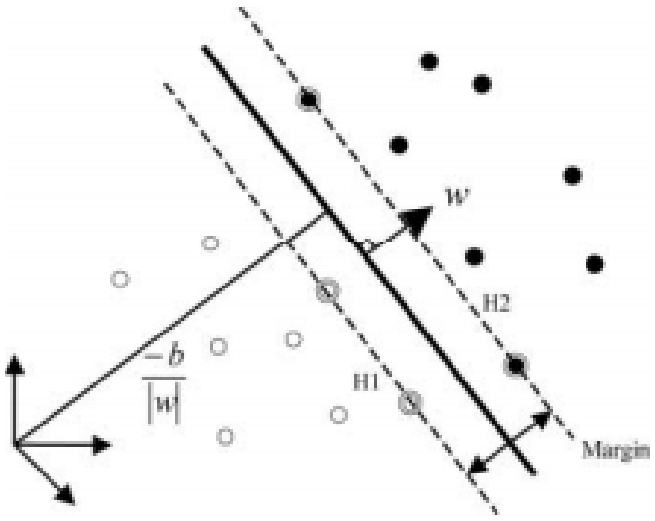

Figura 3.2: *Hiperplano de separação; fonte: Byun e Lee (2002)*

O problema de classificação é determinar o hiperplano que separa as instâncias da amostra de forma que $K(x, y) \geq 1$, para instâncias positivas, e $K(x, y) \leq -1$, para as negativas.

[3] http://crsouza.com/2010/03/kernel-functions-for-machine-learning-applications/#linear

O modelo linear foi escolhido como comparativo por dois principais motivos:

- Por ter apresentado os melhores resultados, quando comparados com outros *kernels* como o polinomial;
- Por ser um modelo mais simples de *kernel*, não especificamente direcionado a *strings* ou análise de sentimentos. Era interessante, do ponto de vista experimental, comparar um *string kernel* complexo, como o criado, com o modelo linear para determinar qual deles tinha o melhor custo-benefício computacional.

Capítulo 4

Implementação Gerais e Resultados

4.1 Tecnologias Utilizadas

4.1.1 Linguagem Python

A linguagem Python[1] é de alto nível, com ênfase em manter o código legível e fácil de entender, mesmo expressando programas em poucas linhas de código quando comparados a programas em outras linguagens.

Foi idealizada e inicialmente implementada por Guido van Rossum, na década de 80. Hoje, sua licença é administrada pela Python Software Foundation, que permite seu uso inclusive para fins comerciais, e seu desenvolvimento é mantido por uma comunidade, uma vez que a linguagem é *open-source*.

Python é uma linguagem de tipagem dinâmica, projetada para atender a diferentes paradigmas de programação como o imperativo, o orientado a objetos e o funcional. Seu interpretador pode ser instalado em diferentes sistemas operacionais para a execução dos programas escritos.

É uma linguagem rica em bibliotecas próprias e também conta com diversas bibliotecas, módulos e *frameworks* desenvolvidos por terceiros, estendendo sua aplicação a diversas áreas, como desenvolvimento *Web*, análises científicas e cálculos numéricos, desenvolvimento de jogos etc.

Entre suas inúmeras bibliotecas e ferramentas, existem várias que são voltadas para a mineração de dados, a aprendizagem de máquina e o processamento da linguagem natural. Isso fez com que a linguagem *Python* fosse escolhida para o desenvolvimento deste trabalho.

4.1.2 Pandas

Pandas[2] é uma biblioteca, escrita em linguagem *Python*, para análise de dados em alta performance. Utiliza *NumPy*, um pacote padrão do *Python* para computar cálculos científi-

[1] http://www.python.org
[2] http://pandas.pydata.org

cos.

A biblioteca *Pandas* fornece estrutura de dados novas e ferramentas para sua manipulação, provendo maior facilidade na execução das análises desejadas.

Aqui, utilizou-se uma dessas estruturas, chamada de *data-frame*. Um *data-frame* é, uma estrutura tabular com indexação integrada, ou seja, basicamente é uma estrutura com linhas e colunas onde cada coluna tem um índice, associado a um conjunto de valores. Cada linha, portanto, tem vários valores, um deles referente a cada coluna indexada do *data-frame*.

