Análise de Sentimentos Aplicada à Política

Lucas Romão Silva

Prof Dr. Roberto Hirata Jr.

23 de novembro de 2017

Sumário

Resumo **1**

1 Introdução **3**

2 Revisão Bibliográfica **5**

3 *Machine Learning* e Análise de Sentimentos **7**
3.1 Considerações 7
3.2 Contextualização 7
3.3 Logistic Regression 8
3.3.1 Método do Gradiente 9
3.3.2 Método de Newton-Raphson 11
3.3.3 Extensão para o caso de várias classes 11
3.4 Support Vector Machine 13
3.4.1 Extensão ao caso de multiclasses 16
3.5 Análise de Sentimentos 17
3.5.1 Análise de *Tweets* de política 18
3.5.2 Obtendo um vetor de *features* 19
3.5.3 Classificação manual da opinião 20

4 Experimentos **25**
4.1 Análise exploratória 25
4.1.1 Distribuição das classes 25
4.1.2 Características dos tuítes de cada classe 26
4.2 Descrição dos experimentos 28
4.2.1 Parâmetros avaliados 29
4.3 Métricas avaliadas 30
4.4 Primeiro experimento 30
4.4.1 Algoritmos próprios 31
4.4.2 Implementações do `scikit` 32

Referências Bibliográficas **37**

Resumo

Com a grande expansão da internet, o uso das redes sociais tem se tornado cada vez mais ativo e, com o passar do tempo, tornou-se um lugar onde os usuários relatam não só informações acerca do cotidiano, mas também para opinar sobre temas como política, esportes, etc. O uso maior das redes sociais também causa maior mobilização dos usuários no cenário político através de petições online ou organização de manifestações.

Este trabalho tem como objetivo estudar as opiniões dos usuários do Twitter acerca do cenário político brasileiro atual utilizando técnicas de processamento de linguagem natural em conjunto com técnicas de aprendizado de máquina.

Para isso coletou-se tuítes sobre grandes decisões atuais como a reforma da previdência (PEC287), a proposta de lei da terceirização e a proposta de emenda constitucional sobre o teto dos gastos públicos (PEC55) bem como sobre políticos em grande visibilidade no cenário político atual. Para a classificação manual dos tuítes, desenvolveu-se um site onde era possível realizar a tarefa de forma menos árdua e permitir a ajuda de outras pessoas. Aplicou-se técnicas de processamento de linguagem natural para extrair informações subjetivas dos tuítes e classificou-se as opiniões associadas a eles utilizando técnicas de aprendizado de máquina. Uma vez desenvolvido os classificadores, analisou-se métricas de desempenho de cada um como acurácia, precisão e revocação para diferentes abordagens de extrair as informações subjetivas do texto.

Ao fim de uma primeira ronda de experimentos, percebeu-se que a baixa quantidade de tuítes com opinião positiva atrapalhava o desempenho geral dos algoritmos e, portanto decidiu executar mais uma ronda sem eles classificando então os tuítes apenas em negativo e neutro. Ao remover estes tuítes obteve-se uma melhoria nos classificadores obtendo resultados médios acima de 67% em todas as métricas e um classificador que pode ser usado com bom desempenho, mesmo com os dados *in natura*.

Capítulo 1

Introdução

Capítulo 2

Revisão Bibliográfica

Diversos estudos recentes têm sido realizados na área de análise de sentimentos. Medhat (2014)[1], sumariza diversos estudos feitos dividindo-os em abordagens (usando técnicas de aprendizado de máquina, dicionário léxico ou híbrida) e em objetivos (classificação de sentimentos, detecção de emoções etc).

Em Pak (2010)[2] é analisado o uso de um corpus compostos de tuítes para a realização de tarefas de análise de sentimentos e mineração de opinião. Nesse estudo é construído um classificador de sentimentos para as opiniões associadas aos tuítes coletados e é discutido desafios encontrados na hora de desenvolver um classificador para o corpus.

Outros estudos, assim como este trabalho, têm como foco o uso de análise de sentimentos aplicados à política utilizando tuítes como o corpus. Em Tumasjan et. al (2010) é estudado se é possível utilizar o Twitter para obter uma previsão para os resultados de uma eleição tendo como base as eleições do parlamento alemão realizadas em 2009.

Risquandl e Petković (2013)[3] realizam experimentos tendo como cenário as eleições presidenciais estadounidenses de 2012 para classificar sentimentos dos usuários do twitter, porém diferente dos outros trabalhos anteriormente mencionados, esse estudo utiliza técnicas de part of speeching tagging para identificar sobre quem os tuítes se referem e, a partir disso, classifica o sentimento baseado a aspectos dessa entidade, no caso pautas eleitorais que foram fortemente discutidas como casamento de pessoas do mesmo gênero, teaparty etc.

Bakliwal et. all (2013)[4] realiza um estudo semelhante ao deste trabalho a classificação de opinião tendo como base três classes (positivo, negativo ou neutro) a cerca de tuítes coletados sobre as eleições gerais da Irlanda em fevereiro de 2011. A diferença entre os trabalhos é que Bakliwal associa a cada partido um sentimento associado a um dos partidos.

Capítulo 3

Machine Learning e Análise de Sentimentos

3.1 Considerações

Ao longo deste capítulo usaremos n para se referir à quantidade de elementos do conjunto de entrada, cada entrada é i é um vetor $x_i \in \mathbb{R}^m$. O conjunto de entrada entrada será referida como X durante as descrições do algoritmo e assim cada entrada será dada tanto como X_i como x_i. Para cada i associaremos duas variáveis t_i e y_i que se referem ao valor esperado e ao valor obtido através do treinamento, respectivamente.

A notação $\mathbb{1}_{i==j}$ é uma função indicadora que vale 1 se i é igual a j e 0 caso contrário.

3.2 Contextualização

Os problemas tratados por *Machine Learning* classificam-se de forma geral em três tipos:

- Aprendizado supervisionado: nesse caso tem-se os elementos de entrada e para cada um desses elementos, tem-se associado um rótulo t_i. Nesse caso o modelo deve ser treinado com base nos elementos dados para que se possa prever o rótulo de uma nova entrada;

- Aprendizado não-supervisionado: nesse caso tem-se apenas os elementos de entrada. O objetivo deste tipo de problema é tentar modelar uma distribuição ou estrutura comum entre os dados para que se possa entendê-los melhor;

- Aprendizado semi-supervisionado: nesse último caso alguns elementos possuem um rótulo associado. Problemas desse tipo aplicam técnicas gtanto de aprendizado supervisionado como de não-supervisionado.

Neste trabalho será tratado um problema de aprendizado supervisionado que é o da classificação.

Na classificação temos k classes e cada elemento i da entrada é associado a uma classe $t_i = \{1..k\}$. O objetivo do problema da classificação é dado entrada $X = (x_1, x_2, \ldots, x_n)$ e $t = (t_1, \ldots, t_n)$ treinar um modelo capaz de prever classes para um x qualquer.

Há diversos algoritmos na literatura que se propõem a resolver o problema da classificação. Bishop (2006)[5] enuncia diversos dos algoritmos comumente utilizados para a classificação, cada algoritmo possui seus prós e contras e utiliza diferentes abordagens.

Para este trabalho escolheu-se implementar os algoritmos *Logistic Regression* e *Support Vector Machines*, que será chamado simplesmente de SVM por facilidade.

Tanto para o *Logistic Regression* quanto SVM será explicado a princípio o problema será inicialmente abordado a partir da classificação binária e, a partir dela, será descrito como estender para o problema com mais de duas classes, que será é o caso deste trabalho.

3.3 Logistic Regression

O nosso modelo será construído de forma probabilística, isto é, a partir de um discriminante linear $w^T x + w_0$ atribuiremos uma probabilidade de um elemento x pertencer à classe C^1 e, consequentemente a probabilidade de pertencer à classe C^2 é dada por $1 - P(C^1|x)$. O termo w_0 é chamado de viés, e para efeito das contas que serão feitas consideraremos vetores w' e x' da forma $x' = (x, 1)$, $w' = (w, w_0)$ note entretanto que os chamaremos daqui pra frente simplesmente de w e x.

No caso da classificação binária, usaremos que $t_n \in \{0, 1\}$ onde $t_n = 1$ se o elemento pertence à classe C^1 e $t_n = 0$ se pertence à classe C^2.

A classificação de um elemento será a classe a qual ele tem maior probabilidade de pertencer.

Para utilizarmos nosso discriminante para atribuir as probabilidades, utiliza-se a função sigmóide definida por:

$$\sigma(a) = \frac{1}{1 + exp(-a)} \tag{3.1}$$

A função sigmóide é usada devido à sua propriedade de mapear todo o conjunto dos números reais dentro do intervalo $[0, 1]$.

