CO544-Text-Classification

June 24, 2020

[36]:
```
import re #importing re module with regular expressions
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
import pickle
from nltk.stem import WordNetLemmatizer #for lemmatization
nltk.download('wordnet')
from sklearn.datasets import load_files
import numpy as np
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\user\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\user\AppData\Roaming\nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

[37]:
```
movie_data = load_files(r"E:/University Works/3rd Year/Semester 6/CO 544 -␣
 ↪Machine Learning and Data Mining/Lab/6/txt_sentoken")
X, y = movie_data.data, movie_data.target
```

[38]:
```
print('y = ',y)
```

```
y =  [0 1 1 … 1 0 0]
```

[39]:
```
documents = []

stemmer = WordNetLemmatizer()

for sen in range(0, len(X)):
    # Remove all the special characters
    document = re.sub(r'\W', ' ', str(X[sen]))

    # remove all single characters
    document = re.sub(r'\s+[a-zA-Z]\s+', ' ', document)

    # Remove single characters from the start
    document = re.sub(r'\^[a-zA-Z]\s+', ' ', document)
```

```
    # Remove numbers
    document= re.sub(r'\d+',' ', document)

    # Substituting multiple spaces with single space
    document = re.sub(r'\s+', ' ', document, flags=re.I)

    # Removing prefixed 'b'
    document = re.sub(r'^b\s+', '', document)

    # Converting to Lowercase
    document = document.lower()

    # Lemmatization
    document = document.split()

    document = [stemmer.lemmatize(word) for word in document]
    document = ' '.join(document)

    documents.append(document)
```

[40]:

```
#TODO 1

#When you have a dataset in bytes format, the alphabet letter "b" is appended␣
↪before every string.
#The regex ^b\s+ removes "b" from the start of a string.
# Removing prefixed 'b'
#document = re.sub(r'^b\s+', '', document)

#Tokenization
#Tokenization is the process of splitting the given text into smaller pieces␣
↪called tokens.
#Words, numbers, punctuation marks, and others can be considered as tokens.

#Remove stop words
#"Stop words" are the most common words in a language like "the", "a", "on",␣
↪"is", "all"
#These words do not carry important meaning and are usually removed from texts.
#Remove using NLTK

#Part of speech tagging (POS)
#Part-of-speech tagging aims to assign parts of speech to each word of a given␣
↪text (such as nouns, verbs, adjectives, and others) based on its definition␣
↪and its context.
```

```
#There are many tools containing POS taggers including NLTK, spaCy, TextBlob, Pattern, Stanford CoreNLP, Memory-Based Shallow Parser (MBSP), Apache OpenNLP, Apache Lucene, General Architecture for Text Engineering (GATE), FreeLing, Illinois Part of Speech Tagger, and DKPro Core.

#Chunking (shallow parsing)
#Chunking is a natural language process that identifies constituent parts of sentences (nouns, verbs, adjectives, etc.) and links them to higher order units that have discrete grammatical meanings (noun groups or phrases, verb groups, etc.).
#Chunking tools: NLTK, TreeTagger chunker, Apache OpenNLP, General Architecture for Text Engineering (GATE), FreeLing.

#Named entity recognition
#Named-entity recognition (NER) aims to find named entities in text and classify them into pre-defined categories (names of persons, locations, organizations, times, etc.).
#Named-entity recognition tools: NLTK, spaCy, General Architecture for Text Engineering (GATE) - ANNIE, Apache OpenNLP, Stanford CoreNLP, DKPro Core, MITIE, Watson Natural Language Understanding, TextRazor, FreeLing

#Coreference resolution (anaphora resolution)
#Pronouns and other referring expressions should be connected to the right individuals.
#Coreference resolution finds the mentions in a text that refer to the same real-world entity.
#For example, in the sentence, "Andrew said he would buy a car" the pronoun "he" refers to the same person, namely to "Andrew".
#Coreference resolution tools: Stanford CoreNLP, spaCy, Open Calais, Apache OpenNLP

#Collocation extraction
#Collocations are word combinations occurring together more often than would be expected by chance.
#Collocation examples are "break the rules," "free time," "draw a conclusion," "keep in mind," "get ready," and so on.

