Machine Learning Lab 11

Simran Kaur

311443

Exercise 0: Preprocessing Text Data

Preprocessing textual data to remove punctuation, stop-words

In [268…
```
from sklearn.datasets import fetch_20newsgroups
import pandas as pd
import numpy as np
import nltk
import re
from nltk.corpus import stopwords
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\simra\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```
Out[268… True

In [269…
```
categories = ['comp.graphics', 'sci.med']
```

In [270…
```
news_data = fetch_20newsgroups(subset = 'all', categories = categories)
```

In [280…
```
def remove_stopwords(file):
    new_file = []
    stop_words = set(stopwords.words('english'))           # stop words
    file = re.sub('[^A-Za-z]+', ' ', file)                  # removes punctuations a
    file = file.lower()
    words = file.split()
    for r in words:
        if r not in stop_words and len(r) > 2:    # words with 2 alphabets won't be
            new_file.append(r)
    return new_file
```

Cleaned data with stop words and punctuations removed

In [272…
```
news_data_cleaned = []
for f in twenty_train['data']:
    news_data_cleaned.append(remove_stopwords(f))
```

Implementing a bag-of-words feature representation for each text sample

For each news item the words and their respective frequency in document is created using bag of words.

In [273…
```
def bag_of_words(l_words):              # passing the list of words after cleaning da
    count = 1
    word_freq = {}
    for w in l_words:
        if w not in word_freq.keys():
            word_freq[w] = count
        else:
            word_freq[w] += count
    return word_freq
```

In [274…
```
news_freq = []                          # list of dictionaries where each dictionar
for i in range(len(news_data_cleaned)):
    news_freq.append(bag_of_words(news_data_cleaned[i]))
```

In [275…
```
corpus_freq = {}                        # dictionary that contains frequencies of word
for d in news_freq:
    for k in d:
        if k in corpus_freq:
            value = corpus_freq.get(k) + d[k]
            corpus_freq.update({k:value})
        else:
            corpus_freq.update({k:d[k]})
```

After applying Bag of words we will make a dataframe with columns as top 1000 words which appear the most and rows as news belonging to two categories.

In [276…
```
def bagOfWords_df(rows, cols):
    col_name = list(cols.keys())
    df_bow = pd.DataFrame(columns = col_name)
    for doc in rows:
        mydict = {}
        for d in doc:
            if(d in col_name):
                mydict.update({d:doc.get(d)})
        df_bow = df_bow.append(mydict, ignore_index=True)

    df_bow = df_bow.replace(np.nan, 0)
    df_bow['Y'] = list(news_data.target)
    return df_bow
```

In [277…
```
corpus_freq_high = {}

for k_dict in sorted(corpus_freq, key = corpus_freq.get, reverse = True)[:1000]:
        corpus_freq_high[k_dict] = corpus_freq[k_dict]
```

In [278…
```
BOW_dataframe = bagOfWords_df(news_freq, corpus_freq_high)
```

In [279…
```
BOW_dataframe
```

Out[279…

	edu	subject	lines	organization	com	one	image	would	graphics	article	...	students
0	2.0	1.0	0.0	1.0	2.0	0.0	0.0	0.0	0.0	1.0	...	0.0
1	2.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	2.0	1.0	...	0.0
2	6.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	2.0	...	0.0

	edu	subject	lines	organization	com	one	image	would	graphics	article	...	students
3	4.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	...	0.0
4	3.0	1.0	1.0	1.0	2.0	1.0	0.0	1.0	0.0	1.0	...	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
1958	3.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0
1959	2.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	...	0.0
1960	4.0	2.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0
1961	5.0	1.0	1.0	1.0	0.0	0.0	0.0	3.0	1.0	1.0	...	0.0
1962	4.0	1.0	1.0	2.0	0.0	0.0	0.0	0.0	1.0	0.0	...	0.0

1963 rows × 1001 columns

Implementing a TF-IDF feature representation for each text sample

TF = term frequency = number of times term appear in document/ total number of terms in document

DF = Inverse Document frequency = number of documents which contain the term/ total number of documents

TFIDF = TF * log(Inverse(DF))

In [281…

```
def tf_idf(l_freq):
    count = 1
    tfIdf = []
    for freq in l_freq:
        freq_tf = {}
        for word in freq.keys():
            freq_tf[word] = (freq[word]/sum(freq.values()))*np.log(len(l_freq)/((sum
        tfIdf.append(freq_tf)
    return tfIdf
```

In [282…

```
lFreq_tfidf = tf_idf(news_freq)
```

Since the words which appear more in one document is important but if this word is in every document then it's impotance decreases as it does not give much new information for classification. This is what we do using TF-IDF.

In [283…

