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

Estudo comparativo de modelos de aprendizado de máquina para detecção de email spam

Nícolas Machado Schumacher

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Dr. Thiago de Paulo Faleiros

Brasília
2020

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

Estudo comparativo de modelos de aprendizado de máquina para detecção de email spam

Nícolas Machado Schumacher

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

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

Prof. Dr. Dibio Leandro Borges
CIC/UnB

Prof. Dr. Alan Demétrius Baria Valejo
DC-UFSCar

Prof. Dr. Marcelo Grandi Mandelli
Coordenador do Bacharelado em Ciência da Computação

Brasília, 24 de novembro de 2020

Dedicatória

Dedico ao meu Pai, que está sempre presente me dando vida plena, consolo e orientação, à minha família pelo suporte e paciência e aos amigos que me apoiaram e incentivaram emocional e espiritualmente, Cayo, Erika, Vinicius, John, Danilo, Denilson, Eva, Lu, Camila, entre tantos outros que estiveram próximos.

Agradecimentos

Agradeço aos amigos e pesquisadores que de alguma forma contribuíram com o desenvolvimento este trabalho: Alex, Rebeca, Rafael, Thales, ao meu orientador, Thiago Faleiros, pelas dicas, acompanhamento e paciência. Também sou grato pela oportunidade de estudar nesta instituição de excelência e pelo curso de *Machine Learning* oferecido gratuitamente na plataforma *Coursera*, pelo professor Dr. Andrew Ng, que muito contribuiu com minha capacitação para esta pesquisa.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

Haja vista o alto tréfego de email spam e a sua natureza inconveniente ou até mesmo, em alguns casos, nociva, este trabalho objetiva examinar a evolução de modelos supervisionados de aprendizado de máquina capazes de classificar emails entre legítimo ou spam ao variar a vetorização das características textuais, dividindo os experimentos em etapas de diferentes níveis de complexidade a fim de explorar a capacidade de aprendizado dos modelos, avaliando numerica e graficamente o desenvolvimento de cada um deles, buscando-se alcançar os maiores resultados na última etapa. A principal medida utilizada para validar a solução é *F1-score*, além das análises das curvas de aprendizado. Foram utilizados os algoritmos SVM, Naive Bayes e KNN, sendo que os modelos SVM apresentaram as melhores respostas quanto ao avanço da complexidade do treinamento, obtendo os maiores resultados em todos os datasets, já os outros dois algoritmos manifestaram maior sensibilidade e incerteza quanto às medidas tomadas em cada etapa. Possíveis incrementos a este trabalho incluem: expansão dos conjuntos de dados utilizados, especialmente para verificar o progresso de modelos SVM de kernel polinomial, implementação de novas *features* extraídas a partir dos textos erroneamente classificados e utilização de técnicas de regressão para melhor avaliação das curvas de aprendizado.

Palavras-chave: email spam, curvas de aprendizado, aprendizado de máquina

Abstract

Because of the high spam traffic and its undesirable or even, in some cases, harmful nature, this present work aims to inspect the progress of supervised machine learning algorithms capable of labeling emails as spam or legitimate by diversifying the text's feature vectorization. This is done by the split of the experiments into phases of different complexity levels, in order to explore the learning ability of the algorithms, numerically and graphically evaluating their development, seeking for the best results in the last phase. The main evaluation method used is the F1-score, also the learning curves analysis. The algorithms used were SVM, Naive Bayes and KNN, and the SVM models presented the best responses as the training complexity increased, obtaining the highest results in all datasets, whereas the other two algorithms showed greater sensitivity and uncertainty regarding the actions taken at each stage. Possible enhancements to this research include: data sets expansion, especially to verify polynomial kernel SVM's progress, development of new features extracted from misclassified emails and the use of regression techniques to better evaluate the learning curves.

Keywords: spam email, learning curves, machine learning

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Email é um meio de comunicação eletrônico de troca de mensagens que passou por diversas evoluções desde sua concepção em 1971[1], de modo que ultimamente ainda é amplamente utilizado, com cerca de 281 bilhões de emails enviados diariamente apenas no ano de 2018[2].

Por ser um efetivo meio de comunicação, suas vulnerabilidades são também exploradas para fins maliciosos. Dessa forma surgiu o email spam[3], que se refere a um email não solicitado, podendo ser enviado em larga escala[4]. Por não solicitado, entende-se que o destinatário não concedeu permissão verificável para a mensagem ser enviada[5].

Em 2018, por volta de 45,3% do total de emails no mundo eram spam[6], de modo que, no mesmo ano, na Austrália foi reportado que cerca de $489 milhões (AUD) foram perdidos por fraudes, já em 2019 foram $634 milhões [1], dos quais $132 milhões devido a emails comerciais falsos, sendo estimado que apenas aproximadamente 13% das vítimas relatam a fraude.

Devido a prejuízos não apenas econômicos, mas também sociais e técnicos por conta de spams, torna-se necessário o esforço para eficazmente detectá-los e uma das formas amplamente utilizadas de realizá-lo é o uso de algoritmos de aprendizado de máquina em servidores de email, que realizam a classificação automaticamente, sem a intervenção do usuário, prevenindo-o de um possível ataque. Porém, estudos recentes demonstram a necessidade de agregar e desenvolver novas técnicas, visto que *spammers*[7] constante-

[1] `https://www.theverge.com/2012/5/2/2991486/ray-tomlinson-email-inventor-interview-i-see-email-bei`

[2] `https://www.statista.com/statistics/456500/daily-number-of-e-mails-worldwide/`

[3] Originalmente, spam se referia apenas a uma marca de presunto enlatado `https://www.bloomberg.com/news/articles/2012-05-17/how-spam-meat-has-survived-spam-e-mail`

[4] `https://kb.iu.edu/d/afne`

[5] `https://www.spamhaus.org/consumer/definition/`

[6] `https://www.statista.com/statistics/420400/spam-email-traffic-share-annual/`

[7] *Spammers* são aqueles que enviam spam.

mente inovam a maneira de enganar os usuários e os trabalhos científicos publicados não acompanham tais tecnologias, conforme aponta o estudo [2].

O presente trabalho propõe o treinamento e a comparação de diferentes modelos de aprendizado de máquina por meio de i) desambiguação do sentido das palavras do corpo do email [3] e ii) desenvolvimento de *features*[8] adicionais. O propósito é obter modelos de melhor performance que a utilização isolada de tais técnicas.

Assim, a pergunta a ser respondida por este trabalho é: unir as técnicas i e ii para classificar spam pode aumentar a performance dos modelos?

A relevância deste trabalho, de pesquisar modelos capazes de identificar spams corretamente, se dá, dentre outros fatores, pelo fato de que tanto empresas quanto consumidores são prejudicados, por exemplo: aqueles por terem seu nome utilizado em um esquema de fraude e perderem a confiança dos consumidores e estes por estarem vulneráveis a ataques sofisticados e, eventualmente, caírem em engano, fato este que motiva novas pesquisas a fim de reduzir o número de vítimas de spam.

1.1 Objetivos

O objetivo deste trabalho é verificar a evolução de modelos de aprendizado de máquina que sejam capazes de identificar se determinado email é legítimo ou spam ao implementar novas características e variar o pré-processamento para melhorar a tarefa de classificação utilizando apenas o corpo do email. A fim de cumprir o objetivo geral explicitado, será necessário realizar os seguintes objetivos específicos:

- Separar os treinamentos em etapas progressivas: daquela de menor para a de maior complexidade, a fim de acompanhar o desenvolvimento dos modelos;
- Aplicar técnicas de desambiguação aos textos para melhorar a performance dos modelos;
- Adicionar novas características, como *POS tagging* e contagem de determinados atributos dos textos variando a forma de normalizá-los, também para contribuir na tarefa de classificação;
- Gerar gráficos do aprendizado de cada modelo em todas as etapas e avaliá-los;

[8] *Features* são as características que representam um email.

1.2 Metodologia

A metodologia deste trabalho se dá em cinco etapas que incluem desde o estudo dos assuntos envolvidos no tema até a análise dos resultados, como se segue:

1. Estudo de teoria, algoritmos e técnicas de treinamento de modelos de ML, inclusive avaliação do desempenho dos modelos.
2. Estudo do domínio em questão, email spam, seu conceito, trabalhos correlatos e pesquisa de datasets rotulados.
3. Desenvolvimento de modelos simples, isto é, nos quesitos de parametrização e de pré-processamento textual, utilizando as técnicas propostas nos objetivos, a fim de evitar a "otimização prematura" [4].
4. Desenvolver modelos que unam ambas as técnicas propostas de otimizações e realizar os treinamentos.
5. Análise dos resultados e conclusão.

1.3 Estrutura do Documento

Este trabalho aborda a fundamentação teórica necessária para a compreensão e o desenvolvimento da pesquisa no Capítulo 2, apresentando conceitos sobre spam, técnicas de vetorização e de pré-processamento textual, algoritmos de apredizado de máquina e técnicas numéricas e gráficas de avaliação dos modelos treinados. Também constam nesse capítulo trabalhos relacionados ao escopo desta pesquisa.

No Capítulo 3 é apresentada a proposta de solução para alcançar os objetivos estabelecidos, envolvendo descrição dos conjuntos de dados utilizados, técnicas aplicadas para processá-los e para avaliar os modelos e, ao final, os resultados obtidos em cada etapa da pesquisa e sua análise.

Por fim, no Capítulo 4, são apresentadas as conclusões e possíveis trabalhos futuros que podem ser realizados para a evolução desta pesquisa.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta alguns conceitos necessários para a compreensão do trabalho e serão referenciados para explicar ações tomadas na metodologia de pesquisa.

2.1 Spam

Como informado na introdução, *spam* se refere a um email não solicitado, que pode ser enviado em larga escala. Nesta seção serão expostos alguns conceitos e motivações que reforçam a necessidade de que os algoritmos classificadores de email sejam treinados de modo a não apresentar maior propensão para rotular uma mensagem como spam nem como ham[1], além de alguns trechos do email que podem contribuir para a classificação destes.

A Tabela 2.1 informa definições de algumas técnicas que podem estar contidas em emails com finalidades maliciosas.

Devido ao email *spoofing* e outras técnicas que possibilitam forjar informações do header, esforços para analisar tais campos podem não surtir tanto efeito quanto esperado, mas o campo *received*, que contém o IP do remetente, é mais difícil de ser falsificado[2], possibilitando, assim, que ele seja uma característica promissora a ser adicionada ao conjunto de dados de treinamento. Isso não quer dizer que os demais campos possam ser desprezados, mas possivelmente o peso deles na determinação de um spam seria menos significativo que o campo *received*.

Apenas o destinatário original da mensagem tem acesso ao cabeçalho completo do email inclusive com o conteúdo para saber exatamente de onde veio a mensagem e o

[1] Ham é um nome dado aos emails legítimos, ou seja, que não são spam.

[2] `https://mediatemple.net/community/products/dv/204643950/understanding-an-email-header`

Termo	Definição
Spoofing	Ocorre quando o *spammer* falsifica o remetente do email.
Phishing	É a tentativa de adquirir dados pessoais ou confidenciais utilizando-se do disfarce de um remetente confiável.
Spear phishing	Variação do *phishing*, a diferença é que neste caso o ataque é direcionado a indivíduos específicos.
Clone phishing	Semelhante ao *phishing*, mas com a utilização de, por exemplo, um site clonado, que imita o original.
Whaling	É o mesmo que o *spear phishing*, porém voltado a "grandes" alvos, como pessoas com altos cargos ou personalidades de relevância.

Tabela 2.1: Possíveis técnicas aplicadas por *spammers*

caminho percorrido na internet, ou seja, emails repassados e cópias de emails não terão tais informações.[3]

Quanto à avaliação dos modelos de aprendizado de máquina, é importante utilizar uma medida de avaliação que equilibra a identificação entre spam e ham sem que haja viés na avaliação para uma classe ou outra, como se segue:

- Caso muitos emails que são realmente spam sejam classificados como legítimos, o usuário que os recebe, dependendo do conteúdo da mensagem, pode receber muita mensagem inconveniente, isto é, que apenas causa perturbação e preenche sua caixa de mensagens, mas, por outro lado, ele pode ser exposto a riscos nos casos em que estas mensagem contenham vírus ou, por exemplo, alguma das técnicas de phishing, ou até mesmo conteúdos sensíveis, que o prejudique social ou emocionalmente.
- Por outro lado, se muitos emails que de fato sejam legítimos acabem sendo classificados como spam, sendo eles ocultados do usuário, isto pode gerar outros tipos de perturbações: o usuário pode aguardar por um email que, aparentemente, nunca chega ou deixar de visualizar propostas e acabar perdendo oportunidades legítimas que foram classificadas como spam. Por exemplo, recentemente no próprio site de suporte do *Gmail*[4] foram registradas ocorrências deste caso, no qual diversos empresários acabaram sendo prejudicados, perdendo oportunidades de negócio ou tendo que visitar frequentemente a seção de spam para encontrar emails legítimos, rela-

[3] `https://kb.iu.edu/d/akij`
[4] `https://support.google.com/mail/thread/4167678?hl=en`

tando a impressão de que os algoritmos ficaram mais propensos a classificar um email como spam.

2.2 Vetorização de Palavras

Os computadores tradicionais não interpretam de forma nativa a linguagem natural. Eles trabalham com números. Até mesmo os algoritmos de aprendizado de máquina e de *Deep Learning* precisam de uma representação numérica dos textos (ou dos sons ou imagens) com que irão trabalhar. Então é preciso transformar palavras em valores numéricos para serem utilizados no processo de mineração de dados. Este processo recebe o nome de vetorização textual (ou *word embedding*).

Dessa forma, a vetorização de palavras é o aprendizado de um vetor denso, ou seja, um vetor cuja maioria dos valores é diferente de zero, que mapeia a representação de tipos discretos (neste caso, palavras de um vocabulário) em um ponto no espaço, isto é, um tipo contínuo [5].

Há diversas formas de representar vetores de palavras, a seguir são apresentadas duas delas.

2.2.1 *One-hot*

Uma das formas mais simples de representar palavras consiste em um vetor V unidimensional, cuja magnitude $||V||$ corresponde ao tamanho do vocabulário, cada índice representa uma palavra p distinta e o valor armazenado em cada índice é igual a 0 ou 1, indicando, respectivamente, ausência ou presença daquela palavra no texto em questão. Assim, $p = \{0, 1\}, \forall\ p \in V$.

Uma das limitações desta técnica é que ela perde informações do texto como: posição relativa das palavras e a quantidade de repetições de cada uma delas.

2.2.2 TF-IDF

Uma técnica semelhante à codificação *one-hot*, porém que não perde a informação quanto à repetição de palavras em um texto, chama-se *Term Frequency* (TF).

O vetor V obtido utilizando esta metodologia possui as mesmas dimensões daquele obtido aplicando *one-hot*, porém, ao invés de receber valores 0 ou 1, eles estão em uma faixa entre 0 e 1, indicando a frequência de cada termo em determinado texto comparado às demais palavras encontradas nele. Então, neste caso:

$$TF(p) = \frac{T}{t_p} \tag{2.1}$$

Sento T o número total de termos do documento em questão e t_p a quantidade de vezes que o termo p aparece neste mesmo documento, com $p \in [0, 1], \forall\ p \in V$.

Há ainda uma variação deste método, que aplica a técnica *Inverse Document-Frequency* (IDF), que atribui maior peso às palavras mais raras entre os textos e, consequentemente, menor peso às mais comuns, conforme a seguinte equação:

$$IDF(p) = \log\left(\frac{D}{d_p}\right) \tag{2.2}$$

Sendo p uma palavra, D o número total de documentos no corpus e d_p a quantidade de documentos em que p aparece. TF-IDF é obtido do produto $TF(p)IDF(p)$, unindo ambas as técnicas, de modo que quanto menos um termo aparece entre os diferentes documentos e mais se repete no documento corrente, maior será seu velor.

2.3 Pré-processamento de Dados

Tradicionalmente os algoritmos não são desenvolvidos para prever qualquer estrutura de dado imaginável, já que é inviável e é uma situação que pode ser contornada padronizando os dados que serão utilizados. Porém, em diversas aplicações de mineração de dados faz-se necessário realizar algum tipo de tratamento de dados para minimamente buscar torná-los mais manipuláveis pelo algoritmo.

Especialmente tratando-se de NLP, os dados de entrada comumente não são estruturados, visto que eles podem ser adquiridos através de diversas fontes completamente distintas, como um site de notícias, comentários de uma rede social, artigos científicos, etc. Ainda que todos os dados sejam de uma mesma fonte, eles podem apresentar caracteres incomuns, como a interrogação invertida, encontrada na língua espanhola, figuras, tabelas, erros de digitação em geral, entre outros. São inúmeras as possibilidades e o trabalho de pré-processamento, no fim, visa melhorar a qualidade da mineração de dados (ou do aprendizado de máquina) e, em alguns casos, tornar mais eficiente o tempo de processamento dos dados.

Seguem alguns exemplos de tipos de pré-processamento:

- Detecção e correção de erros: textos, em geral, são suscetíveis a conterem erros gramaticais. As palavras, por exemplo, “pavil” e “paviu”, que não são encontradas no dicionário de língua portuguesa, poderiam ser encontradas em textos cuja intenção de quem as escreveu seria expressar, na verdade, “pavio”. Há também o problema da abreviação, como “vcs”, significando “vocês”, mas é possível que “VCS” seja uma sigla para o nome de uma empresa ou pessoa, por exemplo. Distinção entre letras minúsculas e maiúsculas: as palavras “Então” e “então” deveriam ser trata-

das como uma só palavra? Letras maiúsculas costumam indicar não apenas início de frase, mas também nomes, sobrenomes, apelidos, instituições, entidades, localizações geográficas, nomes de datas festivas, títulos de periódicos, siglas, símbolos, abreviaturas, etc1. Isto é, letras maiúsculas podem conter informações que serão perdidas caso todas as palavras sejam tratadas da mesma maneira. A questão é se o algoritmo é capaz de captar esta característica ou não.