Abaixo segue um exemplo Reda (2013) ilustrativo da estrutura de um *data-frame*.

```
1 data = {'year': [2010, 2011, 2012, 2011, 2012, 2010, 2011, 2012],
2         'team': ['Bears', 'Bears', 'Bears', 'Packers', 'Packers', 'Lions',
3         'Lions', 'Lions'],
4         'wins': [11, 8, 10, 15, 11, 6, 10, 4],
5         'losses': [5, 8, 6, 1, 5, 10, 6, 12]}
6 football = pd.DataFrame(data, columns=['year', 'team', 'wins', 'losses'])
7 football
```

Id	*Year*	*Team*	*Win*	*Losses*
0	2010	Bears	11	5
1	2011	Bears	8	8
2	2012	Bears	10	6
3	2011	Packers	15	1
4	2012	Packers	11	5
5	2010	Lions	6	10
6	2011	Lions	10	6
7	2012	Lions	4	12

A biblioteca *Pandas* foi importante para a leitura e organização dos dados brutos da resenha, indexando cada uma delas pelo seu rótulo, e transformando o conjunto inicial em algo mais fácil de ser manipulado por outras funções.

4.1.3 Natural Language Toolkit

O *Natural Language Toolkit*[3] *(NLTK)* é uma coleção de bibliotecas para processamento de linguagem natural em *Python*, que contém interfaces para outras bibliotecas também utilizadas em análise de linguagem, como o *WordNet*.

O *NLTK* dispõe de recursos para classificação, tokenização, *tagging*, *parsing*, stemização, lematização (*lemmatization*), análise semântica, dentre outros.

Aqui, o *NLTK* foi empregado na tokenização das palavras que foram consideradas relevantes nas resenhas, mais especificamente no *POS Tagging* e na stemização das palavras que receberam as *tags*.

[3]http://www.nltk.org

4.1.4 Beautiful Soup

Beautiful Soup [4] é uma biblioteca dedicada ao *parsing* de aquivos em formato HTML ou XML. Em resumo, a biblioteca trata o documento como se fosse uma árvore (*parse tree*), definida pelas *tags*, permitindo assim que o usuário navegue facilmente pelo documento, extraindo as informações necessárias.

Neste trabalho, a biblioteca *Beautiful Soup* foi usada para remover as *tags* HTML das resenhas brutas, uma vez que essas vinham de páginas *Web*.

4.1.5 Scikit-Learn

O *Scikit-Learn*[5] é um conjunto de ferramentas para mineração e análise de dados, constituindo uma mistura de pacotes como *NumPy*, *SciPy* e *matplotlib*.

Os principais recursos do *Scikit-Learn* são algoritmos para:

- **Clusterização**: processo de agrupamento de objetos com características similares
- **Regressão**: para predição de atributos de valores contínuos para objetos a eles associados
- **Seleção de modelos**: módulos para comparação e validação de parâmetros e modelos, tais como a validação cruzada, que será citada mais a frente
- **Redução de dimensionamento**: ou diminuição do número de variáveis a serem consideradas para estudo
- **Pré-processamento**: preparação dos dados, extração do vetor de características e normalização
- **Classificação**: métodos de atribuição de um dado a um conjunto específico, tais como o *SVM*, que aqui será o foco na elaboração do estudo

4.2 Escolha dos dados

O conjunto de dados utilizados nos experimentos foi disponibilizados no site do*Kaggle* para a competição que inspirou esse trabalho, e foram obtidos da base de dados do *IMDb*. A amostra consiste de 25 mil resenhas de filmes, previamente rotuladas como positiva ou negativa.

Conforme descrito na página da cometição[6], o rótulo foi aplicado de acordo com a seguinte regra:

[4] http://www.crummy.com/software/BeautifulSoup
[5] http://scikit-learn.org
[6] http://www.kaggle.com/c/word2vec-nlp-tutorial/details/part-2-word-vectors

- 1 se a resenha está associada a uma nota maior igual a 6;
- 0 se a resenha está associada a uma nota menor que 5;

As notas no intervalo $5 \leq n < 6$ foram consideradas neutras e não fazem parte da amostra.

Os dados estão disponíveis no arquivo *labeledTrainData.tsv*. Cada registro é composto de três elementos:

- **id**: identificador da resenha;
- **sentiment**: rótulo, 0 para negativo e 1 para positivo;
- **review**: texto da resenha, uma ou mais orações contendo a opinião do usuário sobre determinado filme não identificado.

A linha abaixo foi retirada do conjunto de dados como exemplo de visualização da estrutura inicial de uma resenha.

"2951_8" 1 ***"Yeah, it\'s a chick flick and it moves kinda slow, but it\'s actually pretty good - and I consider myself a manly man. You gotta love Judy Davis, no matter what she\'s in, and the girl who plays her daughter gives a natural, convincing performance.