Aplicando ao nosso modelo obtêm-se a expressão:

$$P(C^1|x) = y(x) = \sigma(w^T x) \tag{3.2}$$

Importante notar que apesar de utilizarmos o vetor x nas equações, é possível aplicarmos uma transformação linear $\phi : \mathcal{R}^m \to \mathcal{R}^d$ à entrada x para obtermos $\phi(x)$ e usá-lo no lugar de x. O uso de transformação linear no nosso conjunto de entrada nos permite transformar o domínio para que se obtenha uma separação melhor entre as classes ou até mesmo fazer a redução da dimensão do domínio.

Com essa equação em mãos, nosso objetivo é minimizar o erro na classificação dos dados. Tomamos como erro o negativo do logaritmo da verosimilhança de nossa função que é dada por:

$$E(w) = -\sum_{i=1}^{n} p(t|w) = -\sum_{i=1}^{n} \{t_n ln(y_n) + (1 - t_n) ln(1 - y_n)\} \tag{3.3}$$

A fim de minimizar o erro, utiliza-se métodos de otimização linear (note que por mais que se use uma transformação linear ϕ sobre x nosso problema ainda é linear sobre w).

Dois métodos são comumente usados: método do gradiente e método de Newton-Raphson. Esses métodos são utilizados tanto para o caso da classificação binária quanto o caso da classificação com $k > 2$. A diferença entre um problema e outro será abordada com mais detalhes a seguir.

Uma dúvida natural que surge ao ter que resolver um problema de otimização é o caso de parar o procedimento em um mínimo local ao invés de um mínimo local da função. Entretanto, temos que nossa função $E(w)$ é côncava, isto é, $E(\lambda w + (1 - \lambda)w') = \lambda E(w) + (1 - \lambda)w' \ \forall w, w' \in R^m, \lambda \in [0, 1]$, tal propriedade nos garante que existe um único minizador.

3.3.1 Método do Gradiente

Para este método, minimiza-se a função objetivo, no caso $E(w)$ utilizando apenas o gradiente da função junto de um passo α. Com ambos valores em mãos, o valor w é atualizado usando a equação:

$$w^{(novo)} = w^{(antigo)} + \alpha \nabla E(w) \tag{3.4}$$

Com $\nabla E(w)$ sendo o gradiente do vetor de pesos. O gradiente é calculado usando o fato de que a derivada da função sigmóide com respeito a um vetor a é dada por:

$$\frac{d\sigma}{da} = \sigma(1 - \sigma) \tag{3.5}$$

Usando 3.5 tem-se a seguinte equação para o gradiente:

$$\nabla E(w) = X^T(y - t) \tag{3.6}$$

Com $y = (y_1, \ldots, y_n)$ e $t = (t_1, \ldots, t_n)$ onde $y_n = P(C^1|x_n) = \sigma(w^T x)$ e t_n tal qual assumido no começo da seção.

O algoritmo de atualização do vetor de pesos descrito a seguir vale tanto para o método do gradiente quanto para o de Newton-Raphson, portanto para o segundo será focado apenas nas diferenças entre os dois.

Algorithm 1 Logistic Regression usando método do gradiente

Input: Matriz $X \in \mathbb{R}^{n \times m}$, vetor de rótulos $t \in \{0, 1\}^n$
Output: Vetor de pesos $w \in \mathbb{R}^m$

1: $iteracao \leftarrow 0$
2: $w \leftarrow 0$
3: **while** $|E(w)^{(iteracao)} - E(w)^{(iteracao-1)}| \geq \epsilon$ **and** $iteracao < maxIteracoes$ **do**
4: $\quad y \leftarrow (\sigma(w^T x_1), \sigma(w^T x_2), \ldots, \sigma(w^T x_n))^T$
5: $\quad \nabla E(w) \leftarrow X^T(y - t)$
6: $\quad w \leftarrow w - \alpha \nabla E(w)$
7: $\quad E(w)^{(iteracao)} \leftarrow -\sum_{i=1}^{n} \{t_n ln(y_n) + (1 - t_n) ln(1 - y_n)\}$
8: $\quad iteracao \leftarrow iteracao + 1$
9: **end while**

Importante notar que em 3 tem-se duas condições de paradas do algoritmo que são o número de iterações e a diferença da diminuição da função objetivo for menor do que um dado ϵ. Tais condições são chamadas de condições de

convergência e nos garantem que chegamos a um valor suficientemente próximo do ótimo, uma vez que atingir este valor pode exigir um número muito alto de iterações o que acarreta em grande custo computacional. Na implementação do algoritmo, escolheu-se um valores padrão para ϵ e $maxIteracoes$ como 10^{-4} e 200 respectivamente.

A quantidade de iterações necessárias para a convergência é influenciada fortemente pela escolha de α, pois as escolha de um valor pequeno para α acarretaria em muitas iterações para convergir ao passo que um valor muito grande pode fazer com que o algoritmo pare num valor distante do ótimo.

3.3.2 Método de Newton-Raphson

Vimos em 3.3.1 que o método do gradiente apesar de implementação simples pode levar muito tempo para resolver o problema.

O método de Newton-Raphson acaba convergindo mais rápido do que o método do gradiente, contudo ao custo de uma maior complexidade devido à necessidade de calcular outros elementos.

A atualização agora é feita seguindo a equação

$$w^{(novo)} = w^{(antigo)} - H^{-1}\nabla E(w) \tag{3.7}$$

Onde H é a matriz Hessiano da função erro, que é calculado usando $H = \nabla\nabla E(w) = X^T R X$ onde R é uma matriz diagonal $n \times n$ onde as entradas da diagonal principal valem $R_{kk} = y_k(1 - y_k)$. Substituindo os valores de H e usando 3.6 em 3.7 obtemos

$$\begin{aligned} w^{(novo)} &= w^{(antigo)} - (X^T R X)^{-1}\nabla E(w) \\ &= (X^T R X)^{-1}[(X^T R X)w^{(antigo)} - X^T(y - t)] \end{aligned} \tag{3.8}$$

3.3.3 Extensão para o caso de várias classes

Diversas abordagens podem ser usadas para resolver o problema multiclasse, no caso será usado diversos discriminantes y_k com $k = \{1, \dots K\}$ com K sendo o total de classes. Assim nosso vetor w agora é uma matriz $W \in \mathbb{R}^{m \times k}$. Outra representação que usaremos para o algoritmo será $W = (w_1, \dots, w_k)$ com $w_i \in \mathbb{R}^{1 \times mk}$.

Quanto à codificação do vetor de rótulos, segue-se a codificação dada em Bishop (2006)[5] de $1-K$, na codificação tem-se que $t_n \in \{0, 1\}^k$ com $t_{nk} = 1$ se o elemento n pertencer à classe k e 0 nas demais entradas.

Quanto a função de probabilidade que desejamos estimar, utiliza-se a função *softmax* que é dada pela equação:

$$P(C^k|x_n) = y_{nk} = \frac{exp(w_k^T x_n)}{\sum_j exp(w_j^T x_n)} \tag{3.9}$$

Que nos dá verossimilhança e a seguinte função de erro, obtida tomando o negativo do logaritmo da verossimilhança.

$$P(T|w_1, \ldots, w_k) = \prod_{i=1}^{n}\prod_{j=1}^{k} P(C^j|x_i)^{t_{ij}} = \prod_{i=1}^{n}\prod_{j=1}^{k} y_{ij}^{t_{ij}}$$

$$E(W) = -\sum_{i=1}^{n}\sum_{j=1}^{k} t_{ij} ln(y_{ij})$$

Novamente nesse caso pode-se encontrar o valor de W que minimize $E(W)$ usando os dois métodos discutidos em 3.3.1 e 3.3.2, porém agora temos que a derivada com respeito a cada $w_k^T x$ vale:

$$\frac{\partial y_k}{\partial (w_j^T x)} = y_k(\mathbb{1}_{k==j} - y_j) \tag{3.10}$$

Usando essa representação de W como um vetor, podemos calcular o vetor gradiente onde a derivada com respeito a cada w_j é dada pela equação:

$$\nabla_{w_j} E(W) = X^T(Y_j - T_j) \tag{3.11}$$

Com Y_j e T_j correspondendo, respectivamente, às j-ésimas colunas de Y e T.

Com o gradiente em mãos já temos o que é necessário para o método do gradiente e a atualização seria feita da forma $W^{(novo)} = W^{(antigo)} - \alpha \nabla E(W)$.

Para aplicarmos o método de Newton-Raphson, seria necessário computarmos o Hessiano que nesse caso seria uma matriz $m * k \times m * k$ com cada bloco j, i contendo uma matriz $m \times m$ calculada pela equação:

$$\nabla_{w_i}\nabla_{w_j}E(W) = -\sum_{k=1}^{n} y_{ki}(\mathbb{1}_{i==j} - y_{kj})X_k^T X_k \tag{3.12}$$

Onde X_k é a k-ésima linha de X. Com essas equações em mãos nossa atualização de W seria feita usando a fórmula $W^{(novo)} = W^{(antigo)} - H^{-1}\nabla E(W)$.

