bonus

May 12, 2017

1 Part 4 -- Beat the Benchmark (bonus)

The libraries that we are going to use:

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt1
        import matplotlib.pyplot as plt2
        import csv
        import random
        import math
        import operator
        from operator import itemgetter
        from collections import Counter
        from wordcloud import STOPWORDS
        from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
        from sklearn.naive_bayes import MultinomialNB, BernoulliNB
        from sklearn.feature_extraction.text import TfidfTransformer
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn import svm
        from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
        from sklearn.metrics import classification_report, accuracy_score, auc
        from sklearn.model_selection import KFold
        from sklearn.decomposition import TruncatedSVD
        from sklearn.linear_model import SGDClassifier
        from sklearn import metrics
        from sklearn import tree
```

1.1 Setting up

We load our data, we create our assistant-structures and we define our stopwords, our vectorizer *(count vectorizer)* and our category criterion, the *title*:

```
In [2]: # load our data
        test_data = pd.read_csv('test_set.csv', sep='\t')
        train_data = pd.read_csv('train_set.csv', sep='\t')

        # a list of our categories (taken as facts)
```

```
categories = ['Politics','Football','Business','Technology','Film']

# we will use a number to represent each of our categories
category_dict = {'Politics':0, 'Football':1, 'Business':2, 'Technology':3, 'Film':4}

# for our text data, we use a count vectorizer
stopwords = set(STOPWORDS) | set(ENGLISH_STOP_WORDS)
# some additional stopwords based on our own observations
stopwords.add('said')
stopwords.add('say')
stopwords.add('says')
stopwords.add('set')

# our count vectorizer
count_vect = CountVectorizer(stop_words=stopwords)

# we will classify using the 'Title' as a criterion
category_criterion = 'Title'
```

1.2 Data Preprocessing

We preprocess our training and testing data. We then create a *'target'* array where we will note the category of each of our training data and we print a small part of it:

```
In [3]: # DATA PREPROCESSING
        # for training
        X_train_counts = count_vect.fit_transform(train_data[category_criterion])
        tfidf_transformer = TfidfTransformer(use_idf=False).fit(X_train_counts)
        X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
        print(X_train_counts.shape)
        print(X_train_counts.shape)

        # for testing
        X_test_counts = count_vect.transform(test_data[category_criterion])
        X_test_tfidf = tfidf_transformer.transform(X_test_counts)
        print(X_test_counts.shape)
        print(X_test_tfidf.shape)

        # we create a 'target' array where we will note the category of each of our training dat
        target = []
        for x in train_data['Category']:
            target.append(category_dict[x])

        target = np.array(target)
        print("target[] sample:")
        print(target[:40])
```

```
(12266, 13712)
(12266, 13712)
```

```
(3067, 13712)
(3067, 13712)
target[] sample:
[2 2 2 1 1 2 0 1 2 4 2 4 4 4 2 4 0 2 0 0 1 3 0 2 4 1 0 4 2 3 1 0 0 2 1 3 2
 0 3 3]
```

1.3 We will use the Decision Tree Classifier to Beat the Benchmark

By experimenting with various classifiers in Part 3 we ended up choosing the *Random Forest Classifier* as we saw that it produces better evaluation metrics than all the others. For preproccesing we used the *Count Vectorizer* excluding the stopwords that we have setted since Part 1 of this project, we also used the *TfidfTransformer (term-frequency times inverse document-frequency transformer)* because we observed that increases the accuracy of our classifier more than anything else that we tried. We also observed that *Decision Tree Classifier* under the conditions described above can Beat the Random Forest Classifier, which can be proved by the following results.

```
In [4]: # RANDOM FOREST (RF) CLASSIFIER
        RANDOM_STATE = 123

        rndf = RandomForestClassifier(warm_start=True, oob_score=True, max_features="sqrt", rand
        rndf.set_params(n_estimators=30)
        rndf.fit(X_train_tfidf, target)

        predicted = rndf.predict(X_test_tfidf)
```

We experiment with the Latent Semantic Indexing (LSI) for various number of components:

```
In [5]: print("Latent Semantic Indexing (LSI) for various number of components: ")

        accuracy = []
        components = []
        for i in range(6):
            components.append(i*100+100)
            print("For ", components[i], " components:")
            rndf = RandomForestClassifier(warm_start=True, oob_score=True, max_features="sqrt",

            svd = TruncatedSVD(n_components=components[i])
            X_lsi = svd.fit_transform(X_train_tfidf)
            clfSVD = tree.DecisionTreeClassifier().fit(X_lsi, target)
            X_test_lsi = svd.transform(X_train_counts)
            predictedSVD = clfSVD.predict(X_test_lsi)
            print("Accuracy:")
            acc = accuracy_score(target, predictedSVD)
            accuracy.append(acc)
            print("acc == ",acc)
            print(accuracy[i])
```

