Análise de Sentimentos em comentários no Twitter

Danilo Silva Bentes
Gabriel Augusto De Vito D'Abmauia Guimarães

2018, v-1.3

1 Introdução

A análise de sentimentos, também conhecida como mineração de opiniões, refere-se ao uso de *Processamento de Linguagem Natural* para identificar, extrair, quantificar e estudar o estado afetivo e subjetivo da informação.

Este é um recurso muito importante, principalmente para empresas que desejam saber qual é a opinião que as pessoas estão tendo sobre seus produtos lançados no mercado.

A Web é um excelente meio de se extrair dados e de se retirar informações importantes, tanto por meio das redes sociais quanto por sites de notícias, blogs ou outras plataformas. É incontestável que a Web é palco de opiniões de usuários sobre produtos, críticas, elogios, comentários, etc.

2 Objetivo

Nosso objetivo com este artigo é demonstrar como o método probabilístico *Naive Bayes* (explicado na seção 3.2) pode ser útil para determinar a polaridade de uma frase ou comentário, tendo como aprendizado uma base de dados previamente classificada. Ou seja, determinar qual é a eficácia deste método em dizer se uma determinada frase tem cunho **positivo**, **negativo** ou **neutro** dada uma tabela de dados previamente classificada.

3 Desenvolvimento

3.1 Base de dados

Necessitamos de uma base de dados classificada a fim de "treinarmos"o nosso modelo. O objetivo é que o algoritmo que desenvolvermos possa ler esta base e "aprender"qual palavra é 'positiva', 'negativa', ou 'neutra'.

Utilizamos uma base de dados composta por 8199 linhas contendo comentários diversos retirados de tweets do Twitter. Esta base de dados foi retirada do sítio (MINERANDO...,), e possui duas colunas principais: *A frase propriamente dita* e a sua *classificação*.

3.2 Pré-processamento

O pré-processamento inicia-se quando os dados são coletados e organizados na forma de um conjunto de dados. Existem diversos objetivos nesta fase e um deles é solucionar os possíveis problemas com a base com que estamos trabalhando, tais como identificar atributos irrelevantes, valores desconhecidos, possíveis caracteres em lugares errados ou que irão atrapalhar a análise do algoritmo.

Nós utilizamos os seguintes recursos de pré-processamento:

1. Tokenização: É utilizada em decompor cada o documento em cada termo que o compõe. Utilizamos espaços em branco para separar palavra por palavra de cada frase existente no documento.

2. Limpeza: Foram removidas as vírgulas, aspas, dois pontos, arrobas e outros símbolos que estavam sujando a análise. Portanto, deixamos, para cada token, apenas palavras que contenham letras de A-Z, a-z, acentuações básicas e números. Devemos lembrar que a nossa base de dados é composta por tweets e a tokenização separando as palavras por espaços em branco podem causar problemas. Por exemplo: a palavra 'amor' é diferente de 'amor:' ou '@amor'. Portanto, a limpeza é fundamental para o bom funcionamento do método.

Observação: A *remoção de stopwords* é um recurso muito utilizado no pré-processamento de dados. Porém, não o utilizamos, pois afetaria de forma negativa os nossos resultados. Há frases em que remover as stopwords faria com que a sua polaridade mudasse completamente. Um exemplo seria: "Eu não gostei deste governante, não votaria nele novamente"(negativa). Após removermos a stopword "não", a frase tornaria-se positiva.

3.3 Naive Bayes

Utilizaremos o algoritmo *Naive Bayes*, que supõe que exista uma independência entre as palavras (features) do modelo.

Trata-se de um modelo puramente probabilístico, e não se baseia de redes neurais artificiais para fazer predições.

O algoritmo recebe o nome de ingênuo (naive), pois para se classificar uma sentença como positiva, negativa ou neutra, ele faz a classificação palavra por palavra.

Caso o número de palavras classificadas como positiva dentro de uma sentença seja maior do que o número de palavras classificadas como negativa, então essa sentença é considerada positiva. Do contrário, se a frase possui mais palavras negativas do que positivas, ela é considerada uma frase negativa. Caso o número de palavras

positivas seja igual ao número de palavras negativas, então essa frase é taxada como neutra.