A classificação de um novo x é feita a partir do cálculo de $P(C^k|x) = y_k(x), \forall k = \{1, \ldots, k\}$.

A classe de x é dada pelo k que tiver a maior probabilidade sobre os demais.

3.4 Support Vector Machine

Assim como fizemos com a regressão logística, começaremos com a definição para o caso binário e depois iremos estender para mais de uma classe. Nesse caso nossas classes serão $t_n \in \{-1, 1\}$ onde $t_n = 1$ se x pertence à classe C^1 e $t_n = -1$ se x pertence à classe C^2.

No algoritmo SVM a classificação é feita a partir de um discriminante linear da forma

$$y(x) = w^T x + b \tag{3.13}$$

Tal y é chamado de superfície de decisão e a classificação é baseada no sinal de y. Se $y(x) > 0$, x é atribuído à classe C^1, caso contrário é atribuído à classe C^2.

Porém ao invés de procurarmos um w que separe perfeitamente todas as classes (que não necessariamente existe), nosso objetivo é maximizar a margem do discriminante linear, isto é, a menor distância de um ponto à superfície. A distância de um ponto à superfície é dado pela fórmula

$$\frac{[t_n(w^T x_n + b)]}{||w||} \tag{3.14}$$

Na descrição inicial do problema vamos tratar o caso com o conjunto X linearmente separável, o que indica que é possível obter uma superfície que separe sem erro todas as classes para, em seguida, tratarmos o caso real que é o do conjunto que não é linearmente separável. O fato de o conjunto ser linearmente separável nos garante que $t_n y(x_n) > 0 \forall n$.

$$\underset{x,b}{\text{argmax}} \left\{ \frac{1}{||w||} min_n[t_n(w^T x_n + b)] \right\} \tag{3.15}$$

Podemos ajustar w e b de forma a termos que $t_n(w^T x_n + b) = 1$ para o ponto mais próximo da margem e $t_n(w^T x_n + b) \geq 1, \forall n$. Com esse reajuste temos que nosso problema de encontrar um vetor de pesos de margem maximizada seria de maximizar $\frac{1}{||w||}$ que é equivalente ao problema de otimização quadrática:

$$\begin{array}{ll} \text{argmin}_{w,b} & ||w||^2 \\ \textbf{sujeito a} & t_k(w^T x_k + b), \quad k = 1, \ldots, n \end{array} \tag{3.16}$$

Agora iremos supor que não necessariamente nossa entrada não é linearmente separável, isto é, não existe uma superfície de decisão que separe perfeitamente as duas classes. Assim iremos permitir que alguns valores estejam classificados incorretamente, para isso será necessário suavizarmos nossa margem penalizando cada uma das entradas incorretamente classificadas. Cortes e Vapnik (1995)[6] descrevem a penalização através da introdução de variáveis de folga $\xi_n \geq 0$ para cada elemento de X. Um elemento corretamente classificado teŕa $\xi_n = 0$, os demais pontos têm $\xi_n = |t_n - y(x_n)|$.

Com isso nossa restrição de $t_n y(x_n) \geq 1$ é modificada e tem-se $t_n y(x_n) \geq 1 - \xi_n \forall n$.

O valor de ξ_n assim nos indica três possíveis casos:

- Se $\xi_n = 0$, x_n está corretamente classificado e se encontra ou na margem ou do lado correto dela.
- Se $0 < \xi_n \leq 1$, x_n está corretamente classificado e se encontra entre a margem e a superfície.
- Se $\xi_n > 1$, x_n não está classificado corretamente.

A função objetivo agora precisa conter os valores de ξ e para isso colocamos uma constante $C > 0$ que define a compensação entre a penalização das variáveis de folga e a margem. Com isso temos o seguinte problema de otimização quadrática:

$$\begin{array}{ll} \text{argmin}_{w,b,\xi} & C \sum_{i=1}^{n} \xi_i + \frac{1}{2}||w||^2 \\ \text{sujeito a} & t_k(w^T x_k + b) \geq 1 - \xi_k, \quad k = 1, \ldots, n \end{array} \tag{3.17}$$

Para implementar o SVM basta então resolver o problema de otimização quadrática acima. Isto é feito seguindo os seguintes passos

1. Introduz-se multiplicadores de Lagrange α_n e μ_n e obtem-se o Lagrangiano com as restrições

$$L(w,b,\xi) = \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i\{t_i(w^T x_i + b) - 1 + \xi_i\} - \sum_{i=1}^{n}\mu_i\xi_i \tag{3.18a}$$

$$\alpha_n \geq 0 \tag{3.18b}$$

$$t_n y(x_n) - 1 + \xi_n \geq 0 \tag{3.18c}$$

$$\alpha_n\{t_n(w^T x_n + b) - 1 + \xi_n\} = 0 \tag{3.18d}$$

$$\mu_n \geq 0 \tag{3.18e}$$

$$\mu_n\xi_n = 0 \tag{3.18f}$$

2. Derivamos o lagrangiano com respeito a w, b e ξ e igualamos a 0 para obtermos os valores ótios para essas variáveis e, assim obtemos os seguintes valores:

$$\frac{\partial L}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^{n}\alpha_i t_i x_i \tag{3.19}$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{i=1}^{n}\alpha_i t_i = 0 \tag{3.20}$$

$$\frac{\partial L}{\partial \xi} = 0 \Rightarrow \alpha_n = C - \mu_n \tag{3.21}$$

3. Com isso em mãos, resolve-se não o problema primal e sim o dual (utiliza-se aqui o fato de que se as condições mencionadas no item anterior são satisfeitas, tem-se que vale a dualidade forte e o valor ótimo de ambas as funções coincide). O dual é dado pelo problema

$$\begin{aligned} \max_{\alpha} \quad & \tilde{L}(\alpha) = \sum_{i=1}^{n}\alpha_i - 1/2\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j t_i t_j k(x_i, x_j) \\ \text{sujeito a} \quad & 0 \leq \alpha_i \leq C, i = 1, \ldots, n \\ & \sum_{i=1}^{n}\alpha_i t_i = 0 \end{aligned} \tag{3.22}$$

No problema a cima é importante destacar a expressão $k(x_i, x_j)$, essa expressão é um *Kernel* que é uma função onde $k(x, x') = \phi(x)^T \phi(x')$ com ϕ sendo alguma transformação linear. Tal qual no caso da regressão logística, essas transformações são usadas para mudar o domínio da entrada x.

4. Acha-se o valor de b usando a fórmula:

$$b = \frac{1}{N_{\mathcal{M}}} \sum_{n \in \mathcal{M}} \left(t_n - \sum_{m \in \mathcal{S}} \alpha_m t_m k(x_n, x_m) \right) \tag{3.23}$$

Com $\mathcal{M}$ sendo o conjunto de pontos que satisfazem $0 < \alpha_n < C$ e $\mathcal{S}$ o conjunto de vetores de suporte (pontos que possuem $\alpha_n > 0$ e, consequentemente, contribuem para a classificação do modelo).

Uma vez resolvido o problema dual e encontrado valor de b, podemos classificar um novo x usando o sinal do discriminante $y(x)$ dado por

$$y(x) = \sum_{i=1}^{n} \alpha_i t_i k(x, x_i) + b = \sum_{n \in \mathcal{S}} \alpha_n t_n l(x, x_n) + b \tag{3.24}$$

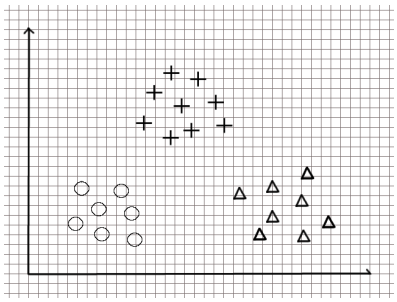
3.4.1 Extensão ao caso de multiclasses

Diversas abordagens são possíveis para o caso de multiclasses. A escolhida entre elas foi o método intuitivo chamado *One-versus-all* (OVA). Nesse método é construído K classificadores, com K sendo o número de classes. Cada classificador y_k define uma superfície de decisão que separa a classe k das demais (por isso o nome *One-versus-all*). Um novo x tem sua classe dada pela que o classificador y_k tem maior valor, isto é

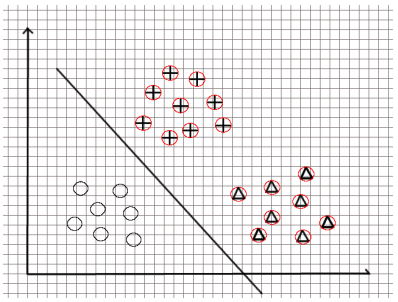
$$y(x) = \underset{k}{\operatorname{argmax}}\, y_k(x) \tag{3.25}$$

Portanto a classificação multiclasses se utiliza de todos os recursos já apresentados no caso binário o que torna simples a construção do classificador.

