Fake news detection

1.

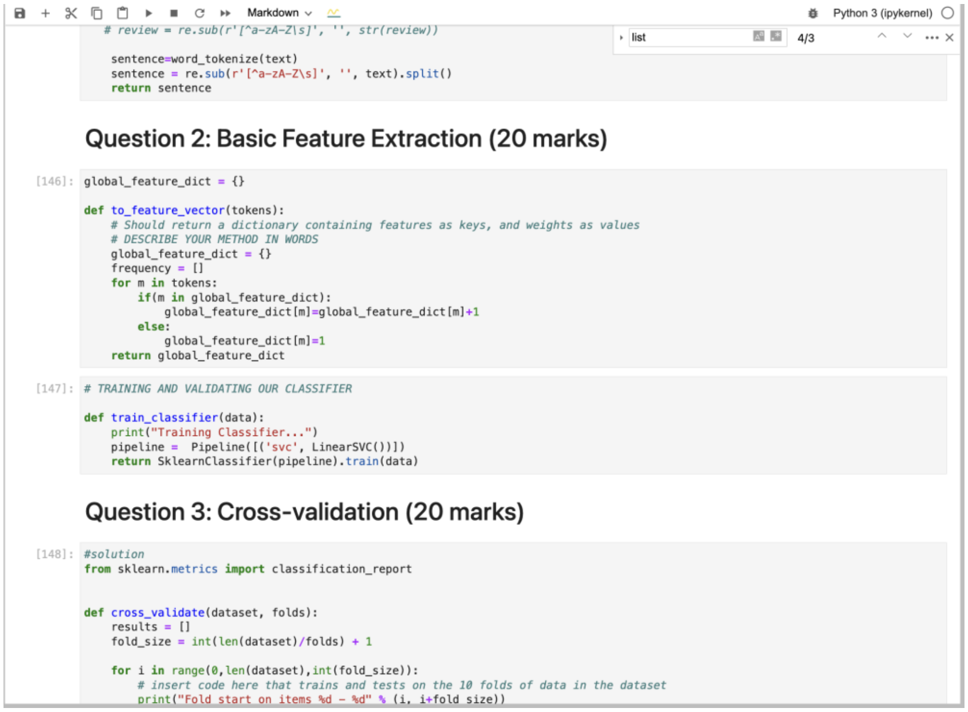
Question 1: Input and Basic preprocessing (10 marks)

```python
def convert_label(label):
    """Converts the multiple classes into two,
    making it a binary distinction between fake news and real."""
    #return label
    # Converting the multiclass labels to binary label
    labels_map = {
        'true': 'REAL',
        'mostly-true': 'REAL',
        'half-true': 'REAL',
        'false': 'FAKE',
        'barely-true': 'FAKE',
        'pants-fire': 'FAKE'
    }
    return labels_map[label]


def parse_data_line(data_line):
    # Should return a tuple of the label as just FAKE or REAL and the statement
    # e.g. (label, statement)
    data_line[1]=convert_label(data_line[1])
    label=data_line[1]
    text=data_line[2]
    data_input=(label,text)
    return tuple(data_input)
```

```python
# Input: a string of one statement
tokens = []
def pre_process(text):
    # Should return a list of tokens
    # DESCRIBE YOUR METHOD IN WORDS
   # review = re.sub(r'[^a-zA-Z\s]', '', str(review))

    sentence=word_tokenize(text)
    sentence = re.sub(r'[^a-zA-Z\s]', '', text).split()
    return sentence
```

1.

Now, we simplify the news afore relegating it as unauthentically spurious or not. We commence by abstracting special characters from the news-designation, then converting it to lower case. Then we tokenize it (split into words) and abstract stop words. Stop words are commonly used words in a language. Search engines ignore the cessation words while indexing data as well as retrieving results for search queries. Now, we simplify the news afore relegating it as fictitiously unauthentic or not. We commence by abstracting special characters from the news-denomination, then converting it to lower case. Then we tokenize it (split into words) and abstract stop words. Stop words are commonly used words in a language. Search engines ignore the cessation words while indexing data as well as retrieving results for search queries.




```
    # review = re.sub(r'[^a-zA-Z\s]', '', str(review))

    sentence=word_tokenize(text)
    sentence = re.sub(r'[^a-zA-Z\s]', '', text).split()
    return sentence
```

