CENG463 NLP HW1 REPORT:

PART1:

In the preproccesing:

First I tokenize the words in every line and apply PortersAlgorithm for stemming.

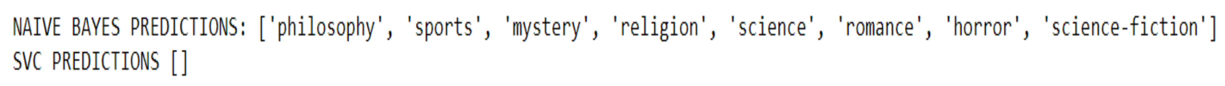
```
processed = []
for line in lines:

    tokens = word_tokenize(line)


    tokens = [word for word in tokens if word.isalpha()]


    stemmer = PorterStemmer()
    tokens = [stemmer.stem(word) for word in tokens]

    preprocessed_line = ' '.join(tokens)
    processed.append((preprocessed_line, filename.split("_")[0]))
```

In the feature selection part:

For every preprocessed documentation I apply TfidVectorizer method which transforms a collection of text documents into a matrix where each row corresponds to a document, and each column corresponds to a unique term in the entire collection.

```
all_features = []

for doc in megadoc:

    texts = [entry[0] for entry in doc]
    label = doc[0][1]

    vectorizer = TfidfVectorizer(strip_accents='ascii', stop_words= 'english')
    x = vectorizer.fit_transform(texts)

    features = vectorizer.get_feature_names_out()
    all_features.append((features,label))

return all_features
```

The classification results for the test data ("philosophy_test.txt","sports_test.txt","mystery_test.txt",...) are as follows:



```
NAIVE BAYES PREDICTIONS: ['philosophy', 'sports', 'mystery', 'religion', 'science', 'romance', 'horror', 'science-fiction']
SVC PREDICTIONS []
```

As seen in the picture above all the test txts are classified correctly. For the SVC predictions I encountered some error during the SVC.fit part in the train() function and I couldn't solve the problem. Anyway, my implementation structure for the SVC classifier is ready in the pythpn file and I believe If this error can be solved then the classifications are predicted automatically for the SVC (I add the SVC structure as comment in the main part).

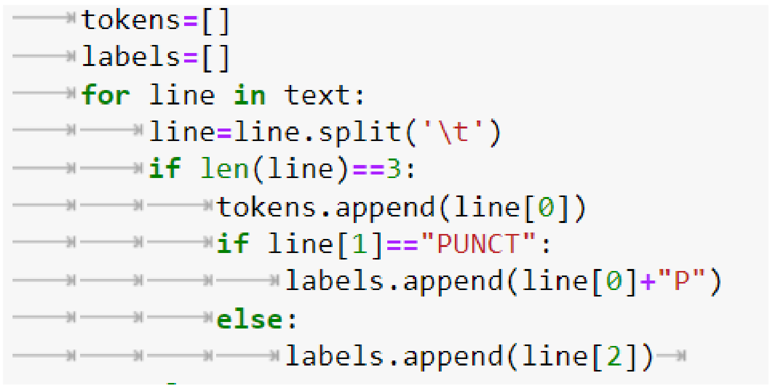
Also, the accuracy is %100 for the NaiveBayes Classifier which I tempted to think I misunderstand the homework description but I implemented it as I understand. Since all the classification results are true there are no false positive or false negatives, therefore I do not calculate performance metrics.

PART2:

1) 2.COLUMN(UPOS): Universal Part of Speech. It stands for the word's allocated universal part-of-speech tag. These tags adhere to the Universal POS tag set, a condensed collection of part-of-speech tags with a goal of being language-neutral.

3.COLUMN(FEATS): This column includes morphological characteristics or further details regarding the form of the word. It may contain information about number, gender, tense, and other language-specific characteristics.