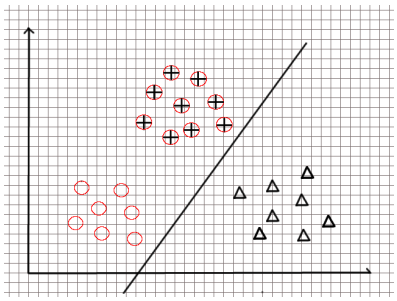
Na imagem abaixo é mostrado a ideia por trás do método OVA nele a classe C^1 é dada pelos círculos, C^2 pelos triângulos e C^3 pelas cruzes. As figuras em vermelho em cada classificador são os pontos onde $t_n = -1$.



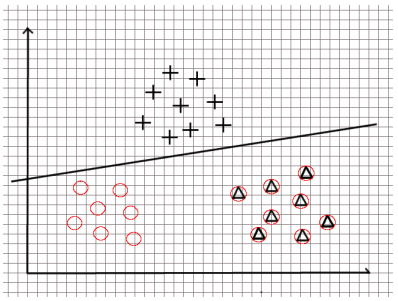
(a) Classes originais



(b) Classificador para a classe 1



(c) Classificador para a classe 2



(d) Classificaodr para a classe 3

3.5 Análise de Sentimentos

O campo de análise de sentimentos, também conhecido como mineração de opinião é uma área da ciência da computação que analisa as opiniões, sentimentos e atitudes das pessoas em relação a uma entidade (Bing Liu, 2012)[7]. Uma entidade pode ser definida como o sujeito ao qual está se observando as opiniões, seja ela uma pessoa, instituição ou produto.

A área possui uma diversa gama de aplicações, dentre elas, a área de classificação de sentimentos que através do uso de informações subjetivas contidas num texto avalia qual a opinião relacionada ao mesmo.

Classificação de sentimentos teve um crescimento devido à expansão da Web 2.0 que fez com que as pessoas manifestassem suas opiniões e sentimentos através de blogs, fóruns, redes sociais e páginas de reviews de produtos que, consequentemente fez com que empresas e pessoas de interesse tivessem um acesso mais aberto e fácil a essas opiniões. O fácil acesso às opiniões dos usuários junto com o grande volume delas tornou a análise manual um processo muito custoso, tornando necessário recorrer a métodos automatizados para a análise e sumarização desses dados que acabou por fim impulsionando estudos na área (Petković e Ringsquandl, 2013)[3].

Bing Liu (2012)[7] divide as formas de classificar sentimentos em um

documento pode se dividir em quatro formas:

- Entidade: um produto, pessoa sobre o qual o texto se refere, seja direta ou indiretamente. Nessa tarefa procura-se analisar se a opinião do texto em torno de uma entidade é positiva ou não.

- Aspecto: características sobre uma entidade. Por exemplo se temos uma entidade que é um produto, aspectos de um produto poderiam ser material ou preço. Nessa forma procura-se avaliar a opinião acerca de cada um dos aspectos.

- Sentença: esse tipo de análise trabalha com o sentimento associado a cada sentença de um documento.

- Documento: nesse caso analisa-se o sentimento associado a um documento como um todo, tratando de uma forma mais genérica em relação aos demais.

Comum a todas as formas de se analizar as opiniões de um documento é que o uso dos métodos para obter a opinião. Há duas abordagens para esse problema de classificação: a de aprendizado de máquina (que iremos utilizar nesse trabalho) que se consiste do uso de algoritmos de classificação em conjunto com técnicas de processamento de texto (que será explicado nas próximas seções) e a abordagem com um dicionário léxico onde a análise é feita baseada na pontuação entre palavras positivas e negativas contidas no documento (Medhat et al., 2014) [1].

No caso desse trabalho, foi escolhido a análise em torno do documento / sentença (devido à estrutura dos *tweets* tem-se que os documentos são, em sua maioria, compostos por apenas uma sentença).

3.5.1 Análise de *Tweets* de política

Dentre os domínios de aplicação da classificação de sentimentos, escolheu-se como domínio o escopo político. Nesse caso, temos que as entidades nos documentos são constituídas por políticos e projetos de lei. E neste trabalho será analisado a análise de *tweets*.

O twitter desde as eleições presidenciais estadunidenses de 2008 mostrou influência na decisão das eleições evidenciado pela campanha online realizada pela equipe do candidato Barack Obama (Petković e Ringsquandl, 2013)[3]. Desde então, a rede social tem sido usada não só para a campanha de políticos, mas também para a avaliação da opinião em relação às medidas tomadas por eles.

A escolha dos *tweets* foi feita tendo em vista o uso da rede social para expressar opiniões sobre diversos assuntos, incluindo política.

3.5.2 Obtendo um vetor de *features*

Para realizar a extração das informações do texto, existe uma etapa de pré-processamento na qual cada texto é transformado num vetor numérico para então ser processado por algum algoritmo de classificação. Neste trabalho foi escolhido usar o modelo *bag of words* (BOW). No modelo, é construída uma matriz de entrada onde cada linha i representa um documento (texto sobre o qual quer extrair a opinião) e as colunas representam a frequência de um termo, o conjunto de todos os textos é chamado de *corpus*.

Uma alternativa ao BOW seria o modelo de N-grama, nele os termos não são apenas as palavras, mas um conjunto de n palavras juntas, o que traz mais informação pelo fato de juntar substantivo e verbo, substantivo e adjetivo, por exemplo, entretanto acaba gerando um vetor de *features* ainda maior que, consequentemente, faz com que os algoritmos demorem mais para convergir. A escolha pelo BOW foi tida baseada no compromisso entre desempenho e performance do algoritmo.

Antes de obter um vetor numérico, é feito um pré-processamento no documento removendo artigos, adicionando espaços antes e depois de pontuações e transformando em um vetor de tokens. Uma vez que foi obtido o vetor de *tokens*, é removido todo elemento que seja apenas espaço e pontuação. Em seguida, monta-se uma nova *string* juntando todas as *tokens* usando espaços para então montar um vetor de frequência a partir do novo *corpus*.

A frequência pode ser medidas simples como a quantidade de vezes em que um termo j aparece no documento i ou se o termo j aparece no documento i (nesse caso cada vetor de entrada seria um vetor binário) bem como pode ser uma medida mais complexa como tf-idf (*term frequency - inverse document frequency*). A relação entre os métodos de computar a frequência e o desempenho dos algoritmos de classificação serão exibidas no capítulo seguinte.

O método tf-idf baseia-se na frequência de cada termo ponderada pela frequência inversa no documento, que é usada para mensurar o quão importante um termo é em um documento (por exemplo a palavra "um"em um conjunto de textos em português aparece várias vezes, apesar de ter uma baixa importância). Ele pode ser calculado pelas expressões:

$$TF(t) = \frac{\#\text{ de aparições de t no documento}}{\text{total de termos no documento}} \tag{3.26a}$$

$$IDF(t) = log\left(\frac{\text{Total de documentos}}{\#\text{ de documentos que contenham t}}\right) \tag{3.26b}$$

$$TF - IDF(t) = TF(t) * IDF(t) \tag{3.26c}$$

O procedimento de obtenção dos vetores de *features* a partir do novo *corpus* pode ser realizado utilizando as bibliotecas `scikit-learn` e `nltk` do Python.

3.5.3 Classificação manual da opinião

Com o conjunto de dados obtido em 3.5.2, podemos aplicar os algoritmos descritos no começo do capítulo. Entretanto como também mencionado no começo deste capítulo, problemas de aprendizado supervisionado exigem que o conjunto de dados seja composto não só dos dados, mas também do valor esperado para cada entrada. No caso da classificação da opinião de textos, tal opinião deve ser primeiro atribuída manualmente para que, com posse dessas opiniões, seja possível treinar os algoritmos.

Para facilitar a etapa de classificação manual, foi desenvolvido um sistema online onde cada usuário informa a quantidade de *tweets* que se deseja classificar e consegue classificá-los de forma mais simples. Deu-se a essa ferramenta o nome de CLAM (de Classificador Manual).

O desenvolvimento do CLAM foi motivado pela ausência de ferramentas no Brasil que permitem a colaboração nessa etapa de classificação manual (uma ferramenta comumente usada é o *Amazon Mechanical Turk*, porém na época do desenvolvimento do CLAM não estava disponível no Brasil) em conjunto com a falta de ferramentas gratuitas de uma forma geral, uma vez que até que a maioria das ferramentas existentes são pagas (vide o próprio *Amazon Mechanical Turk*). Por isso priorizou-se construir um código simples para que fosse fácil a modificiação da base do sistema por colaboradores externos, que fosse de fácil uso para o usuário final e também de fácil hospedagem do sistema em alguma plataforma como Heroku e Firebase.