```
def TfIdf_data(rows, cols):
    col_names = list(cols.keys())
    df_tfidf = pd.DataFrame(columns = col_names)
    for doc in rows:
        mydict = {}
        for d in doc:
            if(d in col_names):
                mydict.update({d:doc.get(d)})
        df_tfidf = df_tfidf.append(mydict, ignore_index=True)

    df_tfidf = df_tfidf.replace(np.nan, 0)
```

```
    df_tfidf['Y'] = list(news_data.target)
    return df_tfidf
```

In [284…

```
tfidf_dataframe = TfIdf_data(lFreq_tfidf, corpus_freq_high)
tfidf_dataframe
```

Out[284…

	edu	subject	lines	organization	com	one	image	would	graphics	
0	0.087778	0.043889	0.000000	0.043889	0.087778	0.000000	0.0	0.000000	0.000000	0.0
1	0.108678	0.054339	0.054339	0.054339	0.000000	0.000000	0.0	0.000000	0.108678	0.0
2	0.136027	0.022671	0.022671	0.022671	0.022671	0.022671	0.0	0.000000	0.000000	0.0
3	0.067455	0.016864	0.016864	0.016864	0.000000	0.016864	0.0	0.016864	0.000000	0.0
4	0.026815	0.008938	0.008938	0.008938	0.017876	0.008938	0.0	0.008938	0.000000	0.0
…	…	…	…	…	…	…	…	…	…	
1958	0.081508	0.027169	0.027169	0.027169	0.000000	0.000000	0.0	0.000000	0.000000	0.0
1959	0.053911	0.026955	0.026955	0.026955	0.000000	0.026955	0.0	0.026955	0.000000	0.0
1960	0.224482	0.112241	0.056120	0.056120	0.000000	0.000000	0.0	0.000000	0.000000	0.0
1961	0.171167	0.034233	0.034233	0.034233	0.000000	0.000000	0.0	0.102700	0.034233	0.0
1962	0.109547	0.027387	0.027387	0.054774	0.000000	0.000000	0.0	0.000000	0.027387	0.0

1963 rows × 1001 columns

Split the dataset randomly into train/validation/test splits according to ratios 80%:10%:10%

In [285…

```
def splitData(df):
    shuffled_df = df.sample(frac = 1)
    size = int(0.8*len(shuffled_df))
    Train = shuffled_df.loc[:size, :]
    Val = shuffled_df.loc[size:size + int(0.1*len(shuffled_df)), :]
    Test = shuffled_df.loc[size + int(0.1*len(shuffled_df)): , :]
    return Train, Val, Test
```

Exercise 1: Implementing Naive Bayes Classifier for Text Data

In [286…

```
# Calculates the prior probabilities for two classes

def prior(df):
    classes = sorted(list(df['Y'].unique()))
    prior_prob = {}
    for c in classes:
        prior_prob[c] = (len(df[df['Y'] == c])/ len(df))
    return prior_prob
```

In [287…

```
# For a given class it calculates the conditional probability for a feature i.e. pro

def conditional_prob(df, feature, label):
```

```
    df_label = df[df['Y'] == label]
    prob_feature_label = len(df_label[df_label[feature] > 0])/len(df_label)
    return prob_feature_label
```

For each word in a document if the word appears in the feature we will calculate the probability for each word given class which when multiplied with prior probability gives the probability for the class given document. In this way we will predict class for each document in the news data.

In [302…

```
def naive_bayes_BOW(df, test_df):
    features = list(df.columns)
    Y_pred = []
    prior_probability = prior(df)
    labels = list(prior_probability.keys())

    for row_test in test_df.iterrows():
        pos_prob = {}
        for y in labels:
            prob = 1
            for i in features:
                if row_test[1][i] > 0:
                    prob *= conditional_prob(df, i, y)
            pos_prob[y] = prob*prior_probability[y]
        Y_pred.append(max(pos_prob, key = pos_prob.get))

    return Y_pred
```

In [290…

```
def accuracy(y, y_pred):
    acc = 0
    for i in range(len(y)):
        if y[i] == y_pred[i]:
            acc += 1
    return (acc/len(y))*100
```

In [289…

```
train_BOW, val_BOW, test_BOW = splitData(BOW_dataframe)
```

Test Accuracy for Naive bayes using BOW as preprocessing

In [298…

```
test_pred_NbBoW = naive_bayes_BOW(train_BOW, test_BOW)
```

In [300…

```
acc_test_NbBoW = accuracy(list(test_BOW['Y']), test_pred_NbBoW)
```

In [301…

```
print(f'Accuracy on test set {acc_test_NbBoW}')
```

```
Accuracy on test set 72.5428027901078
```

For TF-IDF

In [303…

```
def prob_tfidf(df, feature):
    prob = {}
    prob_feature = {}
    for label in df['Y'].unique():
        df_label = df[df['Y'] == label]
        prob[label] = sum(df_label[feature])/(df_label.loc[:,df_label.columns != 'Y'
    for k in prob:
```