The scenery of the small, coastal summer spot is beautiful and plays well with the major theme of the movie. The unknown (at least unknown to me) actors and actresses lend a realism to the movie that draws you in and keeps your attention. Overall, I give it an 8/10. Go see it."***

4.3 Pré-processamento - Limpeza

Após a escolha dos tipos de dados a serem estudados, foi necessário fazer uma limpeza para remover ruídos, tokenizar as resenhas e selecionar apenas as palavras que agregam conteúdo diferencial ao texto.

A função *review_to_features* foi implementada para que dada uma resenha bruta, a saída gere um conjunto apenas com as palavras que colaboram para a dicotomia dos dados. A rotina da função consiste em:

- Remover tags HTML - utilizando a bilbioteca *Beautiful Soup*;

"Yeah, it\'s a chick flick and it moves kinda slow, but it \'s actually pretty good - and I consider myself a manly man. You gotta love Judy Davis, no matter what she \'s in, and the girl who plays her daughter gives a natural, convincing performance.The scenery of the small, coastal summer spot is beautiful and plays

well with the major theme of the movie. The unknown (at least unknown to me) actors and actresses lend a realism to the movie that draws you in and keeps your attention. Overall, I give it an 8/10. Go see it."

- Remover caracteres considerados inválidos - não alfanuméricos - utilizando a biblioteca *re* de expressões regulares;

"Yeah it's a chick flick and it moves kinda slow but it's actually pretty good and I consider myself a manly man You gotta love Judy Davis no matter what she's in and the girl who plays her daughter gives a natural convincing performance The scenery of the small coastal summer spot is beautiful and plays well with the major theme of the movie The unknown at least unknown to me actors and actresses lend a realism to the movie that draws you in and keeps your attention Overall I give it an 8 10 Go see it"

- Filtrar apenas advérbios e adjetivos - com a classificação da função gramatical da palavra gerada utilizando o método *pos_ tagging* de *(Part-of-Speech Tagging)* da biblioteca *NLTK*;

[(Yeah, UH), (it's, VB), (a, DT), (chick, NN), (flick, NN), (and, CC), (it, PRP), (moves, VBZ), (kinda, JJ), (slow, JJ), (but, CC), (it's, JJ), (actually, RB), (pretty, RB), (good, JJ), (and, CC), (I, PRP), (consider, VBP), (myself, PRP), (a, DT), (manly, RB), (man, NN), (Yo, PRP), (gotta, VBP), (love, VB), (Judy, NNP), (Davis, NNP), (no, DT), (matter, NN), (what, WP), (she's, VBZ), (in, IN), (and, CC), (the, DT), (girl, NN), (who, WP), (plays, VBZ), (her, PRP$), (daughter, NN), (gives, VBZ), (a, DT), (natural, JJ), (convincing, NN), (performance, NN), (The, DT), (scenery, NN), (of, IN), (the, DT), (small, JJ), (coastal, JJ), (summer, NN), (spot, NN), (is, VBZ), (beautiful, JJ), (and, CC), (plays, VBZ), (well, RB), (with, IN), (the, DT), (major, JJ), (theme, NN), (of, IN), (the, DT), (movie, NN), (The, DT), (unknown, JJ), (at, IN), (least, JJS), (unknown, JJ), (to, TO), (me, PRP), (actors, NNS), (and, CC), (actresses, NNS), (lend, VBP), (a, DT), (realism, NN), (to, TO), (the, DT), (movie, NN), (that, IN), (draws, VBZ), (yo, PRP), (in, IN), (and, CC), (keeps, VB), (your, PRP$), (attention, NN), (Overall, IN), (I, PRP), (give, VBP), (it, PRP), (an, DT), (8, CD), (10, CD), (Go, NNP), (see, VBP), (it, PRP)]

As *tags* mantidas foram:

- **JJ**: adjective or numeral, ordinal;

- **JJR**: adjective, comparative;
- **JSS**: adjective, superlative;
- **RB**: adverb;
- **RBR**: adverb, comparative;
- **RBS**: adverb, superlative

[kinda, slow, it's, actually, pretty, good, manly, natural, small, coastal, beautiful, well, major, unknown, least, unknown]

- Aplicar uma técnica de stemização - por meio do método *SnowballStemmer*, também do *NLTK*;

[kinda, slow, it, actual, pretti, good, man, natur, small, coastal, beauti, well, major, unknown, least, unknown]

A primeira versão da implementação contava com a remoção de *stop-words*, também chamadas de palavras vazias, como passo final e seguinte ao de remover caracteres inválidos. Esta versão preserva a maior parte do texto original, o que, no que diz respeito em especial para o *kernel* customizado, tornou o cálculo da medida de similaridade entre as resenhas tão custoso que chegou a ser impossível, mesmo com apenas um subconjunto pequeno dos dados, gerar a entrada para o algoritmo do *SVM* com *kernel* pré-computado.

