Sentiment Analysis using SVM

In [ ]:

```
"""Author: Siddhant Shrivastava
<siddhant.shrivastava23@gmail.com>\nSentiment Analysis using word2vec, Naiv
e Bayes, SVM. \
NLP Project; Siddhant Shrivastava, Aditya Srivastava, Pranav Nair; Monsoon
2017, IIIT-Hyderabad"""
print(__doc__)
```

Reference: ***Sentiment analysis on Twitter using word2vec and keras by Ahmed Besbes***

Import Modules

In [ ]:

```
import pandas as pd
pd.options.mode.chained_assignment = None
import numpy as np
from copy import deepcopy
from string import punctuation
from random import shuffle

import gensim
from gensim.models.word2vec import Word2Vec # the word2vec model gensim cla
ss
LabeledSentence = gensim.models.doc2vec.LabeledSentence

from tqdm import tqdm
tqdm.pandas(desc="progress-bar")

from nltk.tokenize import TweetTokenizer
tokenizer = TweetTokenizer()

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
```

Define Functions

Function to load the dataset and extract the columns we need

In [ ]:

```
def ingest():
    """Load dataset, extract the sentiment and tweet's text columns"""
    data = pd.read_csv('../../dataset/cnn_dataset/tweets.csv', encoding="IS
O-8859-1")
    # data.drop(['ItemID', 'Date', 'Blank', 'SentimentSource'], axis=1, inp
lace=True)
```

```
    data.drop(['ItemID', 'SentimentSource'], axis=1, inplace=True)
    data = data[data.Sentiment.isnull() == False]
    data['Sentiment'] = data['Sentiment'].map(int)
    data = data[data['SentimentText'].isnull() == False]
    data.reset_index(inplace=True)
    data.drop('index', axis=1, inplace=True)
    data['Sentiment'] = data['Sentiment'].map({4:1, 0:0})
    print('dataset loaded with shape: ' + str(data.shape))
    return data
```

Tokenizing function

Splits each tweet into tokens and removes user mentions, hashtags and urls as they do not provide enough semantic information for the task

In [ ]:

```
def tokenize(tweet):
    try:
        tweet = tweet.lower()
        tokens = tokenizer.tokenize(tweet)
        return tokens
    except:
        return 'NC'
```

Process tokenized data

Tokenization results should now be cleaned to remove lines with 'NC', resulting from a tokenization error

In [ ]:

```
def postprocess(data, n=1000000):
    data = data.head(n)
    data['tokens'] = data['SentimentText'].progress_map(tokenize)  ## 
progress_map is a variant of the map function plus a progress bar. Handy 
to monitor DataFrame creations.
    print("Tokenization done")
    print(data.head(5))
    # print(data.tokens.value_counts())
    data = data[data.tokens != 'NC']
    data.reset_index(inplace=True)
    data.drop('index', inplace=True, axis=1)
    return data
```

Function to turn tokens to LabeledSentence objects before feeding to the word2vec model

In [ ]:

```
def labelizeTweets(tweets, label_type):
    labelized = []
    for i,v in tqdm(enumerate(tweets)):
        label = '%s_%s'%(label_type,i)
        labelized.append(LabeledSentence(v, [label]))
```

```
    return labelized
```

Function to create averaged tweet vector

In [ ]:

```
def buildWordVector(tokens, size):
    """Given a list of tweet tokens, creates an averaged tweet vector."""
    vec = np.zeros(size).reshape((1, size))
    count = 0.
    for word in tokens:
        try:
            vec += tweet_w2v[word].reshape((1, size)) * tfidf[word]
            count += 1.
        except KeyError: # handling the case where the token is not
                         # in the corpus. useful for testing.
            continue
    if count != 0:
        vec /= count
    return vec
```

Load and Process Data

In [ ]:

```
data = ingest()
data.head(5)
```

In [ ]:

```
data.Sentiment.value_counts()
# {'0': "negative sentiment", '1': "positive sentiment"}
```

Tokenize and clean data

In [ ]:

```
data = postprocess(data)
```

We are considering 1,000,000 (1 million) records.

In [ ]:

```
data.shape
```

In [ ]:

```
n = 1000000
```

Build the word2vec model

Define the training and test dataset

In [ ]:

```
x_train, x_test, y_train, y_test = train_test_split(np.array(data.head(n).t
okens),
                                                    np.array(data.head(n).Se
timent), test_size=0.2)
```

Turn tokens into LabeledSentence Object

Before feeding lists of tokens into the word2vec model, we must turn them into LabeledSentence objects beforehand.

In [ ]:

```
x_train.shape
```

In [ ]:

```
x_train = labelizeTweets(x_train, 'TRAIN')
x_test = labelizeTweets(x_test, 'TEST')
```

In [ ]:

```
print(x_train[0])
```

Build the word2vec model from x_train i.e. the corpus.