O código é de domínio público e está disponível no link: `https://github.com/romaolucas/manual-classifier-helper`

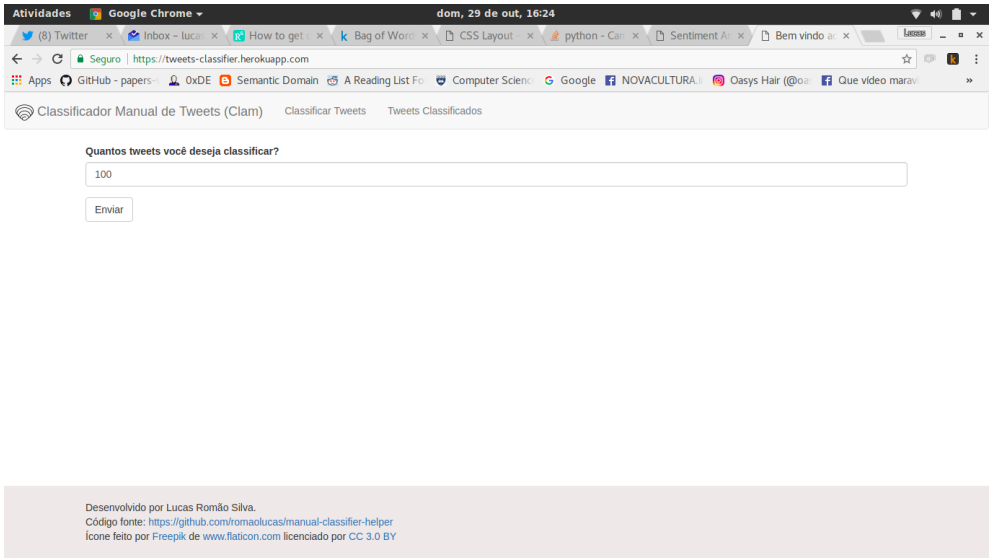
O projeto foi desenvolvido usando a linguagem Python em conjunto do *framework* web Django por já possuir diversas ferramentas integradas não só de gerenciamento do sistema, bem como modelagem do banco de dados e sistema de migração para que possa modificar os modelos de dados já existentes sem precisar realizar um acesso direto à base de dados além de possuir um conjunto de bibliotecas que facilitam o processo de importação

de dados em csv e vasta quantidade de tutoriais de como desenvolver uma aplicação usando Django.

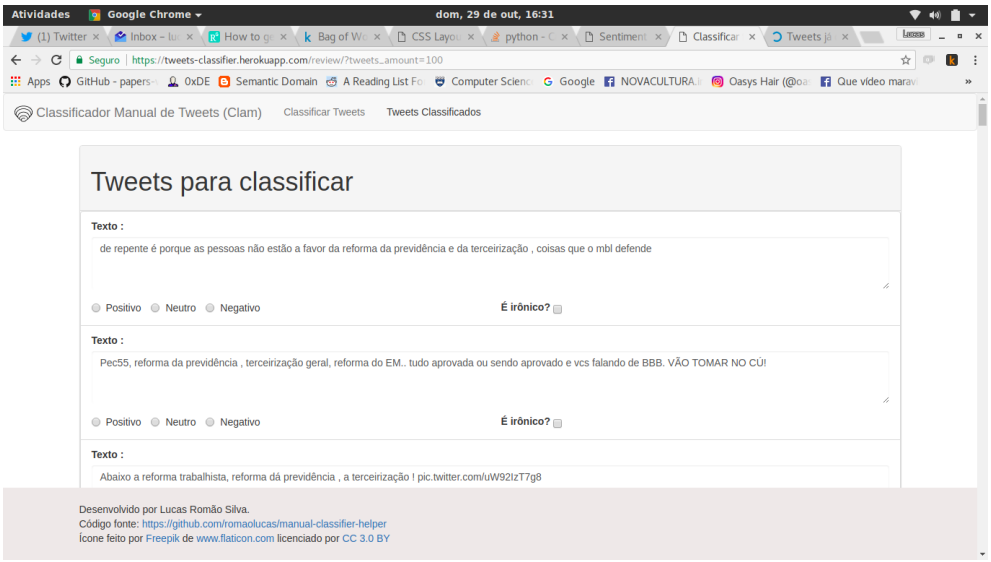
Para a modelagem de dados construiu-se dois modelos: um para *tweets* e outro para as opiniões. Na modelagem, considerou-se que cada usuário irá classificar apenas textos que não possuem uma classificação, não só para evitar ter que lidar com opiniões divergentes em um texto, mas também para garantir um maior número de dados para o treinamento. Uma possível melhoria do sistema seria a possibilidade de mais usuários avaliarem um mesmo conjunto de textos para garantir um consenso na hora de associar uma opinião a um texto.

O CLAM possui o seguinte fluxo para a classificação:

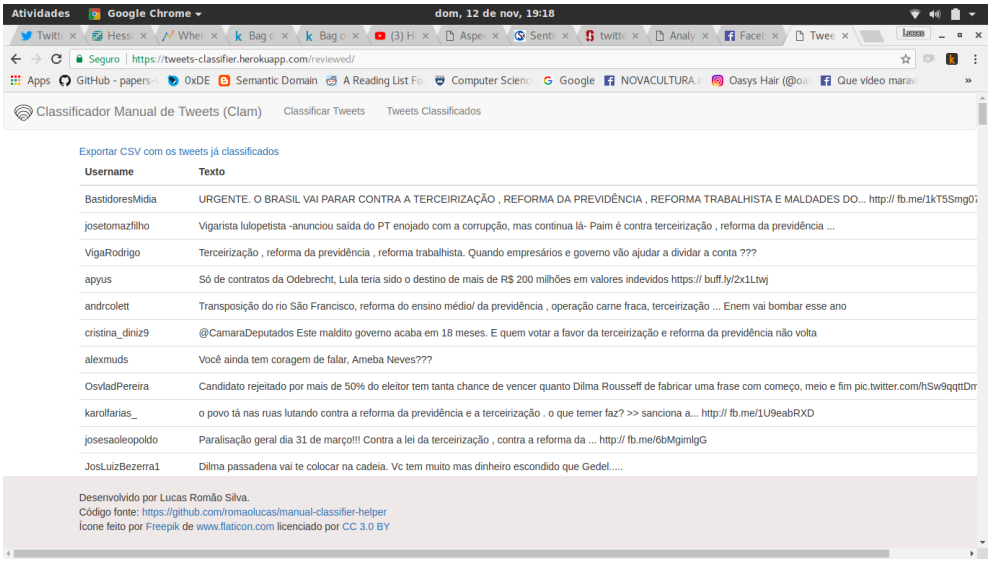
1. Informa a quantidade de *tweets* que se deseja classificar.



2. O usuário é levado a uma página com os n *tweets* para classificar. Enquanto o campo da opinião é obrigatório, existe um campo optativo para informar se o dado *tweet* é irônico ou não, uma vez que textos com ironia atrapalham o treinamento por conter palavras positivas sendo usadas de maneira negativa e vice-versa.

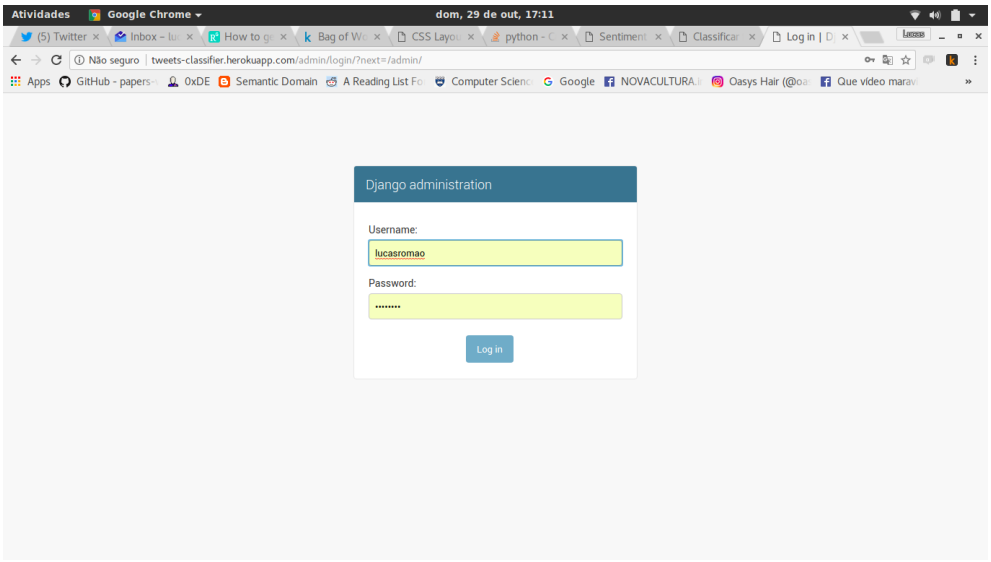


3. Uma vez classificados, o usuário é levado a uma página onde é possível ver todos os *tweets* já classificados e também gerar um arquivo csv contendo todos aqueles que não foram marcados como irônicos.