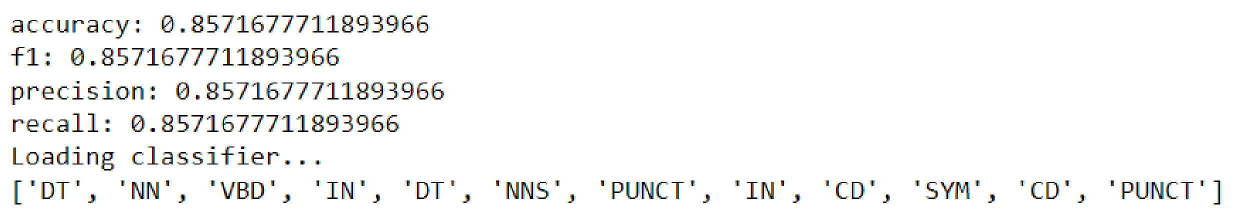
All the columns are used in the implementation:



```
    tokens=[]
    labels=[]
    for line in text:
        line=line.split('\t')
        if len(line)==3:
            tokens.append(line[0])
            if line[1]=="PUNCT":
                labels.append(line[0]+"P")
            else:
                labels.append(line[2])
```

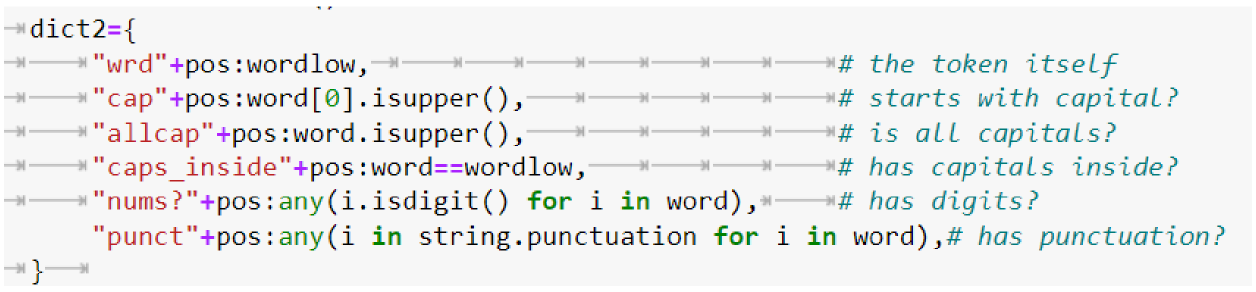
Column1 is used to access the token, Column2 is checked if it is PUNCT but not direcly used(Only for checking purposes), Column3 is used to access the label of that token if Column2 is not PUNCT.

2) For set1 the performance metrics are as follows:



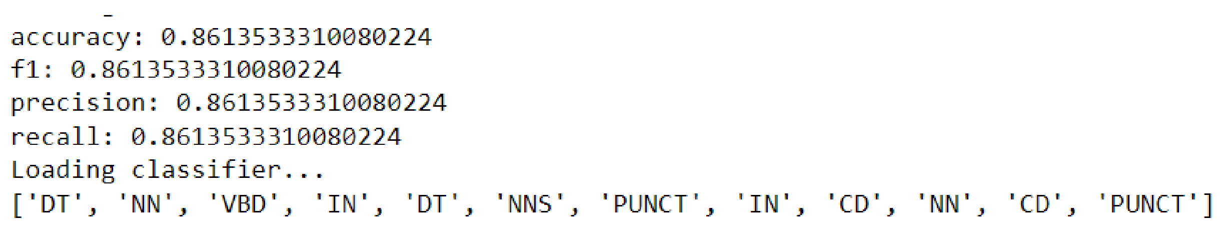
```
accuracy: 0.8571677711893966
f1: 0.8571677711893966
precision: 0.8571677711893966
recall: 0.8571677711893966
Loading classifier...
['DT', 'NN', 'VBD', 'IN', 'DT', 'NNS', 'PUNCT', 'IN', 'CD', 'SYM', 'CD', 'PUNCT']
```

For the set2 :