Foi preciso buscar melhores formas de efetuar a limpeza de dados, restringindo ainda mais a quantidade de palavras que sobrariam em cada resenha, porém sem perder muita informação. Durante as pesquisas que se seguiram, por várias vezes notou-se o uso de classificação gramatical das palavras para análises textuais. Optou-se pelo uso de adjetivos e advérbios, conforme comentado no capítulo 3, chegando-se à versão do trabalho que usa *POS Tagging* e *SnowballStemmer*.

4.4 Método de Validação

O modelo de validação cruzada *(cross validation)* foi utilizado para verificar como o novo kernel se compara com métodos tradicionais no quesito generalização.

Os dados foram separados aleatoriamente em dois conjuntos, um para treino e um para teste, com a função *train_test_split* do módulo *cross_validation* do *Scikit-Learn*. Optou-se pelo uso de 60% dos dados para treino e os 40% restantes para os testes com o classificador.

Essa validação foi repetida apenas 5 vezes, devido a falta de poder computacional, e com os resultados foi calculado a acurácia média e desvio padrão de forma a garantir a consistência do estudo.

4.5 Modelos Comparados

Para definir com quais modelos o *SVM* com o novo *kernel* seria comparado foi rodada uma bateria de testes com modelos mais simples, mas ao mesmo tempo similares ao modelo proposto. Inicialmente os candidatos eram:

- SVM linear com seleção de características por frequência *(Bag of Words)*
- SVM linear com seleção de características pelo valor tf-idf *(TF-IDF Bag)*
- SVM polinomial de grau 2 com *(Bag of Words)*
- SVM polinomial de grau 2 com *(TF-IDF Bag)*
- SVM polinomial de grau 3 com *(Bag of Words)*
- SVM polinomial de grau 3 com *(TF-IDF Bag)*
- SVM polinomial de grau 4 com *(Bag of Words)*
- SVM polinomial de grau 4 com *(TF-IDF Bag)*

Tabela 4.1: *SVM Linear x SVM Polinomial de grau 2 para 25 mil registros*

	Linear (Bag of Words)	Linear (TD-IDF Bag)
score-in	0.9963	0.9394
score-out	0.8284	0.8732
	Polinomial 2 (Bag of Words)	Polinomial 2 (TF-IDF Bag)
score-in	0.5043	0.5045
score-out	0.4939	0.4932

Mas em poucos testes era visível que a separação era melhor representada em um espaço linear, assim foram dispensados os testes com *SVM* polinomial de grau 3, 4, e 5, e a comparação do kernel proposto com os outros modelos além do *SVM* linear.

4.6 Pré-processamento - Seleção de Características

Para gerar o vetor de características foram utilizadas as funções oferecidas pelo *Scikit-Learn*, *CountVectorizer* e *TfidfVectorizer*, sobre os conjuntos de treino e teste.

No caso do *Bag of Words*, a função *CountVectorizer* gera um vetor de características com as N palavras mais frequentes em todas as resenhas, independente do rótulo dado à resenha.

Para o *Tf-idf*, a função *TfidfVectorizer* gera um vetor de características com as N palavras mais significativas entre todas as resenhas do conjunto, ou seja, considerando-se o peso das palavras em cada texto individualmente e no todo, conforme detalhado no capítulo 2.

4.7 Classificador SVM - Kernel Linear

O subconjunto de treino foi submetido à função *SVC*, com parâmetro *linear*, do módulo *SVM*. O parâmetro *linear* seleciona o *kernel* linear para a execução do algoritmo.

O classificador obtido foi então aplicado aos dois subconjuntos, de treino e de testes, criando os valores resultantes que foram comparados com os rótulos originais gerando o *score-in* e *score-out* respectivamente, descrevendo o score ou acurácia dentro e fora do conjunto de treino.

Os dois módulos utilizados para esses cálculos, *SVM* e *cross_ validation*, encontram-se disponíveis também no pacote *Scikit-Learn*.

4.8 Classificador SVM - Kernel Precomputed

Para utilizar o *kernel* proposto, deve-se submeter a *matriz de Gramian* do *kernel* à função *SVC* com parâmetro *precomputed*. A *matriz de Gramian* é a matriz de um conjunto de vetores em um espaço de produtos internos. Neste caso, o produto interno entre dois vetores é o valor resultado da função *kernel*.

Para o treinamento, a matriz é construida usando como domínio o *subconjunto de treino* $\times$ *subconjunto de treino*. E para o teste o *subconjunto de treino* $\times$ *subconjunto de teste*.

