Apply_Logistic_regression_to_Amazon_reviews_data_set_[M]

May 29, 2018

In [33]:
```
!pip install PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
```

```
Requirement already satisfied: PyDrive in /usr/local/lib/python3.6/dist-packages (1.3.1)
Requirement already satisfied: PyYAML>=3.0 in /usr/local/lib/python3.6/dist-packages (from PyDri
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: pyasn1-modules>=0.0.5 in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: rsa>=3.1.4 in /usr/local/lib/python3.6/dist-packages (from oauth2
Requirement already satisfied: httplib2>=0.9.1 in /usr/local/lib/python3.6/dist-packages (from o
Requirement already satisfied: pyasn1>=0.1.7 in /usr/local/lib/python3.6/dist-packages (from oau
Requirement already satisfied: six>=1.6.1 in /usr/local/lib/python3.6/dist-packages (from oauth2
Requirement already satisfied: uritemplate<4dev,>=3.0.0 in /usr/local/lib/python3.6/dist-package
```

In [0]:
```
# 1. Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

In [35]:
```
file_list = drive.ListFile({'q': "'1pbLvjcsi6UtFm3sPciCJGbCG4NK3uyuS' in parents and tr
for file1 in file_list:
  print('title: %s, id: %s' % (file1['title'], file1['id']))
```

```
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 [0]:
```
sql = drive.CreateFile({'id': '1OzLc3k6-T55I-XRMq47ERyCbQbVw4caF'})
sql.GetContentFile('final.sqlite')
```

In [37]:
```
!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 [0]:
```
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.grid_search import GridSearchCV
from sklearn.linear_model import LogisticRegression

import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
import matplotlib.pyplot as plt
import sqlite3

from imblearn.over_sampling import SMOTE
```

In [0]:
```
con = sqlite3.connect('final.sqlite') # this is cleaned dataset
final = pd.read_sql_query("""
SELECT Score, Text_not_included
FROM reviews
""", con)[:12000]
```

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

In [0]:
```
X_train, X_test, y_train , y_test = train_test_split(final['Text_not_included'], final['
```

In [42]:
```
vectorizer = TfidfVectorizer(ngram_range=(1,2), min_df=5   , dtype=float) #play around
vectorizer.fit(X_train)
```

Out[42]:
```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'float'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=5,
        ngram_range=(1, 2), norm='l2', preprocessor=None, smooth_idf=True,
        stop_words=None, strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None)
```

```
In [0]: train_vectors = vectorizer.transform(X_train)

In [44]: train_vectors.get_shape()

Out[44]: (9600, 13399)

In [0]: # Oversampling train set
        over_sampler = SMOTE(ratio='minority')
        X_train_resampled, y_train_resampled = over_sampler.fit_sample(train_vectors, y_train)

In [46]: X_train_resampled.shape

Out[46]: (16006, 13399)

In [0]: test_vectors = vectorizer.transform(X_test)

In [0]: tuned_parameters = [{'C': np.linspace(1.00000000e-05, 1.66581253e+00, 100, dtype=float)}

In [0]: from sklearn.preprocessing import StandardScaler
        scaler=StandardScaler(with_mean=False)
        scaler.fit(X_train_resampled)
        X_train_scaled = scaler.transform(X_train_resampled)
        X_test_scaled = scaler.transform(test_vectors)

In [0]: from datetime import datetime

In [0]: model = LogisticRegression(penalty='l2')

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

        t0 = datetime.now()
        print(gscv.fit(X_train_scaled, y_train_resampled))
        t1=datetime.now()

        print("Execution time = {}".format(t1-t0))

In [50]: gscv.best_estimator_

Out[50]: LogisticRegression(C=0.01683628818181818, class_weight=None, dual=False,
                   fit_intercept=True, intercept_scaling=1, max_iter=100,
                   multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
                   solver='liblinear', tol=0.0001, verbose=0, warm_start=False)

In [51]: gscv.best_estimator_.fit(X_train_scaled, y_train_resampled)

