INM430 Coursework Final Submission Part-2 - Analysis Files

December 11, 2018

1 INM430 Coursework Final Submission Part-2 - Analysis Files

This notebook is to support the written report pdf file. This notebook documents all the steps required for the project submission as listed in the table of contents below.

1.1 Table of Contents

1. Setup

Hide errors for presentation purposes

```
In [25]: import warnings
         warnings.filterwarnings('ignore')
```

Import dependencies

```
In [2]: from gensim.models.doc2vec import Doc2Vec, TaggedDocument
        import matplotlib.pyplot as plt
        import multiprocessing
        import nltk
        from nltk.tokenize import word_tokenize
        import numpy as np
        import pandas as pd
        from pipelinehelper import PipelineHelper
        import re
        import seaborn as sns
        from sklearn import utils
        from sklearn.base import BaseEstimator, TransformerMixin
```

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer, ENGLISH_S
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, precision_recall_
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.pipeline import FeatureUnion, Pipeline
from sklearn.svm import LinearSVC

%matplotlib inline
```

2. Loading the Data

The data was collected through scraping a number of different websites as discussed in the written report. The scripts written for this data collection process are found in the scripts folder. Due to some parsing errors with the web scraper, some lyrics were not returned to appear as empty strings or null values in the CSV file.

In [3]:
```
tracks_by_artist = pd.read_csv("./data/tracks_with_lyrics_for_top_10_artists.csv")
tracks_by_artist = tracks_by_artist[~tracks_by_artist["lyrics"].isnull()]
```

In [4]:
```
tracks_by_artist.head()
```

Out[4]:

	artist	album	track	lyrics
0	T.I.	Trap Muzik	I Can't Quit	[Intro]\nHuh, hell nah, I can't quit\nHell nah...
1	T.I.	Trap Muzik	Be Easy	[Intro]\nUh-uh, uh-uh, uh\nAye, where the pian...
2	T.I.	Trap Muzik	No More Talk	[Verse 1]\nI'm either running for my life or I...
3	T.I.	Trap Muzik	Doin My Job	[T.I. - talking]\nAy I'm working here, know wh...
4	T.I.	Trap Muzik	24's	[Intro]\nYeah\nFor all my real ATL niggas, tha...

In [5]:
```
tracks_by_artist.shape
```

Out[5]: (679, 4)

Have a total of 679 tracks with lyrics with artist name and album title as additional metadata

3. Data Exploration

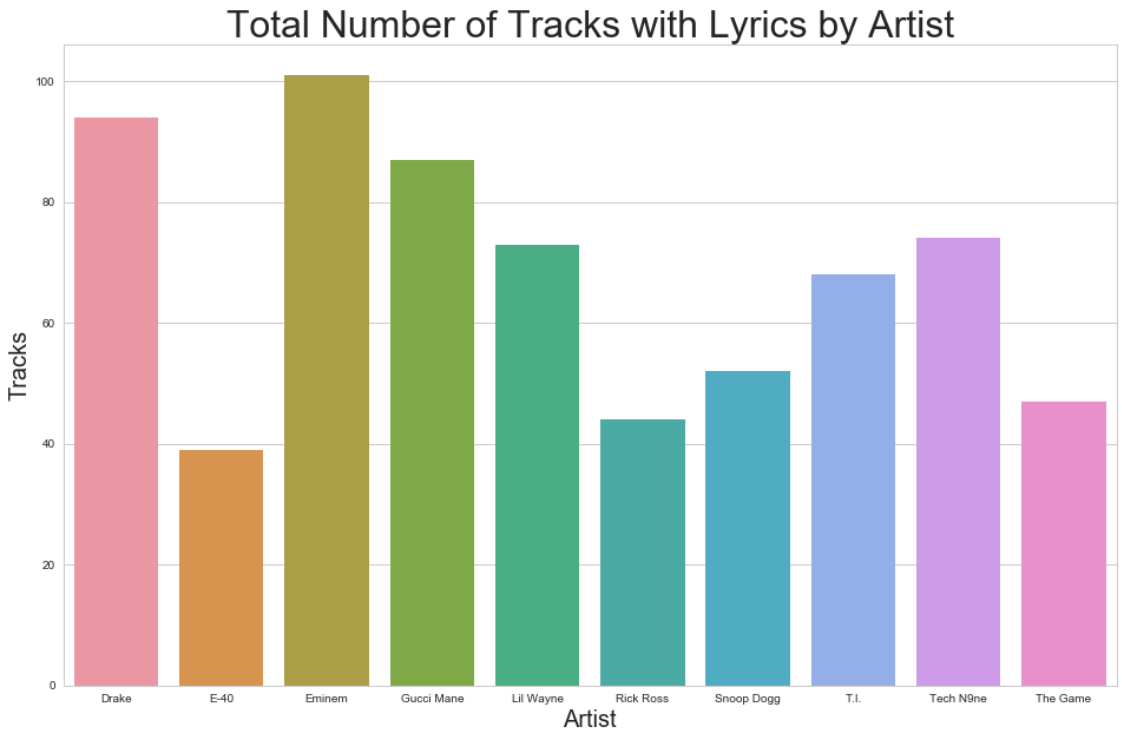
The following illustrates the break down of the number of tracks by artist considered in this project. This has implications for the evaluation metrics I will report in the investigation. The following considerations are important to decide the evaluation metrics,

- The project deals with a supervised, multiclass classification task,
- The classes in the original dataset are unbalanced.

For these reasons, I will report the precision, recall, and weighted fscore for each model. Together, these metrics do a better job of indicating the overall accuracy of a classifier, whereas with accuracy alone, there is a risk that we are not correctly identifying a good-performing classifier but rather just one that is biased towards the more populated classes.