4.9 Resultados

Como dito anteriormente, na primeira versão do pré-processamento a limpeza era muito mais branda em relação a quantas palavras eram removidas da resenha, gerando resultados muito bons. Mas essa forma de limpeza tornava o kernel customizado totalmente inviável, pois era extremamente demorado para construir a *matriz de Gramian*.

Por isso foi necessário utilizar uma seleção de características mais focada, mas que ainda garantisse um bom resultado.

Tabela 4.2: *Acurácia média para amostra de 25 mil*

Count Bag of Words	sem stop-words	adjetivos	adjetivos e advérbios
score-in	0.9963	0.8403	0.8996
score-out	0.8284	0.7955	0.8205
Tf-idf Bag of Words	sem stop-words	adjetivos	adjetivos e advérbios
score-in	0.9394	0.8359	0.8815
score-out	0.8732	0.7981	0.8377

A seleção por adjetivos e advérbios obteve um resultado extremamente alto e satisfatórios apesar do tempo de limpeza e seleção de características deixar a desejar.

Tabela 4.3: *Tempo de execução da limpeza dos dados e seleção de características*

sem stop-words	adjetivos	adjetivos e advérbios
17m 20.79s	10h 00m 35.22s	10h 01m 03.41s

Mesmo reduzindo muito o tamanho do conjunto de características, a montagem da *matriz de Gramian* ainda demorava muito. A tabela a seguir mostra o tempo de execução. No caso do *kernel* customizado, ele representa o custo da montagem da *matriz de Gramian* e no caso do *Bag of Words*, o tempo de seleção das características. Ambos somados ao tempo de treinar e classificar o subconjunto de testes.

Tabela 4.4: *Tempo médio de execução com adjetivos e advérbios*

	1 mil registros	2 mil registros	3 mil registros
Kernel customizado	1h 43m 04.51s	7h 28min 12.45s	14h 07m 54.76s
Count Bag of Words	1.46s	5.71s	12.35s
Tf-idf Bag of Words	1.87s	7.37s	15.74s

Assim, foi necessário diminuir o tamanho da amostra a fim de conseguir colher resultados comparativos entre as diferentes implementações. Ao final, infelizmente o resultado do *kernel* customizado foi o de menor acurária média dentre os experimentos.

Tabela 4.5: *Acurácia média e desvio padrão para amostras menores*

Kernel customizado	1 mil registros	2 mil registros	3 mil registros
score-in	0.9958	0.9967	0.9919
dp score-in	0.0055	0.0047	0.0056
score-out	0.5275	0.5516	0.5592
dp score-out	0.0062	0.0089	0.0052
Count Bag of Words	1 mil registros	2 mil registros	3 mil registros
score-in	0.9974	0.9981	0.9935
dp score-in	0.0044	0.0036	0.0029
score-out	0.7602	0.7489	0.7554
dp score-out	0.0045	0.0039	0.0031
Tf-idf Bag of Words	1 mil registros	2 mil registros	3 mil registros
score-in	0.9967	0.9892	0.9415
dp score-in	0.0017	0.0016	0.0058
score-out	0.7450	0.7733	0.8142
dp score-out	0.0029	0.0027	0.0046

Capítulo 5

Conclusões

A análise dos testes mostrou que o método desenvolvido, com o *kernel* customizado, é muito custoso computacionalmente, o que forçou o experimento a ser executado com uma amostra menor do que se desejava a princípio.

Pelos resultados obtidos com os modelos lineares de *SVM*, pode-ser perceber que a redução drástica do tamanho da amostra afeta significativamente a taxa de acerto do classificador. Seguindo por esse caminho, é possível concluir que um dos motivos que levou o *kernel* customizado a obter resultados aquém do esperado foi o tamanho dos conjuntos de dados que a ele foram submetidos para o treinamento.

Uma observação geral dos resultados aponta para o fato de que o aumento, mesmo que gradativo da amostra, tende a aumentar a acurácia da classificação. Seguindo por esse caminho, e tendo como base as pequenas melhorias que o *kernel* customizado teve quando houve aumento da amostra de mil para dois mil, e depois para três mil resenhas, concluiu-se que, num ambiente de maior poder computacional e algumas otimizações no algoritimo, os valores finais obtidos pelo algoritmo tenderiam a ser melhores com amostras de tamanho mais significativo.

Porém, observa-se também que, quando comparado às implementações lineares, elas obtiveram porcentagens de acerto que superaram, em média, em 20% a taxa alcançada pelo *kernel* customizado. Assim, é provável que, ainda que o *kernel* customizado fosse aplicado a conjuntos maiores de entradas, seu resultado não alcançasse o nível de acerto da implementação linear.