Um exemplo mais concreto de como o algoritmo funciona

Utilizando um exemplo simples, podemos mostrar, passo-a-passo, como funciona o algoritmo *Naive Bayes*:

A partir de uma base de dados, como a que temos no nosso projeto, suponha que exista um trecho desta base com as seguintes classificações:

Tabela 1 – Exemplo fictício de uma base de dados com a classificação de cada palavra

Palavra	**Classe**
cachorro	positivo
amor	negativo
mau	negativo
amor	positivo
amor	positivo
cachorro	positivo
casa	positivo
gato	positivo
gato	negativo
gato	negativo
amor	negativo
casa	positivo
amor	positivo

Agora que temos uma base de dados classificada, o próximo passo é criarmos uma tabela de frequência, mostrando, para cada palavra, quantas vezes ela apareceu como positiva ou negativa nessa base de dados.

Tabela 2 – Tabela de frequência

Palavra	**Positivo**	**Negativo**
cachorro	2	0
amor	3	2
mau	0	1
gato	1	3
casa	2	0
Total	8	6

Tendo em mãos a tabela de frequência, podemos criar uma tabela de probabilidades para, então, podermos calcular qual é a probabilidade de uma determinada palavra ser *positiva* ou *negativa* no nosso 'treino'.

O algoritmo de Naive Bayes consegue calcular a probabilidade de cada palavra ser positiva ou negativa seguindo o seguinte exemplo:

Tabela 3 – Tabela de Probabilidades

Palavra	**Positivo**	**Negativo**	**Frequência**
cachorro	2	0	$\frac{2}{14} = 0.142$
amor	3	2	$\frac{5}{14} = 0.357$
mau	0	1	$\frac{1}{14} = 0.071$
gato	1	3	$\frac{4}{14} = 0.285$
casa	2	0	$\frac{2}{14} = 0.142$
Total	8	6	
	$\frac{8}{14} = 0.571$	$\frac{6}{14} = 0.428$	

$$P(positivo|'amor') = \frac{P('amor'|positivo) * P(positivo)}{P('amor')} \tag{1}$$

$$P(negativo|'amor') = \frac{P('amor'|negativo) * P(negativo)}{P('amor')} \tag{2}$$

Realizando os cálculos necessários para computar a probabilidade da palavra 'amor' ser positiva e negativa, temos:

$$P('amor') = \frac{5}{14} = 0.357 \tag{3}$$

$$P('amor'|positivo) = \frac{3}{8} = 0.375 \tag{4}$$

$$P('amor'|negativo) = \frac{2}{6} = 0.33 \tag{5}$$

$$P(positivo) = \frac{8}{14} = 0.571 \tag{6}$$

$$P(negativo) = \frac{6}{14} = 0.428 \tag{7}$$

$$P(positivo|'amor') = \frac{0.375 * 0.571}{0.357} = \mathbf{0.599} \tag{8}$$

$$P(negativo|'amor') = \frac{0.33 * 0.428}{0.357} = \mathbf{0.395} \tag{9}$$

Facilmente percebemos que a probabilidade da palavra 'amor' ser positiva é maior do que a probabilidade dela ser negativa.

Portanto, sempre que a palavra 'amor' aparecer em uma frase de teste, devemos contabilizá-la como positiva dentro desta frase.

Fazendo esta classificação com todas as palavras deste exemplo, temos uma outra tabela, denominada *tabela de classificação*, contendo a palavra e a classificação (label) que o algoritmo determinou para cada palavra.

3.4 Teste do algoritmo

Tendo em mãos a tabela 4, podemos começar um exemplo de teste. O teste consiste em frases contendo palavras conhecidas na classificação ou não. Percorremos essas

Tabela 4 – Tabela de Classificação

Palavra	Classificação
cachorro	Positivo
amor	Positivo
mau	Negativo
casa	Positivo
gato	Negativo

frases e, para cada frase, contabilizamos quantas palavras desta frase são positivas, negativas e neutras.

Caso o número de palavras positivas seja superior ao número de palavras negativas e neutras, então essa frase é classificada como positiva. A mesma regra é utilizada para frases que possuem maior número de palavras negativas ou neutras.

Caso a frase contenha uma ou mais palavras *desconhecidas*, elas serão descartadas na contagem. Este é um problema do algoritmo e está citado na seção 5 (melhorias a serem feitas), pois se uma frase contém 80% das palavras desconhecidas, não é justo classificá-la baseada em 20% das palavras que a compõe.

Caso nenhuma palavra da frase possua classificação, então essa frase receberá a classificação de maior frequência no treino.