Out[51]: LogisticRegression(C=0.01683628818181818, class_weight=None, dual=False,
                   fit_intercept=True, intercept_scaling=1, max_iter=100,
                   multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
                   solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

```
In [0]: predictions = gscv.best_estimator_.predict(X_test_scaled)
```

```
In [0]: from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
```

```
In [54]: print(classification_report(y_test, predictions))
```

```
             precision    recall  f1-score   support

   negative       0.69      0.63      0.66       733
   positive       0.93      0.95      0.94      4058

avg / total       0.90      0.90      0.90      4791
```

```
In [56]: print(confusion_matrix(y_test, predictions).T)
         tn, fp, fn, tp = confusion_matrix(y_test, predictions).ravel()
```

```
[[ 465  207]
 [ 268 3851]]
```

```
In [57]: print("TPR = {}\n TNR = {}\n FPR = {}\n FNR = {}".format(tp/(fn+tp), tn/(tn+fp), fp/(tn
```

```
TPR = 0.948989650073928
 TNR = 0.6343792633015006
 FPR = 0.3656207366984993
 FNR = 0.05101034992607196
```

```
In [0]: from sklearn.grid_search import RandomizedSearchCV
        from scipy.stats import uniform
```

```
In [0]: tuned_parameters = {'C' : uniform(1.00000000e-05, 1.66581253e+00)}
```

```
In [60]: rscv = RandomizedSearchCV(model, tuned_parameters, scoring = 'accuracy', cv=5, n_iter=1

         t0=datetime.now()
         print(rscv.fit(X_train_scaled, y_train_resampled))
         t1=datetime.now()

         print("Execution time = {}".format(t1-t0))
```

```
RandomizedSearchCV(cv=5, error_score='raise',
          estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
          fit_params={}, iid=True, n_iter=100, n_jobs=1,
          param_distributions={'C': <scipy.stats._distn_infrastructure.rv_frozen object at 0x7fd
```

```
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          scoring='accuracy', verbose=0)
Execution time = 0:14:46.027157
```

```
In [61]: rscv.best_estimator_
```

```
Out[61]: LogisticRegression(C=0.033579382735105565, class_weight=None, dual=False,
                   fit_intercept=True, intercept_scaling=1, max_iter=100,
                   multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
                   solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

```
In [62]: predictions = rscv.best_estimator_.predict(X_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()
```

```
             precision    recall  f1-score   support

   negative       0.68      0.63      0.66       733
   positive       0.93      0.95      0.94      4058

avg / total       0.90      0.90      0.90      4791

[[ 464  214]
 [ 269 3844]]
```

```
In [63]: print("TPR = {}\n TNR = {}\n FPR = {}\n FNR = {}".format(tp/(fn+tp), tn/(tn+fp), fp/(tn
```

```
TPR = 0.9472646623952686
 TNR = 0.6330150068212824
 FPR = 0.3669849931787176
 FNR = 0.0527353376047314
```

0.1 Remarks

Huge improvement in performance over Naive Bayes (note the improved TNR) and not much difference between GridSearch and RandomSearch although the latter is somewhat faster (note the time)

0.2 Performance with L1 regulariser

```
In [64]: rscv = RandomizedSearchCV(LogisticRegression(penalty='l1'), tuned_parameters, scoring =
         rscv.fit(X_train_scaled, y_train_resampled)
         predictions = rscv.best_estimator_.predict(X_test_scaled)
         print(classification_report(y_test, predictions))
         print("\n")
```

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

```
             precision    recall  f1-score   support

   negative       0.70      0.71      0.71       733
   positive       0.95      0.95      0.95      4058

avg / total       0.91      0.91      0.91      4791



[[ 522  220]
 [ 211 3838]]