Também concluiu-se que o melhor custo-benefício, em termos de tempo de execução e porcentagem de acertos, foi obtido pelo algoritmo de *SVM* linear com aplicação da técnica de extração de caracteristicas *Tf-idf*. O percentual obtido por essa versão, apesar de ser minimamente inferior à taxa da versão que conta apenas com o Bag of Words simples (menor que 5%), é compensado com o ganho de tempo de processamento pois, para as aplicações nos vetores de características nos quais foi utilizado o método do *POS Tagging*, ele foi, em média, 50% inferior.

Por fim, a implementação do *kernel* customizado, que de início pareceu promissora,

mostrou-se custosa e menos precisa na classificação. Os resultados atingidos não foram tão bons quanto os esperados e o método de classifição pelo *SVM* linear com *Tf-idf* manteve-se como a melhor opção para o estudo realizado.

5.1 Considerações Finais

Com os dados do vetor de características usado na aplicação do *SVM* com *Bag of Words*, foram construídas duas nuvens de palavras: uma em tons de azul, para as palavras positivas 5.1, e outra em tons de vermelho, para as negativas 5.2.

As palavras usadas já haviam passado pelos processos de *POS Tagging* e stemização, e serviram de entrada para um simples programa em *Scala*, que gerava uma saída nos moldes necessários para o site Wordle[1], onde foram geradas as nuvens. Quanto maior o tamanho da palavra, maior a sua frequência.

Figura 5.1: *Nuvem de palavras mais frequentes em resenhas positivas*

Observou-se que, conforme era esperado, palavra como *good*, *great* e *best* tem relevância na nuvem positiva. Notou-se também a forte presença de advérbios, como *much* e *really* (stemizado como *realli*), que não tem necessariamente conotação positiva nem negativa.

Esse fato apoia o uso do método *Tf-idf* como mediador do peso da frequência das palavras no texto, pois sem ele, observa-se uma grande relevância para palavras mais genéricas.

[1] http://wordle.net

Figura 5.2: *Nuvem de palavras mais frequentes em resenhas negativas*

Quando analisada a nuvem negativa, além do comportamento similar ao da nuvem positiva, com a forte presença de advérbios intensificadores, é possível perceber que não são apenas termos geralmente associados a sentimentos negativos, como *bad*, que têm destaque. Existe aqui uma grande quantidade de termos positivos.

Uma possível justificativa para esse fato é que nem sempre um filme considerado ruim é ruim por completo. É comum encontrar resenhas como o exemplo a seguir.

Some parts are good but the movie is still boring.

Então, quando se faz a separação de palavras em apenas adjetivos e advérbios, perde-se palavras como *but* e sentido de contradição e balanceamento que ela oferece. Dessa forma, o fato de que analisar palavras apenas pela gramática, sem considerar a semântica delas no todo, pode gerar erros, é reforçado.

Com base nesse fator, talvez considerar apenas a frequência das palavras na frase segundo sua gramática, sem incorporar o elemento semântico, não seja a melhor forma de fazer análise de sentimentos com precisão, ainda que os resultados obtidos pelos algoritmos lineares neste trabalho tenham sido melhores que os obtidos pelo *kernel* customizado.

5.2 Sugestões de Melhoria na Implementação

O script implementado é executado de forma sequêncial, o que acarreta em um longo tempo de processamento, então a melhor forma de melhorar o cenário é utilizando paralelismo.

Foram identificados 3 pontos de gargalo que podem ser paralelizados:

1. **Pré-Processamento**
2. **Cálculo da Matriz de Gramian**
3. **Algoritmo do SVM**

5.2.1 Pré-processamento

Entre os três pontos citados, esse com certeza é o mais fácil de identificar e resolver. E foi exatamente essa tarefa que, como um trabalho bônus pelo tempo extra, o grupo resolveu implementar.

Com o intuito também de explorar novas ferramentas e *frameworks* que são voltadas para o contexo de *Big Data*, foi feita a implementação de um pequeno *cluster* com apenas 2 nós no serviço de virtualização *EC2*[2] *(Elastic Compute Cloud)* da *AWS (Amazon Web Services)*.

O gerenciador de cluster utilizado foi o *YARN (Yet Another Resource Negotiator)*, parte do *framework* para computação distribuída *Apache Hadoop*[3].

Para disparar a aplicação no *YARN* e prover paralelismo foi utilizado o *framework Apache Spark*[4], que apesar de se mostrar super simples de programar em *Python*, infelizmente se mostrou ainda muito imaturo com relação as funcionalidades das bibliotecas auxiliares, em especial *MLlib* (biblioteca de aprendizado computacional).