4 O algoritmo desenvolvido

4.1 Treino

Com posse de nossa base de dados citada em 3.1, desenvolvemos um algoritmo em Python, que percorre as 7000 primeiras frases desta base e, para cada frase, monta a tabela de frequência igual à tabela 2, a armazena na memória e calcula a probabilidade de cada palavra ser positiva, negativa ou neutra, baseando-se na equação 1, tendo como resultado uma tabela semelhante à tabela 4, porém com a classificação de cada palavra da base.

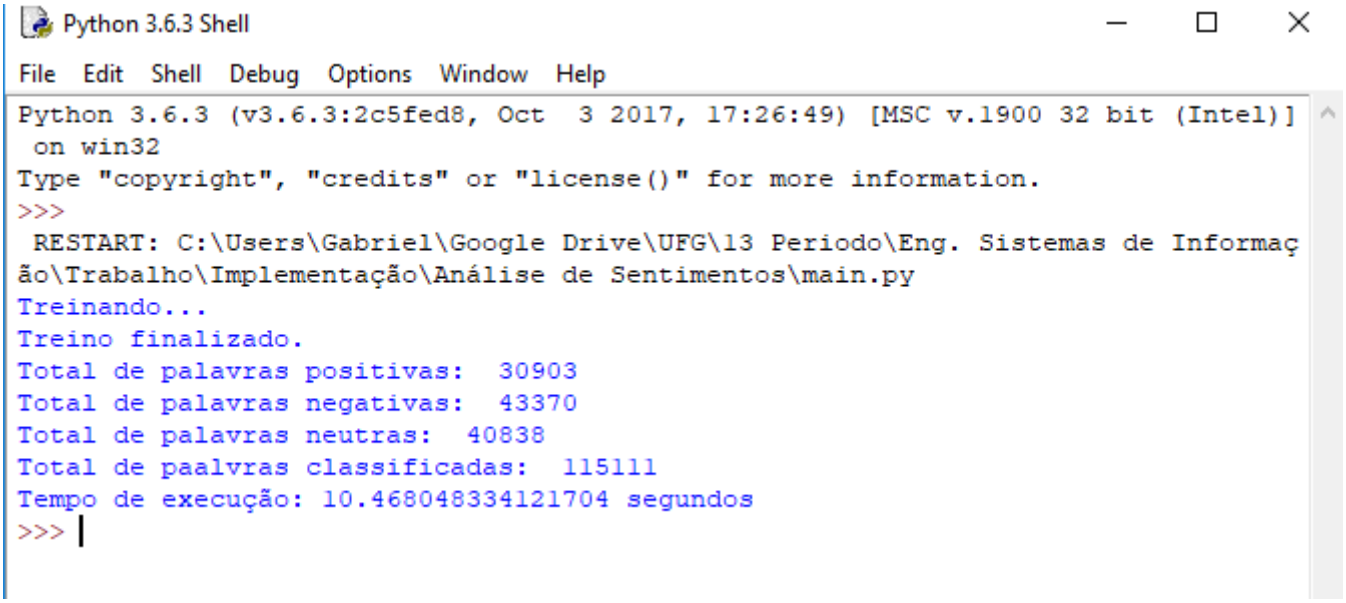
Utilizamos apenas 7000 frases, e não todas, para forçarmos que nosso teste tenha algumas palavras desconhecidas, a fim de fazermos com que nosso modelo se aproxime mais de um modelo real.

Ao executarmos o nosso algoritmo, vimos que a nossa tabela 4 tem

Desta maneira, o algoritmo passa a ter posse de 115111 palavras classificadas, das quais 26.84% foram classificadas como positivas, 37.67% como negativas e 35,49% neutras. O tempo de execução foi 10.46 segundos. A imagem 1 mostra a etapa de treino sendo executada em um processador core i5.

Não utilizamos toda a base de dados para treinar a classificação pois desejamos que algumas palavras no teste sejam desconhecidas, para dar um comportamento mais realista para o nosso modelo.

Figura 1 – Execução da parte de treino do algoritmo



Fonte – Próprio autor

4.2 Teste

O teste é a parte mais importante do algoritmo. Buscamos, dentro desta mesma base de dados, 1700 frases **aleatórias** para testarmos. Devido ao caráter aleatório das frases, a cada instância de execução do teste teremos resultados distintos. Portanto, faremos uma média de execução.

