# lab_11

February 5, 2022

## 0.1 Exercise 0 : Preprocessing Text Data

[1]:
```
import math
import numpy as np
import random
from sklearn.datasets import fetch_20newsgroups
import string
import nltk
from nltk.corpus import stopwords
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn import svm
nltk.download('stopwords')


RANDOM_STATE = 3116
```

```
[nltk_data] Downloading package stopwords to /Users/mean-
[nltk_data]     machine/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[1]:
```
True
```

### Import data

[2]:
```
categories = ['sci.med', 'comp.graphics']
train = fetch_20newsgroups(subset='all', categories=categories, shuffle=False,␣
 ↪random_state=RANDOM_STATE)
```

### Preprocess text data

[3]:
```
# Initialize vocabulary
vocabulary={}
ind = 0

def preprocess_text(text_data):
    """
```

```
    Removes punctuation marks, special characters, lower case text, removes␣
↪stop words and creates list of words
    Returns: list of words
    """
    text_data = text_data.translate(str.maketrans('', '', string.punctuation))

    text_data = text_data.replace('\n', ' ')

    text_list = text_data.split(' ')
    text_list = list(map(lambda x: x.lower(), text_list))

    stop_words = stopwords.words('english')

    text_words = [word for word in text_list if word not in stop_words and word!
↪='']

    global ind
    # Insert items in dictionary and put index as the value
    for word in text_words:
        if word not in vocabulary.keys():
            vocabulary[word] = ind
            ind += 1

    return text_words
```

[4]:
```
news_items = []
for news in train.data:
    news_items.append(preprocess_text(news))
```

### Bag-of-words feature representation

[5]:
```
def bag_of_words(news_item):
    """
    Returns vector representation of the news_item,
    where vector contains frequency of each unique item in vocabulary.
    """
    count_words = {}
    news_vec = np.zeros(len(vocabulary))

    for word in news_item:
        if word not in count_words.keys():
            count_words[word] = 1
        else:
            count_words[word] += 1

    for word in news_item:
        news_vec[vocabulary[word]] = count_words[word]
```

```
    return news_vec
```

[6]:
```
news_vecs = []
for news in news_items:
    news_vecs.append(bag_of_words(news))
```

**TF-IDF feature representation**

[7]:
```
def idf(news_vecs):
    """
    Returns idf dictionary for the available news corpus
    """
    # For every unique word in vocabulary, stores its IDF value
    idf_vec = {}
    N = len(vocabulary)
    for key, item in vocabulary.items():
        count_docs = 0
        # count news_items which contain the word
        for news in news_vecs:
            if news[item] != 0:
                count_docs += 1
        # Store the IDF value for that word(key)
        idf_vec[key] = math.log10(N/count_docs)

    return idf_vec

def tf_idf(news_vecs):
    """
    Returns tf-idf for the news item
    """

    idf_vec = idf(news_vecs)

    for i, news_item in enumerate(news_vecs):
        indexes = list(news_item.nonzero()[0])
        total_words_in_doc = len(news_items[i])
        for index in indexes:
            key = list(vocabulary.keys())[index]
            # Put tf-idf value for very news_item word
            news_item[index] = (news_item[index]/total_words_in_doc)/
 ↪idf_vec[key]
        news_vecs[i] = news_item

    return
```

[8]:
```
# Driver code for TF-IDF
tf_idf(news_vecs)
```

### 0.2 Exercise 2 : SVM classifier

We are trying out three kernels from the sklearn in this section: - `linear`: We are tuning the `C` parameter. - `rbf`: We are tuning the `gamma` parameter. - `poly`: We are tuning the `degree` parameter.

We are tuning one hyperparameter each for the 3 kernels while keeping the others at their default values. We made this choice in interest of time.

We also have the last section with an exhaustive grid seach but that takes long to converge so we had to interrupt there.

[9]:
```
input_indices = np.arange(0, len(news_vecs), 1).tolist()


# train test split
train_indices = random.sample(input_indices, int(0.9*len(news_vecs)))
test_indices = list(set(input_indices) - set(train_indices))