TPR = 0.9457861015278463
 TNR = 0.7121418826739427
 FPR = 0.2878581173260573
 FNR = 0.05421389847215377
```

L1 regulariser outperforms L2 regulariser (higher TNR) only with 50 iterations. When done with 100 iterations of random search, this result is bound to improve.

0.3 Effect of increasing lambda in L1 regularised Logistic Regression

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

```
Out[65]: LogisticRegression(C=1.4158263696229512, class_weight=None, dual=False,
                   fit_intercept=True, intercept_scaling=1, max_iter=100,
                   multi_class='ovr', n_jobs=1, penalty='l1', random_state=None,
                   solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

```
In [0]: best_C = rscv.best_estimator_.C
```

```
In [92]: for i in range(3, 10, 1):
             C = best_C/float(i/2)
             model = LogisticRegression(C=C, penalty = 'l1')
             model.fit(X_train_scaled, y_train_resampled)
             predictions = model.predict(X_test_scaled)
             print("results for {} times best lambda".format(i/2))
             print(classification_report(y_test, predictions))
             print("\n")
             print(confusion_matrix(y_test, predictions).T)
             tn, fp, fn, tp = confusion_matrix(y_test, predictions).ravel()
             print("\n")
```

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

            print("number of nonzero components = {}".format(np.count_nonzero(model.coef_)))
            print("sparsity = {}".format(float((len(vectorizer.get_feature_names())-np.count_nonz
            print('\n*************************************************************************
```

```
results for 1.5 times best lambda
             precision    recall  f1-score   support

   negative       0.70      0.72      0.71       733
   positive       0.95      0.95      0.95      4058

avg / total       0.91      0.91      0.91      4791



[[ 527  221]
 [ 206 3837]]


TPR = 0.9455396747166092
 TNR = 0.7189631650750341
 FPR = 0.2810368349249659
 FNR = 0.054460325283390836
number of nonzero components = 6181
sparsity = 0.7694258962211362

*******************************************************************************************

results for 2.0 times best lambda
             precision    recall  f1-score   support

   negative       0.71      0.72      0.71       733
   positive       0.95      0.95      0.95      4058

avg / total       0.91      0.91      0.91      4791



[[ 526  220]
 [ 207 3838]]


TPR = 0.9457861015278463
 TNR = 0.7175989085948158
 FPR = 0.28240109140518416
 FNR = 0.05421389847215377
number of nonzero components = 6032
```

```
sparsity = 0.7749841459320327

**************************************************************************************************

results for 2.5 times best lambda
             precision    recall  f1-score   support

   negative       0.70      0.71      0.71       733
   positive       0.95      0.94      0.95      4058

avg / total       0.91      0.91      0.91      4791



[[ 524  225]
 [ 209 3833]]


TPR = 0.9445539674716609
 TNR = 0.7148703956343793
 FPR = 0.28512960436562074
 FNR = 0.05544603252833908
number of nonzero components = 5900
sparsity = 0.7799082329242362

**************************************************************************************************

results for 3.0 times best lambda
             precision    recall  f1-score   support

   negative       0.70      0.72      0.71       733
   positive       0.95      0.94      0.95      4058

avg / total       0.91      0.91      0.91      4791



[[ 527  224]
 [ 206 3834]]


TPR = 0.944800394282898
 TNR = 0.7189631650750341
 FPR = 0.2810368349249659
 FNR = 0.05519960571710202
number of nonzero components = 5743
sparsity = 0.7857649121498116
```

```
****************************************************************************************************

results for 3.5 times best lambda
             precision    recall  f1-score   support

   negative       0.70      0.72      0.71       733
   positive       0.95      0.95      0.95      4058

avg / total       0.91      0.91      0.91      4791



[[ 529  223]
 [ 204 3835]]


TPR = 0.9450468210941351
 TNR = 0.7216916780354706
 FPR = 0.2783083219645293
 FNR = 0.05495317890586496
number of nonzero components = 5711
sparsity = 0.7869586302085276