#Relationship extraction
#Relationship extraction allows obtaining structured information from unstructured sources such as raw text.
#Strictly stated, it is identifying relations (e.g., acquisition, spouse, employment) among named entities (e.g., people, organizations, locations).
#For example, from the sentence "Mark and Emily married yesterday," we can extract the information that Mark is Emily's husband.
```

[41]:
```
#(d) Convert text into numbers.
#Bag of Words

from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(max_features=1500, min_df=5, max_df=0.7,␣
↪stop_words=stopwords.words('english'))
X = vectorizer.fit_transform(documents).toarray()
```

[42]:
```
#TODO 2
#Discuss advantages and disadvantages of 'Bag of Words model'

#Advantages
#1.works fine for converting text to numbers.
#2.Very simple to understand and implement.

#Disadvantages
#1.It assigns a score to a word based on its occurrence in a particular␣
↪document.
#It doesn't take into account the fact that the word might also be having a␣
↪high frequency of occurrence in other documents as well.
#TFIDF(Term Frequency Inverse Document Frequency) resolves this issue.

#2.Bag of words leads to a high dimensional feature vector due to large size of␣
↪Vocabulary, V.
#3.It leads to a highly sparse vectors as there is nonzero value in dimensions␣
↪corresponding to words that occur in the sentence.
#4.Although the coding method counts the number of times a word appears in the␣
↪text, it cannot distinguish common words (such as "I", "Yes", "", etc.) and␣
↪keywords (such as: " Natural language processing", "NLP The importance of␣
↪"etc." in the text;
```

[43]:
```
#convert values obtained using the bag of words model into TFIDF values

from sklearn.feature_extraction.text import TfidfTransformer

tfidfconverter = TfidfTransformer()
X = tfidfconverter.fit_transform(X).toarray()
```

[44]:
```
#Note:

#You can also directly convert text documents into TFIDF feature values␣
↪(without first converting documents to bag of words features)
#script as follows

#from sklearn.feature_extraction.text import TfidfVectorizer
```

```
#tfidfconverter = TfidfVectorizer(max_features=1500, min_df=5, max_df=0.7,␣
 ↪stop_words=stopwords.words('english'))
#X = tfidfconverter.fit_transform(documents).toarray()
```

[45]:
```
#(e) Text Classification

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=0)
```

[46]:
```
# Logistic regression model
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression()
log_reg.fit(X_train,y_train)
```

[46]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

[47]:
```
log_reg_predict = log_reg.predict(X_test)
print(log_reg_predict)
```

```
[1 0 0 1 0 0 1 1 1 1 0 1 0 1 1 1 0 0 1 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 0
 0 0 0 1 1 0 0 0 1 0 0 1 1 1 1 0 1 1 1 0 1 1 1 1 0 1 1 1 0 0 0 1 0 0 1 0 1
 0 0 0 0 0 0 1 1 0 0 1 0 1 1 0 1 0 1 1 1 0 0 0 0 0 0 1 0 1 1 0 1 1 0 0 1 1
 0 1 1 1 0 1 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 1 0 0 1 0 1 1 1 1 0 1 1 0 0 0
 1 1 0 1 0 1 1 0 1 1 0 1 0 1 0 0 0 0 1 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 0 1 0
 0 1 1 1 1 1 1 1 0 0 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 1 0
 1 0 1 1 0 0 1 0 0 1 1 1 0 0 0 1 1 1 0 0 1 0 1 0 1 0 1 1 1 1 1 1 0 1 0 1 1
 1 1 0 1 0 0 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 0 1 1 0 0 0 0 1 1 0 0 0 1 0 1 0
 0 0 1 1 0 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 1 0 1
 1 1 1 0 0 0 0 0 1 1 1 0 0 1 1 0 0 0 1 1 0 0 1 1 0 1 0 0 1 0 0 1 0 1 0 1 0
 0 1 0 1 0 0 0 0 1 1 0 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0]
```

[48]:
```
#TODO 3
```

[49]:
```
#Random Forest model

from sklearn.ensemble import RandomForestClassifier

RF_classifier = RandomForestClassifier(n_estimators=1000, random_state=0)
RF_classifier.fit(X_train, y_train)
```

[49]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=1000,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

[50]:
```
RF_predict = RF_classifier.predict(X_test)
print(RF_predict)
```