```
        prob_feature[k] = prob[k]/sum(prob.values())
    return prob_feature
```

The probabilities for Tf- Idf are calculated using the tfidf calculated for each word in a document.

In [308…
```
def naive_bayes_tfidf(df, test_df):
    features = list(df.columns)
    Y_pred = []
    prior_probability = prior(df)
    labels = list(prior_probability.keys())
    for row in test_df.iterrows():
        pos_prob = {}
        for y in labels:
            pro = 1
            for i in features:
                if row[1][i] > 0:
                    pro *= (prob_tfidf(df, i)[y])
            pos_prob[y] = pro*prior_probability[y]
        Y_pred.append(max(pos_prob, key = pos_prob.get))
    return Y_pred
```

In [309…
```
train_tfidf, val_tfidf, test_tfidf = splitData(tfidf_dataframe)
```

Test Accuracy for Naive bayes using TF-IDF as preprocessing

In [310…
```
test_pred_Nbtfidf = naive_bayes_tfidf(train_tfidf, test_tfidf)
```

In [311…
```
acc_test_Nbtfidf = accuracy(list(test_tfidf['Y']), test_pred_Nbtfidf)
```

In [312…
```
print(f'Accuracy on test set {acc_test_Nbtfidf}')
```

```
Accuracy on test set 97.38738738738739
```

Exercise 2: Implementing SVM Classifier via Scikit-Learn

In [259…
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
```

For Bag of Words

In [265…
```
svm_bow_X = train_BOW.iloc[:, :-1].to_numpy()
svm_bow_Y = train_BOW.iloc[:, -1].to_numpy()
svm_bow_testX = test_BOW.iloc[:, :-1].to_numpy()
svm_bow_testY = test_BOW.iloc[:, -1].to_numpy()
```

In [254…
```
hyperparameter_grid = {'C': [0.01, 0.02, 0.03], 'kernel': ['linear', 'rbf', 'sigmoid
```

In [255…
```
svm_classifier = SVC(random_state = 3116)
```

In [257…
```
grid_svm = GridSearchCV(svm_classifier, hyperparameter_grid, cv = 4, return_train_sc
grid_svm.fit(svm_bow_X, svm_bow_Y)
```

Out[257…
```
GridSearchCV(cv=4, estimator=SVC(random_state=3116),
             param_grid={'C': [0.01, 0.02, 0.03], 'gamma': ['scale', 'auto'],
                         'kernel': ['linear', 'rbf', 'sigmoid']},
             return_train_score=True)
```

In [260…
```
svm_bow_testPred = grid_svm.predict(svm_bow_testX)
```

In [262…
```
svm_bow_testPred
```

Out[262…
```
array([0, 0, 0, ..., 1, 1, 1], dtype=int64)
```

In [266…
```
accuracy_score(svm_bow_testY, svm_bow_testPred)
```

Out[266…
```
0.9356828193832599
```

In [267…
```
grid_svm.best_params_
```

Out[267…
```
{'C': 0.03, 'gamma': 'scale', 'kernel': 'linear'}
```

For TF-IDF

In [313…
```
svm_tfidf_X = train_tfidf.iloc[:, :-1].to_numpy()
svm_tfidf_Y = train_tfidf.iloc[:, -1].to_numpy()
svm_tfidf_testX = test_tfidf.iloc[:, :-1].to_numpy()
svm_tfidf_testY = test_tfidf.iloc[:, -1].to_numpy()
```

In [314…
```
grid_svm_tfidf = GridSearchCV(svm_classifier, hyperparameter_grid, cv = 4, return_tr
grid_svm_tfidf.fit(svm_tfidf_X, svm_tfidf_Y)
```

Out[314…
```
GridSearchCV(cv=4, estimator=SVC(random_state=3116),
             param_grid={'C': [0.01, 0.02, 0.03], 'gamma': ['scale', 'auto'],
                         'kernel': ['linear', 'rbf', 'sigmoid']},
             return_train_score=True)
```

In [315…
```
svm_tfidf_testPred = grid_svm_tfidf.predict(svm_tfidf_testX)
```

In [316…
```
svm_tfidf_testPred
```

Out[316…
```
array([0, 0, 1, ..., 1, 0, 1], dtype=int64)
```

In [317…
```
accuracy_score(svm_tfidf_testY, svm_tfidf_testPred)
```

Out[317…
```
0.8198198198198198
```

In [318…
```
grid_svm_tfidf.best_params_
```

Out[318…
```
{'C': 0.03, 'gamma': 'scale', 'kernel': 'sigmoid'}
```