Análise de sentimento.

Igor Tauscher Martynetz

14 de Janeiro de 2020

Conteúdo

0.1 Resumo

Repositório foi criado para análise de sentimento utilizando o dataset público do stanford, para tal feito, foi testado em 4 modelos, modelo de Regressão Logística, SVM, Multinomial Naive Bayes e Convolution Neural Network (CNN), após testes de gridsearch, modelo de SVM foi abandonado pelo fato de alto custo computacional. Depois de determinado os melhores hiperparâmetros para Regressão Logística e Naive Bayes, foi testado os modelos, tendo adquirido uma acurácia de $\approx 90\%$ para Regressão Logística e Naive Bayes, e acurácia superior de 95% para modelo CNN.

0.2 Introdução

NLP (Natural language processing) é uma área que estuda como fazer análise de textos. Como o computador só entende binários, alguns procedimentos são feitos para se converter os textos para uma maneira que o computador consiga entender.

0.2.1 Bag of Word

Uma das maneiras mais simples a se utilizar quando o assunto é texto, é o modelo de Bag of Words, que é literalmente um saco de palavras. O modelo funciona da seguinte maneira, tem-se duas sentenças, $S1$ = "João gosta de assistir filmes.", $S2$ = "Maria gosta de assistir jogos e também gosta de filmes", que pode ser representada como:

	João	gosta	assistir	filmes	Maria	jogos	de	e	também
S1	1	1	1	1	0	0	1	0	0
S2	0	2	1	0	1	1	1	1	1

Cada sentença foi convertida em um vetor, na qual em cada coluna é representada por uma palavra do texto, e cada linha a sentença, os valores representam quantas vezes cada palavra aparece em cada sentença. Problema de se trabalhar com Bag of Words é o fato que não se consegue perceber a influência das palavras, existem palavras que se repetem em quase todas sentenças, e com isso ela acaba tendo um peso alto em relação as outras palavras, e faz com que a particularidade de cada sentença acabe se perdendo.

0.2.2 TF-IDF

TF-IDF (term frequency–inverse document frequency) funciona de maneira semelhante a bag of word, inicialmente é feita uma frequência de termos (TF) onde:

$$TF = \frac{t}{n} \tag{1}$$

onde:

- t é a quantidade de vezes que a palavra aparece na sentença.
- n é total de palavras na senteça.

$$IDF = \log \frac{N}{t_n} \tag{2}$$

onde:

- log é a logaritmo base e.
- N total de sentenças.
- t_n quantidade de sentenças que a palavra t aparece.

Com isso TF-IDF é calculado como

$$TFIDF = TF * IDF \tag{3}$$

0.2.3 Word Embedding

Problema de se trabalhar com Bag of Words e TF-IDF é o fato que primeiro, a matriz resultante é uma matriz esparsa (contém muitos zeros), e segundo que se perde a o significado semântico da palavra. Para isso foi criado os Word Embedding, na qual os mais conhecidos são o Word2Vec e o GloVe.

Para criação do Word2Vec ou GloVe é utilizado uma rede neural porém sem nenhuma hidden layer, suponha a seguinte sentença "O gato pulou sobre a Raposa", no caso do Word2Vec, a rede neural tenta "descobrir"a palavra baseado nas palavras ao redor (Ex. tenta "descobrir"a palavra pulou, baseado em gato e sobre), e o GloVe tenta "descobrir"as palavras ao redor (Ex. tenta descobrir gato e sobre baseado em pulou). Com isso o vetor de cada palavra será representado por um vetor, onde a dimensão do vetor são os pesos gerados pela rede neural. Isso acarreta que palavras com vizinhos iguais tendem a ter pesos semelhantes, e isso faz com que operações algébricas possam ser feitas com palavras, por exemplo, dado a seguinte operação "Rei"− "homem"+"mulher"o resultado será "rainha".

0.3 Procedimento

Primeiramente foi feito um script para fazer um GridSearch junto com Cross Validation. GridSearch é uma biblioteca na qual é setado uma sequência de hiperparâmetros, e o código é executado n vezes, sendo cada vez utilizado uma combinação de hiperparâmetros diferentes. Cross Validation é usado para que o conjunto de dados seja separado em n conjuntos, o treino é feito em cada um e a acurácia é feita utilizando a média de todos os treinos. O código abaixo é mostrado como foi feito o GridSearch com o CrossValidation.

```
pipe = Pipeline([('tfidf', TfidfVectorizer()),
                 ('logreg', LogisticRegression(random_state=42,

                ])

parameter = {"tfidf__ngram_range": [(1, 1), (1, 2), (1, 3)],
             "logreg__solver": ["newton-cg", "lbfgs", "liblinear",
                                "sag", "saga"],
             "logreg__C": np.logspace(-3, 3, 7)
             }
model = GridSearchCV(pipe, param_grid=parameter, cv=5, verbose=1)
```

GridSearch foi feito para os modelos de Regressão Logística, SVM e Naive Bayes, porém o modelo de SVM demorou em torno de 29h para terminar, com isso ele foi excluído de ser utilizado, pois foi visto que o custo computacional dele foi muito alto.