```
[0 0 0 0 0 0 1 1 1 1 1 0 0 0 1 0 0 0 1 1 0 1 1 1 0 1 1 0 1 1 1 0 1 0 0 1 0
 0 1 0 1 1 0 0 1 1 0 0 1 1 0 1 0 1 1 1 0 1 1 1 1 0 1 1 0 0 0 0 1 0 0 1 0 1
 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0 1 1 1 0 0 0 0 0 0 1 1 0 1 0 1 1 0 1 0 1
 0 1 1 0 0 1 1 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 0 1 0 0 1 1 1 0 1 0 0 0 0
 1 1 0 1 0 1 1 1 1 1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1 1 1 0 1 1
 0 1 0 1 1 1 1 1 0 0 1 0 0 1 0 0 0 1 1 1 0 1 1 1 1 0 0 0 0 0 0 1 0 0 1 0 0
 0 0 1 1 0 0 1 0 0 1 1 1 0 0 0 1 0 1 1 0 0 0 1 0 1 0 1 1 1 0 1 0 0 1 0 1 1
 1 1 0 1 0 0 1 0 1 0 0 1 1 0 0 1 0 1 0 0 1 0 1 1 0 0 1 0 0 1 0 1 0 1 0 1 0
 0 0 1 1 1 0 0 0 1 0 0 0 1 0 0 1 0 1 0 1 1 1 0 0 0 1 1 1 1 0 1 1 1 0 1 0 1
 1 1 1 0 1 0 0 0 1 1 1 0 0 1 1 0 0 0 1 1 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 1 0
 0 1 1 1 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 1 0 0 1 1 0 1 0 0 1 0]
```

[51]:
```
#Support Vector Machine

from sklearn.svm import SVC

svm = SVC()
svm.fit(X_train, y_train)
```

[51]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

[52]:
```
svm_predict = svm.predict(X_test)
print(svm_predict)
```

```
[1 0 0 1 0 0 1 1 1 1 0 1 0 1 1 1 0 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0 1 0 0 1 0
 0 0 0 1 1 0 0 0 1 0 0 1 1 1 1 0 1 1 1 0 1 1 1 1 0 1 1 1 0 0 0 1 0 0 1 0 1
 0 0 0 0 0 0 1 1 0 0 1 0 1 1 0 1 0 1 1 1 1 0 0 0 0 0 1 0 1 1 0 1 1 0 0 1 1
 0 1 1 0 0 1 0 0 0 0 1 1 1 0 0 0 1 0 0 0 1 0 1 0 0 1 0 1 1 1 1 0 1 1 0 0 0
 1 1 0 1 0 1 1 0 1 1 0 1 0 1 0 0 0 0 1 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 0 1 0
 0 1 1 1 1 1 1 1 0 0 1 0 0 1 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 1 1
 1 0 1 1 0 0 1 0 0 1 1 1 1 1 0 1 1 1 0 0 0 0 1 0 1 0 1 1 1 1 1 1 0 1 0 1 1
 1 1 0 1 0 0 1 0 1 0 1 1 1 1 1 1 1 1 0 1 1 0 1 1 0 0 0 0 1 1 0 0 0 1 0 1 0
 0 0 1 1 0 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 1 0 1
```

```
 1 1 1 0 0 0 0 0 0 1 1 0 0 1 1 0 0 0 1 1 0 0 1 1 0 1 0 0 1 0 0 1 0 1 0 1 1
 0 1 0 1 0 0 0 0 1 1 0 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0]
```

[53]:
```python
#Naive Bayes Classifier