```
In [6]: total_tracks_by_artist = tracks_by_artist.groupby("artist").agg("count")[["track"]]
        total_tracks_by_artist = total_tracks_by_artist.reset_index()
        sns.set(style="whitegrid")
        fig, ax = plt.subplots(figsize=(16,10))
        sns.barplot(ax=ax, x="artist", y="track", data=total_tracks_by_artist)
        ax.set_xlabel("Artist", fontsize=20)
        ax.set_ylabel("Tracks", fontsize=20)
        ax.set_title("Total Number of Tracks with Lyrics by Artist", fontsize=32)
```

```
Out[6]: Text(0.5,1,'Total Number of Tracks with Lyrics by Artist')
```



Same as above but with numbers

```
In [7]: tracks_by_artist.groupby("artist")["track"].agg("count")
```

```
Out[7]: artist
        Drake          94
        E-40           39
        Eminem        101
        Gucci Mane     87
        Lil Wayne      73
        Rick Ross      44
        Snoop Dogg     52
        T.I.           68
        Tech N9ne      74
```

```
The Game       47
Name: track, dtype: int64
```

4. Baseline Model

For a like-to-like comparison with future text representations and models, this section reports on the testing results of a logistic regression classifier using a Bag-of-Words (BOW) representation. This is very basic but will help indicate whether choices for later models help to improve on the overall classification task. BOW simply counts the frequency of words in each document as a way of representing the documents, where in this case the "documents" are the individual tracks. This introduces a lot of simplifications, which the other text representations approaches below try to address.

The fundamental task for this project is a multiclass classification, where the artist name is the target label, all the other features are inputs to the model.

To get an indication of the effectiveness of a given classifier, we need to perform a test train split on the original dataset. This is to achieve a better measure of the classifier's performance, as the test set will contain data points that are not in the training set. Or in other words, the classifier will not simply rely on overtraining with respect to the training set. Some other things to note, to make sure the test train split is repeatable if required, I am specifying a random state seed. As this is a multiclass dataset, I am specifying a stratified split using the artist name as the class label.

```
In [8]: train, test = train_test_split(tracks_by_artist, test_size=0.3, random_state=34, strati
        y_train, X_train = train["artist"], train
        y_test, X_test = test["artist"], test
```

Here we initialise the BOW model, build vocabulary from training data, and transform both training and test data to BOW representation. In this step, common words or "stop words" are removed according to a standard corpus, as well as using the standard NLTK tokenizer to parse tokens i.e. words, characters from the raw lyrics. It is a common strategy to remove stop words as these do not typically aid the classifier to choose between class labels, but rather just increase the dimensionality of vector space and take up more memory. I've also used the standard NLTK tokenizer simply because it does a good enough job at identifying word tokens.

It should also be mentioned here that I will not stem the words from the lyrics. Although this means that the bow vectors will be larger, as words with similar spelling will instead be mapped to their own unique index e.g. stop, stopping will be treated separately, it means we can get a better grasp of the actual written language being used. This is perhaps especially true for song lyrics where spelling is often non-standard, and slang and other neologisms are frequently used.

```
In [9]: bow = CountVectorizer(stop_words=ENGLISH_STOP_WORDS, tokenizer=word_tokenize)
        X_train_bow = bow.fit_transform(X_train["lyrics"])
        X_test_bow = bow.transform(X_test["lyrics"])

In [10]: print(X_train_bow.shape)
         print(X_test_bow.shape)

(475, 15593)
(204, 15593)
```

The BOW vectors have almost 15,000 components

Perform grid search cross-validation to find the optimal value for the C parameter. In this step, grid search works through the training data, splitting it into a training set and validation set accorinding to the "cv" parameter. By finding the average of these individual runs, it can find the best hyperparamter set. The predictions are then generated from the model with the optimal value for C.

```
In [12]: param_grid = {"C": [1e0, 1e1, 1e2, 1e3]}

         logreg = GridSearchCV(LogisticRegression(), cv=5, param_grid=param_grid)
         logreg.fit(X_train_bow, y_train.ravel())
         y_pred = logreg.predict(X_test_bow)

         prfs = precision_recall_fscore_support(y_test.ravel(), y_pred, average='weighted', lab
```

Reported results for baseline model.

```
In [13]: print('Testing Precision: {}'.format(prfs[0]))
         print('Testing Recall: {}'.format(prfs[1]))
         print('Testing FScore: {}'.format(prfs[2]))
         print("Logistic Regression Param C: {}".format(logreg.best_params_["C"]))
```

```
Testing Precision: 0.7297116622484269
Testing Recall: 0.7254901960784313
Testing FScore: 0.719971983089431
Logistic Regression Param C: 1.0
```

These are the scores to beat!

5. Analysis

Models to be considered,

- Logistic Regression
- Linear SVM

This is because these models have generally been used by previous authors in this task and due to their effectiveness for multilcass problems in general. Additionally, in the case of logistic regression, this provides a direct comparison with the baseline model.

The below function "best_results_by_model" is a simple way to extract the equal best performing classifier for each text representation strategy. For each classifier it finds one with the equal best mean cross-validation score against the test set considered. ("Equal best" because it doesn't consider the case of more than one set of hyperparameters for a given classifier producing the same mean test score.)

```
In [14]: # Helper function
         def best_results_by_model(grid_scores):

             best_results_by_model = {}
             for score in grid_scores:
```