- Stop words: geralmente são as palavras mais comuns de uma linguagem e que não adicionam muito significado ao texto, de modo que é comum removê-las e isso não prejudicará a compreensão textual.

- Lematização: esta técnica mapeia palavras como "programadores" e "programadora" para "programador". Assim, ela mapeia as variações dos verbos para o infinitivo e adjetivos e substantivos para o masculino singular.

- *Stemming*: reduz as palavras à sua base ou raiz. O propósito é semelhante ao da lematização: reduzir as diferentes formas que uma palavra pode assumir a um único formato. Assim, o algoritmo não precisará armazenar e relacionar palavras como, por exemplo, "escrevi", "escreveu", "escrevo" e "escreveríamos", ao invés disso, todas elas serão representadas por "escrev".

- *Feature scaling*: valores altos demais de algumas *features* e outros muito baixos podem acabar prejudicando o aprendizado dos modelos, dependendo do algoritmo, levando-o a forçadamente considerar as características de maior valor como mais importantes na tarefa de predição. Para mitigar este inconveniente, costuma ser muito útil limitar o intervalo que tais valores podem assumir e esta é a tarefa do *feature scaling*. Para algoritmos que utilizam gradiente descentente para atualizar seus pesos, por exemplo, não aplicar esta técnica pode levar a um maior tempo necessário para que a função convirja.

É importante também ressaltar que, dependendo da técnica de pré-processamento utilizada, muitas informações podem ser perdidas: características semânticas e sintáticas caso sejam aplicadas técnicas para remover palavras, pontuações, acentos, transformação de todas as letras para minúsculas, entre outros métodos de pré-processamento. Portanto, ter precaução e realizar testes para verificar se certo pré-processamento auxilia ou não nas predições antes de aplicar diversas técnicas é uma abordagem mais segura.

2.4 Aprendizado de Máquina

"O campo do aprendizado de máquina está relacionado com a questão de como construir programas de computador que automaticamente melhoram com a experiência" [6] (tradução livre).

Assim sendo, diz-se que os algoritmos de ML são capazes de aprender padrões dos dados e há, pelo menos, três abordagens de aprendizado:

- Supervisionado: quando os dados são rotulados, isto é, a classe correta deles é informada de antemão. Assim, os algoritmos podem fazer associações das características dos dados com relação aos resultados disponíveis e isso possibilita métodos objetivos de avaliação dos modelos.
- Não supervisionado: nesta abordagem os dados não possuem rótulos. Os algoritmos buscam relações implícitas entre as próprias características dos dados. Diferentemente do aprendizado supervisionado, como não há um gabarito das classes corretas, isso pode dificultar a avaliação destes modelos.
- Por reforço: os algoritmos aprendem com base em tentativa e erro a priorizar certas características dos dados.

2.4.1 Escolha de Características para Vetorização

Dada uma tarefa de classificação, isto é, que se deseja rotular novos dados de entrada corretamente, é possível considerar esta tarefa de aprendizado de máquina em um outro cenário, fora dos algoritmos. Por exemplo, considerando a pergunta "um carro foi fabricado em 2018, possui 20 mil Km rodados e pesa 1,6T, quanto ele vale?", as informações (características, ou *features*) informadas não são suficientes nem mesmo para um grande especialista e conhecedor de carros poder respondê-la. A primeira pergunta que ele poderia fazer é "qual é o modelo?". Nesse caso, o modelo do carro seria uma característica que, num algoritmo, teria um peso considerável na predição e a falta dela poderia levar a grandes incertezas, assim como no caso deste especialista. Ele poderia listar carros com valores numa faixa de R$70 mil a R$2,7 milhões, por exemplo.

O problema apresentado, aplicando-o ao desempenho de um algoritmo, pode levar ao seguinte pensamento: por maior que seja a quantidade de dados que um modelo de aprendizado tenha sido treinado, se os dados que ele receber para realizar a predição não contiverem informações suficientes para resolver a questão, com precisão, nem mesmo por um especialista, é bastante provável que o algoritmo também tenha um baixo desempenho.

Assim, sem qualquer implementação ou projeto complexo de ML, é possível realizar a seguinte pergunta antes de avançar no projeto: dado um conjunto X de características, um especialista no assunto seria capaz de predizer, precisamente, a resposta y?

2.4.2 Naive Bayes

O algoritmo Naive Bayes assume independência entre as *features* de um texto [7].

Segundo [8], o valor *spamminess* de determinada palavra p de um vocabulário V é dado por:

$$S[p] = \frac{C_{spam}(p)}{C_{spam}(p) + C_{ham}(p)} \tag{2.3}$$

Sendo $C_{spam}(p)$ e $C_{ham}(p)$, respectivamente, o número de documentos spam e ham que contém a palavra p. Assim, o valor de $S[p]$ será alto quanto mais p ocorrer em spams do que em emails legítimos.

Uma forma de calcular esta taxa não apenas por palavra, mas para um documento D inteiro é:

$$S[D] = \prod_{i=1}^{N} S[p_i] \tag{2.4}$$

De modo que $p_i \in D$ tal que $D = \{p_1, p_2, p_3, ..., p_N\}$. O cálculo de *hamminess* do documento $H[D]$ é realizado de maneira equivalente, apenas alterando $S[p_i]$ por $(1-S[p_i])$. Alguns filtros também implementam um cálculo indicador $I[D]$ para não comparar $S[D]$ e $H[D]$ diretamente, o que poderia levar a valores iguais em alguns casos, necessitando de uma decisão para casos de empate. Assim, o indicador é calculado por:

$$I[D] = \frac{1 + S[D] - H[D]}{2} \tag{2.5}$$

Se $I[D] > 0.5$, então D é classificado como spam, caso contrário, ham. Isso possibilita a escolha de um valor de *thresh*, que pode ser 0.5, como apresentado, ou algum outro valor entre 0 e 1.

2.4.3 KNN

O algoritmo *K-Nearest Neighbors* [9] calcula a distância entre os textos a partir do vetor de características deles, o que pode ser feito, por exemplo, por meio da distância euclidiana. Sejam P e Q dois vetores de características, ambos de dimensão N, de modo que $p_i \in P$ e $q_i \in Q$, assim, a distância entre P e Q é dada por:

$$d = \sum_{i=1}^{N} \sqrt{(p_i - q_i)^2} \tag{2.6}$$

Na etapa de treinamento os textos são armazenados em memória, com seu respectivo rótulo. Então, no momento de predição, a classe atribuída ao texto de entrada é determinada pela classe da maioria dentre os K textos do conjunto de treino mais próximos a ele, sendo K um valor inteiro estritamente positivo escolhido previamente.

Há variações da implementação que atibuem diferentes pesos para cada dado vizinho de uma amostra como, por exemplo, atribuir pesos cada vez menores quando os dados vizinhos são mais distantes.

Este algoritmo não está atrelado a uma distribuição de probabilidade conhecida [10], como a distribuição normal, por exemplo, visto que sua classificação depende somente da disposição dos próprios dados no espaço.

2.4.4 SVM

Support Vector Machine [11] é um algoritmo desenvolvido por Corinna Cortes e Vladimir Vapnik que, em sua forma mais simples, é capaz de classificar dados que sejam linearmente separáveis. A ideia é encontrar uma fronteira que separe o conjunto de dados em duas classes distintas.

Esta fronteira é chamada de hiperplano, que é um subespaço cuja dimensão é menor em uma unidade com relação ao espaço em consideração. Neste caso, para um vetor de N características, o hiperplano terá dimensão $N - 1$.

Assumindo que os dados são da forma $(x_1, y_1), (x_2, y_2), (x_3, y_3), ..., (x_D, y_D)$, sendo D o número total de documentos, $y_i \in \{-1, 1\}$, indicando a classe positiva ou negativa e $x_i \in \mathbb{R}^N, i = 1, 2, 3, ..., D$, ou seja, x_i é o vetor de características do documento i.

Um hiperplano é obtido da seguinte equação:

$$v^T u + b = 0 \tag{2.7}$$

Sendo v um ponto pertencente ao hiperplano, b uma constante e u um vetor unitário, ou seja, $||u|| = 1$.

A partir disso, para um texto x, a equação $x^T u + b$ terá resultado negativo indicando que x pertence a um lado do hiperplano e resultado positivo indicando que ele pertence ao outro lado. Se o resultado for igual a zero, então x pertence ao hiperplano. Mas, no caso do SVM para dados linearmente separáveis, nenhum dado fará parte do hiperplano calculado. Assim, de acordo com [12], cada lado do hiperplano representa uma classe e o

seguinte será verdadeiro: $y_i(x_i^T u + b) > 0, \forall i$, pois sempre será o caso de a classe predita ter o mesmo sinal que a classe real.

Além disso, o vetor u é calculado de tal forma que a distância entre este hiperplano e os dados de ambas as classes seja a maior possível, de modo que, quanto este é o caso, tal hiperplano é nomeado hiperplano ótimo. Assim, haverá uma distância entre o hiperplano ótimo e as amostras mais próximas a ele. Para alcançar este hiperplano ótimo, é preciso maximizar a distância M entre o hiperplano e os dados mais próximos, ou seja, segundo [12], encontrar u que resulte em $\max M$:

$$\max_{u,\ b,\ ||u||=1} M,\ \textit{subordinado a}\ y_i(x_i^T u + b) \geq M,\ i = 1, 2, 3, ..., D \tag{2.8}$$

Porém, muitos dados não irão seguir uma distribuição linearmente separável, de modo que o hiperplano ótimo deve ser flexível a fim de tolerar erros de classificação.

2.4.5 Métodos de Avaliação

Modelos de aprendizado de máquina desenvolvidos para classificar textos são treinados para realizar predições acerca de dados não vistos na etapa de treinamento. Há diversas técnicas para avaliar o desempenho de um modelo, medindo o erro relativo ao seu aprendizado.

A Tabela 2.2 informa algumas medidas comumente utilizadas na avaliação do desempenho de modelos de aprendizado de máquina. Considere "VP" como verdadeiro positivo, "FP" como falso positivo, "VN" verdadeiro negativo e, por fim, "FN" como falso negativo.

Definição	Fórmula
Acurácia: afere a proporção de predições corretas com relação ao total de dados.	$\frac{VP+VN}{VP+FP+VN+FN}$
Precisão (p): a partir do número total de predições positivas, informa quantas são, de fato, verdadeiros positivos.	$\frac{VP}{VP+FP}$
Recall (r): informa quantos verdadeiros positivos o modelo foi capaz de predizer do total disponível no conjunto de dados.	$\frac{VP}{VP+FN}$
F1-score: medida que correlaciona precisão e *recall*: será alta apenas se ambas forem altas também.	$\frac{2pr}{p+r} = \frac{VP}{VP+\frac{1}{2}(FP+FN)}$

Tabela 2.2: Medidas de avaliação dos modelos de ML

A acurácia é amplamente utilizada [10], ela é útil especialmente em datasets balanceados, isto é, com proporções equivalentes de classes positivas e negativas, mas quando

este não é o caso, por exemplo, quando a classe negativa é muito maior que a positiva, se um modelo estiver enviasado para rotular dados como pertencentes à classe negativa a acurácia não irá revelar esta anomalia. Já a medida *F1-score* (também chamada de F1-measure ou, simplesmente, F1) cobre esta observação, especialmente neste caso em que a classe positiva possui menos dados, porém ela apresenta limitações como: não considera o valor de verdadeiros negativos (VN). Assim, se muitos dados forem corretamente rotulados como pertencentes à classe negativa, isto não alterará a medida F1, de acordo com a fórmula apresentada.

Como o objetivo de obter classificadores é que eles sejam capazes de predizer novos dados, ou seja, dados que não foram vistos na etapa de treinamento, uma técnica utilizada para avaliar o modelo quanto à generalização de seus resultados é a divisão do dataset, separando-o em um conjunto para realizar o treinamento e outro apenas para testes. A proporção dos dados em cada um dos conjuntos pode ser, por exemplo, 90%/10%, 80%/20%, 70%/30%, etc.

Além disso, é possível explorar ainda mais a divisão do conjunto de dados: muitos algoritmos possuem hiperparâmetros e pode ser uma tarefa bastante custosa encontrar os valores que mais auxiliarão o modelo a predizer novos dados. Utilizar o próprio conjunto de treino para encontrar estes parâmetros pode não ser uma boa prática. Por exemplo, suponha que um modelo treinado seja capaz de predizer corretamente praticamente todos os dados com os quais ele treinou. Isso não é muito significativo, pois deseja-se que ele seja capaz de predizer novos dados. Agora, se este mesmo conjunto que foi utilizado para o treino for, também, utilizado para selecionar os hiperparâmetros deste modelo, ele ficará ainda melhor na tarefa de predizer os dados de treino e, como foi dito, este não é o objetivo. Assim, uma outra técnica possível é a de dividir o dataset em três partes: treino, validação e teste, sendo o conjunto de treino utilizado somente na etapa de treinamento, o de validação para escolha de hiperparâmetros e, finalmente, o de teste apenas para verificar o quanto este modelo é capaz de classificar corretamente dados antes nunca vistos.

2.4.6 Diagnóstico de Aprendizado de Máquina

É possível que mesmo quando o modelo de aprendizado obtém um valor baixo de erro, ao introduzir um novo conjunto de testes sejam observados erros descomedidos sobre as novas predições. Um exemplo é quando ocorre *overfitting* da hipótese sobre um dado conjunto de teste, falhando, assim, o algoritmo na generalização de novos exemplos que não estão presentes naquele conjunto de testes.

Para atenuar o *overfitting*, por exemplo, é possível que se escolha uma técnica, como aumentar o conjunto de dados ou a dimensionalidade do vetor de características, de forma

aleatória ou utilizando simplesmente a intuição, mas há técnicas, que fazem parte do chamado diagnóstico de *machine learning*, que auxiliam nesta escolha para atacar o problema de maneira mais acertiva. Esses diagnósticos são testes que podem ser executados a fim de obter uma percepção mais clara acerca dos pontos fracos e fortes do algoritmo para então melhorar sua performance.

Desta forma, além da utilização de diferentes medidas para avaliar o desempenho de um algoritmo, como informado na seção 2.4.5, outras técnicas podem ser utilizadas para avaliar o aprendizado dos modelos ao longo do tempo ou à medida que são acrescentos novos dados no treinamento.

Uma destas técnicas consiste em gerar um gráfico de curvas de aprendizado: que é a representação do erro associado a determinadas etapas do processo de aprendizagem, fornecendo uma representação matemática do aprendizado à medida que uma tarefa se repete [13].

Estas curvas podem ser utilizadas para verificar se o algoritmo opera corretamente ou até mesmo para aprimorar sua performance ou diagnosticar se o aprendizado contém viés ou variância.

O gráfico pode ser montado atribuindo a variação do número de dados utilizados no conjunto de treino no eixo das abscissas e o erro associado no eixo das ordenadas. Para variar os dados, basta subdividir o conjunto de treino para, por exemplo, 10, 20, 30, ..., até 100% do total de exemplos deste conjunto. Já no eixo de erro, as funções de custo do treino e da validação cruzada podem ser representadas a fim de avaliar suas nuances.

Em muitos casos, quanto menos exemplos houver no conjunto de treino, mais fácil será realizar a predição daquele conjunto, isto é, menor será o erro. Por exemplo, para 1, 2 ou 3 exemplos, é provável que uma função simples de hipótese de grau 1 ou 2 já represente bem aqueles poucos dados. Por outro lado, com poucos exemplos utilizados no treino, o erro da validação cruzada será alto, isto é, é provável que a generalização do modelo não será bem estimada. Por esses motivos, nos gráficos a seguir, para um número baixo de exemplos, o erro do treino começa pequeno e na validação o erro começa alto. Seguem abaixo alguns casos possíveis e suas interpretações.

Alto Viés (*underfitting*)

As curvas de treino e de validação serão altas e próximas, isto é, ambas terão valores de erro altos, conforme é possível verificar na Figura 2.1, em que a medida F1 nos conjuntos de treino e de validação é igual a 0.62. O indicativo é de que há relativamente muitos dados de treino para poucos parâmetros a serem aprendidos a fim de adequar a hipótese aos dados, ou os hiperparâmtros não foram ajustados. Nesse caso, é bastante provável que obter mais dados não implicará na melhora de performance do modelo.

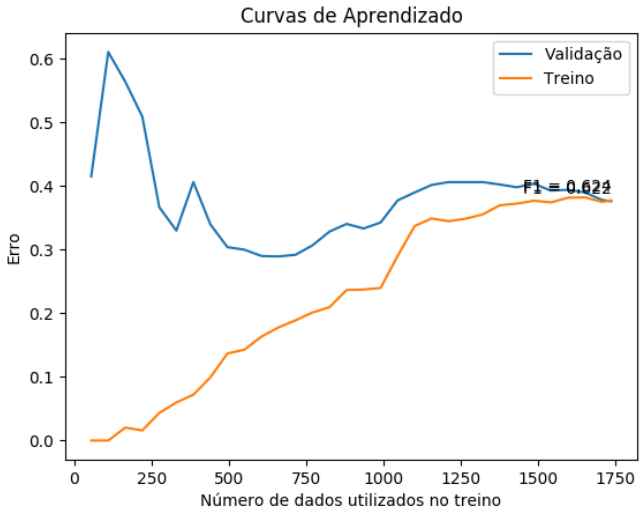


Figura 2.1: Curvas de aprendizado apresentando *underfitting*

Alta Variância (*overfitting*)

Neste caso a hipótese se ajusta bem demais ao conjunto de treino, isto é, possui baixo erro ao predizer aqueles dados, mas não mais que aqueles dados, de modo que o erro de validação será alto, visto que dados fora do conjunto sobre o qual o modelo foi treinado não são abrangidos pela hipótese, assim, o modelo falhará na generalização dos seus resultados. Nesse caso, é possível que aumentar o conjunto de dados possa melhorar o aprendizado, pois, já que o modelo se aproxima bastante do conjunto de treino, caso este conjunto represente cada vez melhor a realidade dos dados, pode ser que ele já tenha a capacidade de que necessita para aprender o padrão mais próximo do real, bastando treinar sobre mais exemplos para aprender novos padrões, melhorando sua generalização. Mas enquanto houver *overfitting*, as curvas de erro de treino e de validação serão distantes uma da outra e especialmente a curva de treino terá um valor de erro bastante baixo, próximo ou igual a zero, conforme se observa na Figura 2.2.