O teste consiste em:

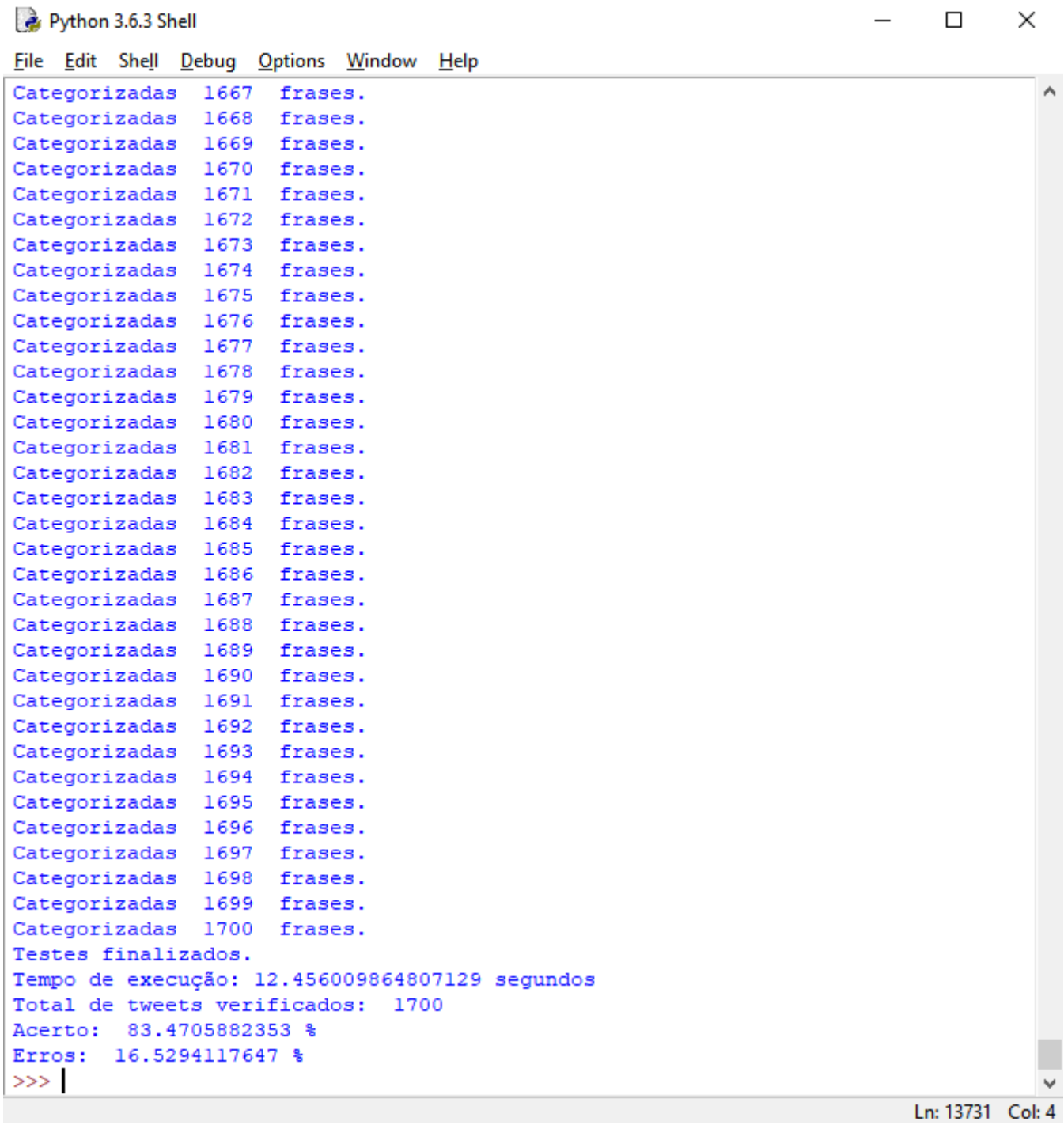
1. Carregar as frases de teste na memória.
2. Para cada palavra de cada frase do teste, verificar a classificação desta palavra na instância de treino (na tabela 4) e somar as contribuições positivas, negativas e neutras de cada palavra dentro desta frase.
3. Caso o número de palavras positivas em uma frase seja maior do que o número de palavras negativas e neutras, então esta frase deve ser classificada como positiva. Repetir a mesma lógica para palavras negativas e neutras.
4. Comparar a classificação do teste com a classificação da base de dados. Caso sejam iguais, então significa que o algoritmo *acertou* o resultado. Caso sejam diferentes, classificar este teste como *erro*.

