Semi-Supervised Text Classification

Shivanshu Verma (IIT Delhi)

May 15, 2019 – July 12, 2019

**INTRODUCTION**

A big company like Schlumberger has a large number of contracts with many other firms and companies and it is nearly impossible to read each and every contract in detail. There are several clauses which company majorly looks in the contract and the aim is to detect them without reading the contracts but with the use of machine learning techniques.

This led to the project of text classification and as the number of contracts is very large, we cannot label huge amount of data therefore using semi supervised technique in which the aim is to extract some useful features from unsupervised classification of complete contracts dataset and then use these features on labeled dataset(supervised) to map these features to the defined clauses.

**APPROACH**

**DATA PREPARATION**

Convert the text files of contracts into a complete dataset with each paragraph as a logical unit and tracking each logical unit by its file name and page number.

|  |  |  |
| --- | --- | --- |
| **Document name** | **Page**  **No.** | **Paragraph text** |
| ---------- | -------- | --------------  -------------- |

Code files: **Data\_gen.py**

**Supervised\_Dataset\_gen.py**

**UNSUPERVISED TECHNIQUES**

**TOPIC MODELING**

1. LATENT DIRICHLET ALLOCATION

Code files:

* Preprocess\_Pipeline\_Training.py
* LDA\_Training.py
* LDA\_SVM.py
* LDA\_XGBoost.py

Working :

Preprocess\_Pipeline\_Training.py :

* Reads complete\_dataset.csv file
* System arguments:
  + - max\_f (int) : Maximum feature vector limit(10000 found optimal)
* Dump fitted SKlearn Pipeline (NLTK Preprocessing + Count Vectorizer) in

'preprocess\_ppl\_' + str(max\_f) + '.pickle'

LDA\_Training.py :

* Reads preprocess pipeline
* System arguments:
  + - max\_f (int) : Maximum feature vector limit(10000 found optimal)
* Dump trained lda model in 'lda\_model\_' + str(max\_f) + '\_features.pickle'

LDA\_SVM.py :

* Reads preprocess pipeline and trained model
* System arguments:
  + - max\_f (int) : Maximum feature vector limit(10000 found optimal)
    - n : Percentage of Others dataset to be taken in supervised classification
* Dump Supervised SVM model in 'Supervised\_svm' + lda\_model\_name.split('.')[0] + str(n) + '% others' + '.joblib'
* Writes confusion matrix in 'confusion\_matrix\_' + lda\_model\_name.split('.')[0] + str(n) + '% others' + '.png'
* Writes classification report in 'classification\_report\_' + lda\_model\_name.split('.')[0] + str(n) + '% others' + '.csv'

LDA\_XGBoost.py :

* Reads preprocess pipeline and trained model
* System arguments:
  + - max\_f (int) : Maximum feature vector limit(10000 found optimal)
    - n : Percentage of Others dataset to be taken in supervised classification
* Dump Supervised SVM model in 'Supervised\_xgb' + lda\_model\_name.split('.')[0] + str(n) + '% others' + '.joblib'
* Writes confusion matrix in 'confusion\_matrix\_xgb' + lda\_model\_name.split('.')[0] + str(n) + '% others' + '.png'
* Writes classification report in 'classification\_report\_xgb' + lda\_model\_name.split('.')[0] + str(n) + ' % others' + '.csv'

1. LDA MALLET

Code files:

* ldamallet\_dictionary.py
* LDA\_mallet\_training.py
* LDA\_mallet\_SVM.py

Ldamallet\_dictionary.py :

* Reads complete\_dataset.csv file
* System arguments: None
* Dump id2word corpora dictionary in 'Corpora\_Dictionary.pickle'
* Dump corpus text file in 'Corpus\_text.pickle'

LDA\_mallet\_training.py :

* Reads corpora dictionary and corpus text file.
* Two different mallet runs on local and cloud. On local, mallet-2.0.8 runs well whereas on cloud mallet2/Mallet runs well.
* Make sure to define MALLET\_HOME environment variable in the bash file on cloud and on local using control; panel environment variables.
* System arguments:
  + - num\_topics(int) : Number of topics of lda model
* Prints coherence score on running.
* Dump trained lda mallet model in 'lda\_mallet\_' + str(num\_topics) + '.pickle'

