Apply_SVM_to_Amazon_reviews_data_set_avg_w2vec_[M]

June 1, 2018

In [1]:
```
!pip install PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# 1. Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

file_list = drive.ListFile({'q': "'1pbLvjcsi6UtFm3sPciCJGbCG4NK3uyuS' in parents and tra
for file1 in file_list:
  print('title: %s, id: %s' % (file1['title'], file1['id']))

sql = drive.CreateFile({'id': '1OzLc3k6-T55I-XRMq47ERyCbQbVw4caF'})
sql.GetContentFile('final.sqlite')
```

```
Requirement already satisfied: PyDrive in /usr/local/lib/python3.6/dist-packages (1.3.1)
Requirement already satisfied: oauth2client>=4.0.0 in /usr/local/lib/python3.6/dist-packages (fr
Requirement already satisfied: google-api-python-client>=1.2 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: PyYAML>=3.0 in /usr/local/lib/python3.6/dist-packages (from PyDri
Requirement already satisfied: pyasn1>=0.1.7 in /usr/local/lib/python3.6/dist-packages (from oau
Requirement already satisfied: httplib2>=0.9.1 in /usr/local/lib/python3.6/dist-packages (from c
Requirement already satisfied: six>=1.6.1 in /usr/local/lib/python3.6/dist-packages (from oauth2
Requirement already satisfied: rsa>=3.1.4 in /usr/local/lib/python3.6/dist-packages (from oauth2
Requirement already satisfied: pyasn1-modules>=0.0.5 in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: uritemplate<4dev,>=3.0.0 in /usr/local/lib/python3.6/dist-package
title: Apply SVM to Amazon reviews data set [M].ipynb, id: 1ElWunFgWZPb1Iq6w4ZMmqoBSVuPt0QQ4
title: Apply Logistic regression to Amazon reviews data set. [M].ipynb, id: 1Es1wP2edJ0vrKasA5wY
title: Apply Naive Bayes to Amazon reviews [M].ipynb, id: 1qPxAZeYQUM-eqaKnOSM5ubK2IPIVmdyo
title: clean_final.sqlite, id: 1TOHyUqaVFyD8HfIQEM6WN8jF8SpEOsAo
title: KNN on Credit Card fraud detection.ipynb, id: 1CkA-RBfXqvubKkQrpnjbYUKVsC7VHlTl
title: creditcard.csv, id: 1VpeqlSOlPVrlzlMIqvQTzc3Pno_Cj4SV
title: creditcard.csv, id: 1bnZktEq3N_5wjoCH85oIXHxNwXUW_jx-
title: Untitled, id: 1KOwwkizWx3WO8d-zw-YewWIUrPdINYmp
```

```
title: final.sqlite, id: 1OzLc3k6-T55I-XRMq47ERyCbQbVw4caF
title: HeavyComputations.ipynb, id: 1aBORe3gqeFY-iNhzMtr-TIkzEyEvFxcG
title: LogisticRegression.ipynb, id: 1WcVTklMZBMu9VTCIWeupOKOr2aYbHk8p
```

```
In [5]: !pip install imblearn
```

```
Requirement already satisfied: imblearn in /usr/local/lib/python3.6/dist-packages (0.0)
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from imbalanced-
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from imbalanced-
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.6/dist-packages (from imba
```

```
In [7]: !pip install gensim
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.6/dist-packages (3.4.0)
Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python3.6/dist-packages (from gen
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.6/dist-packages (from gensim
Requirement already satisfied: scipy>=0.18.1 in /usr/local/lib/python3.6/dist-packages (from gen
Requirement already satisfied: boto>=2.32 in /usr/local/lib/python3.6/dist-packages (from smart-
Requirement already satisfied: boto3 in /usr/local/lib/python3.6/dist-packages (from smart-open>
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from smart-op
Requirement already satisfied: bz2file in /usr/local/lib/python3.6/dist-packages (from smart-ope
Requirement already satisfied: botocore<1.11.0,>=1.10.30 in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: s3transfer<0.2.0,>=0.1.10 in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (fro
Requirement already satisfied: urllib3<1.23,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: idna<2.7,>=2.5 in /usr/local/lib/python3.6/dist-packages (from re
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: python-dateutil<3.0.0,>=2.1; python_version >= "2.7" in /usr/loca
Requirement already satisfied: docutils>=0.10 in /usr/local/lib/python3.6/dist-packages (from bo
```

```
In [0]: from sklearn.model_selection import train_test_split

        from sklearn.grid_search import GridSearchCV
        from sklearn.grid_search import RandomizedSearchCV
        from scipy.stats import uniform
        from scipy.stats import norm

        from imblearn.over_sampling import SMOTE

        import sqlite3

        import pandas as pd
        import numpy as np
```

```
        from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.feature_extraction.text import CountVectorizer
        import gensim

        from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

        from sklearn.preprocessing import StandardScaler
```

```
In [0]: con = sqlite3.connect('final.sqlite') # this is cleaned dataset
        final = pd.read_sql_query("""
        SELECT Score, Text_not_included
        FROM reviews
        """, con)[:2000]
```

```
In [0]: for i, seq in enumerate(final['Text_not_included']):
          final['Text_not_included'][i]=final['Text_not_included'][i].decode('UTF-8')
        X_train, X_test, y_train , y_test = train_test_split(final['Text_not_included'], final['
```

