# Analysis

June 9, 2018

## 1 Implementation

- The purpose of this notebook is to do comparitive analysis of three algorithms: Naive Bayes, Logistic Regression and Support Vector Machines for combinations of n-gram range and TFIDF norm values.
- This is done to select the appropriate value of 'n' in n-grams and to select the better norm for TFIDF.
- Two algorithms giving the best results will be considered as baseline algorithms.
- The hyperparameters of these algorithms will be tuned to further improve the results. The tuned-model with the best results will be finalized for the project.

### 1.1 Data Overview

The dataset used here is a combination of three twitter datasets which were rated by human raters for toxicity. There exist two files, train.csv and test.csv for training and testing respectively.

Hence the comments are tagged in the following three categories

- Hate Speech
- Offensive
- Clean

The tagging was done via crowdsourcing.

### 1.2 Import necessary modules

```
In [1]: import nltk
        import pickle
        import re
        import time
        import warnings

        import numpy as np
        import pandas as pd

        from data.preprocess_data import *
        from nltk.stem.porter import *
        from settings import *
```

```
from visualization.visualize import *
from sklearn import metrics
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
```

## 1.3 Basic EDA

### 1.3.1 Read dataset and take a peak

```
In [2]: df_train = pd.read_csv(TRAIN_DATA, index_col=False, lineterminator='\n')
        df_test = pd.read_csv(TEST_DATA, index_col=False, lineterminator='\n')
        print(df_train.columns)
        df_train.head()
```

```
Index(['output_class', 'text'], dtype='object')
```

Out[2]:

| | output_class | text |
|---|---|---|
| 0 | 1 | @adamjferraro I'm actually with my boys right ... |
| 1 | 1 | Your new bitch my old hoe |
| 2 | 2 | @justenholstein @PNF4LYFE @mistermegative @Dar... |
| 3 | 1 | RT @worst_B_ehavior: If you don't give your bf... |
| 4 | 1 | RT @3High_Tae: Bruh this bitch booty so flat t... |

### 1.3.2 Take a look at dataset size

```
In [3]: print("Train data size:", df_train.shape)
        print("Test data size: ", df_test.shape)
```

```
Train data size: (50454, 2)
Test data size:  (21624, 2)
```

### 1.3.3 Check the number of samples belonging to each class

```
In [4]: print("Train data class distribution: ", df_train.groupby('output_class').size())
        print("Test data class distribution: ", df_test.groupby('output_class').size())
```

```
Train data class distribution:  output_class
0    16747
1    16917
2    16790
dtype: int64
Test data class distribution:  output_class
0    7279
1    7109
2    7236
dtype: int64
```

## 1.4 Performance comparison of algorithms w.r.t diferrent features

### 1.4.1 Generate input and output variables

```
In [5]: X_train = df_train['text'].values
        Y_train = df_train['output_class'].values

        X_test = df_test['text'].values
        Y_test = df_test['output_class'].values
```

### 1.4.2 Train Naive Bayes algorithm iteratively for different sets of features

```
In [7]: estimators_nb = [
            ('vect', CountVectorizer()),
            ('tfidf', TfidfTransformer()),
            ('clf', MultinomialNB())
        ]
        nb_clf = Pipeline(estimators_nb)

        param_grid = dict(
            vect__ngram_range = [(1, 1), (1, 2), (1, 3)],
            tfidf__norm = ['l1', 'l2']
        )
        kfold = KFold(n_splits=10, random_state=23)

        gs_nb_clf = GridSearchCV(nb_clf, param_grid=param_grid,
                                 n_jobs=-1, cv=kfold, scoring='accuracy')

        start = time.time()
        gs_nb_clf = gs_nb_clf.fit(X_train, Y_train)
        end = time.time()
        print("Time taken:", end - start)

        with open("../../models/gs_nb_features.pkl", 'wb') as gs_nb_clf_file:
            pickle.dump(gs_nb_clf, gs_nb_clf_file, pickle.HIGHEST_PROTOCOL)
```

```
Time taken: 157.15308475494385
```

### 1.4.3 Train Logistic Regression algorithm iteratively for different sets of features

```
In [8]: estimators_lr = [
            ('vect', CountVectorizer()),
            ('tfidf', TfidfTransformer()),
            ('clf', LogisticRegression())
        ]
        lr_clf = Pipeline(estimators_lr)

        param_grid = dict(
            vect__ngram_range = [(1, 1), (1, 2), (1, 3)],
            tfidf__norm = ['l1', 'l2']
        )
        kfold = KFold(n_splits=10, random_state=23)
        gs_lr_clf = GridSearchCV(lr_clf, param_grid=param_grid,
                                 n_jobs=-1, cv=kfold, scoring='accuracy')

        start = time.time()
        gs_lr_clf = gs_lr_clf.fit(X_train, Y_train)
        end = time.time()
        print("Time taken:", end - start)

        with open("../../models/gs_lr_features.pkl", 'wb') as gs_lr_clf_file:
            pickle.dump(gs_lr_clf, gs_lr_clf_file, pickle.HIGHEST_PROTOCOL)
```

```
Time taken: 277.4993906021118
```