Uma consideração importante sobre o teste: caso uma palavra no teste não seja reconhecida, ou seja, ainda não tenha sido lida na fase de treino ou não está classificada, **não é atribuída a ela nenhuma classificação**. A classificação desta palavra é **descartada**.

Como citado acima, o caráter aleatório do nosso algoritmo nos força a testar mais de uma vez para termos uma média de erros e acertos. Realizamos, portanto,

dez testes como o ilustrado na figura 2. A tabela 5 mostra os resultados desses testes, bem como a **média de acertos** e **média de erros**.

Figura 2 – Execução de um dos testes



Fonte – Próprio autor

Notamos que o algoritmo se mostra bastante eficaz quando separadas 1700 frases aleatórias para teste, retiradas do mesmo conjunto de dados. Dessas 1700

Tabela 5 – Resultados dos testes realizados

Teste	Acertos %	Erros %	Tempo(s)%
1	85.00	15.0	15.82
2	84.05	15.94	9.24
3	80.82	19.17	12.89
4	83.41	16.58	14.24
5	82.67	17.23	13.09
6	83.41	16.58	12.52
7	81.76	18.23	12.49
8	83.76	16.23	12.48
9	82.23	17.76	12.35
10	83.47	16.53	12.45
Média	83.06	16.92	12.45

frases, houve uma média de acerto de 83.06%, o que corresponde a 1412 frases classificadas da maneira correta.

O tempo de execução dependerá das configurações da sua CPU ou GPU, mas não é um fator limitante para essa quantidade de dados, visto que a média de 12.45 segundos não é um tempo preocupante.

5 Melhorias no método

Ao longo do desenvolvimento deste algoritmo e deste artigo, percebemos que poderíamos fazer algumas modificações para melhorarmos o nosso modelo e torná-lo mais real e aplicável.

A primeira modificação foi adicionar um peso para cada palavra, ou seja, atribuir a ela o *quanto* é positiva, negativa ou neutra, levando em consideração a frequência com que esta palavra aparece em cada uma dessas classificações.

Este "peso"nada mais é do que a divisão da frequência com que esta palavra aparece como positiva, negativa ou neutra dividida pelo total de palavras positivas, negativas ou neutras, respectivamente. Este peso será contabilizado na instância de teste, de tal forma que cada palavra tenha uma contribuição numérica e a soma das contribuições positivas, negativas ou neutras de todas as palavras da frase irão determinar a sua classificação.

Para exemplificar este modelo, podemos tomar como base os dados da tabela 1 e criar uma tabela de pesos (6).

Lembrando que o cálculo do peso segue a seguinte equação:

$$Peso = \frac{P(positiva|palavra)}{P(positiva)} \tag{10}$$

O que percebemos com a tabela 6 é que a palavra "amor"é "mais positiva"do que a palavra "casa", por exemplo. Apesar de ambas serem positivas, existe um peso

Tabela 6 – Tabela de frequência com o peso de cada palavra

Palavra	Positivo	Negativo	Peso (pos)	Peso (neg)
cachorro	2	0	0.25	0
amor	3	2	0.38	0.33
mau	0	1	0	0.16
gato	1	3	0.13	0.5
casa	2	0	0.25	0
Total	8	6	-	-

maior na palavra "amor"e este peso será levado em consideração no treino.

Agora, suponhamos a seguinte sentença de teste:

"cachorro amor mau gato amor"

Gostaríamos de determinar se essa frase é positiva ou negativa.

Não faremos mais o uso da equação 1. Apenas montaremos uma tabela comparando os pesos de cada uma das palavras da sentença na instância de teste.

Tabela 7 – Tabela

Palavra	Peso (pos)	Peso (neg)
cachorro	0.25	0
amor	0.38	0.33
mau	0	0.16
gato	0.13	0.5
amor	0.38	0.33
Total	1.14	1.32

Referências

A Comparison of Event Models for Naive Bayes Text Classification. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.65.9324&rep=rep1&type=pdf>. Accessed: 2018-07-10. Nenhuma citação no texto.

MINERANDO Dados. <http://minerandodados.com.br/index.php/2017/03/15/analise-de-sentimentos-twitter-como-fazer/>. Accessed: 2018-07-10. Citado na página 2.