

CS 777 – BIG DATA ANALYTICS

PYSPARK NLP MODELLING

(Using n-gram/tf-idf/countvectorizer)

(FALL - 2019)

OBJECTIVE:

* Implementing feature engineering using PySpark
* Realizing n-gram/tf-idf/countvectorizer models using PySpark
* These will be used in conjunction with a Logistic Regression to evaluate the effectiveness of the classifier.
* Objective is to build a model that can detect sentiment using PySPark.
* Dataset being used is "Sentiment140" which contains info about 1.6 million tweets
* More info on the dataset can be found from the link >>

<http://help.sentiment140.com/for-students/>  
The dataset can be downloaded from the below link.  
<http://cs.stanford.edu/people/alecmgo/trainingandtestdata.zip>

I/O DESCRIPTION:

By looking at the description of the dataset from the link, the information on each field can be found.

0 - polarity of the tweet (0 = negative, 2 = neutral, 4 = positive)  
1 - id of the tweet  
2 - date of the tweet (Sat May 16 23:58:44 UTC 2009)  
3 - query. If there is no query, then this value is NO\_QUERY.  
4 - UserID  
5 - text of the tweet

APPROACH:

**STAGE1: PRE-PROCESSING**

* **HTML decoding**: Usually HTML encoding will not be converted to text, and end up in text field as '&amp','&quot',etc. Decoding HTML to general text will be the first step of data preparation for which I used BeautifulSoup.
* @**mention**: The second part of the preparation is dealing with @mention.  
  Even though @mention carries a certain information (which another user that the tweet mentioned), this information doesn't add value to build sentiment analysis model.
* **URL** **links**: The other part of the cleaning is dealing with URL links, same with @mention, even though it carries some information, for sentiment analysis purpose, this can be ignored.
* **Hashtag** / **numbers**: Sometimes the text used with a hashtag can provide useful information about the tweet. It might be a bit risky to get rid of all the text together with the hashtag.  
  So, decided to leave the text intact and just remove the '#'.

**STAGE2: PYSPARK MODEL:**

* **TF - IDF + Logistic Regression:** Research on this topic indicated that TF-IDF with Logistic Regression is quite strong combination, and showed robust performance, as high as Word2Vec + Convolutional Neural Network model. So, I implementend TF-IDF + Logistic Regression model with Pyspark.
* **CountVectorizer + Logistic Regression:** CountVectorizer discards infrequent tokens.

CountVectorizer in SparkML can be used to get term frequency for IDF (Inverse Document Frequency) calculation. Apart from the reversibility of the features (vocabularies), there is an important difference in how each of them filters top features.

* **N-gram Implementation:** In Scikit-Learn, n-gram implementation is fairly easy. You can define range of n-grams when you call TfIdf Vectorizer. But with Spark, it is a bit more complicated. It does not automatically combine features from different n-grams, so I had to use VectorAssembler in the pipeline, to combine the features I get from each n-grams.

Instructions for execution:

* Download the data from the link provided in the objective section of this document
* Make sure the folder structure remains unchanged
* Inside the training and test data folder will be input .csv file
* A new intermediate .csv file will be produced by the program which acts as the final input for the models.
* The Project ZIP file already contains all the necessary .csv files
* Execution time of the code varies from a few minutes to couple of hours in local.

CONCLUSION:

* TF-IDF + Logistic Regression produced an accuracy of 86.04
* CountVectorizer + Logistic Regression produced an accuracy of 86.57
* N-Gram model produced an accuracy of 88.35

Since N-Gram model produced the best results, proceeded with N-Gram model on making the predictions on Test data. Achieved a final accuracy of **88.38%** on the testing data set.