### 1.4.4 Train Support Vector Machine iteratively for different sets of features

```
In [9]: estimators_svm = [
            ('vect', CountVectorizer()),
            ('tfidf', TfidfTransformer()),
            ('clf', SGDClassifier(tol=None, max_iter=100))
        ]
        svm_clf = Pipeline(estimators_svm)

        param_grid = dict(
            vect__ngram_range = [(1, 1), (1, 2), (1, 3)],
            tfidf__norm = ['l1', 'l2']
        )
        kfold = KFold(n_splits=10, random_state=23)
        gs_svm_clf = GridSearchCV(svm_clf, param_grid=param_grid,
                                  n_jobs=-1, cv=kfold, scoring='accuracy')
```

```
        start = time.time()
        gs_svm_clf = gs_svm_clf.fit(X_train, Y_train)
        end = time.time()
        print("Time taken:", end - start)

        with open("../../models/gs_svm_features.pkl", 'wb') as gs_svm_clf_file:
            pickle.dump(gs_svm_clf, gs_svm_clf_file, pickle.HIGHEST_PROTOCOL)
```

```
Time taken: 392.3896949291229
```

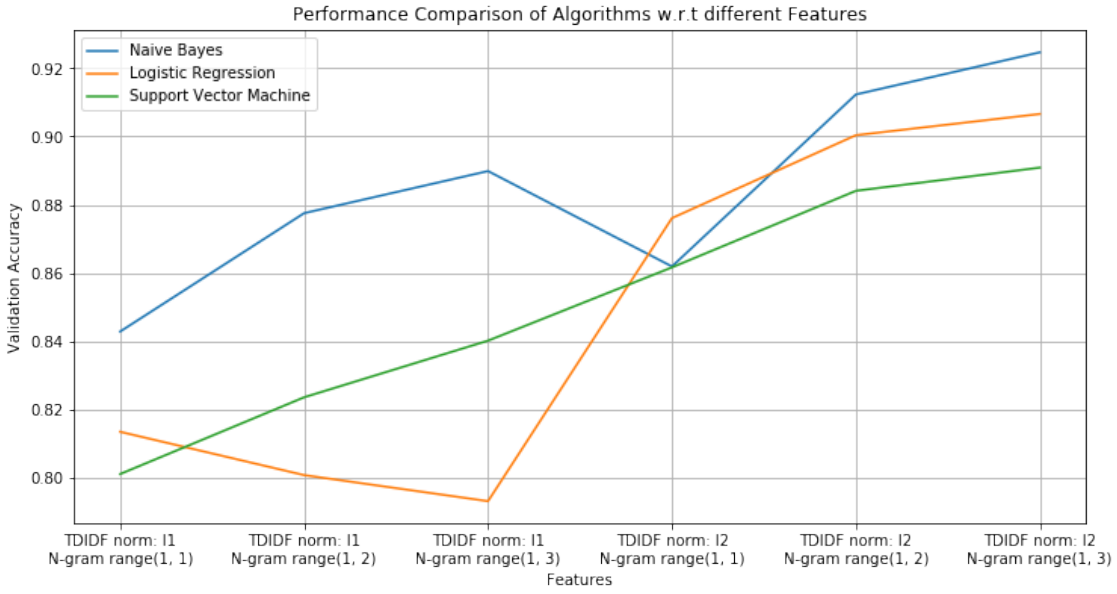
### 1.4.5 Plot line graph of results

```
In [10]: with open("../../models/gs_nb_features.pkl", 'rb') as gs_nb_clf_file:
             gs_nb_clf = pickle.load(gs_nb_clf_file)
         with open("../../models/gs_lr_features.pkl", 'rb') as gs_lr_clf_file:
             gs_lr_clf = pickle.load(gs_lr_clf_file)
         with open("../../models/gs_svm_features.pkl", 'rb') as gs_svm_clf_file:
             gs_svm_clf = pickle.load(gs_svm_clf_file)

         x = list()
         params = gs_nb_clf.cv_results_['params']
         for param in params:
             norm = "TDIDF norm: " + str(param['tfidf__norm'])
             ngram_range = "N-gram range" + str(param['vect__ngram_range'])
             x.append(norm + "\n" + ngram_range)

         nb_means = gs_nb_clf.cv_results_['mean_test_score']
         lr_means = gs_lr_clf.cv_results_['mean_test_score']
         svm_means = gs_svm_clf.cv_results_['mean_test_score']
         y = [nb_means, lr_means, svm_means]

         draw_performance_comparison(x, y)
```