Sem Anomalias Aparentes

Um bom aprendizado é obtido quando as curvas estão em um nível entre o alto viés e a alta variância[5] e pode ser identificado quando ambos os erros de validação e de treino são baixos e decrescem a um ponto de estabilidade com um espaço pequeno entre as curvas e, em geral, o erro no conjunto de treino é menor que o do conjunto de validação. A Figura 2.3 representa este caso.

[5] `https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-perform`

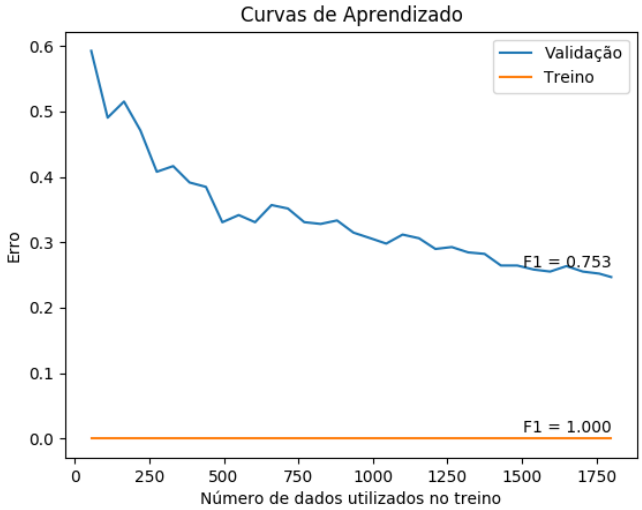


Figura 2.2: Curvas de aprendizado apresentando *overfitting*

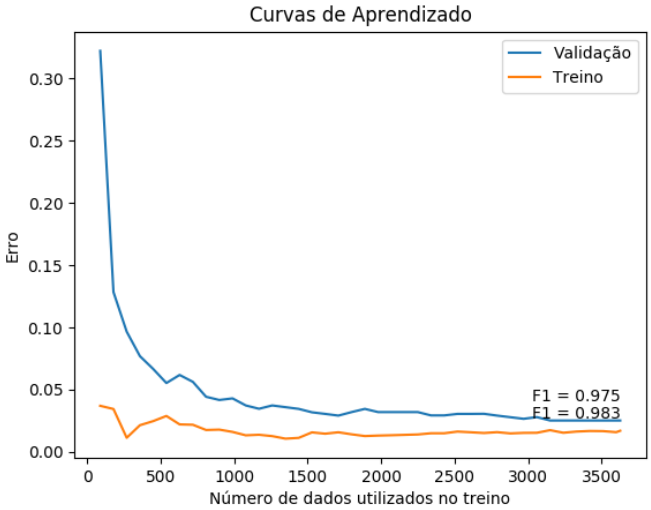


Figura 2.3: Curvas de aprendizado de um modelo que não apresenta anomalias aparentes

Caso um modelo apresente anomalias em seu processo de aprendizagem, há algumas medidas possíveis para atenuá-las, conforme as opções a seguir:

- Obter mais exemplos de treino: é uma alternativa a se considerar quando o modelo apresenta *overfitting*. Isso porque, neste caso, o modelo já se ajusta bem demais aos dados, então coletar mais dados pode levar o modelo a, de fato, aprender novas nuances. Por outro lado, caso o modelo tenha alto viés, essa opção pode não surtir muito efeito, já que o modelo, atualmente, não é capaz de aprender nem mesmo os dados de treino, provavelmente porque o modelo é simples demais para a complexidade do padrão dos dados.
- Aumentar o conjunto de características utilizadas: pode ser uma ação efetiva caso o modelo apresente *underfitting* pelo fato de as características atuais não representarem bem os dados, ou caso haja *overfitting*, pois, neste caso, as novas características poderiam auxiliar o modelo a aprender novos padrões, já que ele mostra potencial para isto.
- Reduzir o conjunto de características utilizadas: pode ser efetivo caso o modelo apresente *underfitting* por carecer em capacidade de correlacionar as características atuais por elas serem complexas demais para o algoritmo escolhido. Assim, reduzir a dimensionalidade do vetor de características de uma forma inteligente, por exemplo, utilizando Ganho de Informação, pode facilitar o aprendizado.
- Decrementar lambda: visto que este parâmetro, lambda, intervém na influência (pesos) dos parâmetros de uma hipótese, decrementá-lo significa aumentar o peso desses parâmetros aprendidos na hipótese, o que pode auxiliar no caso de haver alto viés. Este parâmetro é chamado também de parâmetro de regularização e ele pode funcionar causando o efeito oposto deste que foi explicitado acima, dependendo da implementação.
- Incrementar lambda: consequentemente ao argumento do item anterior, o incremento de lambda reduzirá a influência dos parâmetros treinados e isso é útil quando há *underfitting*, pois a hipótese do modelo, neste caso, precisa ser suavizada e este é um caminho para reduzir a proeminência de algumas características textuais.

2.5 Trabalhos Correlatos

Tendo em vista as taxas de mensagens nocivas ou inconvenientes que trafegam nos sistemas de email, a detecção automática e eficaz delas tem sido um alvo de pesquisa.

O trabalho *A Comprehensive Survey for Intelligent Spam Email Detection* [2] (2019), motivado pela disparidade entre as soluções atuais e as inovações utilizadas pelos spammers, reúne diversos artigos científicos deste domínio, de identificação de spam, abordando métodos de Inteligência Artificial (IA) e de aprendizado de máquina (ML) para tal finalidade. Os autores consideraram quatro partes do email para a tarefa de classificação:

1. Cabeçalho, que provêm informação de roteamento, como endereço IP do receptor e do transmissor.
2. Envelope SMTP, contendo identificações do domínio de origem e de destino.
3. A primeira parte do dado SMTP, com campos "de", "para", data e assunto.
4. A segunda parte do dado SMTP, contendo o corpo do email, incluindo conteúdo textual e anexos.

Afirma-se que as duas últimas partes do emails são as mais comumente utilizadas, enquanto informações como endereço IP e demais campos SMTP são usados com menos frequência. Além disso, identificar spam pela fonte, especialmente usando o campo de endereço IP, é efetivo até certo ponto, pois não é fácil para os spammers forjar este campo, de modo que aplicar esta técnica pode ser um complemento à tarefa de classificação.

O presente trabalho não faz uso das demais partes da mensagem (além do corpo do email), mas estes são alguns exemplos de possibilidades para aumentar a performance dos modelos treinados neste experimento.

O *survey* também dispõe um sumário acerca de certos algoritmos. De três selecionados, destacam-se as seguintes características:

- Naive Bayes: altamente escalável e trabalha bem com dados ruidosos ou em falta, mas é ineficiente ao lidar com datasets desbalanceados e supõe que as características textuais são independentes umas das outras.
- KNN: robusto e efetivo contra dados ruidosos, porém é extremamente sensível a *outliers*[6] e pode ser muito custoso encontrar o valor ótimo de K.
- SVM: é uma boa opção para datasets desbalanceados, pequenas alterações não alteram muito seu desempenho, fazendo-o com que generalize bem os dados e é resistente contra overfitting. Por outro lado, escolher os melhores hiperparâmetros pode ser complexo.

[6]Outliers são dados que se distanciam de forma anormal dos demais valores de uma amostra aleatória, segundo `https://www.itl.nist.gov/div898/handbook/prc/section1/prc16.htm`.

Observa-se que são utilizadas, em grande parte, as métricas precisão e acurácia para avaliação dos modelos, mas não combinadas com *recall* ou *F1-score*, dessa forma o desempenho geral dos modelos é informado, mas isso pode ocultar certas características, especialmente em casos em que os datasets são desbalanceados, o que é bastante frequente.

Procurou-se no *survey* trabalhos que exploram partes dos dados para além do corpo do email, como o endereço IP, mas alguns não são disponibilizados gratuitamente. Apesar disso, segundo [10], a técnica mais bem-sucedida na tarefa de detecção de spam é aquela baseada no conteúdo do email (parte 4 do email) e alguns dos algoritmos utilizados para este fim são: filtros bayesianos, SVM, KNN, redes neurais, AdaBoost, entre outros.

Um dos trabalhos evidenciados, *Word sense disambiguation for spam filtering* [3], utiliza a técnica de desambiguação do sentido de palavras (WSD) para aumentar a performance de modelos treinados utilizando a parte 4 do email. É informado que foi obtido aumento de performance de 2% até 6% ao aplicar esta técnica e pontua-se, também, que não é esclarecida a posição em que os novos termos são acrescentados ao texto.

É mencionado que o sistema WSD utilizado oferece a opção de habilitar ou não a desambiguação de termos utilizando o *WordNet*[7] quando o sistema não é capaz de realizar uma predição do sentido. Neste mesmo artigo [3], observou-se que o *recall* dos modelos Naive Bayes diminuiu substancialmente no dataset TREC, o que revela um ponto fraco destes algoritmos para classificar datasets maiores e com maior diversidade de temas comparado ao dataset Ling-Spam.

Foi buscado utilizar redução de dimensionalidade por meio de Ganho de Informação antes de realizar os experimentos com todas as palavras disponíveis. Isso geralmente é feito para reduzir a carga de processamento, mas é importante verificar antes os efeitos nos resultados, pois também é utilizado em casos de atenuação do *overfitting*. Portanto, analisar antes se há ou não overfitting é uma abordagem mais segura. Além disso, foi utilizada a medida ROC, que é uma boa medida para datasets balanceados [14]. Para o caso de datasets desbalanceados, a medida F1, especialmente quando há uma taxa menor de classes positivas (neste caso, spam), pode ser um melhor indicador de performance [14]. É dito que foi calculada a medida F1, mas não são mostrados seus resultados, ainda assim, é possível aferi-la a partir dos dados de precisão e *recall* informados.

O artigo [15] apresenta resultados de modelos treinados sobre o dataset SpamAssassin. Além do corpo do email, tabém são usados os campos “from” e “subject”, que já mostraram auxiliar na tarefa de classificação em outros experimentos [2]. Os conjuntos de treino e teste utilizados correspondem, respectivamente, a 63% e 37% de um total de 6000 dados. O pré-processamento consistiu em construir um vocabulário, aplicando a

[7]https://wordnet.princeton.edu/

codificação *one-hot* para cada palavra. Foram removidas palavras comuns (*stop words*) e as menos frequentes, então foram selecionadas as 100 palavras mais frequentes dos spams como características. Publicou-se as medidas de precisão, *recall* e acurácia. A partir das duas primeiras medidas fornecidas, afere-se os seguintes resultados de *F1-score* para cada algoritmo: 0.991 para o Naive Bayes, 0.941 para o SVM e o KNN, 0.918. Este último apresentou claramente a menor medida de precisão. O Naive Bayes atingiu a melhor pontuação dentre todos os algoritmos utilizados, sendo que, além dos três citados, também foram treinados os classificadores: Artificial Immune System [16], Rough sets [17] e Redes Neurais.

Por fim, o *survey* [2] indica a necessidade de modelar filtros capazes de acompanhar as novas técnicas que *spammers* frequentemente desenvolvem para esquivar seus ataques dos algoritmos de detecção de spam. Uma técnica utilizada para modelos aprenderem novos padrões constantemente é o aprendizado contínuo[8], o que pode auxiliar algoritmos a alcançar o objetivo citado. A presente pesquisa busca aplicar técnicas de acompanhamento do desenvolvimento dos modelos, como a análise de curvas de aprendizado, o que pode contribuir da seguinte forma: verificar o padrões de desenvolvimento de um modelo em um conjunto limitado de dados pode prover uma estimativa do desempenho deste modelo em um cenário de aprendizado contínuo. A seção 3.5 inclui observações neste assunto a partir dos resultados obtidos.

[8] `https://www.oreilly.com/radar/why-continuous-learning-is-key-to-ai/`

Capítulo 3

Proposta de Solução

Neste capítulo é apresentada a solução proposta de treinamento de diferentes modelos capazes de classificar cada email como sendo spam ou legítimo, comparando os resultados da aplicação de diferentes técnicas de pré-processamento e a adição de novas características ao corpo textual do email, para além da vetorização de palavras.

3.1 Datasets

Foram utilizados quatro conjuntos de dados que reúnem emails rotulados em duas classes: spam ou ham[1]. São eles: Enron[2], SpamAssassin[3], Ling-Spam[4] e 2005 TREC Public Spam Corpus[5]. A Tabela 3.1 resume atributos de cada dataset que foram utilizados neste trabalho.

Dataset	Total de emails	Spam	Ham	Spam/Ham (%)
Enron 3k	3000	1500	1500	50 / 50
Enron	20000	10000	10000	50 / 50
SpamAssassin 3k	3000	500	2500	17 / 83
SpamAssassin	6046	1896	4150	31 / 69
Ling-Spam	2893	481	2412	17 / 83
TREC-05 3k	3000	2463	537	82 / 18
TREC-05	20000	14319	5681	72 / 28

Tabela 3.1: Resumo dos datasets de emails rotulados, com a relação spam/ham

Os maiores datasets utilizados neste trabalho foram truncados em vinte mil emails devido aos recursos computacionais disponíveis. A proporção original de spam e ham dos

[1]Ham é um email legítimo, isto é, não é considerado spam.
[2]`http://nlp.cs.aueb.gr/software_and_datasets/Enron-Spam/index.html`
[3]`http://spamassassin.apache.org/old/publiccorpus/`
[4]`http://nlp.cs.aueb.gr/software.html`
[5]`https://plg.uwaterloo.ca/~gvcormac/treccorpus/`

subconjuntos Enron não foi mantida, bem como do TREC-05, mas gerou maior diversificação para observar o aprendizado dos modelos em distindas condições.

A relação spam/ham do Ling-Spam, por exemplo, é semelhante à relatada em um cenário real da taxa observada de spams trafegada. Apesar disso, esta proporção varia anualmente[6].

Com respeito ao SpamAssassin, o corpus é composto de nove pastas que foram acrescentadas ao dataset entre 2002 e 2006 e cada uma separa os emails ham em categorias de dificuldade de classificação entre fácil e difícil. A dificuldade é maior quando um ham se parece muito com um spam, como quando há conteúdo HTML, ou trechos HTML incomuns, texto colorido e frases que se assemelham às comumente encontradas em spam, por exemplo. No SpamAssasin 3k foram utilizadas apenas as pastas 20030228_easy_ham e 20030228_spam. No dataset completo foram adicionadas as pastas 20030228_easy_ham_2, 20030228_hard_ham e 20050311_spam_2.

3.2 Pré-processamento

Foi removido o cabeçalho de todos os emails, apesar de sua utilidade [18], com a finalidade de utilizar apenas o conteúdo do corpo do email para classificação. Supõe-se que adicionar o cabeçalho pode trazer resultados ainda superiores e há diversas técnicas para fazê-lo [3], que não são exploradas neste trabalho, portanto a ideia deste experimento é realizar o aprendizado apenas por meio do corpo textual.

Dos quatro datasets, apenas o Ling-Spam não é disponibilizado sem qualquer pré-processamento. Ele não possui, por exemplo, arquivos anexados e tags HTML e todas as palavras estão em letra minúscula, ou seja, algumas características textuais são removidas, impossibilitando o uso delas.

Da primeira etapa do experimento até a terceira foi utilizado o mesmo tipo de pré-processamento em todos os datasets: transformação de todas as letras para minúsculas, remoção de tags HTML, substituição de números para o token "number", substituição de endereços web para "httpaddr", emails para "emailaddr", símbolo "$" para "dollar", remoção de caracteres não alfanuméricos e, por fim, *stemming* (utilizando a classe *nltk.stem.PorterStemmer*[7]). Para a vetorização do texto, foi utilizado um vocabulário estático de 1899 palavras[8] aplicando apenas a codificação one-hot, isto é, para cada palavra presente no email, seu valor é igual a 1, caso contrário, 0.

[6] `https://www.statista.com/statistics/420391/spam-email-traffic-share/`

[7] `https://www.nltk.org/howto/stem.html`

[8] O vocabulário completo encontra-se em `https://github.com/nmschumacher/tcc`, arquivo *octave-vocab.txt*. Ele foi adquirido do curso *Machine Learning*, da plataforma *Coursera* `https://www.coursera.org/learn/machine-learning`.

Apenas na quarta etapa foram aplicadas distintas formas de pré-processamentos e adição de novas características dos textos ao vetor de *features*. Foi utilizado TF-IDF com variação de argumentos, como número máximo de características, remoção ou não de stop-words, remoção ou não de acentos e frequência mínima de palavras encontradas nos textos. Nesta etapa foram adicionadas características específicas como: contagem da maior sequência de palavras em letras maíusculas, contagem de URLs e de determinados símbolos (“!”, “?”, “$”, “%”), contagem de tags HTML e de POS tags (verbos, advérbios e adjetivos, por meio da biblioteca nltk.pos_tag[9]) e desambiguação de palavras (biblioteca wrapperWSD[10]).

O processo de desambiguação foi realizado da seguinte forma: aplicou-se o método wsdNLTK() da classe WrapperWSD() sobre cada texto. Este algoritmo retorna uma lista de termos similares a determinadas palavras do texto (não são todas que possuem palavras associadas). Desta lista, variou-se a quantidade de termos relacionados, truncando os resultados retornados para 1, 2, 3, 5 e até 7 termos máximos, adicionando-os logo após a palavra original. Depois disso é aplicada a técnica de one-hot ou TF-IDF. As Tabelas A.2, A.1, A.3 e A.4 apresentam os resultados da aplicação desta técnica na etapa 4 dos experimentos, bem como de diversas outras técnicas aplicadas aos datasets truncados. Observe que para todas as características que não possuírem “TF-IDF” no início na coluna “Característica Adicionada” tiveram a aplicação de one-hot sobre o texto.