# further split of train into train and validation
validation_indices = random.sample(train_indices, int(0.1*len(news_vecs)))
train_indices = list(set(train_indices) - set(validation_indices))


print("size of train set:", len(train_indices))
print("size of validation set:", len(validation_indices))
print("size of test set:", len(test_indices))
```

```
size of train set: 1570
size of validation set: 196
size of test set: 197
```

[10]:
```
news_vecs_df = pd.DataFrame(news_vecs)
target = train.target # train is the name of full dataset

# train set
x_train = news_vecs_df[news_vecs_df.index.isin(train_indices)]
y_train = [target[x] for x in train_indices]

# validation set
x_validation = news_vecs_df[news_vecs_df.index.isin(validation_indices)]
y_validation = [target[x] for x in validation_indices]

# test set
x_test = news_vecs_df[news_vecs_df.index.isin(test_indices)]
```

```
y_test = [target[x] for x in test_indices]
```

[11]:
```
# Write a function to predict the test instances using the learnt SVM. Please␣
 ↪refer to sum.SVC() in sklearn to see how to predict test instances
def predict_test(clf, x_test, y_test):
    y_pred = clf.predict(x_test)
    accuracy = accuracy_score(y_true = y_test, y_pred = y_pred)
    accuracy = np.round(accuracy, 2)

    return accuracy
```

### 0.2.1 linear kernel

[16]:
```
try_cs = [0.1, 1, 10, 100, 1000]
clf_linear = {}
accuracy_linear = {}

for this_c in try_cs:
    print("c:", this_c)
    print("***"*24)
    clf_this = svm.SVC(kernel='linear',
                   C=this_c,
                   max_iter=1000,
                   verbose=True)
    clf_this.fit(x_train, y_train)
    clf_linear[this_c] = clf_this
    acc_this = predict_test(clf=clf_this, x_test=x_validation,␣
 ↪y_test=y_validation)
    accuracy_linear[this_c] = acc_this

print(accuracy_linear)
```

```
c: 0.1
************************************************************************
[LibSVM]*
optimization finished, #iter = 790
obj = -155.691797, rho = 0.997227
nSV = 1560, nBSV = 1560
Total nSV = 1560
```

[16]:
```
SVC(C=0.1, kernel='linear', max_iter=1000, verbose=True)
```

```
c: 1
************************************************************************
[LibSVM]*
optimization finished, #iter = 786
obj = -1529.179720, rho = 0.972267
```

```
nSV = 1560, nBSV = 1560
Total nSV = 1560
```

[16]:
```
SVC(C=1, kernel='linear', max_iter=1000, verbose=True)
```

```
c: 10
**************************************************************************
[LibSVM]*
optimization finished, #iter = 779
obj = -12703.760906, rho = 0.788021
nSV = 1511, nBSV = 1496
Total nSV = 1511
```

[16]:
```
SVC(C=10, kernel='linear', max_iter=1000, verbose=True)
```

```
c: 100
**************************************************************************
[LibSVM]*
optimization finished, #iter = 881
obj = -60404.540988, rho = 0.458921
nSV = 995, nBSV = 795
Total nSV = 995
```

[16]:
```
SVC(C=100, kernel='linear', max_iter=1000, verbose=True)
```

```
c: 1000
**************************************************************************
[LibSVM]WARN: libsvm Solver reached max_iter
optimization finished, #iter = 1000
obj = -127522.888720, rho = 0.421788
nSV = 615, nBSV = 55
Total nSV = 615
/Users/mean-machine/miniconda/envs/plot_utils/lib/python3.9/site-
packages/sklearn/svm/_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=1000).  Consider pre-processing your data with StandardScaler or
MinMaxScaler.
  warnings.warn(
```

[16]:
```
SVC(C=1000, kernel='linear', max_iter=1000, verbose=True)
```

```
{0.1: 0.48, 1: 0.48, 10: 0.53, 100: 0.47, 1000: 0.48}
```

### 0.2.2 rbf kernel

[17]:
```
try_gammas = [0.1, 1, 10, 100]
clf_rbf = {}
accuracy_rbf = {}
```