Set the number of dimensions of the vector space

In [ ]:

```
n_dim = 200
```

In [ ]:

```
tweet_w2v = Word2Vec(size=n_dim, min_count=10)
tweet_w2v.build_vocab([x.words for x in tqdm(x_train)])
tweet_w2v.train([x.words for x in tqdm(x_train)],total_examples=tweet_w2v.c
orpus_count, epochs=tweet_w2v.iter)
```

Check semantic realatioship set by word2vec

In [ ]:

```
tweet_w2v.most_similar('good')
```

Build the Sentiment Classifier

Build the tf-idf matrix

to compute the tf-idf score which is a weighted average where each weight gives the importance of the word with respect to the corpus.

In [ ]:

```
print('building tf-idf matrix ...')
vectorizer = TfidfVectorizer(analyzer=lambda x: x, min_df=10)
matrix = vectorizer.fit_transform([x.words for x in x_train])
tfidf = dict(zip(vectorizer.get_feature_names(), vectorizer.idf_))
print('vocab size : %s' % (len(tfidf)))
```

Convert x_train and x_test to a list of vectors

Also scale each column to have zero mean and unit standard deviation.

In [ ]:

```
from sklearn.preprocessing import scale
train_vecs_w2v = np.concatenate([buildWordVector(z, n_dim) for z in tqdm(ma
p(lambda x: x.words, x_train))])
train_vecs_w2v = scale(train_vecs_w2v)

test_vecs_w2v = np.concatenate([buildWordVector(z, n_dim) for z in tqdm(map
(lambda x: x.words, x_test))])
test_vecs_w2v = scale(test_vecs_w2v)
```

In [ ]:

```
type(train_vecs_w2v)
print(train_vecs_w2v)
print(y_train)
type(y_train)
```

Prepare models

In [ ]:

```
from sklearn.svm import SVC
```

Create Classifier Objects

In [ ]:

```
svClf = SVC(random_state=7)
```

Evaluate Model Performance Metrics

References

- Scikit-learn Model Selection documentation

In [ ]:

```
from sklearn.model_selection import cross_val_score, StratifiedKFold
```

```
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from scipy import interp
from itertools import cycle
import time
```

In [ ]:

```
def cvDictGen(functions, X_train=train_vecs_w2v, y_train=y_train, cv=5, ver
bose=1):
    """Given (a) classifier(s) and training dataset, returns a dictionary c
ontaining\
    cross-validation scores for each classifier passed"""
    cvDict = {}
    for func in functions:
        cvScore = cross_val_score(func, X_train, y_train, cv=cv,
verbose=verbose)
        cvDict[str(func).split('(')[0]] = [cvScore.mean(), cvScore.std()]

    return cvDict
```

In [ ]:

```
def scoreDictGen(functions, data=test_vecs_w2v, target=y_test):
    """Given (a) classifier(s) and test dataset, returns a dictionary conta
ining\
     mean accuracy scores"""
    scoreDict = {}
    for func in functions:
        score = func.score(data, target)
        scoreDict[str(func).split('(')[0]] = score

    return scoreDict
```

Train: Learn to predict each class against the other

In [ ]:

```
svClf.fit(train_vecs_w2v, y_train)
```

Score (mean accuracy) of the trained classifiers

In [ ]:

```
print(scoreDictGen(functions=[svClf]))
```

In [ ]:

```
print(cvDictGen(functions=[svClf]))
```

SVC accuracy = ~80.35%

Hyperparameter Optimization

In [ ]:

```
from sklearn.model_selection import RandomizedSearchCV
```

Support Vector Classifier

In [ ]:

```
tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                     'C': [1, 10, 100, 1000]},
                    {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]

scores = ['precision', 'recall']
```

In [ ]:

```
gridSearchsvClf = GridSearchCV(estimator=svClf, param_distributions=tuned_p
arameters, n_iter=10, cv=10,
                       scoring='%s_macro' % score).fit(train_vecs_w2v, y_tra
in)
```

In [ ]:

```
print("Best SupportVectorClassifier\n\tBest parameters: %s\n\tBest score: %
s" %(gridSearchsvClf.best_params_,

gridSearchsvClf.best_score_))
```

Confusion Matrix

In [ ]:

```
%matplotlib inline
```

In [ ]:

```
import itertools
from sklearn.metrics import confusion_matrix, classification_report
```

In [ ]:

```
class_names = np.array(['0', '1'])
```

In [ ]:

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
```

```
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [ ]:

```
def compute_drive_confusion_plot(y_pred, y_test=y_test):
    # Compute confusion matrix
    cnf_matrix = confusion_matrix(y_test, y_pred)
    np.set_printoptions(precision=2)

    # Plot non-normalized confusion matrix
    plt.figure()
    plot_confusion_matrix(cnf_matrix, classes=class_names,
                          title='Confusion matrix, without normalization')

    # Plot normalized confusion matrix
    plt.figure()
    plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                          title='Normalized confusion matrix')
    plt.show()
```

Support Vector Classifier

In [ ]:

```
compute_drive_confusion_plot(gridSearchsvClf.predict(test_vecs_w2v))
```

Model Persistence

In [ ]:

```
from sklearn.externals import joblib
```

In [ ]:

```
joblib.dump(gridSearchsvClf, 'sentimentAnalysisSVClf.pkl')
```

Later you can load back the pickled model (possibly in another Python process) with

In [ ]:

```
# clf = joblib.load('sentimentAnalysisNBClf.pkl')
```