3.3 Treinamento dos Modelos

Foram treinados modelos de classificação supervisionada para cada dataset a partir de 3 algoritmos: SVM, Naive Bayes e KNN, todos da biblioteca scikit-learn[11].

Do algoritmo SVM, foram utilizados os de kernel linear, sigmoid e polinomial, sendo estes últimos com graus 2, 3, 4, 5 e 7, possibilitando a amplicação dos resultados encontrados em [19], onde foram utilizados os polinômios de grau 1, 2 e 3, sendo que, como informado, o de grau 1 performa da mesma forma que o de kernel linear. Onde houver apenas o nome “SVM”, sem especificar o kernel, admite-se que se trata do linear.

Dos modelos Naive Bayes, foram utilizadas as variações de distribuição multinomial e gaussiana, utilizando, respectivamente, os parâmetros *alpha* e *var_smoothing* para regularização.

Já os modelos KNN treinados foram aqueles com valores de K iguais a 1, 3, 5 e 10, pois são valores próximos aos utilizados em [3] e incluem aqueles recomendados por [20], sendo variado, também, o cálculo do peso dos exemplos vizinhos, atribuindo, em uma

[9] `https://www.nltk.org/api/nltk.tag.html`
[10] `https://pypi.org/project/wrapperWSD/`
[11] `https://scikit-learn.org/stable/supervised_learning.html#supervised-learning`

versão, pesos iguais para todos os vizinhos e, em outra, pesos menores aos exemplos mais distantes. A esta última é adicionada a palavra "Distance" para diferenciá-la daquela de pesos uniformes.

Totalizam-se, assim, 16 modelos que foram treinados para cada dataset.

A metodologia desta pesquisa visa evitar a otimização prematura [4], isto é, primeiro objetiva adquirir evidências, para somente então tomar decisões a respeito de que pontos trabalhar para otimizar aquilo que pode estar causando uma baixa performance do modelo de aprendizagem. Dessa forma é possível acompanhar o aprendizado e as eventuais anomalias. Segue uma possível abordagem para cumprir tal objetivo:

1. Treinar de forma simples um algoritmo de aprendizagem e testá-lo no conjunto de validação e de testes.
2. Aplicar o diagnóstico de ML: representar graficamente as curvas de aprendizado avaliando quais ações poderão surtir maior efeito na performance.
3. Verificar erros manualmente: observar dados preditos incorretamente, procurando identificar que tipos de erros são mais frequentes, categorizando-os.
4. Aplicar técnicas de otimização conforme as observações realizadas nos passos 2 e 3.
5. Repetir os passos 2 a 4 até alcançar os objetivos de desempenho estabelecidos no trabalho.

Buscou-se aplicar tal abordagem ao dividir os experimentos em quatro etapas, daquela de menor complexidade para a mais complexa, conforme resumo apresentado na Tabela 3.2.

	Curvas de Aprendizado	Regularização	Datasets com a carga máxima	Adição de novas características
Etapa 1	✓			
Etapa 2	✓	✓		
Etapa 3	✓	✓	✓	
Etapa 4	✓	✓	✓	✓

Tabela 3.2: Resumo de ações tomadas em cada etapa

Mais especificamente, as seguintes medidas foram tomadas em cada etapa:

1. Etapa 1: todos os datasets com mais de 3 mil emails foram truncados para este valor máximo a fim de realizar rapidamente uma primeira onda de treinamentos e obter os primeiros resultados e gráficos para observar o comportamento de cada modelo utilizando os hiperparâmetros padrões da biblioteca.

2. Etapa 2: os mesmos conjuntos de dados são utilizados, mas desta vez é aplicada a regularização, isto é, há uma seleção do parâmetro que penaliza os pesos encontrados para cada parâmetro aprendido. Apenas os modelos KNN não puderam fornecer resultados nesta etapa, visto que a biblioteca utilizada não provê tal parâmetro.

3. Etapa 3: semelhante à etapa 2, porém utilizando os datasets maiores com a máxima carga de emails suportada, a fim de verificar o aprendizado dos modelos utilizando mais dados. Ao fim desta etapa também é gerado um documento para cada dataset contendo emails erroneamente classificados, utilizando o classificador de maior desempenho. A motivação é cumprir o passo 3 da abordagem metodológica, que possibilita a avaliação de quais possíveis features poderiam ser adicionadas ao modelo para melhor performance.

4. Etapa 4: execução da etapa 3, porém com adição de novas features aos modelos. Há etapas intermediárias, utilizando os datasets truncados e um número reduzido de modelos a fim de realizar diversos testes que demonstrem o desempenho de cada modelo à medida que cada nova feature é acrescentada (ver no apêndice as tabelas A.1, A.3, A.2 e A.4). Foram escolhidos dois conjuntos de features e pré-processamento que demonstraram maior ganho de performance para os resultados finais de cada modelo, aplicando-os aos datasets maiores.

Em todas as etapas foram gerados gráficos das curvas de aprendizado contendo as medidas F1 nos conjuntos de treino e de validação, além de tabelas com resultados de precisão, recall e F1-score no conjunto de teste, como é explicado na seção 3.4.

É importante ressaltar que entre as etapas 3 e 4 foi realizado um passo adicional para selecionar o tipo de normalização de certas features e o tipo de pré-processamento dos dados, utilizando quatro algoritmos: SVM linear, Naive Bayes Multinomial e KNN com K = 1 e o Distance com K = 5, nos quatro datasets truncados. A Tabela A.5 descreve alguns termos utilizados nesta etapa e os resultados encontram-se nas tabelas tabelas A.1, A.2, A.3 e A.4. Foram utilizadas 62 configurações combinando features e técnicas de pré-processamento para cada modelo em cada dataset, totalizando 248 resultados por dataset. Ao fim deste passo, foram selecionadas apenas duas configurações em que cada uma abrangeria o melhor possível os resultados de pelo menos dois dos quatro algoritmos, buscando, assim, configurações que não trouxessem vantagens para um algoritmo específico em detrimento dos demais, para, então, aplicá-las em todos os modelos nos datasets com a carga máxima de dados suportada na Etapa 4. Dentre as duas configurações selecionadas por dataset neste passo, a Etapa 4 apresenta os melhores resultados obtidos.

3.4 Avaliação

Obter muitas medidas de avaliação pode causar confusão e dúvida quanto ao modelo de maior desempenho. Por isso escolheu-se apenas uma medida como sendo a principal, assim, a F1-score obtida sobre o conjunto de testes é tida como o resultado final de cada etapa e o resultado final dos experimentos.

Esta escolha está em consonância com os objetivos estabelecidos, dado o domínio da pesquisa, isto é, a classificação de emails como spam ou ham: a medida F1 informa um valor que correlaciona outras medidas e será alta se, e somente se, tanto o valor de precisão quanto de recall forem altos. Isso evita a escolha de uma medida que mascare possível déficit de aprendizagem dos modelos.

Além disso, devido à natureza dos dados, é importante, como ressaltado na introdução (1), uma medida que informe um equilíbrio entre os resultados de falso positivo e falso negativo, visto que ambos têm a capacidade de prejudicar os usuários dos serviços de email.

Todos os datasets foram divididos em 3 subconjuntos: treino, validação e teste, sendo a proporção, respectivamente, 60%, 20% e 20% do conjunto total de dados, mantendo a proporção spam/ham em cada subconjunto[12].

Os parâmetros de regularização da Etapa 2 selecionados foram aqueles que minimizaram a medida F1 sobre o conjunto de validação, utilizando um algoritmo desenvolvido. Já nas etapas 3 e 4 passou-se a utilizar o Grid Search, pela simplicidade de aplicação, e tanto o conjunto de treino quanto o de validação foram utilizados pelo algoritmo aplicando a técnica de *3-fold cross-validation* para escolha do melhor parâmetro, utilizando uma lista de valores. Por fim, todos os modelos com o parâmetro selecionado (etapas 2 a 4) ou com o padrão (etapa 1) foram treinados utilizando o cojunto de treino e então foram realizadas as predições nos conjuntos de teste, obtendo, assim, o resultado final.

As curvas de aprendizado foram obtidas da seguinte forma: após a escolha do parâmetro de regularização (para os casos em que isso se aplica), incrementou-se gradualmente a quantidade de dados de treino utilizados no treinamento de cada modelo, registrando, para cada *fit()*, o resultado da medida F1 sobre a parte do conjunto de treino utilizada, bem como a medida obtida sobre o conjunto total de validação. Cada par de F1-score de treino e validação foi armazenado subtraindo-se 1 deste valor, obtendo, assim, o erro de treino e o de validação, isto é, os pontos das curvas de aprendizado, então foi utilizada a biblioteca *matplotlib*[13] para, por fim, gerar as curvas.

[12] A função sklearn.model_selection.train_test_split foi utilizada duas vezes para poder separar em 3 subconjuntos, com a opção random_state = 42, a fim de que haja a possibilidade de reproduzir os experimentos.

[13] `https://matplotlib.org/tutorials/introductory/pyplot.html`

3.5 Resultados e Discussão

Visto que muitos dados foram coletados e para melhor visualização dos resultados, optou-se por ressaltar nesta seção a principal medida de avaliação utilizada no trabalho: F1-score sobre o conjunto de teste. Outras medidas, como precisão e recall e mais gráficos de cada etapa poderão ser encontrados no apêndice deste trabalho. Todos os gráficos de cada etapa e o próprio código desenvolvido podem ser acessados em `https://github.com/nmschumacher/tcc`.

A Tabela 3.3 apresenta os resultados dos modelos agrupados por dataset. Os valores destacados são os maiores do dataset e da etapa em questão, elencando medidas F1 que são um número até 0.005 menores que a maior medida alcançada. A Etapa 2 não se aplica aos modelos KNN, pois não possuem parâmetro de regularização.

Modelo	Dataset	Etapa 1	Etapa 2	Etapa 3	Etapa 4	E4 - E3
SVM Linear	**Enron**	**0.949**	0.945	**0.962**	**0.995**	0.033
SVM Sigmoid		0.884	0.915	0.934	**0.993**	0.06
SVM poly 2		0.902	0.922	**0.959**	**0.995**	0.036
SVM poly 3		0.838	0.893	0.956	**0.995**	0.038
SVM poly 4		0.796	0.839	0.952	**0.993**	0.042
SVM poly 5		0.754	0.861	0.935	**0.993**	0.058
SVM poly 7		0.715	0.757	0.847	**0.99**	0.142
GaussianNB		0.801	0.828	0.891	**0.99**	0.099
MultinomialNB		**0.948**	**0.952**	**0.963**	0.988	0.025
KNN K=1		0.872		0.922	0.921	-0.001
KNN K=3		0.87		0.905	0.918	0.012
KNN K=5		0.873		0.889	0.974	0.085
KNN K=10		0.842		0.864	0.985	0.121
KNN K=3 Dist		0.877		0.91	0.92	0.009
KNN K=5 Dist		0.884		0.899	0.976	0.077
KNN K=10 Dist		0.863		0.892	0.983	0.091
SVM Linear	**Spam Assassin**	**0.942**	0.935	**0.953**	**0.967**	0.014
SVM Sigmoid		0.928	0.927	0.89	0.961	0.071
SVM poly 2		0.831	0.922	0.946	**0.97**	0.024
SVM poly 3		0.68	0.852	0.931	**0.965**	0.033
SVM poly 4		0.614	0.706	0.893	0.964	0.072
SVM poly 5		0.46	0.662	0.854	0.96	0.106
SVM poly 7		0.373	0.438	0.734	0.942	0.208
GaussianNB		0.546	0.889	0.934	0.951	0.017

Modelo	Dataset	Etapa 1	Etapa 2	Etapa 3	Etapa 4	E4 - E3 [14]
MultinomialNB		0.933	**0.942**	**0.95**	0.934	-0.016
KNN K=1		0.723		0.865	0.924	0.059
KNN K=3		0.64		0.82	0.937	0.117
KNN K=5		0.67		0.814	0.942	0.128
KNN K=10		0.741		0.811	0.931	0.119
KNN K=3 Dist		0.645		0.832	0.945	0.113
KNN K=5 Dist		0.678		0.823	0.948	0.125
KNN K=10 Dist		0.726		0.815	0.948	0.133
SVM Linear		0.96	**0.971**		**0.983**	0.012
SVM Sigmoid		0.873	0.883		0.971	0.087
SVM poly 2		0.848	0.947		0.958	0.01
SVM poly 3		0.711	0.881		0.918	0.038
SVM poly 4		0.614	0.803		0.846	0.043
SVM poly 5		0.5	0.6		0.602	0.002
SVM poly 7		0.355	0.456		0.635	0.179
GaussianNB	**Ling-**	0.502	0.935		0.936	0.002
MultinomialNB	**Spam**	**0.966**	0.956		0.965	0.009
KNN K=1		0.807			0.87	0.063
KNN K=3		0.829			0.946	0.117
KNN K=5		0.819			0.94	0.122
KNN K=10		0.745			0.965	0.22
KNN K=3 Dist		0.829			0.965	0.136
KNN K=5 Dist		0.834			0.959	0.125
KNN K=10 Dist		0.819			0.965	0.146
SVM Linear		**0.981**	**0.986**	**0.987**	**0.992**	0.004
SVM Sigmoid		0.971	0.971	0.941	0.978	0.037
SVM poly 2		0.968	0.975	**0.988**	**0.99**	0.002
SVM poly 3		0.946	0.963	0.979	**0.99**	0.011
SVM poly 4		0.933	0.945	0.965	**0.99**	0.025
SVM poly 5		0.923	0.936	0.948	0.979	0.031
SVM poly 7		0.92	0.928	0.929	0.976	0.047
GaussianNB	**TREC-**	0.786	0.971	0.901	0.968	0.067
MultinomialNB	**05**	0.936	0.963	0.933	0.961	0.028
KNN K=1		**0.976**		0.976	0.97	-0.006

[14]Visto que o Ling-Spam é o menor dataset, ele não foi ampliado na etapa 3, portanto, as medidas da coluna E4 - E3, na realidade, correspondem à Etapa 1 (E4 - E1).

Modelo	Dataset	Etapa 1	Etapa 2	Etapa 3	Etapa 4	E4 - E3 [14]
KNN K=3		0.967		0.97	0.955	-0.015
KNN K=5		0.96		0.965	0.944	-0.021
KNN K=10		0.956		0.954	0.934	-0.02
KNN K=3 Dist		0.967		0.972	0.958	-0.014
KNN K=5 Dist		0.961		0.968	0.949	-0.018
KNN K=10 Dist		0.958		0.961	0.94	-0.021

Tabela 3.3: F1-score de cada etapa para cada dataset

Da Etapa 1 para a 2, observa-se melhora do desempenho de praticamente todos os algoritmos que possuem parâmetro de regularização, com exceção daqueles cujo argumento padrão já era igual ou muito próximo ao valor do melhor parâmetro encontrado. Além disso, há redução da medida F1 no conjunto de teste para o SVM linear nos datasets Enron e SpamAssassin e para o MultinomialNB no Ling-Spam, o que pode ser explicado pelo fato de que o método para encontrar o melhor parâmetro utiliza o conjunto de validação e isso nem sempre refletirá em uma melhora de resultado no conjunto de teste. Apesar disso, esta técnica se mostrou eficaz para generalizar o aprendizado na maioria dos casos. Uma observação é a de que foi selecionado o maior valor possível (imposto em código) do parâmetro de regularização C para alguns modelos SVM de kernal polinomial e isso foi ajustado nas demais etapas, pois a seleção deste parâmetro se mostrou essencial especialmente neste kernel.

A Etapa 3, de expandir os datasets Enron, SpamAssassin e TREC-05, mostrou ser benéfica para grande parte dos algoritmos. Porém o SVM sigmoid decaiu no SpamAssassin e no TREC-05. Neste último também ambos os Naive Bayes decaíram. É interessante ressaltar que este dataset, TREC-05, é o que mais possui spam, comparado aos outros três, em número e em proporção, segundo a Tabela 3.1, com cerca de 72% dos emails sendo spam, o que pode indicar certa dificuldade de tais algoritmos para aprender padrões de email spam, característica esta que pode ter sido evidenciada ao ampliar este conjunto de um total de 3 mil para 20 mil emails.

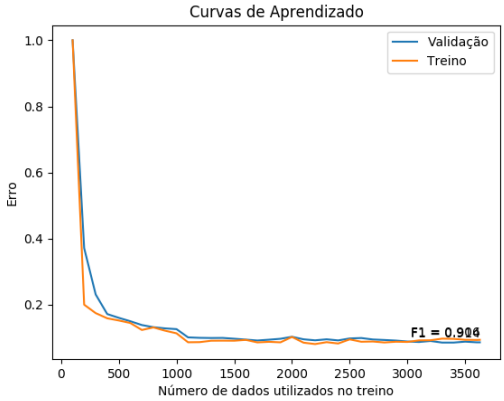
Entre as etapas 3 e 4 foi realizado um passo adicional, como expressado na seção 3.3, para aplicar TF-IDF variando parâmetros e para diversificar a normalização das novas features e observar o desempenho dos modelos ao acrescentar cada uma delas. Nota-se que aquelas não normalizadas, isto é, com valores fora da faixa padrão (entre 0 e 1), por vezes imprimiram melhores resultados, mas ao combinar diversas features não normalizadas o desempenho é prejudicado e é, em geral, pior que combinar as mesmas features normalizadas. Nota-se, também, que algumas delas não surtem efeito algum no dataset Ling-Spam, visto que ele é disponibilizado já com certo pré-processamento, como

informado na seção de pré-processamento 3.2 e, se este não fosse o caso, alguns resultados poderiam ser melhores neste dataset.