LDA\_mallet\_SVM.py :

* Reads lda mallet model
* System arguments:
  + - num\_topics(int) : Number of topics of lda model
* Dump supervised SVM model in 'Supervised\_svm\_ldamallet\_' + str(num\_topics) + 'topics.joblib'
* Dump confusion matrix in 'confusion\_matrix\_ldamallet\_' + str(num\_topics) + '\_topics.png'

1. LDA2VEC (Incomplete)

Code files:

* lda2vec\_preprocess.py
* lda2vec\_model.py
* lda2vec\_run.py

lda2vec\_preprocess.py :

* Reads complete\_dataset.csv, NLTK preprocessed text list which is in 'NLTK\_Preprocessed\_text\_list.txt' and GoogleNews-vectors-negative300.bin
* System arguments: None
* Dump vocabulary in 'vocab.pkl'
* Dump Corpus in 'corpus.pkl'
* Save numpy arrays of flatten , doc\_ids, pruned, bow, and vectors in "flattened.npy", "doc\_ids.npy", "pruned.npy", "bow.npy" and "vectors.npy" respectively.

lda2vec\_model.py :

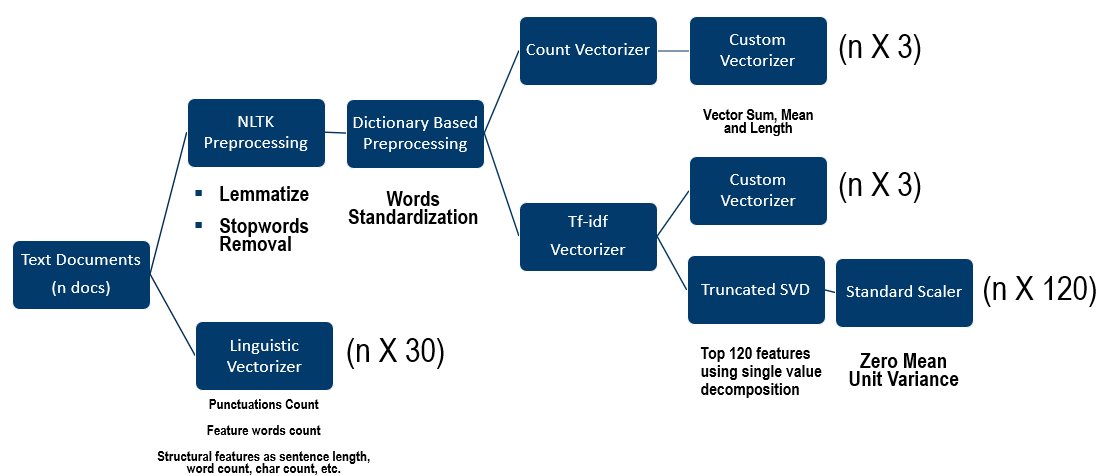
* Class of Lda2vec model

lda2vec\_run.py :

* Reads corpus, vocab, flatten , doc\_ids and vectors
* System arguments: None
* Finds a dictionary data which contains term document matrix, document topic matrix and few more things.
* Can calculate top words in each topic with print\_top\_words\_per\_topic(data).
* Data dictionary can be directly use to visualize topics using pyLDAvis.
* Could not figure out how to predict using this model.
* Also they used epochs loop for calculating weights and topics matrix, make sure to use that as I didn’t have time and just ran the code to check its working.

**CLUSTERING**

**Feature Engineering**

****All the text documents were converted into the feature vectors to be classified by clustering. Here is the detailed figure shows the vector of 156 features defining each document.