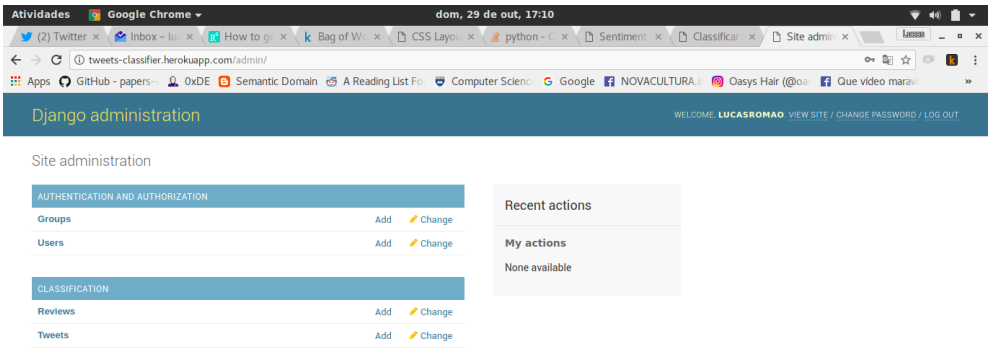


Para um usuário que deseja hospedar a plataforma e usá-la para si, existe também o admin onde é possível importar *tweets* para serem classificados. Abaixo é descrito o fluxo para a importação de novos dados no admin.

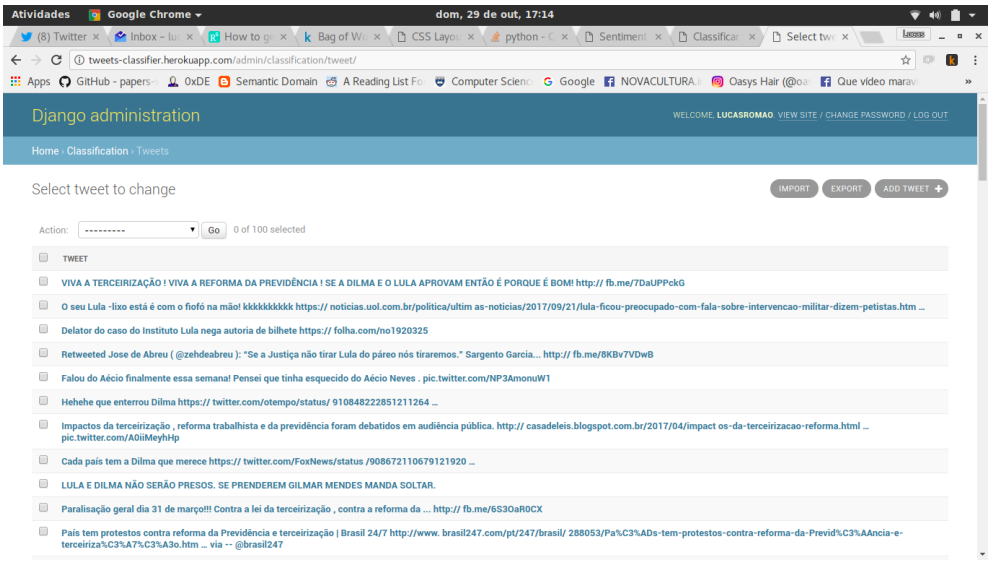
1. Realiza o login no sistema utilizando seu usuário e senha de administrador.



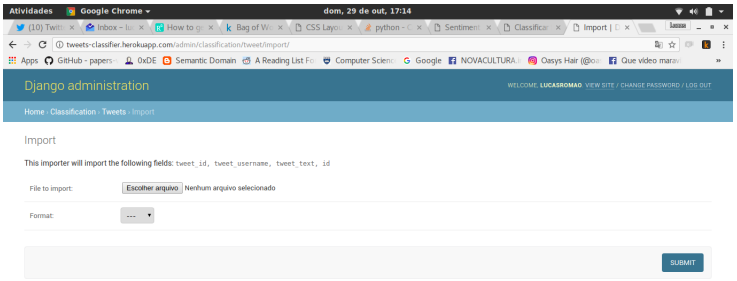
2. Seleciona a parte de Tweets na tela principal.



3. Uma vez na parte de visualizar todos os *tweets* já armazenados na base de dados, seleciona a opção de importar no canto superior direito.



4. Na página que segue, basta informar um arquivo .csv, .xls (formato do excel) ou .json contendo nessa ordem: o id do *tweet* (fornecido pelo Twitter), usuário que escreveu, texto, espaço vazio para representar o id que será preenchido automaticamente na hora de importar no banco de dados.



5. Caso todos os dados sejam fornecidos corretamente, será passado a uma nova página para confirmar se deseja importar e, por fim, será redirecionado à pagina que contém todos os *tweets*.

Capítulo 4

Experimentos

4.1 Análise exploratória

4.1.1 Distribuição das classes

Antes de mais nada, analisaremos a composição do nosso conjunto de dados e como se dá a distribuição de classes. Na figura 4.1 temos um histograma das classes associadas a cada tuíte.

Classe	Total
Positivo	115
Negativo	1248
Neutro	1223

Quantidade de tuítes pertencentes a cada classe

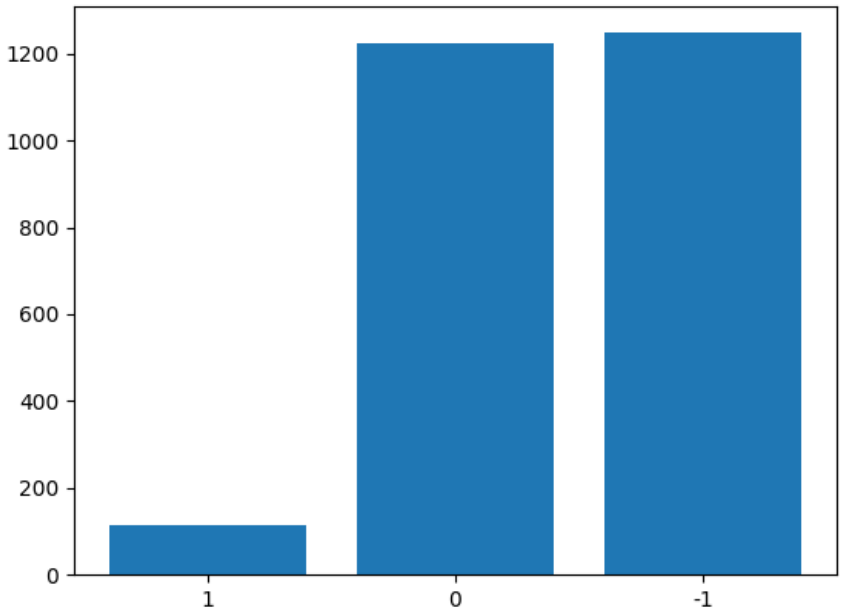


Figura 4.1: Histograma da frequência de tuítes

Como podemos observar a partir da figura 4.1 e tabela ??, temos que enquanto as classes negativa e neutra possuem distribuição próxima (48,26% e 47,29% respectivamente) a classe positiva corresponde a apenas 4,45% de todos os tuítes. Mais a frente veremos que esse desbalanceamento entre as classes irá acarretar em um desempenho ruim por parte dos algoritmos de classificação, sejam os desenvolvidos para este trabalho quanto os da biblioteca `scipy`.

4.1.2 Características dos tuítes de cada classe

A fim de entender como os classificadores realizariam o treinamento, analisou-se o conjunto de dados a procura de padrões de características para cada classe.

Para os tuítes avaliados como neutro, notou-se o padrão que são tuítes derivados de portais de notícias ou são indagações, porém sem nenhuma crítica feita nelas. A tabela abaixo mostra como exemplo alguns tuítes avaliados como neutro.

usuário	tuíte
Tica_Fernandes	Cidades têm protestos contra reforma da Previdência e terceirização http://g1.globo.com/politica/noticia/cidades-tem-protestos-contra-reforma-da-previdencia-e-terceirizacao.ghtml
andrcolett	Transposição do rio São Francisco, reforma do ensino médio/ da previdência , operação carne fraca, terceirização ... Enem vai bombar esse ano
VictoorAugustoo	Via @estadao : Manifestantes protestam em capitais do País contra reforma da Previdência e terceirização - http:/ln.is/estadao.com.brb29w1
EdmilsonPequeno	O meu deputado @luizcoutopt votou contra a terceirização e é contra a reforma da previdência

Já para os tuítes negativos, nota-se a presença de palavrões e xingamentos junto de algumas *hashtags* de cunho negativo como por exemplo #ForaTemer, além disso é comum a presença de tuítes onde as pessoas são convocadas para manifestações. Outros termos comuns são escravidão e retrocesso, referindo-se diretamente às reformas da previdência e terceirização.