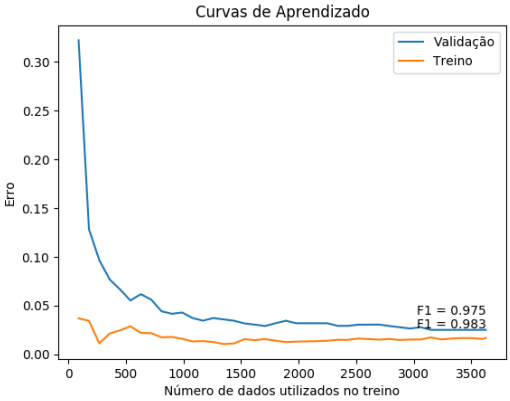
Adicionar novas features e selecionar o tipo de pré-processamento, na Etapa 4, também surtiu efeitos positivos na maioria dos casos, especialmente para todos os SVMs, que superaram seus próprios resultados com relação a todas as etapas anteriores e, em geral, alcançaram as maiores pontuações, mostrando serem modelos robustos: capazes de corresponder positivamente à expansão de dados e à adição de features. Apenas os de kernel polinomial de maiores graus não obtiveram resultados satisfatórios no dataset Ling-Spam nesta etapa, o que é rasoável, já que é o menor dataset e estes modelos mostraram que a ação de expandir o conjunto de dados é bastante eficaz para eles (perceba as tendências das curvas de aprendizado nos gráficos da Figura A.1).

É interessante observar a evolução do SVM sigmoid no SpamAssassin nesta etapa para além do simples aumento da medida F1: os gráficos da Figura 3.1 demonstram que ambas as curvas de erro de treino e de teste caíram na Etapa 4 ao aplicar TF-IDF com 3000 *features* em comparação com o pré-processamento utilizado anteriormente (*one-hot encoding* descrito na seção 3.2), indicando que sua capacidade de aprendizado foi mais bem aroveitada nesta nova configuração, embora o modelo não apresentasse anomalias na etapa anterior. Este resultado, como mencionado na seção 2.5, exemplifica a aplicabilidade da análise destes gráficos para selecionar um algoritmo que pode ser utilizado em um cenário de aprendizado contínuo: visto que um determinado tipo de pré-processamento, como dito, melhor aproveitou a capacidade de aprendizado de um modelo ao treinar com uma carga maior de dados, então, a partir dos gráficos apresentados, é provável que utilizar este pré-processamento em um cenário de aprendizagem contínua seja mais benéfico que aplicar a vetorização realizada anteriormente. Novos estudos podem ser feitos para maior precisão da análise das curvas de aprendizado e para reforçar a observação acima, por exemplo, utilizando regressão polinomial para prever a tendência da curva de validação.

Um outro resultado interessante deste mesmo modelo foi obtido no dataset Enron. A Figura 3.2 revela um resultado semelhante ao obtido no SpamAssassin, quando um determinado tipo de pré-processamento é realizado, aumentando a capacidade do modelo de aprender com mais dados, levando-o de uma curva de validação com um comportamento aparentemente assintótico (gráfico 3.2a) para uma curva de erro decrescente (gráfico 3.2b), reduzindo o erro e tornando-o capaz de aprender com mais dados. Mas ao adicionar novas features (gráfico 3.2c), é possível observar um padrão crescente de erro a partir do uso de, aproximadamente, 8 mil dados de treino. Se o treinamento fosse interrompido neste momento, provavelmente o resultado no conjunto de teste seria até mesmo superior àquele obtido. Assim, neste caso, as novas features reduziram a capacidade de aprendizado do modelo. Porém, em geral, nos modelos SVM de kernel polinomial ou linear, as novas



(a) Etapa 3, C = 0.33, F1 Validação = 0.914



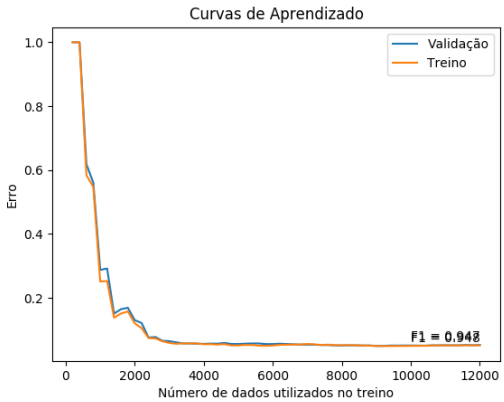
(b) Etapa 4, C = 1, TF-IDF com 3000 features, F1 Validação = 0.975

Figura 3.1: Curvas de aprendizado de modelos SVM sigmoid no dataset SpamAssassin

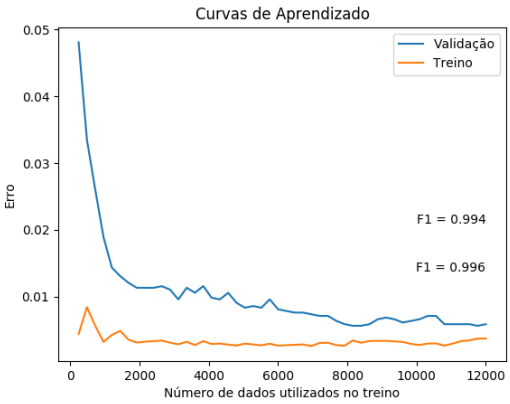
features aumentaram seu desempenho, inclusive é possível observar nos gráficos obtidos, como, por exemplo, na Figura 3.3.

Ainda acerca da Etapa 4, os modelos KNN decaíram apenas no TREC-05, todos eles, com relação a todas as etapas anteriores. Ou seja, mesmo havendo um esforço para selecionar a técnica de pré-processamento e as features que obtiveram melhor desempenho no passo entre as etapas 3 e 4 (ver Tabela A.4), estes modelos demonstraram um comportamento inesperado quanto à adição de novas features simultaneamente com a expansão do conjunto de dados, especialmente neste dataset. É possível verificar na Figura 3.4 que não foram introduzidos, por exemplo, ruídos na Etapa 4, mas aplicar TF-IDF e adicionar novas features apenas aumentou o erro em ambas as curvas de treino e validação, refletindo no aumento do erro no conjunto de teste. Por outro lado, no Enron, por exemplo, que é do mesmo tamanho, porém com uma proporção completamente balanceada de spam/ham, o resultado foi diferente: aplicar TF-IDF com remoção de acentos e de stop words (Figura 3.5b) introduziu ruidos, mas aplicar a mesma técnica sem remover stop words e adicionando as novas features (Figura 3.5c) reduziu significativamente os ruídos anteriormente vistos, bem como o erro de validação e, mais importante, de teste.

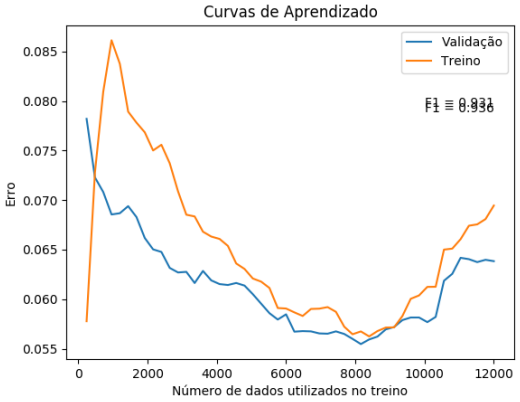
Vale ressaltar que a medida F1, como informado na seção de métodos de avaliação dos modelos 2.4.5, não leva em consideração a medida de verdadeiros negativos. Ou seja, os emails que corretamente foram classificados como sendo legítimos não são contados. Os modelos KNN, em muitos casos, apresentaram medida de *recall* superior à de precisão por uma margem notável, indicando uma baixa taxa de falso negativo, então é possível que estes modelos tenham melhorado seu desempenho na Etapa 4, mas seria necessário obter os valores de verdadeiros negativos. Assim, a medida F1, em particular no dataset TREC-05, pode ter ocultado o desempenho destes e dos demais modelos, visto que este



(a) Etapa 3, C = 0.03, F1 Validação = 0.947

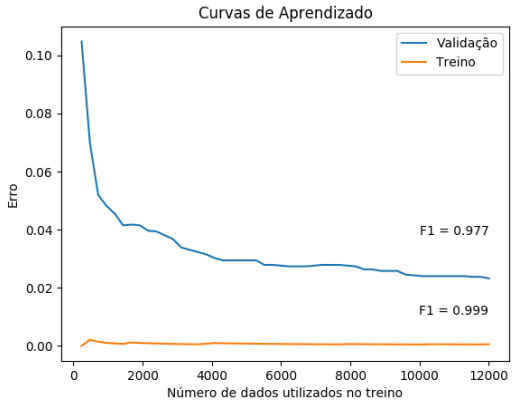


(b) Etapa 4, C = 1, TF-IDF sem acentos, com remoção de stop words, F1 Validação = 0.994

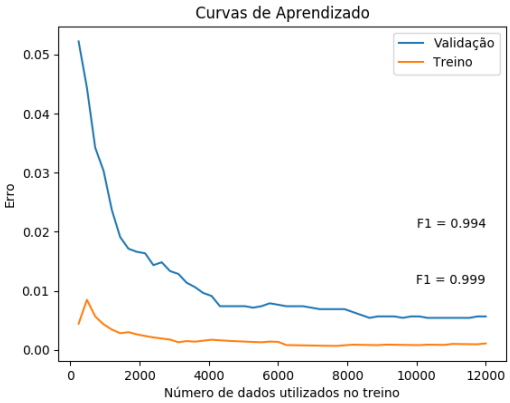


(c) Etapa 4, C = 0.03, TF-IDF com adição de todas as novas features, F1 Validação = 0.936

Figura 3.2: Curvas de aprendizado de modelos SVM sigmoid no dataset Enron

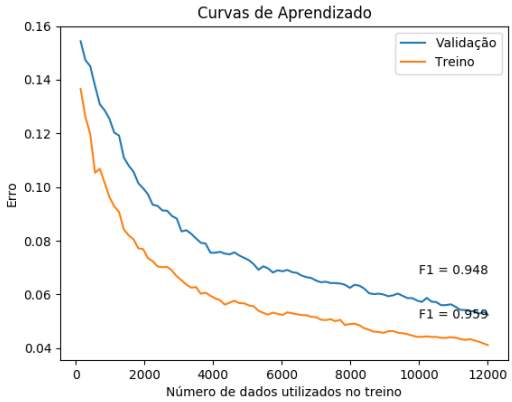


(a) Etapa 4, C = 3, TF-IDF com remoção de acentos e de stop words, F1 Validação = 0.977

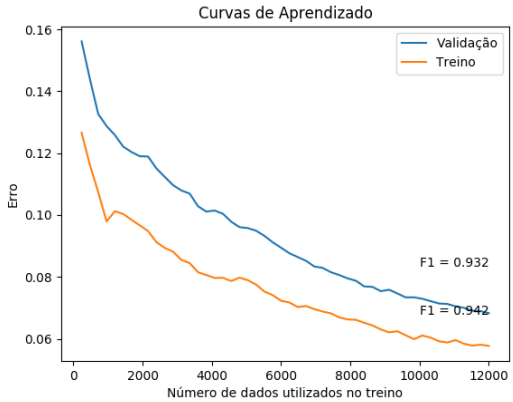


(b) Etapa 4, C = 30, TF-IDF com remoção de acentos e todas as features adicionadas, F1 Validação = 0.994

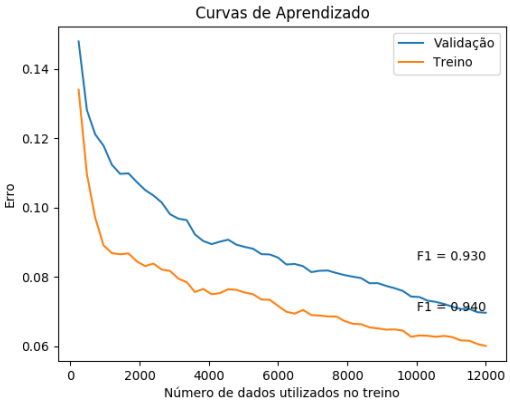
Figura 3.3: Curvas de aprendizado de modelos SVM polinomial de grau 3 no dataset Enron



(a) Etapa 3

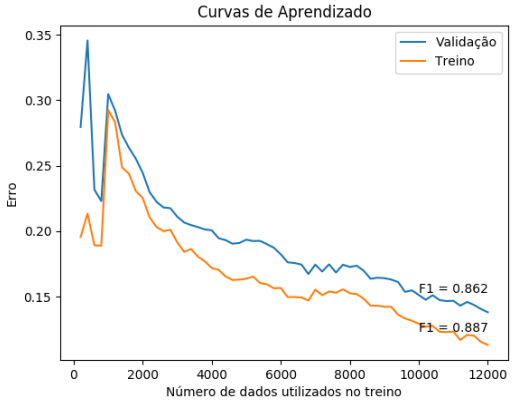


(b) Etapa 4, TF-IDF com *max_features = 1000*

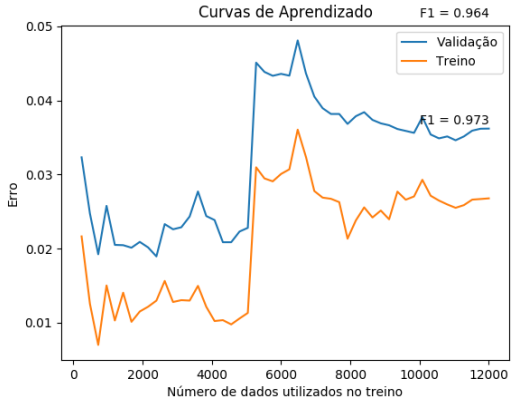


(c) Etapa 4, TF-IDF com todas as features adicionadas

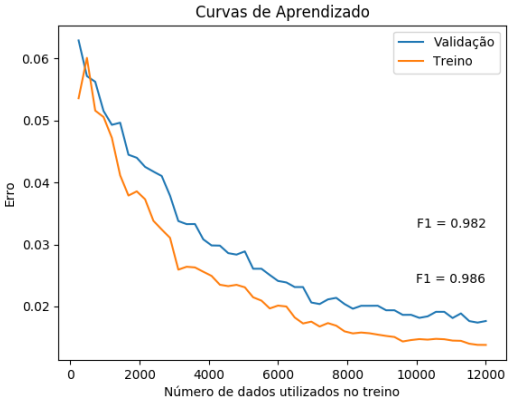
Figura 3.4: Curvas de aprendizado de modelos KNN com K = 10 no dataset TREC-05



(a) Etapa 3



(b) Etapa 4, TF-IDF com remoção de acentos e de stop words



(c) Etapa 4, TF-IDF com remoção de acentos e todas as features adicionadas

Figura 3.5: Curvas de aprendizado de modelos KNN com K = 10 no dataset Enron

é o dataset com maior porcentagem de dados da classe negativa, isto é, spam. Então, apesar de ter sido escolhida para todas as etapas, esta não foi uma boa escolha de método de avaliação sobre o desempenho dos modelos neste dataset.

Além disso, foi realizado um experimento à parte, inspirado nos resultados do trabalho [15]. Foi utilizado TF-IDF com limites de 50, 70, 100, 200, 500, 1000, 2000, 3000 e 5000 features, além de também deixar esta opção desativada (com o parâmetro *max_features = None*, no qual foram obtidas 16534 features), as letras foram transformadas para minúsculas e foi variada a opção de utilizar ou não a técnica de remoção de stop words fornecida pela biblioteca[15]. A melhor medida F1 encontrada utilizando ambos os Naive Bayes nestas condições foi de 0.964, com *max_features = None*, sem remoção de stop words, com o modelo Multinomial com *alpha = 1e-6*, e esta medida, 0.964, é superior às encontradas nas etapas apresentadas neste trabalho para este modelo, porém é ainda

[15] `https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html`, opção stop_words = "english".

muito distante da encontrada pelos pesquisadores em [15], que é de 0.991 e, ainda assim, ela é menor que a medida encontrada pelo SVM polinomial de grau 2 na Etapa 4 deste trabalho, modelo este que obteve o melhor resultado, com F1 igual a 0.97.

A razão disto pode ser, por exemplo, devido ao fato de eles utilizarem também as informações do cabeçalho: os campos "subject" e "from", ou por terem executado a codificação one-hot no lugar de TF-IDF ou até mesmo por utilizarem uma fração diferente do corpus disponibilizado (o que não é informado, não há como saber), visto que há pastas de diferentes versões do dataset[16], é preciso ignorar algumas delas devido ao versionamento realizado[17].

Em suma quanto ao desempenho dos modelos, nota-se que o algoritmo que mais se sobressai nas últimas duas etapas é o SVM de kernel linear, obtendo o maior valor na Etapa 4 com relação aos demais algoritmos e aos resultados anteriores (com exceção do SpamAssassin, em que o polinomial de grau 2 foi ligeiramente superior). Alguns polinomiais também atingiram a mesma medida no dataset Enron. O Naive Bayes Multinomial se mostrou bastante promissor no início, nas etapas 1 e 2, em datasets menores, inclusive atingindo medidas superiores ao SVM linear, mesmo depois da regularização deste, porém, ao aumentar o dataset, ele não acompanhou o ritmo de aprendizado do SVM e isso, em parte, foi previsto na análise das curvas de aprendizado dos modelos Naive Bayes: não era esperado seu aumento de performance, mas já era possível visualizar desde a etapa 2 que tais modelos estavam tendendo ao *underfitting*. Quanto aos modelos KNN, na maioria dos casos o aumento de dados foi benéfico, mas a tarefa de selecionar features e o tipo de pré-processamento se mostrou bastante trabalhosa e, por vezes, imprevisível de se saber seu efeito ao expandir o dataset. Este algoritmo se apresentou bastante sensível ao tipo de pré-processamento e de normalização e não se mostrou muito robusto, no sentido de demonstrar uma evolução consistente ao longo das etapas e, como citado em [2], ele é muito sensível aos *outliers*, fatos tais que talvez seriam atenuados caso fossem selecionados valores de K maiores.