from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
gnb.fit(X_train, y_train)
```

[53]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

[54]:
```python
gnb_predict = gnb.predict(X_test)
print(gnb_predict)
```

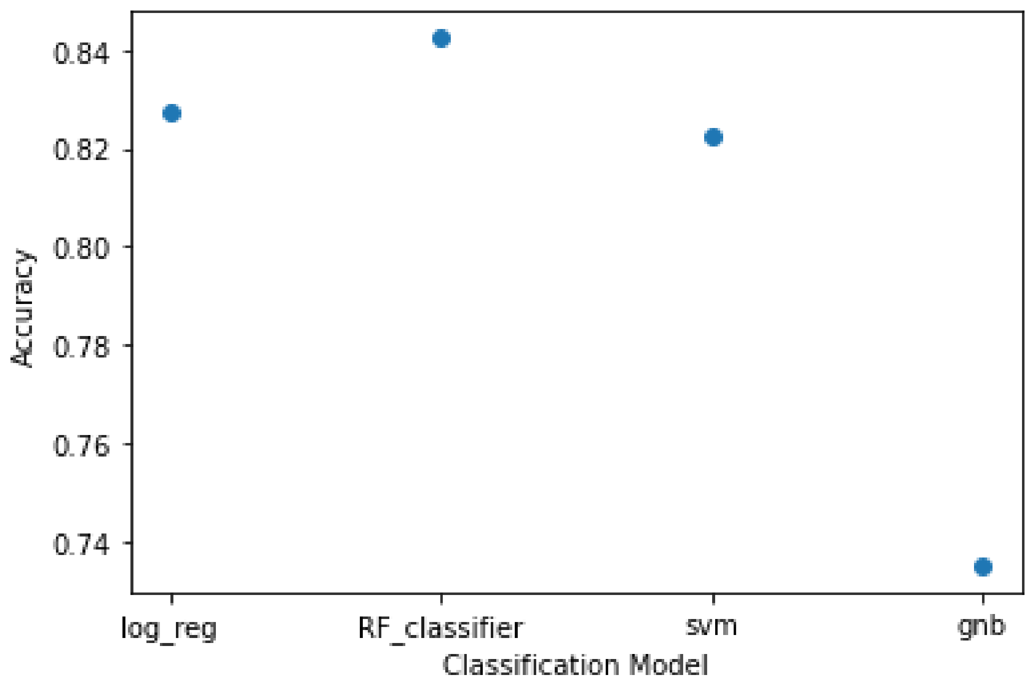
```
[0 0 0 1 0 0 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 0 0 1 0 1 1 0 1 0 1 0 1 0 0 0 0
 0 1 1 1 1 0 0 0 1 0 1 1 1 1 1 0 1 1 1 0 1 1 1 1 0 1 1 1 0 0 0 1 0 0 1 1 1
 0 0 0 0 0 0 1 1 1 0 1 0 1 1 0 1 0 1 1 1 0 1 1 0 0 0 1 0 0 0 0 1 1 0 1 1 0
 0 1 1 0 0 1 0 0 0 0 0 1 1 1 1 0 1 0 0 1 1 0 0 0 0 1 0 1 1 0 1 0 1 0 0 1 1
 1 0 0 1 0 1 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 1 0 0 0 0 0 0 0 1 1 1 0
 0 1 1 1 1 1 1 1 0 0 1 1 0 0 0 0 0 0 1 1 1 1 1 1 0 0 1 1 0 1 0 1 0 0 0 0 0
 1 0 1 1 0 0 1 0 0 1 1 1 1 1 0 1 0 0 0 1 1 0 1 0 1 0 1 1 0 1 1 1 1 1 0 1 0
 1 0 0 1 0 1 1 0 0 0 1 1 1 0 1 1 1 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0 1 1 0 1 0
 0 0 1 1 1 0 0 0 1 0 1 1 0 0 0 1 0 1 0 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 1 0 1
 1 1 1 0 0 0 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 0 1 0 0 1 0 0 1 0 1 0 0 1
 0 1 0 1 0 0 0 1 1 1 1 1 1 0 0 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0]
```

[55]:
```python
#Evaluating the models
```

[56]:
```python
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score

#Logistic regression model
print("Logistic regression accuracy = ",accuracy_score(y_test, log_reg_predict))

#Random Forest model
print("Random Forest accuracy = ",accuracy_score(y_test, RF_predict))

#Support Vector Machine
print("Support Vector Machine accuracy = ",accuracy_score(y_test, svm_predict))

#Naive Bayes Classifier
print("Naive Bayes accuracy = ",accuracy_score(y_test, gnb_predict))
```

```
Logistic regression accuracy =  0.8275
Random Forest accuracy =  0.8425
Support Vector Machine accuracy =  0.8225
Naive Bayes accuracy =  0.735
```

[57]:
```
import matplotlib.pyplot as plt

models = ['log_reg','RF_classifier','svm','gnb']
scores = []
scores.append(accuracy_score(y_test, log_reg_predict))
scores.append(accuracy_score(y_test, RF_predict))
scores.append(accuracy_score(y_test, svm_predict))
scores.append(accuracy_score(y_test, gnb_predict))

plt.figure()
plt.xlabel('Classification Model')
plt.ylabel('Accuracy')
plt.scatter(models, scores)
plt.show()
```



[58]:
```
#As you can see in the graph the highest accuracy is for Random forest␣
↪classifier
#Therefore best model is the Random forest classifier
```

[ ]:
```
```