usuário	tuíte
MgracaGalvao	Dória é o maior Fake de todos os tempos. Se f... Palhaço
adamastaquio	Acorda POVO trabalhador. Previdencia + Terceirização + Reforma da CLT é P*** NO RABO DO POVO! #ReformaTrabalhistaNÃO
TheMairaBastos	Reforma da previdência , desmonte da CLT, terceirização ,congelamento de gastos com a saúde e educação #ForaTemerLadrao #TemerGolpista
neyeverest	O PT esteve no poder por 14 anos o bandido do Lula e a burra da Dilma e o país está um caos pela instituição a roubalheira deles também !!
carinasotero	GREVE GERAL DIA 28/04 CONTRA RETROCESSOS DE TEMER • Reforma da Previdência • Reforma Trabalhista • Terceirização irrestrita

Assim como para os tuítes classificados como negativo, os tuítes positivos também eram caracterizados pela presença de *hashtags*. Sejam elas declarando apoio a um candidato (como #Aecio2018 ou #Lula2018) ou simplesmente manifestando alguma opinião positiva.

usuário	tuíte
Verinha_Lu	É Aécio pelo Brasil !!! #AécioPresidenteDoBrasil2018 #EstamosComAécio #DeusÉmaior e #VitóriaVem #FÉ #SouAécio
Haddad_Femando	O salário mínimo aumentou 71% durante os governos Lula e Dilma #BrasilQueOPovoQuer
rovisacro	A favor da terceirização e tmb da reforma da previdência ! Mas óbvio, de forma justa e igualitária para todos, principalmente na previdência

4.2 Descrição dos experimentos

Realizou-se duas etapas de experimentos, uma primeira etapa utilizando todas as classes onde se observou que a baixa quantidade de elementos da classe positiva acabava por atrapalhar o desempenho geral dos algoritmos, em especial os que utilizavam a abordagem OVA para o caso de multiclasses. Com isso, decidiu-se descartar esses elementos e rodar uma nova sequência de experimentos considerando apenas as classes negativa e neutra (que para efeito dos algoritmos será a classe positiva) e, com isso, verificar a melhoria dos algoritmos nas métricas.

Para ambas as etapas, procurou-se não só medir o desempenho dos algoritmos, mas também encontrar uma escolha certa de parâmetros para os mesmos que obtivesse a melhor performance (como a escolha do Kernel no caso do SVM ou a constante de regularização usada tanto no SVM quanto na regressão logística).

Tais testes de escolha de parâmetro, caso feitos usando apenas a divisão do conjunto de dados entre conjunto de treino e conjunto de testes acabaria por dar uma escolha enviesada de parâmetros, uma vez que uma escolha de parâmetros que tenha bons números em determinado conjunto de testes não indica que ele possui uma boa generalização, isto é, apresentará uma boa precisão para novos dados.

A fim de garantir maior generalização, utiliza-se um método de validação chamado de validação cruzada, que se consiste de um modo de dividir o conjunto de dados e testá-lo com um conjunto de validação. Uma primeira abordagem da validação cruzada seria dividir nosso conjunto em três: um de treino, um de validação e outro de teste, assim testaríamos as escolhas de parâmetros no conjunto de treino e, por fim, mediríamos a performance do melhor no conjunto de teste.

Outra abordagem seria ainda manter a divisão do conjunto de dados em

conjunto de treino e teste porém realizar esse procedimento diversas vezes alternando as porções que irão corresponder a cada conjunto. Tal método é chamado de k-fold. No k-fold divide-se o conjunto total em k partições e, a cada iteração, é construído um conjunto de teste usando uma das partições enquanto as k-1 serão o de treino. O procedimento de treinamento é realizado k vezes e, a escolha de parâmetros com melhor desempenho médio será a escolhida. A figura 4.2 mostra o funcionamento do método.

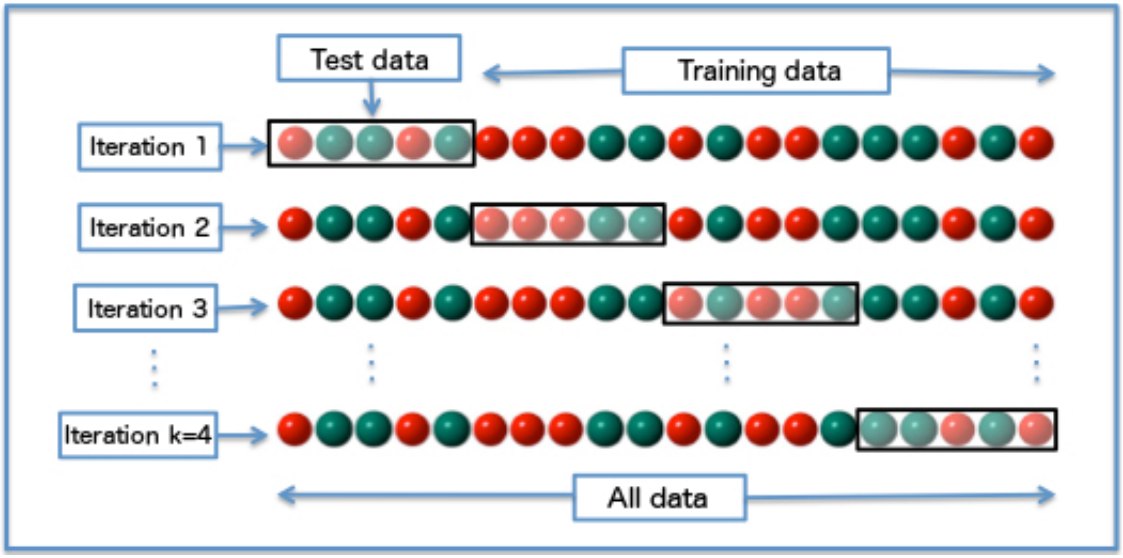


Figura 4.2: Funcionamento do método k-fold da validação cruzada
fonte: https://en.wikipedia.org/wiki/Cross-validation_(statistics)#/media/File:K-fold_cross_validation_EN.jpg

4.2.1 Parâmetros avaliados

SVM

Para o SVM, textou-se o desempenho do algoritmo testando diferentes kernels e com diferentes parâmetros para cada um. Os seguintes kernels foram utilizados:

- Kernel Linear: $K(x, y) = x^T y$
- Kernel RBF: $K(x, y) = exp(-\gamma||x - y||^2)$
- Kernel Polinomial: $K(x, y) = (x^T y + c)^d$

Em comum a todos os kernels, variou-se o parâmetro C do SVM utilizado para penalizar os valores mal classificados. Já para o RBF, varia-se o parâmetro γ. Por último para o polinomial, varia-se tanto o parâmetro d que determina o grau do polinômio (aqui variamos entre 3, 4 e 5) e o coeficiente.

Regressão Logística

Para a regressão logística, adicionamos um parâmetro extra λ que será usado para penalizar nosso vetor de pesos e assim, evitar *overfitting*, que ocorre quando um modelo apresenta baixo erro no conjunto de treino, porém baixa generalização. Importante ressaltar que λ só é aplicado nos índices de 1 a $m+1$ do vetor de pesos, com o valor do termo w_0 não sendo penalizado.

4.3 Métricas avaliadas

Para a realização dos experimentos, levou-se em consideração quatro métricas para avaliar a eficiência dos classificadores.

- Acurácia: taxa calculada pela quantidade de dados corretamente classificados sobre o total de dados. Quanto mais próximo de 1, melhor a precisão.
- Precisão: taxa calculada pela expressão $\frac{tp}{tp+fp}$ onde tp corresponde aos verdadeiro positivos, isto é, elementos corretamente classificados como da classe C e fp são elementos erroneamente classificados na classe C. Quanto mais próximo de 1, melhor.
- Revocação: taxa calculada pela expressão $\frac{tp}{tp+fn}$ onde tp corresponde aos verdadeiro positivos, como na precisão e fn corresponde aos falso negativos, isto é, os elementos incorretamente classificados como não pertencentes a uma classe C quando na verdade pertencem. Assim como a precisão, quanto mais próximo de 1, melhor.
- Medida F: pode ser interpretado como uma média harmônica entre precisão e revocação é dado pela expressão: $2 * \frac{precis*revocao}{precis+revocao}$. Assim como as demais medidas, quanto mais próximo de 1, melhor o valor.

4.4 Primeiro experimento

Nesse primeiro experimento, avaliou-se o desempenho dos classificadores desenvolvidos quanto dos já disponíveis pela biblioteca `scikit` a fim de comparar o desempenho das implementações. Além disso, testou-se diferentes escolhas de parâmetro para cada abordagem de vetorizar o corpus que no

caso foram vetor binário, vetor de frequência e vetor com os valores tf-idf conforme descrito em 3.5.2. Note que neste primeiro experimento, todos os dados foram tratados *in-natura*, sem nenhum método de seleção de características ou qualquer abordagem semelhante a fim de melhorar o desempenho.

4.4.1 Algoritmos próprios

Dos algoritmos desenvolvidos para este trabalho, por questão do tempo levado a cada rodada de testes acabou dificultando a execução de múltiplas interações do algoritmo, porém testou-se o SVM utilizando um valor para γ que apresentou bom desempenho na implementação do `scikit`. E a partir dele, mediu-se acurácia, precisão, revocação e f1-score para cada classe.