Inclusive, quanto à sensibilidade do KNN e do Naive Bayes relativa às alterações das *features*, foram incluídos ao fim das tabelas A.1, A.2, A.3 e A.4 o valor mínimo, máximo, a média aritmética e o desvio padrão de F1-score dentre todas as pontuações obtidas por modelo a fim de observar o intervalo de desempenho possível apenas com as alterações implementadas (para visualizar apenas o desvio padrão, veja a Tabela 3.4). O maior desvio padrão observado nos quatro datasets foi evidentemente obtido para os algoritmos KNN e, em segundo lugar, para o Naive Bayes multinomial, comprovando a sensibilidade

[16]No site `https://spamassassin.apache.org/old/publiccorpus/`

[17]No presente trabalho, inclusive, tais pastas não foram removidas inicialmente, chegando a um total de 10744 emails neste dataset e o SVM linear atingiu uma medida F1 de 0.995 utilizando-as. Depois os experimentos foram refeitos.

destes algoritmos quanto às alterações nas *features* (como tipo de normalização e adição ou remoção de características). Já o SVM linear obteve, em todos os casos, os menores valores de desvio padrão.

Dataset	SVM	MultinomialNB	KNN K = 1	KNN K = 5 Dist
Enron 3k	0.017	0.017	**0.044**	**0.046**
SpamAssassin 3k	0.014	**0.085**	0.081	**0.095**
Ling-Spam	0.007	0.016	**0.042**	**0.055**
TREC-05 3k	0.003	**0.014**	0.007	**0.008**

Tabela 3.4: Desvio padrão de 62 medidas F1 obtidas de cada modelo por dataset entre as etapas 3 e 4

Acerca da aplicação da técnica de desambiguação do sentido das palavras (WSD), também verificado nas tabelas A.1, A.2, A.3 e A.4, notou-se que, dos quatro algoritmos treinados, o que obteve aumento de desempenho com maior frequência nas variações do WSD foi o MultinomialNB e, em seguida, o SVM linear, possibilitando aumento de até aproximadamente 1.7% da medida F1 com relação à medida sem aplicação de WSD e redução de até 1.3% nestes algoritmos nos piores casos. Os melhores resultados obtidos de todos os modelos somam uma média de 0.2% de melhora de performance (em alguns casos a melhor opção de WSD reduziu o desempenho), por outro lado, os piores resultados, em média, somam 1.4% de redução da performance dos modelos. Os modelos KNN obtiveram as piores performances, para alguns modelos constatou-se redução de 5% e até 8% do desempenho e na maior parte dos casos houve redução de F1. Portanto, dos resultados apresentados, esta técnica tem potencial para aumentar as predições, mas, depende do número de sentidos associados a cada termo, do algoritmo que será utilizado e da própria composição do dataset (como tamanho e temas recorrentes), tornando o resultado inesperado, pois é preciso buscar um número ideal de termos, se é que haverá. No trabalho [3] constatou-se melhora de desempenho na maioria dos casos, provavelmente porque foi utilizado um sistema diferente de desambiguação e foram variados diferentes parâmetros, indicando que esta técnica pode exigir maior dedicação para extrair os melhores resultados dela.

A análise das curvas de aprendizado mostrou ser uma técnica bastante efetiva para avaliar e prever o comportamento dos modelos quanto ao aumento do conjunto de dados e para verificar a propensão ao *overfitting* ou *underfitting*. Em alguns casos foi concluido que um modelo reduziria seu desempenho de classificação caso fosse tomada a medida de aumentar o conjunto de dados ou adicionar novas features, como nos modelos bayesianos, mas eles acabaram melhorando em alguns datasets e piorando em outros, demonstrando certa imprevisibilidade, diferentemente dos modelos SVM, que, em geral, corresponderam

confirmando as análises. Também foi difícil prever a resposta dos modelos KNN em alguns casos, mas na maioria deles as análises das curvas se concretizaram.

Notou-se que os SVMs polinomiais possuem grande capacidade de aprendizado, inclusive com potencial para ultrapassar o desempenho do linear à medida que se aumenta o dataset, o que pode ser observado nas tabelas, bem como e, especialmente, nas curvas de aprendizado. De acordo com os resultados de [19], estes modelos decaíram em desempenho à medida que se aumentou o grau do polinômio, o que também foi observado nos resultados do presente trabalho, porém a quantidade de dados pode compensar e, talvez, fazer com que tais modelos alcancem resultados ainda mais promissores.

Outra observação é a de que utilizar o mesmo tipo de pré-processamento e normalização de features para todos os algoritmos não fornecerá uma comparação justa do aprendizado dos modelos, visto que cada um calcula os pesos das features de maneira diferente e um determinado pré-processamento pode favorecer alguns modelos em detrimento de outros. Apesar disso, o SVM se mostrou mais flexível quanto a esta observação e os bayesianos e os KNNs mais sensíveis.

Por fim, a Tabela 3.5 apresenta trechos de alguns emails incorretamente classificados pelo SVM linear treinado na Etapa 4 no dataset TREC-05. É possível verificar a presença de grandes valores monetários nos spams, o que poderia ser uma nova feature a ser implementada para identificar estes casos.

#	Dataset	Classe Real	Predito	Trecho do email
1	TREC	Ham	Spam	<tr> <td width="49%">Resident Hunting </td> <td width="27%">RH</td> <td width="24%">$12.50</td> </tr> <tr> <td width="49%">Resident Combination Hunting/Fishing </td> <td width="27%">CHF</td> <td width="24%">$21</td> </tr>
2	TREC	Spam	Ham	- Frank Moss, who took Tivoli Systems from a $50-million startup into a $743-million IBM acquisition, on the progress of Bowstreet's plans to take Web services from buzzword to profitable business model
3	TREC	Spam	Ham	All you will be required to do is to draft an agreement from your end that you will be supplying Medical and Agricultural Equipment to the tune of $18,500,000 (Eighteen Million,Five Hundred Thousand United States Dollars) in my Company's favour.Once this agreement is set up,I will need to see the contents and have a copy submitted to my bankers here so they can immediately begin the transfer processes of the stipulated amount in favour of goods to be supplied(Supposedly) by you.

Tabela 3.5: Trechos de alguns emails incorretamente classificados pelo SVM linear na Etapa 4

Capítulo 4

Conclusões

Um inconveniente ainda hoje é o tráfego de spams e a dificuldade em detectá-los, o que pode causar danos tanto a empresas quanto a consumidores, além de a toda uma gama de usuários de correio eletrônico. Este trabalho avalia três algoritmos de aprendizado de máquina na tarefa de classificar emails entre legítimo ou spam utilizando quatro datasets, aplicando técnicas de seleção de hiperparâmetros dos modelos, variação do pré-processamento dos textos, implementação de novas características e o efeito delas no aprendizado dos padrões dos emails.

Em concordância com os objetivos deste trabalho, os experimentos foram divididos em quatro etapas de complexidade progressiva a fim de acompanhar a resposta de cada modelo à aplicação de diferentes técnicas por meio do diagnóstico de *Machine Learning*, especialmente avaliando curvas de aprendizado. O ideal desta abordagem é tratar cada modelo de forma individual, tomando ações condizentes com o diagnóstico aplicado. Por exemplo, se o modelo apresenta *underfitting*, ao invés de adicionar mais dados no treinamento, reduzir a dimensionalidade pode ser uma atitude mais eficaz. Porém, para simplificar os experimentos, da primeira até a terceira etapa as mesmas ações foram tomadas para todos os modelos e isso possibilitou, inclusive, confirmar ou não o diagnóstico realizado em cada etapa.

Verificou-se que os modelos SVM, em geral, apresentaram um desempenho condizente com o diagnóstico aplicado e tiveram a capacidade de aprender novos padrões dos emails à medida que novas características textuais foram adicionadas e o dataset expandido. Assim, eles alcançaram os maiores resultados dentre os três algoritmos utilizados, destacando-se o de kernel linear, que em três datasets obteve as avaliações finais mais altas com as seguintes medidas F1: 0.995 no Enron, 0.983 no Ling-Spam e 0.992 no TREC-05. O SVM de kernel polinomial de grau 2 obteve o maior resultado no SpamAssassin, com F1-score igual a 0.97. Os algoritmos bayesianos apresentaram resultados positivos nas duas primeiras etapas, mas ao expandir o dataset e adicionar novas características eles se

mostraram imprevisíveis: em alguns datasets foram de encontro ao diagnóstico aplicado, aumentando seu desempenho mesmo com curvas de erro crescentes na etapa anterior e, em outros casos, reduziu o desempenho, concordando com a tendência dos gráficos ao *underfitting*. Já os modelos KNN apresentaram ainda maior imprevisibilidade e bastante ruído em alguns gráficos, em geral para os menores valores de K, impossibilitando o diagnóstico em alguns deles, além de demonstrarem bastante sensibilidade quanto à normalização e seleção de *features*, tornando esta tarefa bastante trabalhosa, de modo que, entre as etapas 3 e 4, para diversas configurações de vetores de características testados (mais precisamente 62 por modelo em cada dataset), estes modelos apresentaram o maior desvio padrão das medidas F1, o que evidencia que, de fato, pequenas mudanças na forma de representar o texto pode influenciar grandemente no desempenho destes modelos, de modo que tipos diferentes de pré-processamento também possibilitou reduzir ruídos e gerar curvas com formatos conhecidos, que seguem padrões observáveis.

Por fim, em resposta à pergunta apresentada na introdução deste trabalho, sobre ser possível aumentar a performance dos modelos ao unir técnicas de desambiguação e adicionar novas características: a união destas técnicas produziu resultados positivos em boa parte dos casos, em especial nos datasets maiores, auxiliou a remover ou reduzir ruídos, suavizando curvas de aprendizagem, principalmente nos modelos KNN, bem como em modelos SVM de kernel linear e polinomial, embora nem sempre refletiu em melhora da medida F1. A técnica de desambiguação, isoladamente, demonstrou

Propõe-se as seguintes possíveis evoluções deste trabalho: utilizar os datasets Enron e TREC-05 com sua carga máxima, especialmente para verificar o progresso de modelos SVM de kernel polinomial, implementação de novas características extraídas a partir dos textos erroneamente classificados e utilização de técnicas de regressão polinomial para melhor avaliação da curva de aprendizado de validação.

Referências

[1] *Targeting scams 2019.* https://www.accc.gov.au/publications/targeting-scams-report-on-scam-activity/targeting-scams-2019-a-review-of-scam-activity-since-2009, 2020. 1

[2] Karim, A., S. Azam, B. Shanmugam, K. Kannoorpattik e M. Alazab: *A comprehensive survey for intelligent spam email detection.* IEEE Access, 2019. 2, 18, 19, 20, 35

[3] Laorden, C., I. Santos, B. Sanz, G. Alvarez e P. G. Bringas: *Word sense disambiguation for spam filtering.* Elsevier, 2012. 2, 19, 22, 23, 36

[4] R., Hyde: *The fallacy of premature optimization.* ACM Ubiquity, 2009. 3, 24

[5] Rao, Delip e Brian McMahan: *Natural Language Preprocessing with PyTorch.* O'Reilly, 2019. 6

[6] Mitchell, Tom M.: *Machine Learning.* McGraw-Hill Science/Engineering/Math, 1997. 9

[7] Androutsopoulos, Ion, John Koutsias, Konstantinos V. Chandrinos, George Paliouras e Constantine D. Spyropoulos: *An evaluation of naive bayesian anti-spam filtering.* Proceedings of the workshop on Machine Learning in the New Information Age, G. Potamias, V. Moustakis and M. van Someren (eds.), 11th European Conference on Machine Learning, 2000. 10

[8] Li, Kang e Zhenyu Zhong: *Fast statistical spam filter by approximate classifications.* Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems, 2006. 10

[9] Cover, T. e P. Hart: *Nearest neighbor pattern classification.* IEEE Transactions on Information Theory, 13, 1967. 10

[10] Dada, Emmanuel Gbenga, Joseph Stephen Bassi, Haruna Chiroma, Shafi'i Muhammad Abdulhamid, Adebayo Olusola Adetunmbi e Opeyemi Emmanuel Ajibuwa: *Machine learning for email spam filtering: review, approaches and open research problems.* Heliyon, Elsevier, 2019. 11, 12, 19

[11] Cortes, C. e V. Vapnik: *Support-vector networks.* Machine learning, 1995. 11

[12] Hastie, Trevor, Robert Tibshirani e Jerome Friedman: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2009. 11, 12

[13] Anzanello, Michel Jose e Flavio Sanson Fogliatto: *Learning curve models and applications: Literature review and research directions*, volume 41. Elsevier, 2011. 14

[14] *Choosing the right metric for evaluating machine learning models — part 2*. https://medium.com/usf-msds/choosing-the-right-metric-for-evaluating-machine-learning-models-part-2-86d5649a5428, 2018. 19

[15] Awad, W.A. e S.M. ELseuofi: *Machine learning methods for spam e-mail classification*. International Journal of Computer Science & Information Technology (IJCSIT), 3, 2011. 19, 34, 35

[16] Guzella, T. S. e W. M. Caminhas: *A review of machine learning approaches to spam filtering*. Expert Systems with Applications, 36, 2009. 20

[17] Pawlak, Z.: *Rough sets*. International Journal of Computer & Information Sciences, 11, 1982. 20

[18] Al-Jarrah O., Khater I., Al Duwairi B.: *Identifying potentially useful email header features for email spam filtering*. ICDS, IARIA, 2012. 22

[19] Chhabra, Priyanka, Rajesh Wadhvani e Sanyam Shukla: *Spam filtering using support vector machine*. International Journal of Computer and Communication Technology, 1, 2010. 23, 37

[20] Batista, Gustavo E.A.P.A. e Diego Furtado Silva: *How k-nearest neighbor parameters affect its performance*. Argentine symposium on artificial intelligence, 2009. 23

Apêndice A

Resultados em tabelas e gráficos

Dataset Enron 3k				
Característica Adicionada	**SVM**	**MultiNB**	**KNN** 1	**KNN 5 Dist**
One-hot vocab	0.949	0.95	0.872	0.884
TF-IDF min_df=1	0.992	0.992	0.988	0.995
TF-IDF min_df=5	0.992	0.992	0.988	0.995
TF-IDF min_df=10	0.992	0.992	0.988	0.995
TF-IDF min_df=30	0.992	0.992	0.99	0.995
TF-IDF min_df=5 max_features=1000	0.992	0.99	0.992	0.993
TF-IDF min_df=5 max_features=3000	0.992	0.99	0.977	0.995
TF-IDF min_df=5 no_accents	0.992	0.992	0.988	0.995
TF-IDF min_df=5 no_accents stop_words	0.993	0.993	0.99	0.993
TF-IDF min_df=5 no_accents stop_words preproc	0.966	0.958	0.862	0.821
TF-IDF min_df=5 no_accents stop_words preproc stem	0.971	0.965	0.854	0.821
WSD max_senses=1	0.961	0.956	0.876	0.872
WSD max_senses=2	0.948	0.953	0.863	0.873
WSD max_senses=3	0.944	0.957	0.853	0.869
WSD max_senses=5	0.951	0.953	0.861	0.862
WSD max_senses=7	0.954	0.955	0.858	0.863
pos_tag	0.961	0.97	0.874	0.86
pos_tag norm	0.949	0.95	0.895	0.893
pos_tag norm f_max_val=50	0.966	0.952	0.896	0.897

Dataset Enron 3k				
Característica Adicionada	**SVM**	**MultiNB**	**KNN** 1	**KNN 5 Dist**
pos_tag percent	0.95	0.95	0.902	0.901
pos_tag percent norm	0.951	0.95	0.898	0.897
upper_case	0.958	0.912	0.907	0.895
upper_case 1-hot	0.95	0.95	0.887	0.883
upper_case norm	0.95	0.95	0.891	0.888
upper_case norm f_max_val=100	0.953	0.955	0.891	0.888
upper_case percent	0.951	0.955	0.873	0.882
upper_case percent norm	0.947	0.955	0.873	0.882
punctuations	0.944	0.958	0.913	0.903
punctuations 1-hot	0.959	0.954	0.911	0.897
punctuations norm	0.949	0.95	0.902	0.897
punctuations norm f_max_val=5	0.959	0.955	0.9	0.896
punctuations norm f_max_val=3	0.959	0.955	0.9	0.896
seek_tag	0.966	0.948	0.904	0.923
seek_tag 1-hot	0.967	0.963	0.89	0.91
seek_tag norm	0.951	0.953	0.876	0.885
seek_tag percent	0.97	0.959	0.896	0.901
seek_tag percent norm	0.97	0.959	0.896	0.901
seek_tag percent norm2	0.97	0.958	0.881	0.889
seek_URL	0.978	0.955	0.95	0.941
seek_URL 1-hot	0.977	0.947	0.932	0.912
seek_URL norm	0.951	0.95	0.913	0.901
seek_URL norm f_max_val=7	0.969	0.954	0.913	0.901
seek_URL norm f_max_val=5	0.975	0.95	0.922	0.901
seek_URL norm f_max_val=3	0.977	0.948	0.928	0.904
upper_case + seek_URL	0.972	0.94	0.94	0.927
upper_case + seek_URL + punctuations	0.97	0.945	0.944	0.93
upper_case + seek_URL + punctuations + seek_tag	0.987	0.969	0.974	0.974
upper_case + seek_URL + punctuations + seek_tag + pos_tag percent	0.987	0.969	0.974	0.974

Dataset Enron 3k				
Característica Adicionada	**SVM**	**MultiNB**	**KNN** 1	**KNN 5 Dist**
TF-IDF min_df=5 no_accents + upper_case + seek_URL + punctuations + seek_tag + pos_tag percent	0.992	0.983	0.969	0.969
upper_case + seek_URL + punctuations + seek_tag + pos_tag percent + WSD max_senses=1	0.984	0.968	0.977	0.974
TF-IDF min_df=5 no_accents + upper_case + seek_URL + punctuations + seek_tag + pos_tag percent + WSD max_senses=1	0.992	0.988	0.969	0.969
upper_case norm f_max_val=100 + seek_URL 1-hot	0.968	0.947	0.949	0.916
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot	0.974	0.953	0.932	0.925
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot	0.99	0.966	0.955	0.951
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50	0.993	0.968	0.958	0.953
TF-IDF min_df=5 no_accents + upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50	0.992	0.993	0.985	0.982
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=1	0.99	0.966	0.952	0.944
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=2	0.985	0.969	0.946	0.948