```
for this_gamma in try_gammas:
    print("gamma:", this_gamma)
    print("***"*24)
    clf_this = svm.SVC(kernel='rbf',
                       gamma=this_gamma,
                   max_iter=1000,
                   verbose=True)
    clf_this.fit(x_train, y_train)
    clf_rbf[this_gamma] = clf_this
    acc_this = predict_test(clf=clf_this, x_test=x_validation,␣
 ↪y_test=y_validation)
    accuracy_rbf[this_gamma] = acc_this

print(accuracy_rbf)
```

```
gamma: 0.1
**************************************************************************
[LibSVM]*
optimization finished, #iter = 790
obj = -1553.837992, rho = 0.933500
nSV = 1560, nBSV = 1560
Total nSV = 1560
```

[17]: SVC(gamma=0.1, max_iter=1000, verbose=True)

```
gamma: 1
**************************************************************************
[LibSVM]*
optimization finished, #iter = 783
obj = -1498.563783, rho = 0.338010
nSV = 1560, nBSV = 1560
Total nSV = 1560
```

[17]: SVC(gamma=1, max_iter=1000, verbose=True)

```
gamma: 10
**************************************************************************
[LibSVM]*
optimization finished, #iter = 752
obj = -1099.561614, rho = -1.548211
nSV = 1394, nBSV = 1362
Total nSV = 1394
```

[17]: SVC(gamma=10, max_iter=1000, verbose=True)

```
gamma: 100
**************************************************************************
[LibSVM]WARN: libsvm Solver reached max_iter
```

```
optimization finished, #iter = 1000
obj = -497.589133, rho = -0.849282
nSV = 958, nBSV = 615
Total nSV = 958
```

```
/Users/mean-machine/miniconda/envs/plot_utils/lib/python3.9/site-
packages/sklearn/svm/_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=1000).  Consider pre-processing your data with StandardScaler or
MinMaxScaler.
  warnings.warn(
```

[17]:
```
SVC(gamma=100, max_iter=1000, verbose=True)
```

```
{0.1: 0.48, 1: 0.48, 10: 0.49, 100: 0.49}
```

### 0.2.3 poly Kernel

[18]:
```
try_degrees = [0, 1, 2, 3, 4, 5, 6]
clf_poly = {}
accuracy_poly = {}

for this_degree in try_degrees:
    print("degree:", this_degree)
    print("***"*24)
    clf_this = svm.SVC(kernel='poly',
                   degree=this_degree,
                   max_iter=1000,
                   verbose=True)
    clf_this.fit(x_train, y_train)
    clf_poly[this_degree] = clf_this
    acc_this = predict_test(clf=clf_this, x_test=x_validation,␣
 ↪y_test=y_validation)
    accuracy_poly[this_degree] = acc_this

print(accuracy_poly)
```

```
degree: 0
************************************************************************
[LibSVM]*
optimization finished, #iter = 780
obj = -1560.000000, rho = 1.000000
nSV = 1560, nBSV = 1560
Total nSV = 1560
```

[18]:
```
SVC(degree=0, kernel='poly', max_iter=1000, verbose=True)
```

```
degree: 1
************************************************************************
[LibSVM]WARN: libsvm Solver reached max_iter
```

```
optimization finished, #iter = 1000
obj = -291.667699, rho = 0.381838
nSV = 707, nBSV = 302
Total nSV = 707

/Users/mean-machine/miniconda/envs/plot_utils/lib/python3.9/site-
packages/sklearn/svm/_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=1000).  Consider pre-processing your data with StandardScaler or
MinMaxScaler.
  warnings.warn(
```

[18]: SVC(degree=1, kernel='poly', max_iter=1000, verbose=True)

```
degree: 2
**************************************************************************
[LibSVM]WARN: libsvm Solver reached max_iter
optimization finished, #iter = 1000
obj = -471.466496, rho = 0.604781
nSV = 1090, nBSV = 486
Total nSV = 1090

/Users/mean-machine/miniconda/envs/plot_utils/lib/python3.9/site-
packages/sklearn/svm/_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=1000).  Consider pre-processing your data with StandardScaler or
MinMaxScaler.
  warnings.warn(
```

[18]: SVC(degree=2, kernel='poly', max_iter=1000, verbose=True)

