Binary classification of COVID-19 tweets using tf-idf vectorization & PCA

Import modules & data

1. `Import modules and data`

In [1]:
```
import pandas as pd
import numpy as np
import re
from nltk import word_tokenize, pos_tag
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.decomposition import NMF, PCA, TruncatedSVD, FastICA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_validate, KFold, GridSearchCV
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, roc_auc_score

import nltk
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to C:\Users\Sunday
[nltk_data]     Okechukwu\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[1]:
```
True
```

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

Out[2]:

	Is_Unreliable	Category	Tweet
0	1	1, 3, 6, 9	We are living in scary times in Canada. Gov't ...
1	1	1, 6, 8, 9	Just as bad in Canada. In fact, our government...
2	1	1, 4, 9	It was only a matter of time before the mainst...
3	1	6, 8	Russia's taking no chances: Foreigners infecte...
4	1	6, 8, 9	Although there is now a presumptive confirmed ...
...	...	...	...
555	0	NaN	BREAKING: Harvard classes will move online sta...
556	0	NaN	Singularity University is hosting a FREE Virtu...
557	0	NaN	Coronavirus: how does it spread and what are t...
558	0	NaN	Stanford just cancelled classes for the rest o...
559	0	NaN	Tech conferences were cancelled in #Waterloo R...

560 rows × 3 columns

2. Clean and normalize text

In [3]:
```
def clean_text(str_list, lemmatize=True):
    clean_list = []
    for text in str_list:
        text = re.sub(r'#', '', text)
        words = word_tokenize(text)

        clean_words = []
        for word in words:
            if(len(word) > 1) and (re.match(r'^\w+$', word)):
                if lemmatize:
                    lemmatizer = WordNetLemmatizer()
                    word1 = lemmatizer.lemmatize(word)
                clean_words.append(word1)
        clean_text = ' '.join(clean_words)
        clean_list.append(clean_text)
    return clean_list
```

In [4]:
```
tweets['clean_tweet'] = clean_text(tweets['Tweet'], lemmatize=True)
tweets
```

Out[4]:

	Is_Unreliable	Category	Tweet	clean_tweet
0	1	1, 3, 6, 9	We are living in scary times in Canada. Gov't ...	We are living in scary time in Canada Gov refu...
1	1	1, 6, 8, 9	Just as bad in Canada. In fact, our government...	Just a bad in Canada In fact our government is...
2	1	1, 4, 9	It was only a matter of time before the mainst...	It wa only matter of time before the mainstrea...
3	1	6, 8	Russia's taking no chances: Foreigners infecte...	Russia taking no chance Foreigners infected wi...
4	1	6, 8, 9	Although there is now a presumptive confirmed ...	Although there is now presumptive confirmed ca...
...	...	...	...	...
555	0	NaN	BREAKING: Harvard classes will move online sta...	BREAKING Harvard class will move online starti...
556	0	NaN	Singularity University is hosting a FREE Virtu...	Singularity University is hosting FREE Virtual...
557	0	NaN	Coronavirus: how does it spread and what are t...	Coronavirus how doe it spread and what are the...
558	0	NaN	Stanford just cancelled classes for the rest o...	Stanford just cancelled class for the rest of ...
559	0	NaN	Tech conferences were cancelled in #Waterloo R...	Tech conference were cancelled in Waterloo Reg...

560 rows × 4 columns

3. Instantiate vectorizers and topic models.

For this part use PCA to reduce your feature dimensions. Remember to create a sparse-to-dense transformer.

In [5]:
```
# Instantiate vectorizer
```

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

In [6]:
```
pca = PCA()
ncomps = [50, 75, 100]
```

In [7]:
```
# Create sparse-to-dense transformer
from sklearn.base import TransformerMixin
```

In [8]:
```
class SparseToDense(TransformerMixin):

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

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

In [9]:
```
# pull out relevant data
X = tweets['clean_tweet']
y = tweets['Is_Unreliable']

X_count = tfidf.fit_transform(X)
```

In [10]:
```
X_count.shape
```

Out[10]:
```
(560, 2331)
```

In [11]:
```
y
```

Out[11]:
```
0      1
1      1
2      1
3      1
4      1
      ..
555    0
556    0
557    0
558    0
559    0
Name: Is_Unreliable, Length: 560, dtype: int64
```

In [12]:
```
y.value_counts() # the data is balanced
```

Out[12]:
```
1    280
0    280
Name: Is_Unreliable, dtype: int64
```

4. Set up a cross validation scheme, optimize the hyperparameters of SVM, and print out the accuracy, precision, recall, and f1 score.