A muito tempo queria utilizar Word Embedding em algum algoritmo de machine learning, problema de se utilizar ele era, como represento uma frase? Como cada palavra tem um vetor correspondente não sabia como utilizar num modelo de regressão linear ou outros mais simples, pois esses modelos aceitam tokens como input do modelo. Uma ideia que me surgiu foi, se cada palavra é um vetor, então uma sentença pode ser representado por uma matriz, e com isso posso utilizar uma CNN semelhante ao utilizado por imagens, porém é, como cada hidden layer deve ser montado pois no caso de imagens cada hidden layer é composto por uma convolution layer de dimensão NxN (geralmente 2x2 ou 3x3), e isso no caso da matriz de word embedding, não seria correto, pois a informação da matriz está restrita as linhas, e não nas colunas.

A resposta para isso veio num curso que fiz na udemy sobre NLP, na qual foi mostrado uma arquitetura que foi elaborada por Ye Zhang and Byron Wallace no artigo intitulado "A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification", ele elabora uma arquitetura contendo 3 convolution layer, uma 2xn, outra 3xn e outra 4xn, onde elas representam bigram, trigram e fourgram, e após essas convolution layer elas são concatenadas, o esquema dela pode ser visto na figura 1

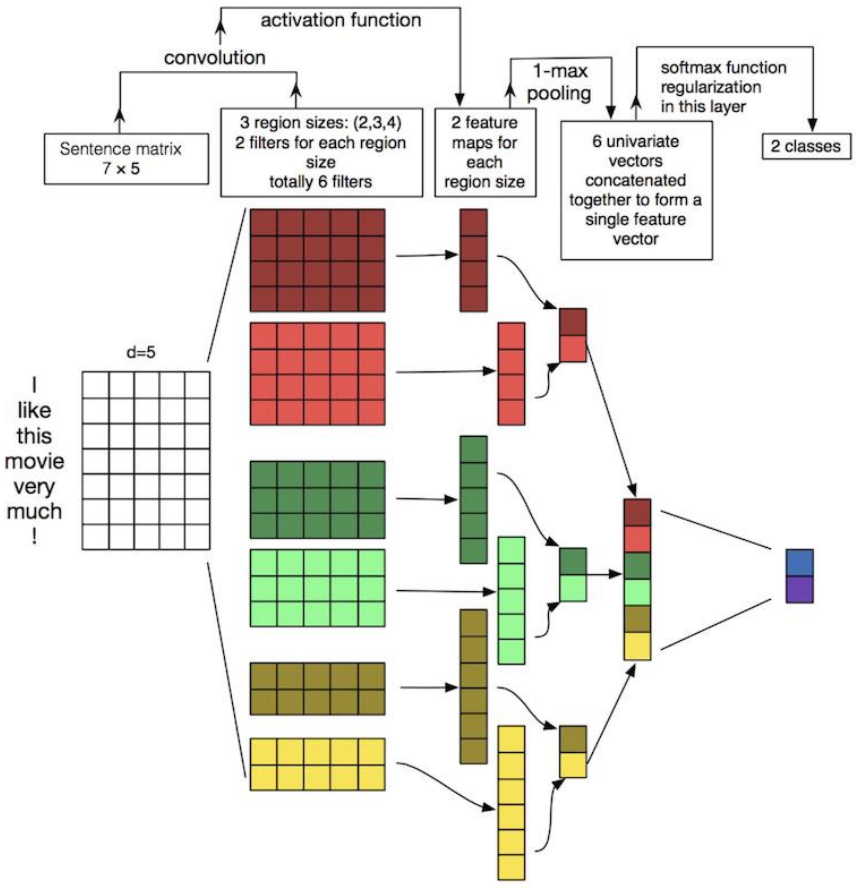

Figura 1: Zhang, Y., Wallace, B. (2015) "A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification"

Nessa arquitetura foi utilizado em todas hidden layer uma relu como função de ativação, e na última layer foi utilizado tanh, pois ela gera valores entre -1 e 1, o otimizador utilizado foi Adam, e loss function utilizado foi o Hinge, porém foi observado que se ao invés de determinar outputs entre -1 e 1 utilizasse para determinar outputs binários (0,1) e trocar a tanh da última layer por sigmoid, e loss function como binary cross entropy, obteve-se um aumento na acurácia do modelo, por esse fator foi gerado outputs binários e depois alterados para valores -1 e 1.

0.4 Resultados

Todos modelos foram rodados numa máquina cujo especificação são:

- Processador: Intel(R) Core(TM) i5-7400 CPU @ 3.00GHz
- Memoria: 8gb
- placa de video: GeForce GTX 1050

obtendo os seguintes resultados

0.4.1 Regressão Logistica

No conjunto de treino obteve-se a seguinte matriz de confusão

	Positive	Negative
Positive	12500	0
Negative	0	12500

- F1-score = 1
- accuracy = 1

E no conjunto de teste obteve-se

	Positive	Negative
Positive	11283	1217
Negative	1218	11282

- F1-score = 0.9025
- accuracy = 0.9026

0.4.2 Naive Bayes

No conjunto de treino obteve-se a seguinte matriz de confusão

	Positive	Negative
Positive	11283	8
Negative	23	12477

- F1-score = 0.9987
- accuracy = 0.9987

E no conjunto de teste obteve-se

	Positive	Negative
Positive	11403	1097
Negative	1689	10811

- F1-score = 0.8858
- accuracy = 0.8885

0.4.3 CNN

Para a CNN foi determinado somente a acurácia

- train data accuracy: 0.97548
- test data accuracy: 0.91432