The figure shows all the three algorithms perform better for 'l2' norm of TFIDF. Futhermore, Naive Bayes and Logistic Regression show better performance than SVM for 'l2' norm. Therefore, we consider Naive Bayes and Logistic Regression for TFIDF norm 'l2' and N-gram range (1, 3) as baseline algorithms and tune their hyperparameters.

## 1.5 Hyperparameter tuning

### 1.5.1 Tune hyperparameters of Naive Bayes

```
In [11]: estimators_nb = [
             ('vect', CountVectorizer(ngram_range=(1,3))),
             ('tfidf', TfidfTransformer(norm='l2')),
             ('clf', MultinomialNB())
         ]
         nb_clf = Pipeline(estimators_nb)

         param_grid = dict(
             clf__alpha = [0.01, 0.1, 1, 10],
         )
         kfold = KFold(n_splits=10, random_state=23)
         gs_nb_clf = GridSearchCV(nb_clf, param_grid=param_grid,
                                  n_jobs=-1, cv=kfold, scoring='accuracy')

         start = time.time()
         gs_nb_clf = gs_nb_clf.fit(X_train, Y_train)
         end = time.time()
         print("Time taken:", end - start)
```

```
        with open("../../models/gs_nb_tuned.pkl", 'wb') as gs_nb_clf_file:
            pickle.dump(gs_nb_clf, gs_nb_clf_file, pickle.HIGHEST_PROTOCOL)
```

```
Time taken: 161.61590766906738
```

### 1.5.2 Result

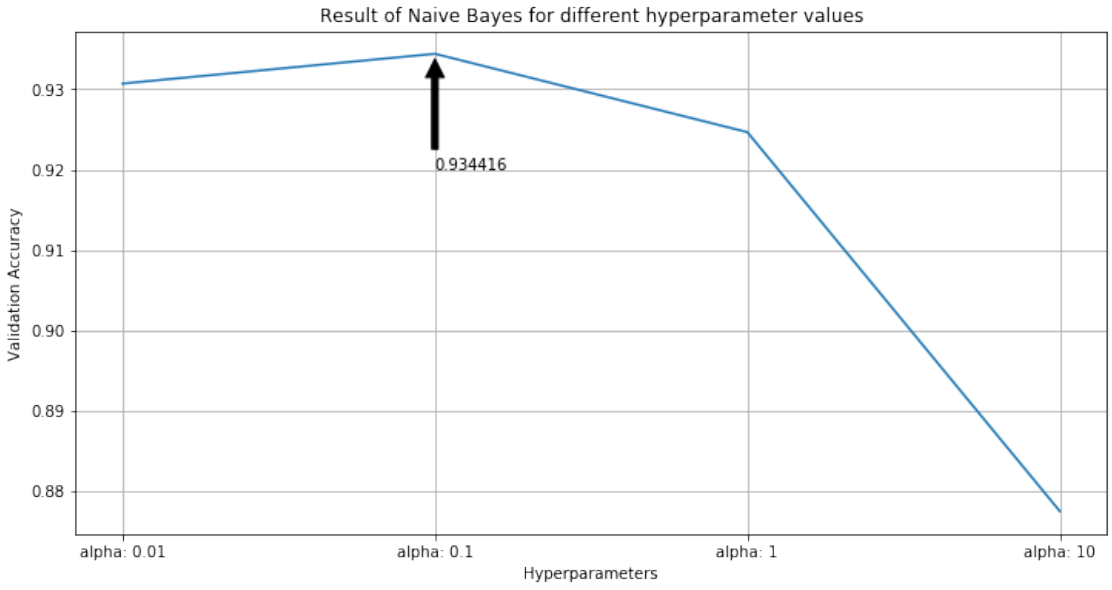
```
In [12]: # NB Results
        with open("../../models/gs_nb_tuned.pkl", 'rb') as gs_nb_clf_file:
            gs_nb_clf = pickle.load(gs_nb_clf_file)

        x = list()
        params = gs_nb_clf.cv_results_['params']
        for param in params:
            alpha = "alpha: " + str(param['clf__alpha'])
            x.append(alpha)

        nb_means = gs_nb_clf.cv_results_['mean_test_score']
        y = nb_means

        draw_hp_performance_nb(x, y)

        print("Best: %f using %s" % (gs_nb_clf.best_score_, gs_nb_clf.best_params_))
```




```
Best: 0.934416 using {'clf__alpha': 0.1}
```