Testou-se o kernel RBF com $\gamma = 10^{-4}$ e $C = 10$ utilizando uma vetorização baseada na frequência dos termos, foi obtida uma acurácia média de 46.8%.

classe	precision	recall	f1score	support
-1	0.47	1.00	0.64	306
0	0.00	0.00	0.00	312
1	0.00	0.00	0.00	29
média / total	0.22	0.47	0.30	647

Pelos resultados do experimento, percebe-se que a implementação do SVM multiclasses usando a abordagem OVA priorizou a classe negativa dando maiores valores para ela constatado pelo valor do revocação de -1, uma vez que quanto maior o valor maior as chances de recuperarmos um elemento de determinada classe. Interpretamos também, por outro lado, que o classificador desconsiderou as demais classes, visto pelos valores das mesmas.

Cre-se que é possível implementar melhorias no método OVA para que esse desbalanceio não seja tão drástico, ou até mesmo normalizar os valores de y_k para cada classe k de modo que seja possível garantir que estão em faixas de valores próximos, pois um dos perigos dessa abordagem é que cada classificador gere valores de magnitudes diferentes uns dos outros e assim enviesando a comparação entre si.

Além disso, também existe mais espaço para melhorias no resultado utilizando outras escolhas de Kernel ou até mesmo escolha de outros valores para γ usando o Kernel RBF.

4.4.2 Implementações do `scikit`

Abaixo, separaremos os testes baseados em como a obtenção do vetor de *features* foi feita ao invés dos algoritmos, uma vez que para todos esses realizou-se os mesmos testes. Importante ressaltar que só serão colocados aqui os melhores resultados de cada um, uma vez que existe uma grande quantidade de resultados.

Os resultados dos experimentos serão comentados apenas ao final desta subseção, após mostrar os valores encontrados para todas as formas de vetorização.

Frequência

Tabela 4.1: Resultados para o SVM

Kernel	C	parâmetros	Acurácia média
Linear	0,1		0,666
Linear	1		0,637
RBF	100	$\gamma = 10^{-3}$	0,663
RBF	100	$\gamma = \frac{1}{\#amostras}$	0,653
Polinomial	0,01	$c = 10$, $d = 5$	0,666
Polinomial	100	$c = 1$, $d = 4$	0,663

Importante ressaltar que em casos de empate, o primeiro melhor será selecionado, no caso o melhor foi o kernel linear com $C = 0.1$, utilizando esses valores no conjunto de testes obteve-se os seguintes resultados:

classe	precisão	recall	f1score	support
-1	0.70	0.73	0.71	324
0	0.67	0.69	0.68	298
1	1.00	0.04	0.08	25
média / total	0.70	0.68	0.67	647

Para a regressão logística, variou-se o valor de λ entre 10^{-4} a 1 e, além disso variou-se a abordagem para o caso multiclasses, uma vez que o `scikit` disponibiliza uma implementação usando o método OVA e outro usando o método multinomial (assim como implementou-se).

Tabela 4.2: Resultados da regressão logística

Método	λ	Acurácia média
OVA	0.1	0.654
OVA	1	0.651
Multinomial	0.1	0.654
Multinomial	1	0.649

Assim como para o SVM, escolheu-se o primeiro melhor que no caso é a abordagem OVA com $\lambda = 0.1$. Deu os seguintes resultados no conjunto de testes:

classe	precisão	recall	f1score	support
-1	0.70	0.68	0.69	320
0	0.68	0.73	0.70	307
1	0.50	0.15	0.23	20
média / total	0.68	0.69	0.68	647

Vetor binário

Tabela 4.3: Resultados para o SVM

Kernel	C	parâmetros	Acurácia média
Linear	0,1		0,658
Linear	1		0,643
RBF	100	$\gamma = 10^{-3}$	0,649
RBF	100	$\gamma = \frac{1}{\#amostras}$	0,646
Polinomial	100	$c = 1$, $d = 4$	0,656
Polinomial	1	$c = 5$, $d = 4$	0,658

Assim como para o vetor de frequências, a melhor escolha de parâmetros encontrada na validação cruzada foi C = 0,1 e kernel linear. Executando no conjunto de testes obteve-se os seguintes resultados:

classe	precisão	recall	f1score	support
-1	0.66	0.76	0.71	310
0	0.69	0.66	0.67	302
1	1.00	0.06	0.11	35
média / total	0.69	0.68	0.66	647

Tabela 4.4: Resultados da regressão logística

Método	λ	Acurácia média
OVA	0.1	0.631
OVA	1	0.634
Multinomial	0.1	0.631
Multinomial	1	0.630

Tal qual com o vetor de frequências, escolheu-se a abordagem OVA e $\lambda = 1$ obtendo os seguintes resultados no conjunto de testes:

classe	precisão	recall	f1score	support
-1	0.68	0.73	0.71	305
0	0.70	0.69	0.69	317
1	0.33	0.04	0.07	25
média / total	0.67	0.69	0.68	647

Tf-idf

Tabela 4.5: Resultados para o SVM

Kernel	C	parâmetros	Acurácia média
Linear	1		0,658
Linear	10		0,627
RBF	100	$\gamma = 10^{-3}$	0,651
RBF	1	$\gamma = 10^{-4}$	0,483
Polinomial	1	$c = 5$, $d = 5$	0,678
Polinomial	10	$c = 10$, $d = 3$	0,677

Com essa vetorização, diferente das demais, os melhores parâmetros escolhidos foram kernel polinomial com coeficiente e grau 5. Utilizando esses valores, obteve-se o seguinte resultado no conjunto de testes:

classe	precisão	recall	f1score	support
-1	0.64	0.79	0.71	311
0	0.69	0.60	0.64	301
1	0.00	0.00	0.00	35
média / total	0.63	0.66	0.64	647

Tabela 4.6: Resultados da regressão logística

Método	λ	Acurácia média
OVA	0.1	0.659
OVA	1	0.665
Multinomial	0.1	0.660
Multinomial	1	0.662

Neste teste, assim como os demais usando a regressão logística, escolheu-se a abordagem OVA com $\lambda = 1$ obtendo seguintes resultados no conjunto de testes:

classe	precisão	recall	f1score	support
-1	0.66	0.75	0.70	312
0	0.69	0.66	0.68	310
1	0.00	0.00	0.00	25
média / total	0.65	0.68	0.66	647

Observa-se que, para ambos os algoritmos tem-se que nenhum valor foi classificado como positivo quando usou-se a vetorização através do valor tf-idf de cada termo, apesar de ainda apresentar desempenho médio aceitável.

Isso pode ser explicado de a vetorização usando tf-idf leva-se em consideração a relevância das palavras na sentença como um todo e, possivelmente algumas palavras associadas a documentos positivos estão também associadas a documentos neutros portanto não possuem um valor tf-idf alto, aliado ao fato de o vocabulário ser montado levando em conta apenas as 5000 palavras com melhores pontuações ou presença, portanto algumas palavras explicitamente associadas a documentos positivos não compuseram o vetor de *features*.

Por outro lado se olharmos os desempenhos utilizando frequências simples ou um vetor binário obtemos resultados semelhantes para ambos os algoritmos, todavia ainda existe um melhor desempenho com a vetorização por frequências tal como o SVM apresenta melhores resultados em relação à regressão logística, mesmo que com pouca diferença.

Comum a todos os experimentos foram resultados ruins associados à classe positiva. No caso do SVM obteve-se alta precisão em alguns casos, porém as taxas de revocação e f1-score foram baixas. Valores baixos para revocação e f1-score podem ser interpretados como um sinal de que os classificadores são ruins para detectar opiniões positivas. A exemplo do melhor classificador, tem-se uma taxa de revocação de 0,08 que indica que 92% das amostras positivas não são detectadas pelo algoritmo.

Motivado pelo baixo desempenho visto na classe positiva em todos os experimentos aliado à baixa quantidade de tuítes nessa classe, rodou-se mais experimentos, porém dessa vez usando apenas as outras classes.

Referências Bibliográficas

[1] Hoda Korashy Walaa Medhat, Ahmed Hassan. Sentiment analysis algorithms and applications: A survey. *Ain Shams Engineering Journal*, pages 1093–1113, 2014.

[2] Patrick Paroubek Alexander Pak. Twitter as a corpus for sentiment analysis and opinion mining. *Proceedings of the International Conference on Language Resources and Evaluation*, pages 1320–1326, 2010.

[3] Martin Ringsquandl and Dušan Petković. Analyzing political sentiment on twitter. *AAAI Spring Symposium*, 2013.

[4] Jennifer van der Puil et. al Akshat Bakliwal, Jennifer Foster. Sentiment analysis of political tweets: Towards an accurate classifier. *Proceedings of the Workshop on Language in Social Media*, pages 49–58, 2013.

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 1 edition, 2006.

[6] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, pages 273–297, 1995.

[7] Bing Liu. *Sentiment Analysis and Opinion Mining*. Morgan & Claypool Publishers, 1 edition, 2012.