Dataset Enron 3k				
Característica Adicionada	**SVM**	**MultiNB**	**KNN** 1	**KNN 5 Dist**
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=3	0.985	0.969	0.938	0.948
TF-IDF min_df=5 no_accents + upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=1	0.992	0.992	0.985	0.985
TF-IDF min_df=5 no_accents + upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=2	0.992	0.992	0.985	0.985
TF-IDF min_df=5 no_accents + upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=3	0.992	0.992	0.987	0.985
F1 Máximo	0.993	0.993	0.992	0.995
F1 Mínimo	0.944	0.912	0.853	0.821
F1 Médio	0.971	0.963	0.925	0.922
Desvio padrão	0.017	0.017	**0.044**	**0.046**

Tabela A.1: F1-score de modelos treinados no dataset Enron 3k com variação de pré-processamento e adição de novas características, processo anterior à etapa 4

Dataset SpamAssassin 3k				
Característica Adicionada	**SVM**	**MultiNB**	**KNN** 1	**KNN 5 Dist**
One-hot vocab	0.941	0.942	0.723	0.678
TF-IDF min_df=1	0.974	0.968	0.572	0.953
TF-IDF min_df=5	0.974	0.968	0.578	0.953
TF-IDF min_df=10	0.974	0.968	0.577	0.964
TF-IDF min_df=30	0.974	0.974	0.564	0.948
TF-IDF min_df=5 max_features=1000	0.979	0.946	0.664	0.938

Dataset SpamAssassin 3k				
Característica Adicionada	**SVM**	**MultiNB**	**KNN** 1	**KNN 5 Dist**
TF-IDF min_df=5 max_features=3000	0.974	0.974	0.513	0.953
TF-IDF min_df=5 no_accents	0.974	0.968	0.578	0.953
TF-IDF min_df=5 no_accents stop_words	0.974	0.968	0.57	0.933
TF-IDF min_df=5 no_accents stop_words preproc	0.984	0.952	0.503	0.809
TF-IDF min_df=5 no_accents stop_words preproc stem	0.968	0.952	0.5	0.824
WSD max_senses=1	0.952	0.947	0.716	0.692
WSD max_senses=2	0.946	0.952	0.726	0.688
WSD max_senses=3	0.941	0.948	0.705	0.669
WSD max_senses=5	0.942	0.952	0.717	0.667
WSD max_senses=7	0.942	0.947	0.726	0.675
pos_tag	0.947	0.948	0.592	0.699
pos_tag norm	0.941	0.942	0.732	0.683
pos_tag norm f_max_val=50	0.946	0.942	0.732	0.672
pos_tag percent	0.941	0.942	0.711	0.667
pos_tag percent norm	0.941	0.942	0.717	0.669
upper_case	0.957	0.714	0.72	0.816
upper_case 1-hot	0.941	0.941	0.696	0.717
upper_case norm	0.941	0.947	0.72	0.656
upper_case norm f_max_val=100	0.957	0.952	0.733	0.672
upper_case percent	0.952	0.942	0.723	0.68
upper_case percent norm	0.952	0.942	0.723	0.678
punctuations	0.957	0.948	0.706	0.72
punctuations 1-hot	0.946	0.947	0.726	0.712
punctuations norm	0.952	0.952	0.72	0.69
punctuations norm f_max_val=5	0.963	0.948	0.72	0.696
punctuations norm f_max_val=3	0.952	0.948	0.717	0.693
seek_tag	0.952	0.724	0.726	0.761
seek_tag 1-hot	0.952	0.969	0.718	0.732
seek_tag norm	0.952	0.958	0.726	0.7
seek_tag percent	0.946	0.958	0.718	0.712

Dataset SpamAssassin 3k				
Característica Adicionada	**SVM**	**MultiNB**	**KNN** 1	**KNN 5 Dist**
seek_tag percent norm	0.946	0.958	0.718	0.712
seek_tag percent norm2	0.957	0.958	0.72	0.7
seek_URL	0.951	0.948	0.672	0.746
seek_URL 1-hot	0.973	0.947	0.709	0.712
seek_URL norm	0.941	0.942	0.723	0.717
seek_URL norm f_max_val=7	0.952	0.942	0.723	0.723
seek_URL norm f_max_val=5	0.952	0.942	0.723	0.723
seek_URL norm f_max_val=3	0.947	0.942	0.723	0.73
upper_case + seek_URL	0.957	0.726	0.761	0.832
upper_case + seek_URL + punctuations	0.958	0.755	0.766	0.819
upper_case + seek_URL + punctuations + seek_tag	0.974	0.72	0.808	0.888
upper_case + seek_URL + punctuations + seek_tag + pos_tag percent	0.957	0.728	0.796	0.878
TF-IDF min_df=5 no_accents + upper_case + seek_URL + punctuations + seek_tag + pos_tag percent	0.974	0.676	0.879	0.884
upper_case + seek_URL + punctuations + seek_tag + pos_tag percent + WSD max_senses=1	0.968	0.742	0.806	0.869
TF-IDF min_df=5 no_accents + upper_case + seek_URL + punctuations + seek_tag + pos_tag percent + WSD max_senses=1	0.974	0.648	0.883	0.884
upper_case norm f_max_val=100 + seek_URL 1-hot	0.973	0.952	0.733	0.705
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot	0.957	0.947	0.741	0.726
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot	0.979	0.969	0.745	0.788

Dataset SpamAssassin 3k				
Característica Adicionada	**SVM**	**MultiNB**	**KNN** 1	**KNN 5 Dist**
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50	0.973	0.969	0.736	0.788
TF-IDF min_df=5 no_accents + upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50	0.989	0.963	0.833	0.874
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=1	0.969	0.969	0.752	0.792
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=2	0.968	0.974	0.759	0.762
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=3	0.952	0.974	0.743	0.73
TF-IDF min_df=5 no_accents + upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=1	0.984	0.934	0.808	0.854
TF-IDF min_df=5 no_accents + upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=2	0.984	0.899	0.8	0.857

Dataset SpamAssassin 3k				
Característica Adicionada	**SVM**	**MultiNB**	**KNN** 1	**KNN 5 Dist**
TF-IDF min_df=5 no_accents + upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=3	0.989	0.88	0.808	0.832
F1 Máximo	0.989	0.974	0.883	0.964
F1 Mínimo	0.941	0.648	0.5	0.656
F1 Médio	0.960	0.917	0.711	0.772
Desvio padrão	0.014	**0.085**	0.081	**0.095**

Tabela A.2: F1-score de modelos treinados no dataset SpamAssassin 3k com variação de pré-processamento e adição de novas características, processo anterior à etapa 4

Dataset Ling-Spam				
Característica Adicionada	**SVM**	**MultiNB**	**KNN** 1	**KNN 5 Dist**
One-hot vocab	0.966	0.95	0.807	0.834
TF-IDF min_df=1	0.952	0.971	0.783	0.935
TF-IDF min_df=5	0.952	0.971	0.783	0.953
TF-IDF min_df=10	0.952	0.971	0.783	0.953
TF-IDF min_df=30	0.959	0.966	0.783	0.954
TF-IDF min_df=5 max_features=1000	0.977	0.959	0.87	0.959
TF-IDF min_df=5 max_features=3000	0.971	0.982	0.766	0.943
TF-IDF min_df=5 no_accents	0.952	0.971	0.783	0.953
TF-IDF min_df=5 no_accents stop_words	0.959	0.976	0.711	0.927
TF-IDF min_df=5 no_accents stop_words preproc	0.97	0.976	0.775	0.901
TF-IDF min_df=5 no_accents stop_words preproc stem	0.983	0.965	0.8	0.905
WSD max_senses=1	0.982	0.943	0.809	0.777
WSD max_senses=2	0.976	0.959	0.791	0.79
WSD max_senses=3	0.953	0.965	0.778	0.767
WSD max_senses=5	0.959	0.952	0.757	0.762
WSD max_senses=7	0.959	0.946	0.775	0.788
pos_tag	0.96	0.927	0.584	0.612

Dataset Ling-Spam				
Característica Adicionada	**SVM**	**MultiNB**	**KNN** 1	**KNN 5 Dist**
pos_tag norm	0.966	0.95	0.807	0.834
pos_tag norm f_max_val=50	0.971	0.95	0.812	0.834
pos_tag percent	0.966	0.95	0.805	0.827
pos_tag percent norm	0.966	0.95	0.812	0.842
upper_case	0.966	0.95	0.807	0.834
upper_case 1-hot	0.966	0.95	0.807	0.834
upper_case norm	0.966	0.95	0.807	0.834
upper_case norm f_max_val=100	0.966	0.95	0.807	0.834
upper_case percent	0.966	0.95	0.807	0.834
upper_case percent norm	0.966	0.95	0.807	0.834
punctuations	0.96	0.96	0.849	0.886
punctuations 1-hot	0.953	0.95	0.795	0.834
punctuations norm	0.966	0.95	0.807	0.813
punctuations norm f_max_val=5	0.966	0.95	0.815	0.842
punctuations norm f_max_val=3	0.965	0.95	0.815	0.842
seek_tag	0.966	0.95	0.807	0.834
seek_tag 1-hot	0.966	0.95	0.807	0.834
seek_tag norm	0.966	0.95	0.807	0.834
seek_tag percent	0.966	0.95	0.807	0.834
seek_tag percent norm	0.966	0.95	0.807	0.834
seek_tag percent norm2	0.966	0.95	0.807	0.834
seek_URL	0.966	0.95	0.807	0.834
seek_URL 1-hot	0.966	0.95	0.807	0.834
seek_URL norm	0.966	0.95	0.807	0.834
seek_URL norm f_max_val=7	0.966	0.95	0.807	0.834
seek_URL norm f_max_val=5	0.966	0.95	0.807	0.834
seek_URL norm f_max_val=3	0.966	0.95	0.807	0.834
upper_case + seek_URL	0.966	0.95	0.807	0.834
upper_case + seek_URL + punctuations	0.96	0.96	0.849	0.886
upper_case + seek_URL + punctuations + seek_tag	0.96	0.96	0.849	0.886
upper_case + seek_URL + punctuations + seek_tag + pos_tag percent	0.96	0.96	0.869	0.892

Dataset Ling-Spam				
Característica Adicionada	**SVM**	**MultiNB**	**KNN** 1	**KNN 5 Dist**
TF-IDF min_df=5 no_accents + upper_case + seek_URL + punctuations + seek_tag + pos_tag percent	0.959	0.888	0.831	0.823
upper_case + seek_URL + punctuations + seek_tag + pos_tag percent + WSD max_senses=1	0.959	0.954	0.852	0.867
TF-IDF min_df=5 no_accents + upper_case + seek_URL + punctuations + seek_tag + pos_tag percent + WSD max_senses=1	0.971	0.888	0.828	0.827
upper_case norm f_max_val=100 + seek_URL 1-hot	0.966	0.95	0.807	0.834
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot	0.953	0.95	0.795	0.834
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot	0.953	0.95	0.795	0.834
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50	0.953	0.944	0.788	0.842
TF-IDF min_df=5 no_accents + upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50	0.958	0.976	0.881	0.862
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=1	0.97	0.944	0.824	0.818
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=2	0.97	0.961	0.811	0.805

Dataset Ling-Spam				
Característica Adicionada	**SVM**	**MultiNB**	**KNN** 1	**KNN 5 Dist**
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=3	0.965	0.959	0.791	0.815
TF-IDF min_df=5 no_accents + upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=1	0.971	0.965	0.879	0.88
TF-IDF min_df=5 no_accents + upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=2	0.965	0.977	0.88	0.886
TF-IDF min_df=5 no_accents + upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=3	0.959	0.965	0.875	0.875
F1 Máximo	0.983	0.982	0.881	0.959
F1 Mínimo	0.952	0.888	0.584	0.612
F1 Médio	0.964	0.953	0.806	0.848
Desvio padrão	0.007	0.016	**0.042**	**0.055**

Tabela A.3: F1-score de modelos treinados no dataset Ling-Spam com variação de pré-processamento e adição de novas características, processo anterior à etapa 4

Dataset TREC 3k				
Característica Adicionada	**SVM**	**MultiNB**	**KNN** 1	**KNN 5 Dist**
One-hot vocab	0.989	0.963	0.976	0.961
TF-IDF min_df=1	0.99	0.984	0.966	0.947
TF-IDF min_df=5	0.989	0.984	0.967	0.946
TF-IDF min_df=10	0.989	0.983	0.967	0.945
TF-IDF min_df=30	0.991	0.984	0.96	0.943
TF-IDF min_df=5 max_features=1000	0.989	0.977	0.976	0.97

Dataset TREC 3k				
Característica Adicionada	**SVM**	**MultiNB**	**KNN** 1	**KNN 5 Dist**
TF-IDF min_df=5 max_features=3000	0.992	0.987	0.95	0.939
TF-IDF min_df=5 no_accents	0.99	0.983	0.966	0.946
TF-IDF min_df=5 no_accents stop_words	0.992	0.987	0.955	0.942
TF-IDF min_df=5 no_accents stop_words preproc	0.988	0.989	0.955	0.944
TF-IDF min_df=5 no_accents stop_words preproc stem	0.99	0.992	0.95	0.938
WSD max_senses=1	0.983	0.96	0.97	0.957
WSD max_senses=2	0.984	0.959	0.975	0.962
WSD max_senses=3	0.985	0.963	0.974	0.962
WSD max_senses=5	0.984	0.964	0.973	0.959
WSD max_senses=7	0.985	0.963	0.974	0.957
pos_tag	0.982	0.967	0.953	0.958
pos_tag norm	0.989	0.963	0.978	0.96
pos_tag norm f_max_val=50	0.982	0.963	0.977	0.959
pos_tag percent	0.989	0.963	0.975	0.962
pos_tag percent norm	0.988	0.963	0.977	0.961
upper_case	0.99	0.92	0.968	0.962
upper_case 1-hot	0.989	0.963	0.969	0.953
upper_case norm	0.989	0.963	0.979	0.965
upper_case norm f_max_val=100	0.99	0.964	0.979	0.966
upper_case percent	0.989	0.964	0.975	0.962
upper_case percent norm	0.989	0.964	0.975	0.962
punctuations	0.983	0.946	0.973	0.956
punctuations 1-hot	0.985	0.965	0.977	0.964
punctuations norm	0.989	0.963	0.977	0.963
punctuations norm f_max_val=5	0.981	0.964	0.976	0.962
punctuations norm f_max_val=3	0.981	0.964	0.976	0.962
seek_tag	0.982	0.953	0.974	0.961
seek_tag 1-hot	0.99	0.971	0.976	0.962
seek_tag norm	0.989	0.97	0.976	0.962
seek_tag percent	0.99	0.972	0.976	0.962

Dataset TREC 3k				
Característica Adicionada	**SVM**	**MultiNB**	**KNN** 1	**KNN 5 Dist**
seek_tag percent norm	0.99	0.972	0.976	0.962
seek_tag percent norm2	0.98	0.972	0.976	0.962
seek_URL	0.983	0.969	0.976	0.966
seek_URL 1-hot	0.991	0.966	0.976	0.964
seek_URL norm	0.989	0.963	0.978	0.964
seek_URL norm f_max_val=7	0.99	0.965	0.978	0.964
seek_URL norm f_max_val=5	0.99	0.965	0.978	0.964
seek_URL norm f_max_val=3	0.989	0.966	0.978	0.964
upper_case + seek_URL	0.992	0.93	0.972	0.965
upper_case + seek_URL + punctuations	0.988	0.934	0.963	0.958
upper_case + seek_URL + punctuations + seek_tag	0.989	0.958	0.966	0.966
upper_case + seek_URL + punctuations + seek_tag + pos_tag percent	0.989	0.958	0.966	0.965
TF-IDF min_df=5 no_accents + upper_case + seek_URL + punctuations + seek_tag + pos_tag percent	0.992	0.942	0.97	0.968
upper_case + seek_URL + punctuations + seek_tag + pos_tag percent + WSD max_senses=1	0.984	0.955	0.966	0.965
TF-IDF min_df=5 no_accents + upper_case + seek_URL + punctuations + seek_tag + pos_tag percent + WSD max_senses=1	0.993	0.944	0.969	0.968
upper_case norm f_max_val=100 + seek_URL 1-hot	0.991	0.966	0.978	0.968
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot	0.989	0.972	0.978	0.967
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot	0.99	0.973	0.977	0.967

Dataset TREC 3k				
Característica Adicionada	**SVM**	**MultiNB**	**KNN** 1	**KNN 5 Dist**
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50	0.989	0.973	0.977	0.969
TF-IDF min_df=5 no_accents + upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50	0.993	0.983	0.965	0.96
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=1	0.985	0.977	0.97	0.965
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=2	0.992	0.976	0.976	0.967
upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=3	0.992	0.971	0.976	0.966
TF-IDF min_df=5 no_accents + upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=1	0.994	0.985	0.96	0.961
TF-IDF min_df=5 no_accents + upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=2	0.992	0.982	0.966	0.963

Dataset TREC 3k				
Característica Adicionada	**SVM**	**MultiNB**	**KNN** 1	**KNN 5 Dist**
TF-IDF min_df=5 no_accents + upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=3	0.994	0.981	0.964	0.966
F1 Máximo	0.994	0.992	0.979	0.97
F1 Mínimo	0.98	0.92	0.95	0.938
F1 Médio	0.988	0.967	0.971	0.960
Desvio padrão	0.003	**0.014**	0.007	**0.008**

Tabela A.4: F1-score de modelos treinados no dataset TREC 3k com variação de pré-processamento e adição de novas características, processo anterior à etapa 4

Termo	Descrição
Pré-processamento	
TF-IDF min_df=N	Os termos que irão compor o vocabulário do TF-IDF devem aparecer no mínimo N vezes.
TF-IDF max_features=N	Limita o tamanho do vocabulário que o TF-IDF formará para N palavras. Quando este parâmetro não é informado, a dimensão é a mesma do one-hot, isto é, N = 1899.
TF-IDF no_accents	Remoção de acentos antes de aplicar de TF-IDF.
TF-IDF stop_words	Remoção de stop words (lista padrão da biblioteca, argumento stop_words = "english").
TF-IDF preproc	Mesmo pré-processamento descrito na seção 3.2, sem aplicar stemming.
TF-IDF preproc stem	Mesmo pré-processamento descrito na seção 3.2, aplicando stemming às palavras.
Características Adicionadas e Normalização	
pos_tag	Contagem de verbos, advérbios e adjetivos.
norm	Normalização dos valores utilizando o método *fit_transform()* da classe *sklearn.preprocessing.MinMaxScaler*.
norm f_max_val=N	Estabelece um teto de N para os valores, depois eles são normalizados.
percent	Normaliza segundo a porcentagem daquele atributo com relação a outro, por exemplo, a porcentagem de uma tag HTML específica com relação ao total de tags ou a porcentagem de uma determinada POS tag com relação ao total de palavras do texto.
,percent norm	Aplica a técnica de porcentagem, depois a de normalização.
upper_case	Conta a maior sequência de palavras em letras maiúsculas (w_sequences_len), maior sequência de letras maiúsculas (l_sequences_len) e total de palavras do texto grafadas em maiúsculo (total_upper_w).
1-hot	Atribui 1 ou 0 caso certa característica esteja presente ou não no texto.
punctuations	Contagem dos símbolos "!", "?", "$" e "%".
seek_tag	Conta todas as tags do email (total_any_tag) e outras específicas: img, input, a, object, source, script, iframe, br e p.
seek_tag percent norm2	seek_tag com percent e norm com f_max_val = 30 aplicados.
seek_URL	Contagem de quantas URLs são encontradas.