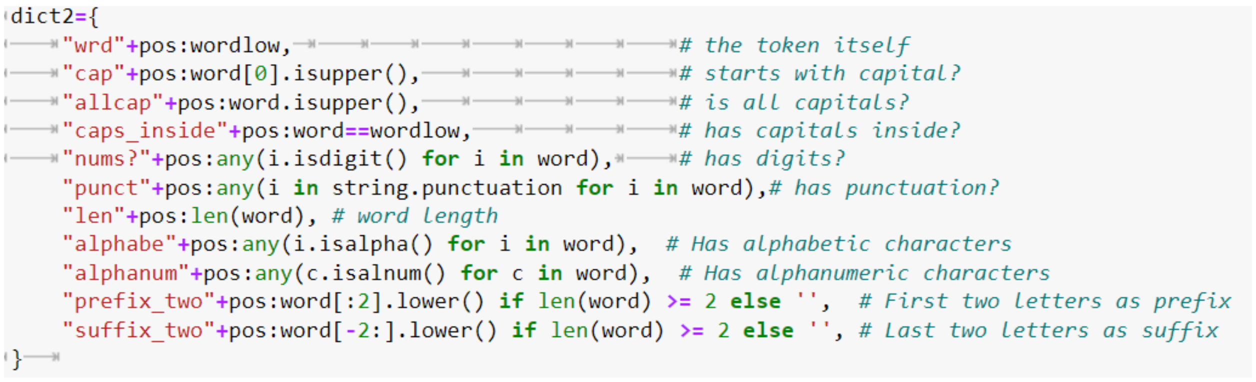
```
	dict2={
		"wrd"+pos:wordlow,								# the token itself
		"cap"+pos:word[0].isupper(),					# starts with capital?
		"allcap"+pos:word.isupper(),					# is all capitals?
		"caps_inside"+pos:word==wordlow,				# has capitals inside?
		"nums?"+pos:any(i.isdigit() for i in word),		# has digits?
		"punct"+pos:any(i in string.punctuation for i in word),# has punctuation?
	}
```

The performance metrics are as follows:



```
accuracy: 0.8613533310080224
f1: 0.8613533310080224
precision: 0.8613533310080224
recall: 0.8613533310080224
Loading classifier...
['DT', 'NN', 'VBD', 'IN', 'DT', 'NNS', 'PUNCT', 'IN', 'CD', 'NN', 'CD', 'PUNCT']
```

For the set3:



```
dict2={
    "wrd"+pos:wordlow,                          # the token itself
    "cap"+pos:word[0].isupper(),                # starts with capital?
    "allcap"+pos:word.isupper(),                # is all capitals?
    "caps_inside"+pos:word==wordlow,            # has capitals inside?
    "nums?"+pos:any(i.isdigit() for i in word), # has digits?
    "punct"+pos:any(i in string.punctuation for i in word),# has punctuation?
    "len"+pos:len(word), # word length
    "alphabe"+pos:any(i.isalpha() for i in word),  # Has alphabetic characters
    "alphanum"+pos:any(c.isalnum() for c in word),  # Has alphanumeric characters
    "prefix_two"+pos:word[:2].lower() if len(word) >= 2 else '',  # First two letters as prefix
    "suffix_two"+pos:word[-2:].lower() if len(word) >= 2 else '', # Last two letters as suffix
}
```

The performance metrics are as follows:

```
accuracy: 0.8957098011859086
f1: 0.8957098011859086
precision: 0.8957098011859086
recall: 0.8957098011859086
Loading classifier...
['DT', 'NN', 'VBD', 'IN', 'DT', 'NNS', 'RB', 'IN', 'CD', 'VBD', 'CD', 'PUNCT']
```

3) Positive aspects of LR: With LR, we might see shorter training periods and reduced computational costs. If there are no significant sequential dependencies in your data, LR may still function rather well.

Negative aspects of LR: Losing the ability to represent complex sequential patterns could impair our performance, particularly if our task benefits from taking the complete sequence into account.

CRF could be more suitable for our task because I think our task relies on capturing intricate sequential dependencies rather than simplicity and efficiency.