Question 2: Basic Feature Extraction (20 marks)

```
[146]: global_feature_dict = {}

def to_feature_vector(tokens):
    # Should return a dictionary containing features as keys, and weights as values
    # DESCRIBE YOUR METHOD IN WORDS
    global_feature_dict = {}
    frequency = []
    for m in tokens:
        if(m in global_feature_dict):
            global_feature_dict[m]=global_feature_dict[m]+1
        else:
            global_feature_dict[m]=1
    return global_feature_dict
```

```
[147]: # TRAINING AND VALIDATING OUR CLASSIFIER

def train_classifier(data):
    print("Training Classifier...")
    pipeline =  Pipeline([('svc', LinearSVC())])
    return SklearnClassifier(pipeline).train(data)
```

Question 3: Cross-validation (20 marks)

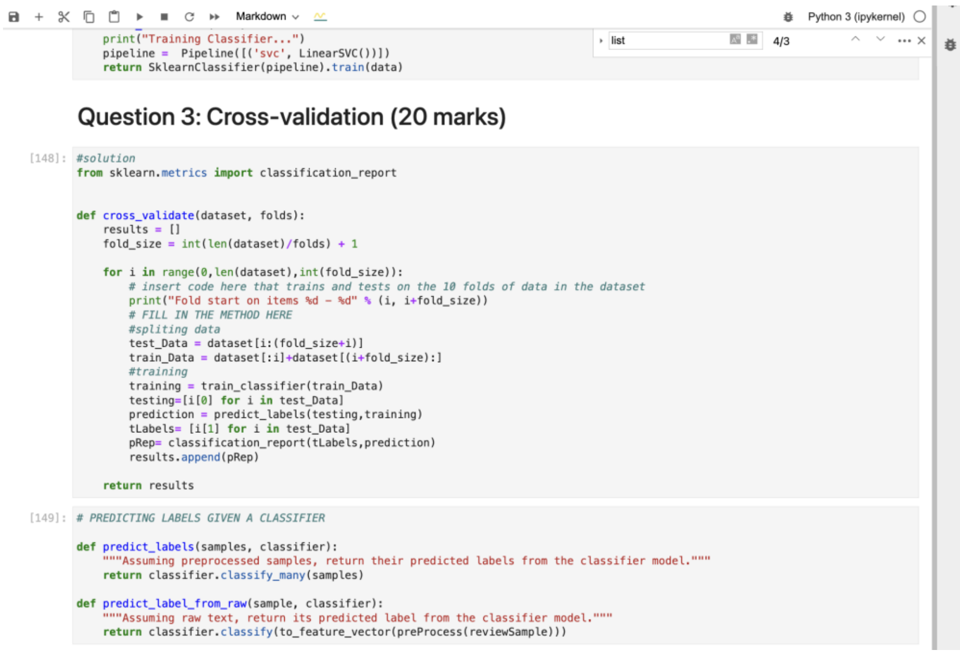
```
[148]: #solution
from sklearn.metrics import classification_report


def cross_validate(dataset, folds):
    results = []
    fold_size = int(len(dataset)/folds) + 1

    for i in range(0,len(dataset),int(fold_size)):
        # insert code here that trains and tests on the 10 folds of data in the dataset
        print("Fold start on items %d - %d" % (i, i+fold_size))
```

2.

Once the data is simplified by converting it to lower case, abstracting special characters, stemming and abstracting stop words, we build a Bag of Words model. This will represent the text as a bag (multiset) of words and keep a count of the words. It gives us features or most frequently used words within a range. The count vectorizer tokenizes an amassment of documents and builds a lexicon of unique words. It can withal encode incipient documents utilizing this lexicon. Using words of bag method we can easily extract features from the data which we need and put it into a bag of list which we are returning here.

Label 1 represents genuine news and 0 represents fake news.