Tabela A.5: Descrição de termos utilizados na Etapa 4

Características	Modelo	Conjunto de Teste			Treino	Validação
		Precisão	Recall	F1	F1	F1
TF-IDF min_df=5 no_accents stop_words	SVM C=1	0.996	0.993	0.995	0.998	0.995
	SVM C=1 sigmoid	0.996	0.991	0.993	0.996	0.994
	SVM C=3 poly 2	0.998	0.987	0.992	0.999	0.990
	SVM C=3 poly 3	0.999	0.960	0.979	0.999	0.977
	SVM C=3 poly 4	0.999	0.947	0.972	0.999	0.971
	SVM C=3 poly 5	1.000	0.936	0.967	0.999	0.967
	SVM C=3 poly 7	1.000	0.904	0.950	0.999	0.950
	Naive Bayes var_smoothing=1e-05	0.994	0.986	0.990	0.992	0.989
	Multinomial NB alpha=1e-10	0.993	0.984	0.988	0.989	0.989
	KNN K=1	0.870	0.975	0.919	0.999	0.926
	KNN K=3	0.873	0.967	0.918	0.987	0.924
	KNN K=5	0.990	0.958	0.974	0.987	0.970
	KNN K=10	0.993	0.945	0.968	0.973	0.964
	KNN K=3 Dist	0.875	0.969	0.920	0.999	0.927
	KNN K=5 Dist	0.991	0.962	0.976	0.999	0.974
	KNN K=10 Dist	0.993	0.957	0.975	0.999	0.973
TF-IDF min_df=5 no_accents + upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50 + WSD max_senses=3	SVM C=3	0.994	0.995	0.995	0.999	0.994
	SVM C=0.03 sigmoid	0.954	0.924	0.939	0.931	0.936
	SVM C=30 poly 2	0.997	0.993	0.995	1.000	0.996
	SVM C=30 poly 3	0.996	0.993	0.995	0.999	0.994
	SVM C=1000 poly 4	0.995	0.991	0.993	1.000	0.993
	SVM C=3000 poly 5	0.996	0.990	0.993	1.000	0.992
	SVM C=1e+05 poly 7	0.996	0.983	0.990	1.000	0.988
	Naive Bayes var_smoothing=1e-03	0.985	0.984	0.985	0.982	0.980
	Multinomial NB alpha=1e-10	0.993	0.940	0.966	0.964	0.962

	Conjunto de Teste			Treino	Validação
Modelo	Precisão	Recall	F1	F1	F1
KNN K=1	0.857	0.994	0.921	1.000	0.923
KNN K=3	0.838	0.992	0.908	0.940	0.912
KNN K=5	0.831	0.990	0.904	0.920	0.909
KNN K=10	0.985	0.985	0.985	0.986	0.982
KNN K=3 Dist	0.846	0.992	0.913	1.000	0.918
KNN K=5 Dist	0.841	0.992	0.910	1.000	0.915
KNN K=10 Dist	0.975	0.991	0.983	1.000	0.980

Tabela A.6: Resultados parciais da etapa 4 utilizando dois tipos diferentes de pré-processamento sobre o dataset Enron

Características	Modelo	Conjunto de Teste			Treino	Validação
		Precisão	Recall	F1	F1	F1
TF-IDF min_df=5 max_features=3000	SVM C=3	0.964	0.969	0.966	0.996	0.984
	SVM C=1 sigmoid	0.956	0.966	0.961	0.983	0.975
	SVM C=30 poly 2	0.968	0.958	0.963	1.000	0.973
	SVM C=100 poly 3	0.970	0.924	0.946	1.000	0.964
	SVM C=300 poly 4	0.971	0.882	0.925	1.000	0.952
	SVM C=300 poly 5	0.978	0.812	0.887	1.000	0.908
	SVM C=300 poly 7	0.986	0.746	0.849	1.000	0.863
	Naive Bayes var_smoothing=1e-03	0.941	0.961	0.951	0.986	0.954
	Multinomial NB alpha=1e-10	0.961	0.908	0.934	0.963	0.945
	KNN K=1	0.631	0.990	0.771	1.000	0.751
	KNN K=3	0.917	0.958	0.937	0.977	0.953
	KNN K=5	0.933	0.950	0.942	0.968	0.954
	KNN K=10	0.948	0.914	0.931	0.954	0.960
	KNN K=3 Dist	0.927	0.963	0.945	1.000	0.957
	KNN K=5 Dist	0.938	0.958	0.948	1.000	0.962
	KNN K=10 Dist	0.943	0.953	0.948	1.000	0.965
TF-IDF min_df=5 no_accents + upper_case norm f_max_val=100 + seek_URL 1-hot + punctuations 1-hot + seek_tag 1-hot + pos_tag norm f_max_val=50	SVM C=10	0.969	0.966	0.967	0.999	0.976
	SVM C=0.3 sigmoid	0.828	0.717	0.769	0.780	0.785
	SVM C=100 poly 2	0.969	0.971	0.970	1.000	0.979
	SVM C=1000 poly 3	0.966	0.963	0.965	1.000	0.977
	SVM C=3000 poly 4	0.971	0.958	0.964	1.000	0.977
	SVM C=30000 poly 5	0.968	0.953	0.960	1.000	0.976
	SVM C=3e+05 poly 7	0.967	0.919	0.942	1.000	0.962
	Naive Bayes var_smoothing=1e-04	0.945	0.953	0.949	0.980	0.951

Modelo	Conjunto de Teste			Treino	Validação
	Precisão	Recall	F1	F1	F1
Multinomial NB alpha=1e-10	0.907	0.866	0.886	0.915	0.900
KNN K=1	0.920	0.929	0.924	1.000	0.919
KNN K=3	0.919	0.895	0.907	0.957	0.922
KNN K=5	0.926	0.882	0.903	0.936	0.914
KNN K=10	0.926	0.846	0.884	0.900	0.900
KNN K=3 Dist	0.921	0.916	0.919	1.000	0.936
KNN K=5 Dist	0.928	0.908	0.918	1.000	0.938
KNN K=10 Dist	0.922	0.895	0.908	1.000	0.938

Tabela A.7: Resultados parciais da etapa 4 utilizando dois tipos diferentes de pré-processamento sobre o dataset SpamAssassin

Características	Modelo	Conjunto de Teste Precisão	Recall	F1	Treino F1	Validação F1
	SVM C=3	0.977	0.977	0.977	0.998	0.983
	SVM C=3 sigmoid	0.976	0.965	0.971	0.993	0.983
	SVM C=3 poly 2	1.000	0.907	0.951	1.000	0.970
	SVM C=3 poly 3	1.000	0.849	0.918	1.000	0.912
	SVM C=3 poly 4	1.000	0.733	0.846	1.000	0.814
	SVM C=1 poly 5	1.000	0.547	0.707	1.000	0.724
	SVM C=3 poly 7	1.000	0.465	0.635	1.000	0.561
TF-IDF min_df=5 max_features=1000	Naive Bayes var_smoothing=1e-03	0.931	0.942	0.936	0.979	0.970
	Multinomial NB alpha=1e-01	0.965	0.953	0.959	0.984	0.991
	KNN K=1	0.985	0.779	0.870	1.000	0.863
	KNN K=3	0.975	0.919	0.946	0.976	0.949
	KNN K=5	0.963	0.919	0.940	0.966	0.945
	KNN K=10	0.976	0.953	0.965	0.962	0.949
	KNN K=3 Dist	0.976	0.953	0.965	1.000	0.958
	KNN K=5 Dist	0.965	0.953	0.959	1.000	0.954
	KNN K=10 Dist	0.965	0.965	0.965	1.000	0.958
	SVM C=3	0.977	0.988	0.983	1.000	0.983
	SVM C=1 sigmoid	0.976	0.953	0.965	0.993	0.969
	SVM C=3 poly 2	1.000	0.919	0.958	1.000	0.937
	SVM C=3 poly 3	1.000	0.709	0.830	1.000	0.837
	SVM C=3 poly 4	1.000	0.547	0.707	1.000	0.689
	SVM C=3 poly 5	1.000	0.430	0.602	1.000	0.587
TF-IDF min_df=5 no_accents stop_words preproc stem	SVM C=3 poly 7	1.000	0.349	0.517	1.000	0.427
	Naive Bayes var_smoothing=1e-03	0.902	0.965	0.933	0.982	0.975
	Multinomial NB alpha=1e-01	0.976	0.953	0.965	0.987	0.983
	KNN K=1	0.983	0.674	0.800	1.000	0.776
	KNN K=3	0.961	0.849	0.901	0.961	0.902
	KNN K=5	0.923	0.837	0.878	0.946	0.917
	KNN K=10	0.948	0.849	0.896	0.927	0.890

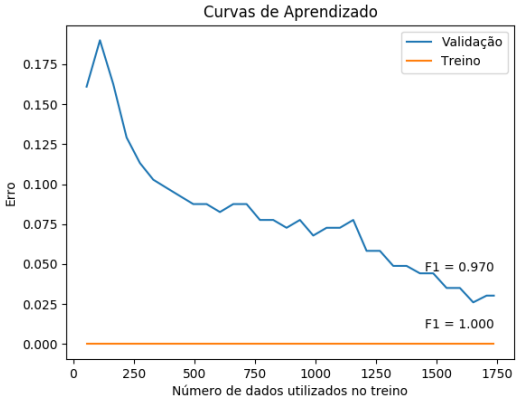
	Conjunto de Teste			Treino	Validação
Modelo	Precisão	Recall	F1	F1	F1
KNN K=3 Dist	0.962	0.884	0.921	1.000	0.912
KNN K=5 Dist	0.927	0.884	0.905	1.000	0.931
KNN K=10 Dist	0.949	0.872	0.909	1.000	0.914

Tabela A.8: Resultados parciais da etapa 4 utilizando dois tipos diferentes de pré-processamento sobre o dataset Ling-Spam

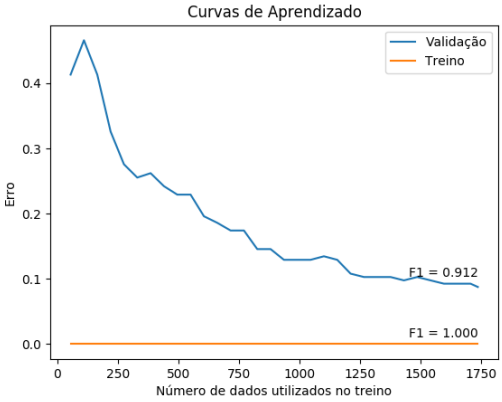
Características	Modelo	Conjunto de Teste Precisão	Recall	F1	Treino F1	Validação F1
	SVM C=3	0.988	0.991	0.990	0.995	0.990
	SVM C=0.1 sigmoid	0.968	0.988	0.978	0.980	0.981
	SVM C=30 poly 2	0.987	0.993	0.990	0.999	0.992
	SVM C=1 poly 3	0.986	0.993	0.990	0.999	0.992
	SVM C=1 poly 4	0.989	0.992	0.990	0.999	0.993
	SVM C=1 poly 5	0.993	0.965	0.979	0.984	0.980
TF-IDF min_df=5	SVM C=1 poly 7	0.997	0.949	0.972	0.984	0.972
max_features=1000	Naive Bayes var_smoothing=1e-01	0.970	0.929	0.949	0.959	0.957
	Multinomial NB alpha=3e+01	0.966	0.956	0.961	0.966	0.967
	KNN K=1	0.947	0.994	0.970	0.999	0.964
	KNN K=3	0.920	0.993	0.955	0.973	0.949
	KNN K=5	0.900	0.993	0.944	0.959	0.942
	KNN K=10	0.879	0.993	0.932	0.942	0.932
	KNN K=3 Dist	0.925	0.994	0.958	0.999	0.952
	KNN K=5 Dist	0.908	0.994	0.949	0.999	0.946
	KNN K=10 Dist	0.888	0.995	0.938	0.999	0.937
	SVM C=3	0.992	0.992	0.992	0.998	0.992
	SVM C=0.03 sigmoid	0.891	0.963	0.926	0.929	0.925
TF-IDF min_df=5	SVM C=100 poly 2	0.987	0.991	0.989	0.999	0.991
no_accents +	SVM C=30 poly 3	0.994	0.966	0.980	0.984	0.980
upper_case norm	SVM C=1000 poly 4	0.976	0.990	0.983	1.000	0.983
f_max_val=100 + seek_URL 1-hot +	SVM C=1000 poly 5	0.994	0.963	0.978	0.984	0.979
punctuations 1-hot + seek_tag 1-hot +	SVM C=30000 poly 7	0.991	0.962	0.976	0.985	0.977
pos_tag norm	Naive Bayes					
f_max_val=50 + WSD max_senses=1	var_smoothing=1e-02	0.979	0.958	0.968	0.976	0.975

Modelo	Conjunto de Teste			Treino	Validação
	Precisão	Recall	F1	F1	F1
Multinomial NB alpha=1e-10	0.987	0.936	0.961	0.966	0.966
KNN K=1	0.917	0.989	0.952	1.000	0.948
KNN K=3	0.910	0.987	0.947	0.965	0.945
KNN K=5	0.898	0.987	0.941	0.952	0.937
KNN K=10	0.890	0.984	0.934	0.940	0.930
KNN K=3 Dist	0.912	0.990	0.950	1.000	0.948
KNN K=5 Dist	0.904	0.991	0.945	1.000	0.942
KNN K=10 Dist	0.895	0.990	0.940	1.000	0.938

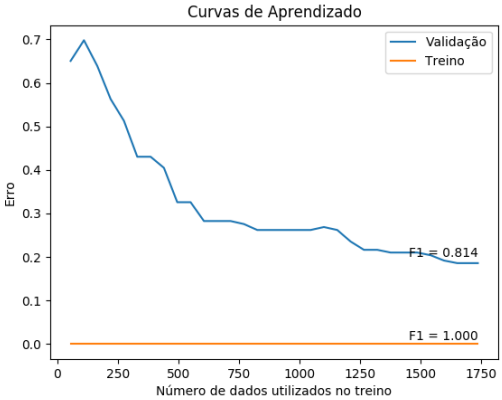
Tabela A.9: Resultados parciais da etapa 4 utilizando dois tipos diferentes de pré-processamento sobre o dataset TREC-05



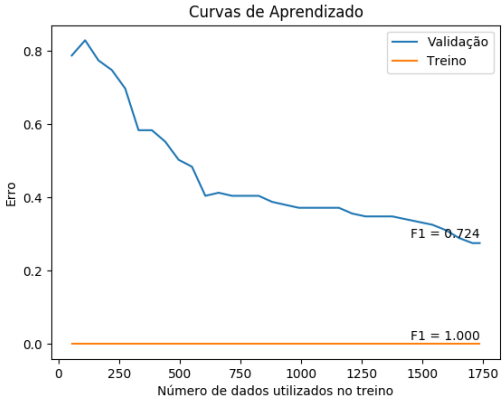
(a) Grau 2, C = 3



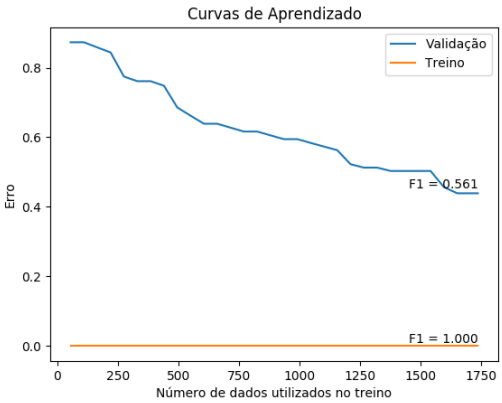
(b) Grau 3, C = 3



(c) Grau 4, C = 3



(d) Grau 5, C = 1



(e) Grau 7, C = 3

Figura A.1: Curvas de aprendizado de modelos SVM polinomiais na Etapa 4 no dataset Ling-Spam