Final Matrix Shape : n X 156

1. K MEANS

Code files:

* Text\_features.py
* KMeans\_Training.py
* KMeans\_SVM.py

Text\_features.py :

* Reads **complete\_dataset.csv**
* System arguments: None
* Imports **preprocess\_text()** function from **classes.py** for feature engineering.
* Save numpy arrays of preprocessed text documents feature vectors in 'preprocess\_text.npy'

KMeans\_Training.py :

* Reads preprocess text numpty array
* System arguments:
  + - n\_clusters(int) : Number of KMeans clusters
* Dump KMeans model in 'KMeans\_Clusters\_' + str(n\_clusters) + '.pickle'

KMeans\_SVM.py :

* Reads KMeans Model
* Using model, calculate the distance of particular document feature from the means of each cluster, reciprocate it and then divide it with the sum of total reciprocals to calculate the probability of each document in each cluster and get a feature vector of size(n\_clusters)
* Use this n\_cluster size feature vector to calculate optimal SVM model using GRIDSearchCV.
* System arguments:
  + - n\_clusters(int) : Number of KMeans clusters
* Dump confusion matrix in 'confusion\_matrix\_'+ kmeans\_model\_name.split('.')[0] + '.png'
* Dump supervised model of KMeans features as input in 'Supervised\_svm\_'+ kmeans\_model\_name.split('.')[0] + '.joblib'

1. GAUSSIAN MIXTURE MODELS

Code files:

* Text\_features.py
* GMM\_Training.py
* GMM\_SVM.py

Text\_features.py :

* Reads **complete\_dataset.csv**
* System arguments: None
* Imports **preprocess\_text()** function from **classes.py** for feature engineering.
* Save numpy arrays of preprocessed text documents feature vectors in 'preprocess\_text.npy'

GMM\_Training.py :

* Reads preprocess text numpty array
* System arguments:
  + - n\_components(int) : Number of GMM clusters
* Dump GMM model in 'GMM\_Components\_'+str(n\_components) + '.pickle'

GMM\_SVM.py :

* Reads GMM Model
* Using model, finds the probability of each document vector using **predict\_proba()** function of sklearn GMM library
* System arguments:
  + - n\_components(int) : Number of GMM clusters
* Dump confusion matrix in 'confusion\_matrix\_'+ gmm\_model\_name.split('.')[0] + '.png'
* Dump supervised model of GMM features as input in 'Supervised\_svm\_'+ gmm\_model\_name.split('.')[0] + '.joblib'

1. HEIRARCHICAL CLUSTERING

Code files:

* Text\_features.py
* GMM\_Training.py
* GMM\_SVM.py

Text\_features.py :

* Reads **complete\_dataset.csv**
* System arguments: None
* Imports **preprocess\_text()** function from **classes.py** for feature engineering.
* Save numpy arrays of preprocessed text documents feature vectors in 'preprocess\_text.npy'

Hierarchical\_Training.py :

* Reads preprocess text numpty array
* System arguments:
  + - n\_clusters(int) : Number of Hierarchical Clusters
* Dump model in 'Hierarchical\_clusters'+ str(n\_clusters) + '.pickle'

Hierarchical\_SVM.py :

* Reads Model
* System arguments:
  + - n\_clusters(int) : Number of Hierarchical Clusters
* Dump confusion matrix in 'confusion\_matrix\_'+ hierarchical\_model\_name.split('.')[0] + '.png'
* Dump supervised model of Hierarchical features as input in 'Supervised\_svm\_'+ hierarchical\_model\_name.split('.')[0] + '.joblib'

**FUNCTIONS USED**

Code files:

* functions.py

Functions:

* get\_dataframe()
  + - * Input : file name with path, sheet name
      * Returns dataframe of xlsx or csv file
* preprocess\_dataframe()
  + - * Input : dataframe, similarity threshold
      * Returns dataframe which removes similar rows within that threshold
* Best\_estimator\_grid()
  + - * Input : X, Y, classifier, parameters dictionary, cross validation number(int)
      * Returns best model among the parameters given using GridSearchCV
* cm\_anaysis()
  + - * Input : true label, predicted label, labels, ymap, figure size
      * Returns colored confusion matrix with accuracy
* NLTK\_Preprocessor() {*class*}
  + - * Input : None, kwarg : stopwords list(manual), punctuation, lower(bool), strip(bool)
      * transform() : Preprocess text with lemmatization, tokenization, stopwords removal, bad character removal and returns a list of cleaned text.