```
Latent Semantic Indexing (LSI) for various number of components:
For  100  components:
Accuracy:
acc ==  0.506766672102
0.506766672102
For  200  components:
Accuracy:
acc ==  0.481656611772
0.481656611772
For  300  components:
Accuracy:
acc ==  0.437795532366
0.437795532366
For  400  components:
Accuracy:
acc ==  0.413908364585
0.413908364585
For  500  components:
Accuracy:
acc ==  0.412359367357
0.412359367357
For  600  components:
Accuracy:
acc ==  0.404858959726
0.404858959726
```

In [6]:
```
X_lsi
```

Out[6]:
```
array([[ 0.00115585,  0.00079303,  0.00109195, ..., -0.01235207,
        -0.01853712,  0.02652396],
       [ 0.0029537 ,  0.0036768 ,  0.01050912, ..., -0.03479176,
        -0.01400699, -0.01348819],
       [ 0.00362246,  0.0031201 ,  0.00949418, ...,  0.00758439,
        -0.00697445, -0.01399747],
       ...,
       [ 0.00215124,  0.00457309,  0.02670653, ..., -0.00714358,
         0.00670417, -0.01355189],
       [ 0.01173553,  0.02352226,  0.11453489, ..., -0.01125241,
         0.01209246,  0.0085153 ],
       [ 0.00057599,  0.00324241,  0.00214254, ..., -0.01677111,
        -0.01079069,  0.01509295]])
```

In [7]:
```
# DECISION TREE (DT) CLASSIFIER
dt_clf = tree.DecisionTreeClassifier()
dt_clf.fit(X_lsi, target)
```

Out[7]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
```

```
                     min_impurity_split=1e-07, min_samples_leaf=1,
                     min_samples_split=2, min_weight_fraction_leaf=0.0,
                     presort=False, random_state=None, splitter='best')
```

Our Cross Validation function:

```
In [8]: cross_val_instance = 0

        def cross_validate(clf):
            global cross_val_instance    # Needed to modify global copy of a global variable

            kf = KFold(n_splits=10)

            fold = 0
            for train_index, test_index in kf.split(train_data[category_criterion]):
                cross_val_instance += 1

                X_train_counts = count_vect.transform(train_data[category_criterion][train_index
                X_test_counts = count_vect.transform(train_data[category_criterion][test_index].

                tfidf_transformer = TfidfTransformer(use_idf=False)
                X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)

                clf_cv = clf.fit(X_train_tfidf, target[train_index])
                X_test_tfidf = tfidf_transformer.fit_transform(X_test_counts)

                yPred = clf_cv.predict(X_test_tfidf)
                fold += 1
                print ("Fold " + str(fold))

                accuracy = accuracy_score(target[test_index], yPred)
                print("Accuracy: ", accuracy)

                A = auc(target[test_index], yPred, reorder=True)
                print("AUC: ", A)

                p = metrics.precision_score(target[test_index], yPred, average='macro')
                print("PRESICION: ", p)

                recall = metrics.recall_score(target[test_index], yPred, average='macro')
                print("Recall: ", recall)
                f_1 = metrics.f1_score(target[test_index], yPred, average='micro')
                print("F-1: ", f_1)

                fpr, tpr, thresholds = metrics.roc_curve(target[test_index], yPred, pos_label=2)
                roc_auc = metrics.auc(fpr, tpr)
                print("Roc: ",roc_auc)

In [9]: cross_validate(dt_clf)
```

```
Fold 1
Accuracy:  0.858190709046
AUC:  8.0
PRESICION:  0.856592702861
Recall:  0.847663782264
F-1:  0.858190709046
Roc:  0.55264319887
Fold 2
Accuracy:  0.823960880196
AUC:  8.5
PRESICION:  0.829131070605
Recall:  0.806900556423
F-1:  0.823960880196
Roc:  0.597483198464
Fold 3
Accuracy:  0.828035859821
AUC:  8.5
PRESICION:  0.82725245634
Recall:  0.813829808445
F-1:  0.828035859821
Roc:  0.594596106512
Fold 4
Accuracy:  0.826405867971
AUC:  8.0
PRESICION:  0.83221138996
Recall:  0.815289019143
F-1:  0.826405867971
Roc:  0.592304646251
Fold 5
Accuracy:  0.831295843521
AUC:  8.0
PRESICION:  0.827841859791
Recall:  0.817717481729
F-1:  0.831295843521
Roc:  0.581341454216
Fold 6
Accuracy:  0.820700896496
AUC:  8.0
PRESICION:  0.826525126451
Recall:  0.807508133007
F-1:  0.820700896496
Roc:  0.583420411007
Fold 7
Accuracy:  0.825448613377
AUC:  8.0
PRESICION:  0.828891598213
Recall:  0.816100463636
F-1:  0.825448613377
```

```
Roc:  0.580352617909
Fold 8
Accuracy:  0.827895595432
AUC:  8.0
PRESICION:  0.830417258674
Recall:  0.818428013079
F-1:  0.827895595432
Roc:  0.597696730211
Fold 9
Accuracy:  0.826264274062
AUC:  8.0
PRESICION:  0.834414063588
Recall:  0.816872366609
F-1:  0.826264274062
Roc:  0.582634547366
Fold 10
Accuracy:  0.831973898858
AUC:  8.0
PRESICION:  0.831638470106
Recall:  0.814496698889
F-1:  0.831973898858
Roc:  0.597990605428
```