Rodando sequencialmente em um notebook a limpeza durou 10h 01m 03s (i7-3537U), enquanto que no cluster com 30 *cores* virtuais (em */proc/cpuinfo* aparece E5-2680) durou 37m 19s. Speedup de 16.24, porém Eficiência de 0.54, o que indica um *overhead* considerável devido ao gasto em comunicação.

5.2.2 Cálculo da Matriz de Gramian

O cálculo da função *kernel* não é paralelizável, uma vez que ele é feito por um algoritmo que utiliza programação dinâmica, ou seja, existe uma ordenação topológica que deve ser seguida para o cálculo.

[2] http://aws.amazon.com/pt/ec2
[3] http://hadoop.apache.org
[4] http://spark.apache.org

Mas é possível preencher a *matriz de Gramian* de forma paralela. E além disso, a matriz construída para o treinamento possui algumas propriedades que podem reduzir o custo de processamento.

Como visto no capítulo 4, para o treinamento, a matriz é construida usando como domínio o $subconjunto\ de\ treino \times subconjunto\ de\ treino$. Devido a isso a matriz é simétrica e possui apenas 1 na sua diagonal principal.

Gramian	r1	r2	r3	r4	r5
r1	1	x	y	z	w
r2	x	1			
r3	y		1		
r4	z			1	
r5	w				1

Figura 5.3: *Exemplo de uma matriz de Gramian para treino genérica*

5.2.3 Algoritmo do SVM

O *SVM* é outro ponto cuja paralelização não é simples, mas existem soluções já implementadas.

Inicialmente foi planejado utilizar a biblioteca *MLlib* do *Apache Spark* que possui um algoritmo de *SVM* linear, porém ela não possui suporte a um algoritmo de *SVM* com *kernel* pré-calculado.

Uma outra possível solução seria utilizar a plataforma H_2O[5].

[5]http://www.h20.ai

Capítulo 6

Subjetivo

Paulo

Experiências Comecei a trabalhar com dados *(BI)* durante o primeiro estágio em 2010, e no começo desse ano um segundo estágio em análise de dados de fato. Nesse trabalho houve diversas falhas, e fico feliz por elas terem ocorrido, várias dessas falhas passariam despercebidas em um ambiente corporativo, já que normalmente eu não sou o responsável pelo projeto.
Esse trabalho foi desafiador mesmo com a experiência. Decidimos trabalhar com textos, o que está muito fora da minha zona de conforto e decidimos implementar em *Python*, uma linguagem que não me considero completamente fluente.
Um ponto importante foram os direcionamentos que o professor Gubi proporcionou. Em vários momentos nos sentíamos perdidos no que fazer e qual rumo o projeto deveria caminhar. E as dicas e sugestões sempre foram muito precisas.

Matérias Importantes MAC0332 - Engenharia de Software: A disciplina abriu meus olhos com relação ao que eu queria para a minha vida. Me mostrou que
o meu gosto por programação na verdade era gosto por resolver problemas, e que eu achava sistemas em que existe um usuário interagindo algo realmente chato.
MAC0431 - Introdução à Computação Paralela e Distríbuida: A disciplina foi extremamente útil, pois me deu mais flexibilidade para enfrentar problemas. Tentar ver as coisas de um ângulo diferente e tentar resolver de uma forma criativa.
B. Matemática - Não é exatamente uma disciplina, mas o curso todo em si. Fico feliz de ter gasto um tempo na Matemática para descobrir que não era o que eu queria, e ao mesmo tempo, acredito que foi um período útil para o meu desenvolvimento pessoal.

Luciana

Experiências Começando pelas partes boas, o primeiro fato que posso citar foi o grande aprendizado que tive fazendo este trabalho, cujo assunto não era algo ao qual eu estava familiarizada até então. Por meio do trabalho pude aprender não só sobre temas atuais e interessantes, diferentes dos que estou acostumada a ver, mas também a mexer com ferramentas diferentes.

Outro fator positivo foi a decisão de fazer o trabalho em dupla. Foi realmente muito bom poder contar com a ajuda de um amigo em todas as situações que surgiram, pra ajudar a resolver os problemas e pra comemorar junto quando algo legal era concluído.

Por fim, um ponto muito positivo do nosso TCC deve-se ao professor Gubi. Seu apoio e direcionamento foi muito importante, ao mesmo tempo que a liberdade que ele nos deu para tomarmos nossas decisões. Também não posso deixar de citar sua compreensão com nossos horários e as divertidas reuniões que ele nos proporcionou. Nós até apelidamos nosso *kernel* customizado de Gubi's *kernel*.

Quanto aos problema, o tempo, é claro, foi um deles. Cada um de nós tinha suas próprias atividades, e nem sempre dispúnhamos de todo o tempo de que gostaríamos.