3.



```
    print("Training Classifier...")
    pipeline =  Pipeline([('svc', LinearSVC())])
    return SklearnClassifier(pipeline).train(data)
```

Question 3: Cross-validation (20 marks)

```
[148]:
#solution
from sklearn.metrics import classification_report


def cross_validate(dataset, folds):
    results = []
    fold_size = int(len(dataset)/folds) + 1

    for i in range(0,len(dataset),int(fold_size)):
        # insert code here that trains and tests on the 10 folds of data in the dataset
        print("Fold start on items %d - %d" % (i, i+fold_size))
        # FILL IN THE METHOD HERE
        #spliting data
        test_Data = dataset[i:(fold_size+i)]
        train_Data = dataset[:i]+dataset[(i+fold_size):]
        #training
        training = train_classifier(train_Data)
        testing=[i[0] for i in test_Data]
        prediction = predict_labels(testing,training)
        tLabels= [i[1] for i in test_Data]
        pRep= classification_report(tLabels,prediction)
        results.append(pRep)

    return results
```

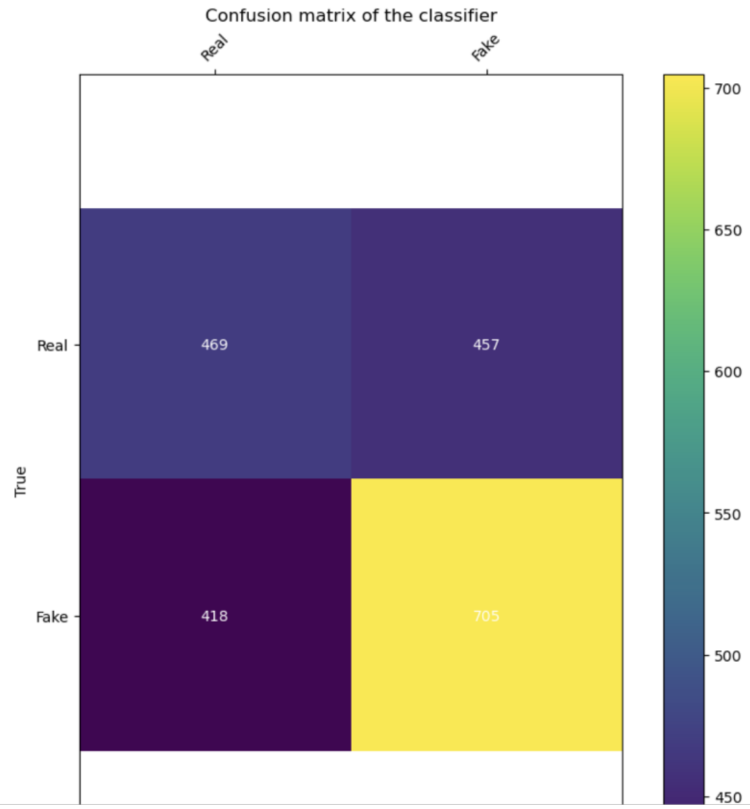
```
[149]:
# PREDICTING LABELS GIVEN A CLASSIFIER

def predict_labels(samples, classifier):
    """Assuming preprocessed samples, return their predicted labels from the classifier model."""
    return classifier.classify_many(samples)

def predict_label_from_raw(sample, classifier):
    """Assuming raw text, return its predicted label from the classifier model."""
    return classifier.classify(to_feature_vector(preProcess(reviewSample)))
```

Case folding describes the process of consolidating multiple spellings of a single word that differ only in capitalization. Now, let us move into the relegation models. First we require to split our data into training and testing data since it is a supervised learning model. This normalization technique is withal kenned as case normalization. Case folding is one way to truncate our lexicon size and sanction for better generalization of our NLP pipeline .We train the data utilizing the train set and test it utilizing the test set. 80% of the data has been utilized for training and the rest 20 % for testing for optimal results. Therefore in cross validation were are checking the accuracy of training and test data and compare.

4.

```
[153]: #classifier = PassiveAggressiveClassifier()
#classifier.fit(training,testing)
training = train_classifier(train_data)
testing=[i[0] for i in test_data]
prediction = predict_labels(testing,training)
tData=[i[1] for i in test_data]
confusion_matrix_heatmap(tData,prediction,["Real","Fake"])
```

```
Training Classifier...
```



After cross validation by using heatmap confusion matrix of real and fake I can see that its coming as 705 which is not so good therefore we can make changes on the model using different processes.

```
    text=data_line[2]
    dinput=(label,text)
    return tuple(dinput)
```

```
[7]: import re
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords
from collections import Counter

sp = PorterStemmer()
s_words = stopwords.words('english')
swords_dict = Counter(s_words)

def pre_process(text):
    #text = clean_text(text)
    t = re.sub(r'[^\w\s]', '', text).split()
    text = [sp.stem(word) for word in t if not word in swords_dict]
    return  text
```