0.1 Generate Count BoW vectors

```
In [41]: count_vect = CountVectorizer(ngram_range=(1,2) )
         count_vect.fit(X_train)
```

```
Out[41]: CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                lowercase=True, max_df=1.0, max_features=None, min_df=1,
                ngram_range=(1, 2), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, vocabulary=None)
```

```
In [0]: bow_train=count_vect.transform(X_train)
        bow_test=count_vect.transform(X_test)
```

0.2 Generate TF IDF vectors

```
In [0]: tf_idf_vect=TfidfVectorizer(ngram_range=(1,2), min_df=10, dtype=float)
        tf_idf_vect.fit(X_train)
        tf_idf_train=tf_idf_vect.transform(X_train)
        tf_idf_test=tf_idf_vect.transform(X_test)
```

0.3 Generate average Word2Vec representations

```
In [0]: sentences=[]
        for review in X_train:
          sentence=[]
          for word in review.split():
            sentence.append(word)
          sentences.append(sentence)
```

```
w2vec_model=gensim.models.word2vec.Word2Vec(sentences, min_count=10)

avg_w2vec_train=np.zeros(shape=(len(X_train), 100), dtype=float)

for i, sentence in enumerate(sentences):
  for word in sentence:
    try:
      avg_w2vec_train[i]+=w2vec_model.wv[word]

    except KeyError:
      pass

  avg_w2vec_train[i]/=len(sentence)

sentences=[]
for review in X_test:
  sentence=[]
  for word in review.split():
    sentence.append(word)
  sentences.append(sentence)

avg_w2vec_test=np.zeros(shape=(len(X_test), 100), dtype=float)

for i, sentence in enumerate(sentences):
  for word in sentence:
    try:
      avg_w2vec_test[i]+=w2vec_model.wv[word]

    except KeyError:
      pass

  avg_w2vec_test[i]/=len(sentence)
```

0.4 Generate TF IDF weighted Word2Vec representations

```
In [0]: sentences=[]
        for review in X_train:
          sentence=[]
          for word in review.split():
            sentence.append(word)
          sentences.append(sentence)

        tf_idf_w2vec_train=np.zeros((len(X_train), 100), dtype=float)
        feat=tf_idf_vect.get_feature_names()
        for i, sentence in enumerate(sentences):
          tf_idf_sum=0
          for word in sentence:
```

```
            try:
              tf_idf_w2vec_train[i]+=w2vec_model.wv[word]*tf_idf_train[i, feat.index(word)]
              tf_idf_sum+=tf_idf_train[i, feat.index(word)]
            except KeyError:
              pass
            except ValueError:
              pass
          tf_idf_w2vec_train[i]/=tf_idf_sum

        sentences=[]
        for review in X_test:
          sentence=[]
          for word in review.split():
            sentence.append(word)
          sentences.append(sentence)

        tf_idf_w2vec_test=np.zeros((len(X_test), 100), dtype=float)

        for i, sentence in enumerate(sentences):
          tf_idf_sum=0
          for word in sentence:
            try:
              tf_idf_w2vec_test[i]+=w2vec_model.wv[word]*tf_idf_test[i, feat.index(word)]
              tf_idf_sum+=tf_idf_test[i, feat.index(word)]
            except KeyError:
              pass
            except ValueError:
              pass
          tf_idf_w2vec_test[i]/=tf_idf_sum
```

0.5 Upsampling followed by standardization

```
In [45]: # Upsampling minority class
         over_sampler = SMOTE(ratio='minority')
         bow_train_resampled, y_train_resampled = over_sampler.fit_sample(bow_train, y_train)
         tf_idf_train_resampled, y_train_resampled = over_sampler.fit_sample(tf_idf_train, y_tra
         avg_w2vec_train_resampled, y_train_resampled = over_sampler.fit_sample(avg_w2vec_train,
         #tf_idf_w2vec_train_resampled, y_train_resampled = over_sampler.fit_sample(tf_idf_w2vec

         scaler_bow=StandardScaler(with_mean=False)
         scaler_tf_idf=StandardScaler(with_mean=False)
         scaler_w2vec=StandardScaler()
         #scaler_tf_w2vec=StandardScaler()

         scaler_bow.fit(bow_train_resampled)
         scaler_tf_idf.fit(tf_idf_train_resampled)
         scaler_w2vec.fit(avg_w2vec_train_resampled)
         #scaler_tf_w2vec.fit(tf_idf_w2vec_train_resampled)
```

```
        bow_train_scaled=scaler_bow.transform(bow_train_resampled)
        tf_idf_train_scaled=scaler_tf_idf.transform(tf_idf_train_resampled)
        avg_w2vec_train_scaled=scaler_w2vec.transform(avg_w2vec_train_resampled)
        #tf_idf_w2vec_train_scaled=scaler_tf_w2vec.transform(tf_idf_w2vec_train_resampled)

        bow_test_scaled=scaler_bow.transform(bow_test)
        tf_idf_test_scaled=scaler_tf_idf.transform(tf_idf_test)
        avg_w2vec_test_scaled=scaler_w2vec.transform(avg_w2vec_test)
        #tf_idf_w2vec_test_scaled=scaler_tf_w2vec.transform(tf_idf_w2vec_test)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:475: DataConversionWarning: D
  warnings.warn(msg, DataConversionWarning)
```

```
In [0]: from sklearn.svm import SVC
```

