**Software Development Project**

**Movie Review Sentiment Analysis**

**2019-2020**

**Swar Patel Priyank Chaudhari**

**( 17CEUOG132 ) ( 17CEUOG123 )**

**Guided By: Hariom Pandya.**

**February 15, 2020**

**Dharmsinh Desai University**

**Faculty of Technology**

**Nadiad.**

1. **Problem Statement**

People always express their views and feelings about movies in the form of writing reviews on some of the well-known websites like imdb, rottontomatoes, google.

Our task is to gather that review data, clean that data and on that cleaned data train machine learning model and then predict review is it positive or negative? Based on number of total positive and negative review give ratings to movie from total 10 stars. We

Also need to create our website to display our predicated rating and also users can write reviews so that no more we need to fetch reviews from other websites 😅 .

Our Workflow for Creating model.

Steps:

1. Scaping review from google.

* We find already created good dataset of imdb movie reviews. But we don’t want use that so…… we scraped reviews from google using NodeJS with puppeteer library ( we wasted our so much precious internet data and it take so much time to scape more than 1000 movie’s reviews. ).
* How we did scraping?

We download csv file which contains [list of Bollywood movies](https://github.com/calci/bollywood-movie-dataset/blob/master/BollywoodMovieDetail.csv).

We read csv file and then combine movie name with it’s release year

and used that for searching in google (only movie name is sometimes not working). after loading movie page we search for particular html tag which gives us list of reviews and then we store that reviews in json file. see code in this file. Hey we know that you have question about labelling review it’s in 3rd step but don’t skip step 2.

1. Cleaning Review

* It’s most important step for training your model. Why? Because raw text contains maybe spelling mistakes, punctuation marks, emojis, words like goooood. and if we don’t clean that then our model is trained on this misspelled word and emojis and punctuation marks that we don’t want.
* How we cleaned our data?

We first observed some of our scarped reviews and we found that………..

1. People uses emojis so much frequently. 6 out of 10 reviews has emojis so we can’t remove emojis from raw text because it’s important factor for identify positivity or negativity of review. So what we did? We created list of positive emojis and list of negative emojis and then replace positive emoji in our raw text with “emo\_pos” and negative emoji with “emo\_neg”.

emo\_pos - 😊 👌 😍 😇 😌 ☺ and so many more…

emo\_neg - 😡 😤 😥 😓 👎 and so many more….

We know that there are so many emojis but we repleced only few of them which we know that it is conveying meaning positivity or negativity.

1. In raw text few words are in uppercase and others are in lowercase. We have to convert all words into lower case why? Because “good” and “Good”

have same meaning for us but model treat them differently.

1. There is unnecessary space in raw text so we removed them. There is puncution marks, symbols and digits we also removed that because at the end our model is only going to work with words.
2. STOPWORDS - Stop words are generally the most common words in a language. e.g. the, is, a ,an, but, again, some, there, once, of, am, for, do, yours,…….. we have to remove them because they are not going to convey any positivity or negativity and it’s also reduce our dataset file size.
3. Stemming - stemming is the process of reducing inflected (or sometimes derived) words to their word stem.

There is Two type of popular stemming algorithm. (i) Porter (ii) Lancaster.

We used Porter stemmer which is less strict and faster.

Lemmatization - Lemmatization usually refers to the morphological analysis of words, which aims to remove inflectional(related) endings. It helps in returning the base or dictionary form of a word, which is known as the lemma.

e.g.

text:

Stemming is funnier than a bummer says the sushi loving computer scientist movie moved best beautiful moves studies study studying cries

After applying porter stemmer:

Stem is funnier than a bummer say the sushi love comput scientist movi

Move best beauty studi studi studi cri.

After applying lemmatization :

Stemming is funnier a bummer say the sushi loving computer scientist movie moved best beautiful move study study studying cry.

What is difference between lemmatization and stemming?

Somewhat similar but there is difference that a stemmer operates on single word without knowledge of context and therefore cannot discriminate between words which have different meaning while lemmatization preserve the meaning. e.g. we know that “marketing“ and “market” has different meaning but if we do stemming then “marketing” is replaced with “market”

While in lemmatization “marketing” remain same.

Why stemming and lemmatization?

It reduces the word density in the given text and helps in preparing the accurate features for training machine. Cleaner the data, the more intelligent and accurate your machine learning model, will be.

We used “text-miner” and “lemmatizer” npm package to do cleaning.

Check this website to do online [lemmatization and stemming](https://text-processing.com/demo/stem/).

1. Labelling our scraped reviews.

So we first downloaded list of [positive](https://gist.github.com/mkulakowski2/4289437) and [negative](https://gist.github.com/mkulakowski2/4289441) words. And then based on count of positive and negative words in reviews we labelled them as 1 (positive) and 0 (negative). Way we did this is wrong for some of the reviews. why….? Idiot is negative word. And we know that 3 idiots is great movie with 8+ imdb rating. But most of the reviews contains idiot word which not referencing negativity but it referring to movie name. so due to the idiot word-count half of reviews of that movie are labelled as negative. So what’s solution…? Whatever the idea we used for labelling review is not good but it happening with very few reviews and after the training model we again predicted reviews of 3 idiots most of are predicted as positive as expected. Good way to label the movie review data is based on the stars given by each user(more than 7 then positive, less than 4 then negative, for 5 6 neutral). On google every movie don’t have reviews with stars but imdb have that. So good way is to fetch reviews and stars from imdb.