5.

```
[8]: # Input: a string of one statement
tokens = []
def pre_process(text):
    # Should return a list of tokens
    # DESCRIBE YOUR METHOD IN WORDS
   # review = re.sub(r'[^a-zA-Z\s]', '', str(review))

    sentence=word_tokenize(text)
    sentence = re.sub(r'[^a-zA-Z\s]', '', text).split()
    return sentence
```

```
Training Classifier...
[20]: ['            precision    recall  f1-score   support\n\n        FAKE       0.55      0.32      0.41       339\n        REAL       0.63      0.81
0.71       481\n\n    accuracy                           0.61       820\n   macro avg       0.59      0.57      0.56       820\nweighted avg       0.60
0.61      0.58       820\n',
 '            precision    recall  f1-score   support\n\n        FAKE       0.61      0.38      0.47       375\n        REAL       0.60      0.80
0.69       445\n\n    accuracy                           0.60       820\n   macro avg       0.61      0.59      0.58       820\nweighted avg       0.61
0.60      0.59       820\n',
 '            precision    recall  f1-score   support\n\n        FAKE       0.62      0.33      0.43       392\n        REAL       0.57      0.81
0.67       428\n\n    accuracy                           0.58       820\n   macro avg       0.59      0.57      0.55       820\nweighted avg       0.59
0.58      0.56       820\n',
 '            precision    recall  f1-score   support\n\n        FAKE       0.59      0.39      0.47       357\n        REAL       0.63      0.79
0.70       463\n\n    accuracy                           0.62       820\n   macro avg       0.61      0.59      0.59       820\nweighted avg       0.61
0.62      0.60       820\n',
 '            precision    recall  f1-score   support\n\n        FAKE       0.57      0.34      0.42       348\n        REAL       0.62      0.81
0.70       472\n\n    accuracy                           0.61       820\n   macro avg       0.60      0.57      0.56       820\nweighted avg       0.60
0.61      0.59       820\n',
 '            precision    recall  f1-score   support\n\n        FAKE       0.56      0.34      0.42       349\n        REAL       0.62      0.80
0.70       471\n\n    accuracy                           0.60       820\n   macro avg       0.59      0.57      0.56       820\nweighted avg       0.59
0.60      0.58       820\n',
 '            precision    recall  f1-score   support\n\n        FAKE       0.52      0.33      0.40       349\n        REAL       0.61      0.77
0.68       471\n\n    accuracy                           0.59       820\n   macro avg       0.56      0.55      0.54       820\nweighted avg       0.57
0.59      0.56       820\n',
 '            precision    recall  f1-score   support\n\n        FAKE       0.60      0.37      0.46       357\n        REAL       0.63      0.81
0.70       463\n\n    accuracy                           0.62       820\n   macro avg       0.61      0.59      0.58       820\nweighted avg       0.61
0.62      0.60       820\n',
 '            precision    recall  f1-score   support\n\n        FAKE       0.54      0.35      0.43       363\n        REAL       0.60      0.76
0.67       457\n\n    accuracy                           0.58       820\n   macro avg       0.57      0.55      0.55       820\nweighted avg       0.57
0.58      0.56       820\n',
 '            precision    recall  f1-score   support\n\n        FAKE       0.54      0.36      0.43       333\n        REAL       0.64      0.79
0.70       479\n\n    accuracy                           0.61       812\n   macro avg       0.59      0.57      0.57       812\nweighted avg       0.60
0.61      0.59       812\n']
```

4. Error Analysis (10 marks)

We can see that the accuracy increased after using stemming on previous model.

As we saw the accuracy is coming upto arount 70 % and confusion matrix score was coming upto 705 therefore this can me increased in many ways like lemmatization, stemming etc, using lemmatization it didn't help me to increase my accuracy to my model therefor I used stemming, stemming data to remove data from stem of the phrase datas. This gives the root form of the word. Search engines ignore the cessation words indexing data as well as results for search queries. We withal need to stem the words utilizing PorterStemmer().