In [13]:
```
# create pipeline
pipe = Pipeline([
    ('vectorize', tfidf),
    ('densify', SparseToDense()),
    ('scale', StandardScaler()),
    ('dim_red', pca),
    ('classify', SVC())
])
```

In [14]:
```
# SVC hyperparams to optimize
```

```
kernel = ['rbf', 'linear', 'poly', 'sigmoid']
C = [0.001, 0.01, 0.1, 1, 10]
# set up parameter grid
params = {
    'dim_red__n_components': ncomps,
    'classify__kernel': kernel,
    'classify__C': C
}
```

In [15]:
```
# Set CV scheme for inner and outer loops
inner_cv = KFold(n_splits = 3, shuffle = True, random_state = 1)
outer_cv = KFold(n_splits = 5, shuffle = True, random_state = 1)

# Set up GridSearch for inner loop
grid_SVC = GridSearchCV(pipe, params, cv = inner_cv)
#grid_SVC.fit(X, y)

# 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)
```

In [16]:
```
# Extract the results and store them in a dataframe
results = pd.DataFrame({'auc': scores['test_roc_auc'],
                        'accuracy': scores['test_accuracy'],
                        'f1': scores['test_f1'],
                        'precision': scores['test_precision'],
                        'recall': scores['test_recall']})

# Print the results in a dataframe format

results.loc[len(results.index)] = [scores['test_roc_auc'].mean(), scores['test_accuracy'
                                   scores['test_f1'].mean(), scores['test_precision'].me
                                   scores['test_recall'].mean()]

results = results.rename(index={5: 'Mean of each column'})

# Display the formatted last row
results
```

Out[16]:

	auc	accuracy	f1	precision	recall
0	0.768495	0.732143	0.732143	0.732143	0.732143
1	0.839193	0.714286	0.692308	0.642857	0.750000
2	0.798724	0.705357	0.702703	0.722222	0.684211
3	0.822436	0.741071	0.738739	0.803922	0.683333
4	0.760154	0.705357	0.713043	0.732143	0.694915
Mean of each column	0.797800	0.719643	0.715787	0.726657	0.708920

What are the parameters that provide the best classification performance?

In [17]:
```
grid_SVC.fit(X, y)

print(grid_SVC.best_params_)
```

```
{'classify__C': 1, 'classify__kernel': 'sigmoid', 'dim_red__n_components': 75}
```

In [18]:
```
estimators = scores['estimator']
```

In [19]:
```
print(estimators)
```

```
[GridSearchCV(cv=KFold(n_splits=3, random_state=1, shuffle=True),
              estimator=Pipeline(steps=[('vectorize',
                                         TfidfVectorizer(stop_words='english')),
                                        ('densify',
                                         <__main__.SparseToDense object at 0x0000017DBA6C
C2B0>),
                                        ('scale', StandardScaler()),
                                        ('dim_red', PCA()),
                                        ('classify', SVC())]),
              param_grid={'classify__C': [0.001, 0.01, 0.1, 1, 10],
                          'classify__kernel': ['rbf', 'linear', 'poly',
                                               'sigmoid'],
                          'dim_red__n_components': [50, 75, 100]}), GridSearchCV(cv=KFold
(n_splits=3, random_state=1, shuffle=True),
              estimator=Pipeline(steps=[('vectorize',
                                         TfidfVectorizer(stop_words='english')),
                                        ('densify',
                                         <__main__.SparseToDense object at 0x0000017DBA79
70D0>),
                                        ('scale', StandardScaler()),
                                        ('dim_red', PCA()),
                                        ('classify', SVC())]),
              param_grid={'classify__C': [0.001, 0.01, 0.1, 1, 10],
                          'classify__kernel': ['rbf', 'linear', 'poly',
                                               'sigmoid'],
                          'dim_red__n_components': [50, 75, 100]}), GridSearchCV(cv=KFold
(n_splits=3, random_state=1, shuffle=True),
              estimator=Pipeline(steps=[('vectorize',
                                         TfidfVectorizer(stop_words='english')),
                                        ('densify',
                                         <__main__.SparseToDense object at 0x0000017DBE79
3C40>),
                                        ('scale', StandardScaler()),
                                        ('dim_red', PCA()),
                                        ('classify', SVC())]),
              param_grid={'classify__C': [0.001, 0.01, 0.1, 1, 10],
                          'classify__kernel': ['rbf', 'linear', 'poly',
                                               'sigmoid'],
                          'dim_red__n_components': [50, 75, 100]}), GridSearchCV(cv=KFold
(n_splits=3, random_state=1, shuffle=True),
              estimator=Pipeline(steps=[('vectorize',
                                         TfidfVectorizer(stop_words='english')),
                                        ('densify',
                                         <__main__.SparseToDense object at 0x0000017DBE79
3E50>),
                                        ('scale', StandardScaler()),
                                        ('dim_red', PCA()),
                                        ('classify', SVC())]),
              param_grid={'classify__C': [0.001, 0.01, 0.1, 1, 10],
                          'classify__kernel': ['rbf', 'linear', 'poly',
                                               'sigmoid'],
                          'dim_red__n_components': [50, 75, 100]}), GridSearchCV(cv=KFold
(n_splits=3, random_state=1, shuffle=True),
              estimator=Pipeline(steps=[('vectorize',
                                         TfidfVectorizer(stop_words='english')),
                                        ('densify',
                                         <__main__.SparseToDense object at 0x0000017DBA79
74C0>),
                                        ('scale', StandardScaler()),
                                        ('dim_red', PCA()),
                                        ('classify', SVC())]),
```

```
                  param_grid={'classify__C': [0.001, 0.01, 0.1, 1, 10],
                              'classify__kernel': ['rbf', 'linear', 'poly',
                                                   'sigmoid'],
                              'dim_red__n_components': [50, 75, 100]})]
```