What we used for finding count of positive and negative words?

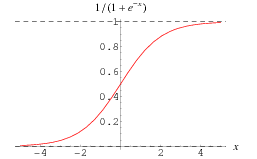
Popular string matching algorithm rabin-karp, knuth-morris-pratt takes O(n) (where n is length of text) for single word and we have list of 1000+ words of positive and 1000+ words of negative and total 25000 reviews. so we need to run kmp or rabinkarp for 2000 times for one review. Looks terrible……….

But our boat is not going to sink because there is great aho-corasick string matching algorithm which runs with time complexity of O(N + L + Z) where Z is number of pattern, N is length of text and L is total number of characters in all words(patterns). This algorithm search whole 1000 words in 1 go. This algorithm uses Trie data structure for all pattern words. For this we used aho-corasick npm package. See this [visualization](http://jovilab.sinaapp.com/visualization/algorithms/strings/aho-corasick) of this algorithm.

1. Training A Model.

What type of algorithm can be used for classification?

As we have to do classification linear regression is not going to work because it is used to determine continuous value(numeric). Unlike linear regression, the dependent variable is categorical ( in our case positive and negative ). **If we take the weighted sum of inputs as the output as we do in Linear Regression, the value can be more than 1 but we want a value between 0 and 1. That’s why Linear Regression can’t be used for classification tasks.**

****

**Sigmoid function**

**( credit:** <http://mathworld.wolfram.com/SigmoidFunction.html> )

Logistic Regression is a generalized Linear Regression in the sense that we don’t output the weighted sum of inputs directly, but we pass it through a function that can map any real value between 0 and 1. That function is called sigmoid function. check out this [online sigmoid calculator](https://keisan.casio.com/exec/system/15157249643325).

A Linear Regression model can be represented by the equation.



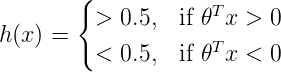
We then apply the sigmoid function to the output of the linear regression

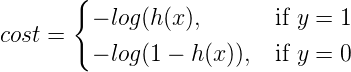


where the sigmoid function is represented by,



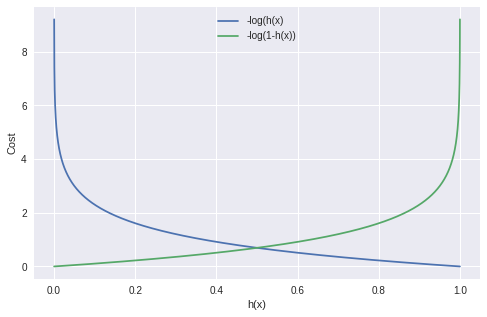




The cost function for a single training example can be given by: 

Cost function intuition

If the actual class is 1 and the model predicts 0, we should highly penalize it and vice-versa. As you can see from the below picture, for the plot -log(h(x)) as h(x) approaches 1, the cost is 0 and as h(x) nears 0, the cost is infinity(that is we penalize the model heavily). Similarly for the plot -log(1-h(x)) when the actual value is 0 and the model predicts 0, the cost is 0 and the cost becomes infinity as h(x) approaches 1.



( credit : <https://towardsdatascience.com/building-a-logistic-regression-in-python-301d27367c24>)

For minimizing cost gradient descent is used. Gradient descent is algorithm to find local minimum of function. If we take small step in opposite direction of gradient then we reach to local minimum. ( Gradient is derivation of cost function & we know that dy/dx is used for finding minima and maxima ).

Check out this [page](https://web.stanford.edu/class/archive/cs/cs109/cs109.1178/lectureHandouts/220-logistic-regression.pdf) for understanding cost function and gradient cost.

But now what to pass to this logistic regression model? model cannot directly work with text (reviews). it only works with numeric value so some how we have to convert our reviews to numeric values.

