Data 641 - Lab 3

Homayoon Fotros

Exercise 1

Importing Base Modules and Data

[1]:
```
import numpy as np
import pandas as pd
import re
import string
```

[2]:
```
dt_twt = pd.read_csv('COVID19_Dataset-text_labels_only.csv')
```

Exercise 2

Extracting and Normalizing Text

[3]:
```
## Removing Hashtags

dt_twt['No Hash'] = [re.sub('#', '', line) for line in dt_twt['Tweet']] ##␣
 ↪Removing pound sign

## Example:
print('Original Tweet:\n', dt_twt['Tweet'].iloc[100])
print('\nTweet without hashtag:\n', dt_twt['No Hash'].iloc[100])
```

```
Original Tweet:
 #coronavirus #virus Reasons to buy back stocks today 1500 new cases 60 deaths
2700 recovered #markets #portfolio

Tweet without hashtag:
 coronavirus virus Reasons to buy back stocks today 1500 new cases 60 deaths
2700 recovered markets portfolio
```

[4]:
```
## Dropping words with less than 2 characters

from nltk import word_tokenize

twt_longs = []

for tweet in dt_twt['No Hash']:
```

```
    twt_words = word_tokenize(tweet)
    twt_n = ' '.join([word for word in twt_words if re.match('\w{2,}|\d+',word)])
    twt_longs.append(twt_n)

dt_twt['No Short Word'] = twt_longs
```

[5]:
```
dt_twt['No Short Word'].iloc[326]
```

[5]:
```
'5 Rings Podcast Road To Tokyo March 10th 2020 COVID-19 and the Possibility of
an Olympic Cancellation 24thminute KevLaramee coronavirus COVID19 Tokyo2020 IOC
OlympicPodcast'
```

[6]:
```
## Lemmatizing

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

tw_temp=[]

for tweet in dt_twt['No Short Word']:
    twt_lm = [lemmatizer.lemmatize(word) for word in word_tokenize(tweet)]
    tw_temp.append(' '.join(twt_lm))

dt_twt['Lemmatized'] = tw_temp
```

[8]:
```
(dt_twt['Lemmatized'].iloc[444])
```

[8]:
```
'State of emergency declared in New York a number of coronavirus patient climb'
```

Exercise 3

Instantiating Vectorizer (tf-idf, PCA, Sparse to Dense)

[10]:
```
## Instantiating tf-idf

from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(lowercase = True, stop_words = 'english', ngram_range =␣
 ↪(1,1))
```

[11]:
```
## Instantiating PCA

from sklearn.decomposition import PCA

pca = PCA()

## Setting the range of components to try
```

```
ncomps = [5, 10, 20, 50, 75, 100]
```

[12]:
```
## Instantiating Dense Transformer

from sklearn.base import TransformerMixin

class SparseToDense(TransformerMixin):
    def fit(self, X, y = None, **fit_params):
        return self

    def transform(self, X, y = None, **fit_params):
        return X.toarray()
```

Exercise 4

Setting up ML Process (Transformation, Pipeline, SVM hyperparams, CV, etc.)

[13]:
```
## Setting up data set

X = dt_twt['Lemmatized']
y = dt_twt['Is_Unreliable']

X_count = tfidf.fit_transform(X)
```

[14]:
```
# Setting up the pipeline

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

pipe = Pipeline([
    ('vectorize', tfidf),
    ('densify', SparseToDense()),
    ('scale', StandardScaler()),
    ('dim_red', pca),
    ('classify', SVC())
])
```

[15]:
```
# SVC hyperparameters

kernel = ['rbf', 'linear', 'poly', 'sigmoid'] ## kernels

C = [0.001, 0.01, 0.1, 1, 10]

# parameters' grid
params = {
    'dim_red__n_components': ncomps,
```

```
    'classify__kernel': kernel,
    'classify__C': C
}
```

[16]:
```
# Setting Cross Validation scheme for inner and outer loops

from sklearn.model_selection import cross_validate, KFold, GridSearchCV

inner_cv = KFold(n_splits = 3, shuffle = True, random_state = 1)
outer_cv = KFold(n_splits = 5, shuffle = True, random_state = 1)

# Setting GridSearch for inner loop

grid_SVC = GridSearchCV(pipe, params, cv = inner_cv)

# Nested CV scores

scores = cross_validate(grid_SVC, X = X, y = y, cv = outer_cv,
                        scoring = ['roc_auc', 'accuracy', 'f1',␣
 ↪'precision','recall'],
                        return_estimator = True)

print ('Done!')
```

[17]:
```
## Model Performance: AUC, Accuracy, F1

auc = scores['test_roc_auc']
accuracy = scores['test_accuracy']
f1 = scores['test_f1']
precision = scores['test_precision']
recall = scores['test_recall']
estimators = scores['estimator']
```

[24]:
```
## Average scores for models

print('Average Scores:\n')
print('Accuracy: {} \nPrecision: {}\nRecall: {}\nF1: {}'.format(
    accuracy.mean().round(4), precision.mean().round(4), recall.mean().round(4),␣
 ↪f1.mean().round(4)))
```

```
Average Scores:

Accuracy: 0.7696
Precision: 0.7771
Recall: 0.76
F1: 0.7661
```

[19]:
```
## Best Performance

for score in estimators:
    print(score.best_params_)
    print('\n')
```

```
{'classify__C': 0.01, 'classify__kernel': 'linear', 'dim_red__n_components':
100}


{'classify__C': 0.01, 'classify__kernel': 'linear', 'dim_red__n_components':
100}


{'classify__C': 0.01, 'classify__kernel': 'linear', 'dim_red__n_components':
100}


{'classify__C': 1, 'classify__kernel': 'sigmoid', 'dim_red__n_components': 50}


{'classify__C': 1, 'classify__kernel': 'linear', 'dim_red__n_components': 50}
```