0.6 Classification using avg_word2Vec

```
In [62]: tuned_parameters = {'C': np.linspace(10, 15, 20, dtype=float),'gamma' : np.linspace(0.0

         #Using GridSearchCV
         gscv = GridSearchCV(SVC(), tuned_parameters, scoring = 'accuracy', cv=5)

         print(gscv.fit(avg_w2vec_train_scaled, y_train_resampled))

         tuned_parameters = {'C' : uniform(10,15), 'gamma' : uniform(0, 1)}

         #Using RandomizedSearchCV
         rscv = RandomizedSearchCV(SVC(), tuned_parameters, scoring = 'accuracy', cv=5, n_iter=2

         print(rscv.fit(avg_w2vec_train_scaled, y_train_resampled))
```

```
GridSearchCV(cv=5, error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'C': array([10.     , 10.26316, 10.52632, 10.78947, 11.05263, 11.31579,
       11.57895, 11.84211, 12.10526, 12.36842, 12.63158, 12.89474,
       13.15789, 13.42105, 13.68421, 13.94737, 14.21053, 14.47368,
       14.73684, 15.     ]), 'gamma': array([0.001  , 0.05358, 0.10616, 0.15874, 0.21132, 0.2638
       0.36905, 0.42163, 0.47421, 0.52679, 0.57937, 0.63195, 0.68453,
       0.73711, 0.78968, 0.84226, 0.89484, 0.94742, 1.     ])},
       pre_dispatch='2*n_jobs', refit=True, scoring='accuracy', verbose=0)
RandomizedSearchCV(cv=5, error_score='raise',
          estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
```

```
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
          fit_params={}, iid=True, n_iter=25, n_jobs=1,
          param_distributions={'C': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7fe
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          scoring='accuracy', verbose=0)
```

```
In [63]: predictions = gscv.best_estimator_.predict(avg_w2vec_test_scaled)
         print(classification_report(y_test, predictions))
         print(confusion_matrix(y_test, predictions).T)
         tn, fp, fn, tp = confusion_matrix(y_test, predictions).ravel()

         print("TPR = {}\n TNR = {}\n FPR = {}\n FNR = {}".format(tp/(fn+tp), tn/(tn+fp), fp/(tn

         predictions = rscv.best_estimator_.predict(avg_w2vec_test_scaled)
         print(classification_report(y_test, predictions))
         print(confusion_matrix(y_test, predictions).T)
         tn, fp, fn, tp = confusion_matrix(y_test, predictions).ravel()

         print("TPR = {}\n TNR = {}\n FPR = {}\n FNR = {}".format(tp/(fn+tp), tn/(tn+fp), fp/(tn
```

```
             precision    recall  f1-score   support

   negative       0.33      0.07      0.11        87
   positive       0.79      0.96      0.87       313

avg / total       0.69      0.77      0.70       400

[[  6  12]
 [ 81 301]]
TPR = 0.9616613418530351
 TNR = 0.06896551724137931
 FPR = 0.9310344827586207
 FNR = 0.038338658146964855
             precision    recall  f1-score   support

   negative       0.45      0.11      0.18        87
   positive       0.80      0.96      0.87       313

avg / total       0.72      0.78      0.72       400

[[ 10  12]
 [ 77 301]]
TPR = 0.9616613418530351
 TNR = 0.11494252873563218
 FPR = 0.8850574712643678
 FNR = 0.038338658146964855
```

```
In [64]: gscv.best_estimator_
```

```
Out[64]: SVC(C=10.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma=0.9474210526315789,
           kernel='rbf', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
In [65]: rscv.best_estimator_
```

```
Out[65]: SVC(C=23.865376755248537, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape='ovr', degree=3, gamma=0.8557541623170543,
           kernel='rbf', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

0.7 Conclusions

avg_word2vec performs better than other representations especially BoW and TF IDF. TNR is somewhat improved. gamma : 0.855 20 < C < 25