### 1.5.3 Tune hyperparameters of Logistic Regression

```
In [14]: estimators_lr = [
             ('vect', CountVectorizer(ngram_range=(1,3))),
             ('tfidf', TfidfTransformer(norm='l2')),
             ('clf', LogisticRegression(class_weight='balanced'))
         ]
         lr_clf = Pipeline(estimators_lr)

         param_grid = dict(
             clf__C = [10, 100],
             clf__solver = ['newton-cg', 'liblinear', 'saga']
         )
         kfold = KFold(n_splits=10, random_state=23)
         gs_lr_clf = GridSearchCV(lr_clf, param_grid=param_grid,
                                  n_jobs=-1, cv=kfold, scoring='accuracy')

         start = time.time()
         with warnings.catch_warnings():
             warnings.filterwarnings('ignore')
             gs_lr_clf = gs_lr_clf.fit(X_train, Y_train)
         end = time.time()
         print("Time taken:", end - start)

         with open("../../models/gs_lr_tuned.pkl", 'wb') as gs_lr_clf_file:
             pickle.dump(gs_lr_clf, gs_lr_clf_file, pickle.HIGHEST_PROTOCOL)
```

```
Time taken: 1323.6134932041168
```

### 1.5.4 Result

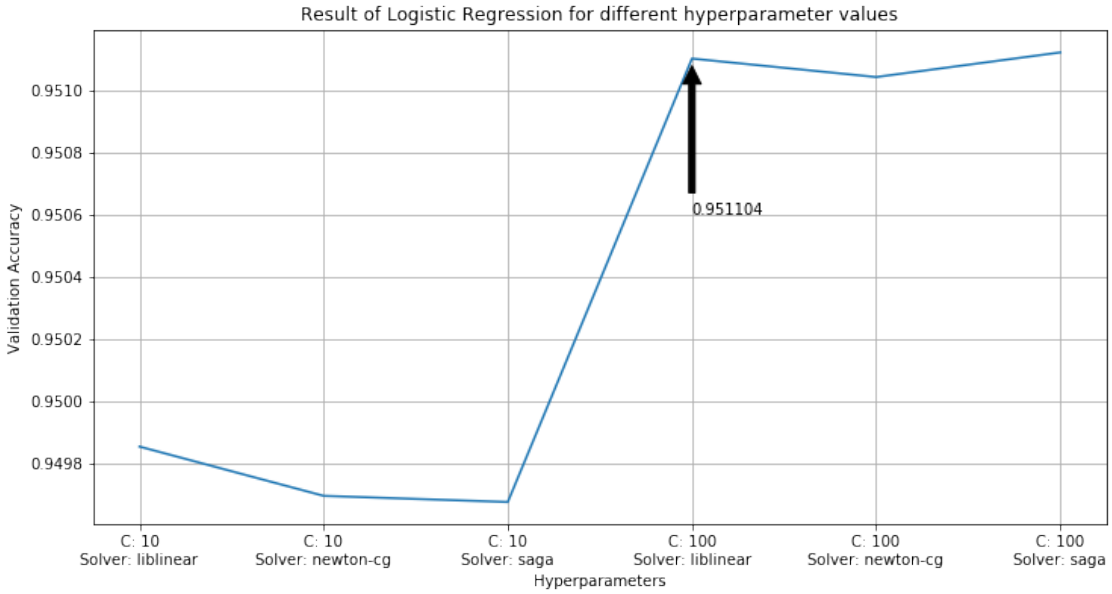
```
In [15]: # LR Results
         with open("../../models/gs_lr_tuned.pkl", 'rb') as gs_lr_clf_file:
             gs_lr_clf = pickle.load(gs_lr_clf_file)

         x = list()
         params = gs_lr_clf.cv_results_['params']
         for param in params:
             C = "C: " + str(param['clf__C'])
             solver = "Solver: " + str(param['clf__solver'])
             x.append(C + "\n" + solver)

         lr_means = gs_lr_clf.cv_results_['mean_test_score']
         y = lr_means

         draw_hp_performance_lr(x, y)

         print("Best: %f using %s" % (gs_lr_clf.best_score_, gs_lr_clf.best_params_))
```




```
Best: 0.951124 using {'clf__C': 100, 'clf__solver': 'saga'}
```