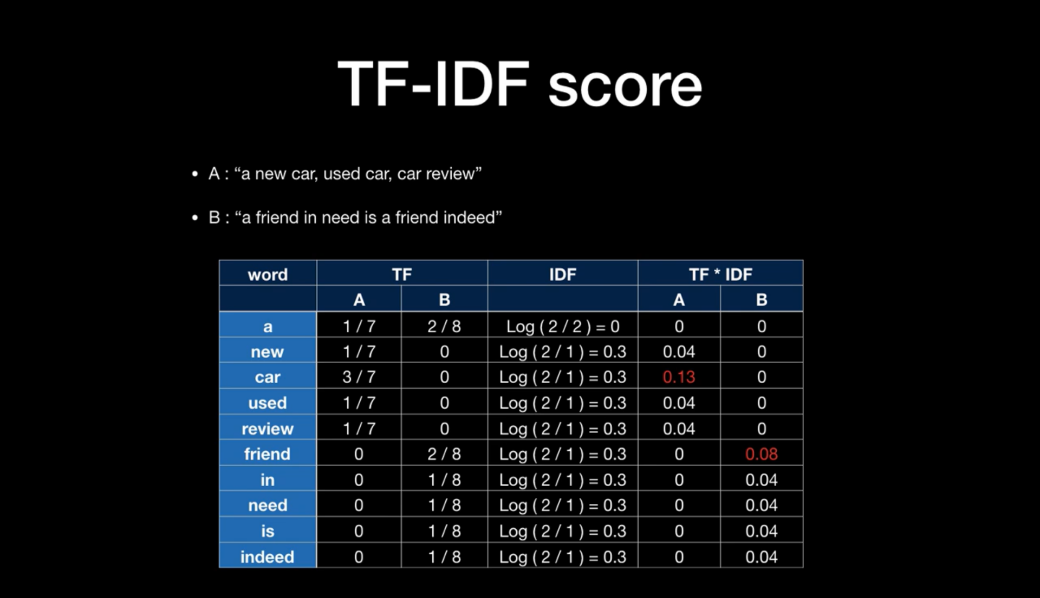
That’s where TF\_IDF vectorizer come into picture.

TF-IDF help us to find correlation between reviews.

TF-IDF (term frequency-inverse document frequency) is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents.

TF-IDF was invented for document search and information retrieval. It works by increasing proportionally to the number of times a word appears in a document, but is offset by the number of documents that contain the word. So, words that are common in every document, such as this, what, and if, rank low even though they may appear many times, since they don’t mean much to that document in particular.

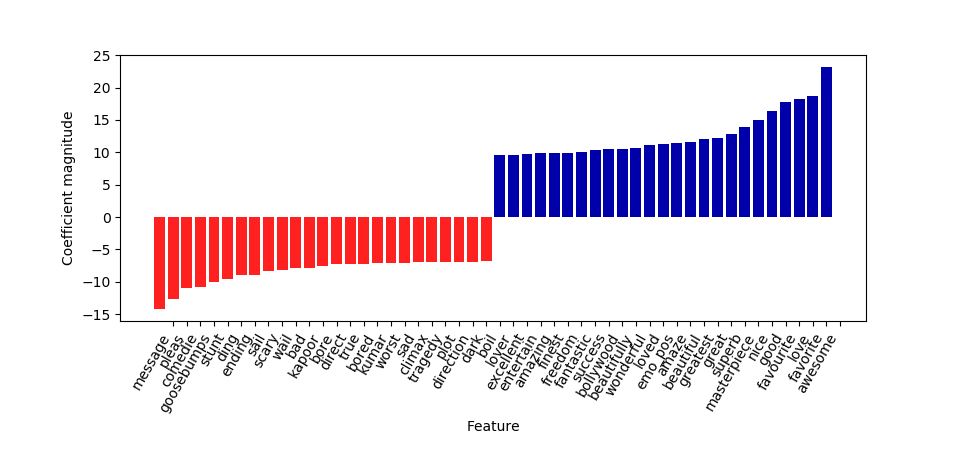
How TF-IDF is calculated?



(credit:<https://www.youtube.com/watch?v=G1bof7UL9RU&list=PLwyIYCshlcyHtMTL8dvUoeM_CF2h0X2JC&index=4>)  
The **term frequency** of a word in a document. There are several ways of calculating this frequency, with the simplest being a raw count of instances a word appears in a document. Then, there are ways to adjust the frequency, by length of a document, or by the raw frequency of the most frequent word in a document.  
  
The **inverse document frequency** of the word across a set of documents. This means, how common or rare a word is in the entire document set. The closer it is to 0, the more common a word is. This metric can be calculated by taking the total number of documents, dividing it by the number of documents that contain a word, and calculating the logarithm. So, if the word is very common and appears in many documents, this number will approach 0. Otherwise, it will approach 1.

Why TF-IDF works in machine learning?

Vectorizing a document is taking the text and creating one of these vectors, and the numbers of the vectors somehow represent the content of the text. TF-IDF enables us to gives us a way to associate each word in a document with a number that represents how relevant each word is in that document. Then, documents with similar, relevant words will have similar vectors, which is what we are looking for in a machine learning algorithm.



After Doing TF-IDF in our vocabulary which top 25 words in positive words has highest magnitude and which 25 Words in negative words has lowest highest magnitude.

Vocabulary created based on our positive reviews and negative review corpus we also considered n-gram of size 2.

Our vocabulary size has total 9462 words.

Why only 9642 words……?

We have dataset of size approx 25000 reviews which contains so many words. Vocabulary size is reduced because we(tfidfvectorizer) removing words which occurring so much frequently and also the word which occurring so much rarely.

Our focusing on medium frequency words. So Each review has vector size of 9642.

Word2vec !!!

As we found on internet Word2vec is also used for finding correlation between words. If we sum vector of words of review then we get doc2vec. Which used for finding correlation between reviews.

What happen if you combine word2vec and tfidf?

What is accuracy of your model?

What about precision and recall and f1 score?

What about learning cost and learning curve?

What is epoch? How is it helpful?

What is gridsearchcv?