Além disso, nossa ideia inicial era trabalhar com textos literários e, bem, ela acabou sendo apenas uma ideia mesmo e tivemos que buscar outro tema pelo qual ambos se interessavam.

No entanto, a parte mais frustrante foi não ter conseguido lidar com tantos dados quanto gostaríamos, e ter que limitar nossas amostras, fazendo com que os resultados não saíssem tão bons quanto gostaríamos.

Matérias Importantes MAC0110 - Introdução à Computação: Essa disciplina foi o primeiro contato que tive com a linguagem Python. O prévio conhecimento ajudou na decisão de que ela seria a linguagem que usaríamos no trabalho.

MAC0338 - Análise de Algoritmos: Foi muito importante para introduzir o conceito de programação dinâmica e também para aprendermos mais sobre como analisar um algoritmo, o que basicamente, resume o nosso trabalho.

MAC0425 - Inteligência Artificial: Nessa disciplina introduz-se o conceito de aprendizagem de máquina, que para nós foi fundamental na utilização de *kernels* e do SVM.

MAC0444 - Sistemas Baseados em Conhecimento: Nessa disciplina pude ter um contato, ainda que inicial, com o processamento de linguagem natural, que também foi um tema de destaque em nosso trabalho.

Referências Bibliográficas

Baccarini *et al.* **(2011)** Lane Maria Rabelo Baccarini, Valceres Vieira Rocha e Silva, Benjamim Rodrigues De Menezes e Walmir Matos Caminhas. Svm practical industrial application for mechanical faults diagnostic. *Expert Systems with Applications*, 38(6):6980–6984. Citado na pág. 10

Benamara *et al.* **(2007)** Farah Benamara, Carmine Cesarano, Antonio Picariello, Diego Reforgiato Recupero e Venkatramana S Subrahmanian. Sentiment analysis: Adjectives and adverbs are better than adjectives alone. Em *ICWSM*. Citado na pág. 12

Byun e Lee (2002) Hyeran Byun e Seong-Whan Lee. Applications of support vector machines for pattern recognition: A survey. Em *Pattern recognition with support vector machines*, páginas 213–236. Springer. Citado na pág. 18

Cai *et al.* **(2004)** Yu-Dong Cai, Pong-Wong Ricardo, Chih-Hung Jen e Kuo-Chen Chou. Application of svm to predict membrane protein types. *Journal of Theoretical Biology*, 226(4):373–376. Citado na pág. 10

Chu *et al.* **(2005)** Feng Chu, Guosheng Jin e Lipo Wang. Cancer diagnosis and protein secondary structure prediction using support vector machines. Em *Support Vector Machines: Theory and Applications*, páginas 343–363. Springer. Citado na pág. 10

Garreta e Moncecchi (2013) Raul Garreta e Guillermo Moncecchi. *Learning scikit-learn: Machine Learning in Python*. Packt Publishing Ltd. Citado na pág. 4, 5, 9

Gonçalves *et al.* **(2013)** Pollyanna Gonçalves, Matheus Araújo, Fabrício Benevenuto e Meeyoung Cha. Comparing and combining sentiment analysis methods. Em *Proceedings of the first ACM conference on Online social networks*, páginas 27–38. ACM. Citado na pág. 1

Kecman (2005) Vojislav Kecman. Support vector machines–an introduction. Em *Support vector machines: theory and applications*, páginas 1–47. Springer. Citado na pág. 9

Lodhi *et al.* **(2002)** Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini e Chris Watkins. Text classification using string kernels. *The Journal of Machine Learning Research*, 2:419–444. Citado na pág. 14

Ng (2015) Andrew Ng. Aprendizado de máquina - universidade de stanford | coursera. https://pt.coursera.org/learn/machine-learning, 2015. Último acesso em 05/11/2015. Citado na pág. 4

Osuna *et al.* **(1997)** Edgar Osuna, Robert Freund e Federico Girosi. Training support vector machines: an application to face detection. Em *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, páginas 130–136. IEEE. Citado na pág. 10

Reda (2013) Greg Reda. Intro to pandas data structures. http://www.gregreda.com/2013/10/26/intro-to-pandas-data-structures/, 2013. Último acesso em 25/10/2015. Citado na pág. 21

Sharma *et al.* **(2014)** Richa Sharma, Shweta Nigam e Rekha Jain. Opinion mining of movie reviews at document level. *IJIT International Journal on Information Theory*, 3. Citado na pág. 1

Shawe-Taylor e Cristianini (2004) John Shawe-Taylor e Nello Cristianini. *Kernel methods for pattern analysis*. Cambridge university press. Citado na pág. 12, 14, 16