****************************************************************************************************

results for 4.0 times best lambda
             precision    recall  f1-score   support

   negative       0.70      0.72      0.71       733
   positive       0.95      0.95      0.95      4058

avg / total       0.91      0.91      0.91      4791



[[ 530  222]
 [ 203 3836]]


TPR = 0.9452932479053721
 TNR = 0.723055934515689
 FPR = 0.27694406548431105
 FNR = 0.05470675209462789
number of nonzero components = 5624
sparsity = 0.7902040511806617

****************************************************************************************************
```

```
results for 4.5 times best lambda
             precision    recall  f1-score   support

   negative       0.70      0.72      0.71       733
   positive       0.95      0.94      0.95      4058

avg / total       0.91      0.91      0.91      4791



[[ 528  225]
 [ 205 3833]]


TPR = 0.9445539674716609
 TNR = 0.7203274215552524
 FPR = 0.27967257844474763
 FNR = 0.05544603252833908
number of nonzero components = 5555
sparsity = 0.7927780057447682

**************************************************************************************************
```

0.4 Remarks

With increase in lambda (decrease in C) number of nonzero components decrease however there is no appreciable change in performance

0.5 Check for multicollinearity

```
In [0]: noise = np.reshape(np.random.normal(loc=0, scale=0.01, size=X_train_scaled.get_shape()[0
```

```
In [0]: #X_noisy = X_train_scaled.todense() + noise

        model = LogisticRegression(C=0.0168)
        model.fit(X_train_scaled, y_train_resampled)

        w_non_noisy = model.coef_

        #model.fit(X_noisy, y_train_resampled)
        #w_noisy = model.coef_
```

```
In [0]: diff = np.linalg.norm(w_non_noisy-w_noisy)
```

```
In [26]: np.linalg.norm(w_non_noisy)
```

```
Out[26]: 4.6978174895120945
```

In [28]: np.linalg.norm(w_noisy)

Out[28]: 4.698050250116066

In [29]: diff

Out[29]: 0.04061884783793817

0.6 Remarks

Since difference vector has very small magnitude as compared to w_non_noisy we can conclude, there is very low multicollinearity between features ## Feature Importance

```
In [0]: important = np.abs(w_non_noisy[0]).argsort()[:100]
```

```
In [0]: # top 100 important feature names
        imp = np.array(vectorizer.get_feature_names())[important]
```

```
In [82]: imp
```

```
Out[82]: array(['standard poodl', 'easi grow', 'love brand', 'coffe creamer',
                'mash', 'easi find', 'not total', 'ziplock bag', 'pulp',
                'texa bbq', 'pleas use', 'supplier', 'experi', 'also moder',
                'also keep', 'brew tea', 'like ive', 'sweeter', 'chip product',
                'packag one', 'not requir', 'loos stool', 'premium brand', 'bounc',
                'food help', 'west coast', 'nice tast', 'eat howev', 'bait',
                'vitamin supplement', 'msg artifici', 'tri hook', 'drastic',
                'keebler', 'boil water', 'sever varieti', 'delici healthi',
                'bag make', 'amazon quick', 'week ago', 'pet', 'new formula',
                'enjoy best', 'not firm', 'bun', 'serv packag', 'amount water',
                'cost bit', 'enjoy coffe', 'price definit', 'treat pet',
                'recommend item', 'chihuahua', 'pack make', 'know much',
                'like mani', 'littl honey', 'good old', 'order futur', 'far far',
                'husband absolut', 'didnt mind', 'seed butter', 'began search',
                'oili', 'know well', 'good packag', 'anyth ive', 'urin', 'bush',
                'price make', 'tuna', 'minti', 'live room', 'use type', 'versus',
                'product tri', 'almost not', 'seattl', 'mail', 'break apart',
                'line organ', 'think get', 'formul', 'older one', 'good like',
                'food yet', 'ect', 'first tast', 'bacon', 'health issu', 'includ',
                'carri happi', 'sure expect', 'twine english', 'cooki best',
                'also say', 'unhealthi', 'snack bag', 'use need'], dtype='<U22')
```