Using RandomizedSearchCV

In [20]:
```
from scipy.stats import uniform
from sklearn.model_selection import RandomizedSearchCV
```

In [21]:
```
# create pipeline
pipe = Pipeline([
    ('vectorize', tfidf),
    ('densify', SparseToDense()),
    ('scale', StandardScaler()),
    ('dim_red', pca),
    ('classify', SVC())
])

# SVC hyperparams to optimize
kernel = ['rbf', 'linear', 'poly', 'sigmoid']
C = uniform(0.001, 10) # use a continuous distribution for C
# set up parameter grid
param_distributions = {
    'dim_red__n_components': ncomps,
    'classify__kernel': kernel,
    'classify__C': C
}

# Set CV scheme for inner and outer loops
inner_cv = KFold(n_splits = 3, shuffle = True, random_state = 1)
outer_cv = KFold(n_splits = 5, shuffle = True, random_state = 1)

# Set up RandomizedSearch for inner loop
random_SVC = RandomizedSearchCV(pipe, param_distributions, n_iter=20, cv = inner_cv)
#random_SVC.fit(X, y)

# Nested CV scores
scores = cross_validate(random_SVC,
                        X = X,
                        y = y,
                        cv = outer_cv,
                        scoring = ['roc_auc', 'accuracy', 'f1', 'precision', 'recall'],
                        return_estimator = True)

# Extract the results and store them in a dataframe
results = pd.DataFrame({'auc': scores['test_roc_auc'],
                        'accuracy': scores['test_accuracy'],
                        'f1': scores['test_f1'],
                        'precision': scores['test_precision'],
                        'recall': scores['test_recall']})

# Print the results in a dataframe format

results.loc[len(results.index)] = [scores['test_roc_auc'].mean(), scores['test_accuracy'
                                   scores['test_f1'].mean(), scores['test_precision'].me
                                   scores['test_recall'].mean()]

results = results.rename(index={5: 'Mean of each column'})

# Display the formatted last row
results
```

Out[21]:

	auc	accuracy	f1	precision	recall

0	0.804528	0.758929	0.765217	0.745763	0.785714
1	0.831055	0.767857	0.745098	0.703704	0.791667
2	0.801276	0.687500	0.672897	0.720000	0.631579
3	0.853526	0.785714	0.789474	0.833333	0.750000
4	0.711545	0.687500	0.690265	0.722222	0.661017
Mean of each column	0.800386	0.737500	0.732590	0.745004	0.723995

In [22]:
```
random_SVC.fit(X, y)

print(random_SVC.best_params_)
```

```
{'classify__C': 0.5872685952748159, 'classify__kernel': 'sigmoid', 'dim_red__n_components': 100}
```