```
degree: 3
**************************************************************************
[LibSVM]WARN: libsvm Solver reached max_iter
optimization finished, #iter = 1000
obj = -684.879667, rho = 0.744514
nSV = 1310, nBSV = 717
Total nSV = 1310

/Users/mean-machine/miniconda/envs/plot_utils/lib/python3.9/site-
packages/sklearn/svm/_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=1000).  Consider pre-processing your data with StandardScaler or
MinMaxScaler.
  warnings.warn(
```

[18]: SVC(kernel='poly', max_iter=1000, verbose=True)

```
degree: 4
**************************************************************************
[LibSVM]WARN: libsvm Solver reached max_iter
optimization finished, #iter = 1000
obj = -803.469330, rho = 0.752092
```

```
nSV = 1406, nBSV = 839
Total nSV = 1406

/Users/mean-machine/miniconda/envs/plot_utils/lib/python3.9/site-
packages/sklearn/svm/_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=1000).  Consider pre-processing your data with StandardScaler or
MinMaxScaler.
  warnings.warn(
```

[18]:
```
SVC(degree=4, kernel='poly', max_iter=1000, verbose=True)
```

```
degree: 5
**********************************************************************
[LibSVM]WARN: libsvm Solver reached max_iter
optimization finished, #iter = 1000
obj = -855.047614, rho = 0.778344
nSV = 1458, nBSV = 885
Total nSV = 1458

/Users/mean-machine/miniconda/envs/plot_utils/lib/python3.9/site-
packages/sklearn/svm/_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=1000).  Consider pre-processing your data with StandardScaler or
MinMaxScaler.
  warnings.warn(
```

[18]:
```
SVC(degree=5, kernel='poly', max_iter=1000, verbose=True)
```

```
degree: 6
**********************************************************************
[LibSVM]WARN: libsvm Solver reached max_iter
optimization finished, #iter = 1000
obj = -873.272955, rho = 0.848353
nSV = 1474, nBSV = 906
Total nSV = 1474

/Users/mean-machine/miniconda/envs/plot_utils/lib/python3.9/site-
packages/sklearn/svm/_base.py:284: ConvergenceWarning: Solver terminated early
(max_iter=1000).  Consider pre-processing your data with StandardScaler or
MinMaxScaler.
  warnings.warn(
```

[18]:
```
SVC(degree=6, kernel='poly', max_iter=1000, verbose=True)
```

```
{0: 0.48, 1: 0.47, 2: 0.47, 3: 0.51, 4: 0.51, 5: 0.51, 6: 0.49}
```

### 0.2.4 Summary

We observe that the best accuracy for the validation set is with the `linear` kernel at `C=10`. However the accuracy is really low for best model, a tiny bit better than a completely random fit. This could be due to multiple reasons, one of them being the low tuning time.

[26]:
```
print("linear kernel", accuracy_linear)
print("rbf kernel", accuracy_rbf)
print("poly kernel", accuracy_poly)
```

```
linear kernel {0.1: 0.48, 1: 0.48, 10: 0.53, 100: 0.47, 1000: 0.48}
rbf kernel {0.1: 0.48, 1: 0.48, 10: 0.49, 100: 0.49}
poly kernel {0: 0.48, 1: 0.47, 2: 0.47, 3: 0.51, 4: 0.51, 5: 0.51, 6: 0.49}
```

[27]:
```
# evaluate the best model on test set
print("Best model accuracy on test set:", predict_test(clf_linear[10], x_test,␣
 ↪y_test))
```

```
Best model accuracy on test set: 0.54
```

### 0.2.5 Altogether

[ ]:
```
parameters = {'kernel':('linear', 'rbf', 'poly'),
              'C':[1, 10, 100],
             'gamma': [0.1, 1, 10, 100],
             'degree': [0, 1, 2, 3, 4, 5, 6]}
svc = svm.SVC()
clf = GridSearchCV(svc, parameters)
clf.fit(x_train, y_train)
GridSearchCV(estimator=svc,
             param_grid=parameters,
             scoring='accuracy',
             n_jobs=8,
             cv=1, # in interest of time, but even this wasn't fast enough
             verbose=4)

print("---Cross validation results---")
print(sorted(clf.cv_results_.keys()))
```

[ ]:
```

```