Question 1: Input and Basic preprocessing (10 marks)

```
[30]: def convert_label(label):
    """Converts the multiple classes into two,
    making it a binary distinction between fake news and real."""
    #return label
    # Converting the multiclass labels to binary label
    labels_map = {
        'true': 'REAL',
        'mostly-true': 'REAL',
        'half-true': 'REAL',
        'false': 'FAKE',
        'barely-true': 'FAKE',
        'pants-fire': 'FAKE'
    }
    return labels_map[label]


def parse_data_line(data_line):
    # Should return a tuple of the label as just FAKE or REAL and the statement
    # e.g. (label, statement)
    data_line[1]=convert_label(data_line[1])
    label=data_line[1]
    text=data_line[2]+data_line[3]+data_line[5]+data_line[6]+data_line[7]
    dinput=(label,text)
    return tuple(dinput)
```

```
[31]: import re
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords
from collections import Counter

sp = PorterStemmer()
s_words = stopwords.words('english')
swords_dict = Counter(s_words)

def pre_process(text):
    #text = clean_text(text)
    t = re.sub(r'[^\w\s]', '', text).split()
```

43 / 12.00 GB   Mode: Command   Ln 1, Col 1   English (United States)   question6 NLP Assignment1.ipynb

6.

As per question we are adding all the column's together to check the accuracy as hit and trial method in this case the accuracy didn't increased much and was as similar to the previous model.

```
Training classifier...
[38]: ['              precision    recall  f1-score   support\n\n        FAKE       0.53      0.33      0.41       339\n        REAL       0.63      0.79      0.70       481\n\n    accuracy                           0.60       820\n   macro avg       0.58      0.56      0.55       820\nweighted avg       0.59      0.60      0.58       820\n',
 '              precision    recall  f1-score   support\n\n        FAKE       0.62      0.37      0.46       375\n        REAL       0.60      0.81      0.69       445\n\n    accuracy                           0.61       820\n   macro avg       0.61      0.59      0.58       820\nweighted avg       0.61      0.61      0.59       820\n',
 '              precision    recall  f1-score   support\n\n        FAKE       0.60      0.34      0.44       392\n        REAL       0.57      0.79      0.66       428\n\n    accuracy                           0.58       820\n   macro avg       0.58      0.57      0.55       820\nweighted avg       0.58      0.58      0.55       820\n',
 '              precision    recall  f1-score   support\n\n        FAKE       0.57      0.38      0.45       357\n        REAL       0.62      0.78      0.69       463\n\n    accuracy                           0.60       820\n   macro avg       0.59      0.58      0.57       820\nweighted avg       0.60      0.60      0.59       820\n',
 '              precision    recall  f1-score   support\n\n        FAKE       0.58      0.36      0.44       348\n        REAL       0.63      0.81      0.71       472\n\n    accuracy                           0.62       820\n   macro avg       0.61      0.58      0.58       820\nweighted avg       0.61      0.62      0.60       820\n',
 '              precision    recall  f1-score   support\n\n        FAKE       0.56      0.32      0.41       349\n        REAL       0.62      0.81      0.70       471\n\n    accuracy                           0.60       820\n   macro avg       0.59      0.57      0.56       820\nweighted avg       0.59      0.60      0.58       820\n',
 '              precision    recall  f1-score   support\n\n        FAKE       0.53      0.34      0.41       349\n        REAL       0.61      0.78      0.69       471\n\n    accuracy                           0.59       820\n   macro avg       0.57      0.56      0.55       820\nweighted avg       0.58      0.59      0.57       820\n',
 '              precision    recall  f1-score   support\n\n        FAKE       0.62      0.39      0.48       357\n        REAL       0.64      0.82      0.72       463\n\n    accuracy                           0.63       820\n   macro avg       0.63      0.61      0.60       820\nweighted avg       0.63      0.63      0.61       820\n',
 '              precision    recall  f1-score   support\n\n        FAKE       0.58      0.38      0.46       363\n        REAL       0.61      0.78      0.69       457\n\n    accuracy                           0.60       820\n   macro avg       0.60      0.58      0.57       820\nweighted avg       0.60      0.60      0.59       820\n',
 '              precision    recall  f1-score   support\n\n        FAKE       0.59      0.37      0.45       333\n        REAL       0.65      0.82      0.73       479\n\n    accuracy                           0.64       812\n   macro avg       0.62      0.59      0.59       812\nweighted avg       0.63      0.64      0.61       812\n']
```