News Group Classification

Content :

What is the news group classification project ?

The aim of this project is to preprocess a group of articles splitted between classes using NLP techniques and train it using machine learning or deep learning to get a model capable of classifying future articles to its specific news group.

Preprocessing

The data set contain of 20 folders each folder represent a class and inside each folder group of articles in text format

1) Remove unwanted headers from each article in the dataset

Difine Path and Initialize Dictionary

In [ ]:
```
path = "C:\\Users\\Shehab\\Desktop\\NLP\\Dataset"
dic = {}
for folder in os.listdir(path):
    dic.update( {folder : []} )

```

Fill the Dictionary with articles and remove first 2 lines

In [ ]:
```

for folder in os.listdir(path):
    folder_path = os.path.join(path,folder)
    for file in os.listdir(folder_path):
        file_path = os.path.join(folder_path,file)
        with open(file_path, 'r') as f:
            contents = f.readlines()
            contents = contents[2:]
            tmp = dic[folder]
            tmp.append(contents)
            dic[folder] = tmp


```

2) convert the dataset to dataframe so that each row in the data frame consists of 2 columns article and class of the article and encode class labels.

From Dictionary to Data Frames

In [ ]:
```
dfs=[]
labels = []
i =0
for cls in dic:
    tmp_dic = {"article" : dic[cls] , "group" : [i]*len(dic[cls])}
    labels.append(cls)
    i+=1
    df = pd.DataFrame(tmp_dic)
    dfs.append(df)


all_dfs = pd.concat(dfs)
all_dfs.to_csv("All.csv" , index=False)
```

3) Tokenize each article in the dataframe using regex Tokenizer and remove stop words

Tokinize and preprocessing

In [ ]:
```
all_dfs = pd.read_csv("All.csv")
stop = list(stopwords.words('english'))
def tokenize(text):
    text = text.lower()
    text = text.replace("\\n", "")
    text = text.replace("\\", "")
    text = text.replace("'", "")
    pattern = r"^[a-z]+(-[a-z]+)*.?$"
    tokens = word_tokenize(text)
    tokens = [token for token in tokens if  re.match(pattern, token)]
    tokens = [x for x in tokens if x not in stop]

    return list(tokens)


all_dfs["article"] = all_dfs["article"].apply(tokenize)

```

4) Apply POS tagging to each article in the data frame then apply lemmatizing to each token and save the data frame in CSV format

Apply POS and Lemma

In [ ]:
```
tag_map = {
    "N": "n",
    "V": "v",
    "R": "r",
    "J": "a"
}

{token: "hello" , tag:"n"}
def pos_lemma(tokens):
    tags=pos_tag(tokens)
    lemmatizer=WordNetLemmatizer()
    w_net_lemma=[]
    for token ,tag in tags:
        if(tag_map.get(tag[0]) is not None ):
            w_net_lemma.append(lemmatizer.lemmatize(token,pos=tag_map[tag[0]]))
        else:
            w_net_lemma.append(lemmatizer.lemmatize(token,pos='n'))

    return list(w_net_lemma)

all_dfs["article"] = all_dfs["article"].apply(pos_lemma)
```

Save Data Frame

In [ ]:
```
all_dfs.to_csv("All.csv" , index=False)
```

Models Learning

1) Load saved CSV dataframe and split it to train and test with the percentage of 80% train and 20 % test

Load Data Frame

```
[13]: 1 all_dfs = pd.read_csv("All.csv")
```

Split Data to Train and Test

```
[14]: 1 train_x, valid_x, train_y, valid_y = train_test_split(all_dfs['article'], all_dfs['group'],test_size=0.2, random_state=42)
```

2) Apply TF-IDF on both train and test data frames and save the results in PK1 file and save tfidf_vect to joblib file

Apply TF-IDF on Train and Test Data

```
In [15]: 1 tfidf_vect = TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}', max_features=5000)
         2 tfidf_vect.fit(all_dfs['article'])
         3 xtrain_tfidf =  tfidf_vect.transform(train_x)
         4 xvalid_tfidf =  tfidf_vect.transform(valid_x)
```

Save Test data and its Labels

```
In [ ]: 1 joblib.dump(xvalid_tfidf, 'pk1/xvalid_tfidf.pk1')
        2 joblib.dump(valid_y, 'pk1/valid_y.pk1')
```

3) Declare train template function which take a model data and both train and test data then fit the train data using the given model and show the train/test accuracy plot during training and save model weights to PK1 file

Train Template

In [ ]:
```
def All_Models(model,model_name,x_train,y_train,x_test,y_test):
    #-----------------model fitting---------------
    history = model.fit(x_train,y_train)
    #----------Train Plot-----------------
    train_sizes, train_scores, test_scores = learning_curve(model, x_train, y_train, cv=5, n_jobs=-1, train_sizes=np.linspac
    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)
    test_mean = np.mean(test_scores, axis=1)
    test_std = np.std(test_scores, axis=1)
    plt.figure(figsize=(8, 6))
    #plt.plot(train_sizes, train_mean, 'o-', color='b', label='Training score')
    plt.plot(train_sizes, test_mean, 'o-', color='r', label='Cross-validation score')
    #plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.2, color='b')
    plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.2, color='r')
    plt.xlabel('Number of training samples')
    plt.ylabel('Score')
    plt.title('Learning Curve')
    plt.legend(loc='best')
    plt.show()
    # ------------------------------save model---------------------------------
    joblib.dump(model, f"pk1/{model_name}.pk1")
    ##########################################################################################################################

```

4) Pass models to the train template function

Naive Bayes

In [ ]:
```
model = naive_bayes.MultinomialNB(alpha=0.2)
All_Models(model,"Naive Bayes Multinominal",xtrain_tfidf,train_y,xvalid_tfidf,valid_y)
```

Logistice Regression

In [ ]:
```
model = linear_model.LogisticRegression(max_iter=10000000)
All_Models(model,"Logistic Regression",xtrain_tfidf,train_y,xvalid_tfidf,valid_y)
```

Support Vector Machine

In [ ]:
```
model = svm.SVC()
All_Models(model,"Support Vector Classifier",xtrain_tfidf,train_y,xvalid_tfidf,valid_y)
```

Random Forest

In [ ]:
```
model =RandomForestClassifier(n_estimators=100, random_state=42)
All_Models(model,"Random Forest Classifier",xtrain_tfidf,train_y,xvalid_tfidf,valid_y)
```

5) Train a neural network that consists of 7 layers of dropout to reduce overfitting and dense layers with relu function and classification layer with 20 class and softmax activation function. Compile the network using adam optimizer and train it then save the weights to H5 file

Neural Network

In [18]:
```
model = Sequential()
model.add(Dense(256, activation='relu', input_dim=xtrain_tfidf.shape[1]))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(20, activation='softmax'))
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
x = xtrain_tfidf.toarray()
x_valid = xvalid_tfidf.toarray()
model.fit(x, train_y, epochs=10, batch_size=32, verbose=1)
model.save("pk1/NN_model.h5")


```

Model evaluation

1) Load tf-idf test data and its labels from the PK1 files then declare an accuracy calc function that uses the passed model weights and information to calculate accuracy and plot confusion matrix using test data.

Load Test data and its Labels

In [3]:
```
x_test = joblib.load('pk1/xvalid_tfidf.pk1')
y_test=  joblib.load('pk1/valid_y.pk1')
```

Accuracy Calculation Template

In [4]:
```
def accuracy_calc(model,model_name):
    predictions = model.predict(x_test)
    print(f"Accuracy For {model_name} is = {metrics.accuracy_score(predictions, y_test)}")
    #---------------Confussion Matrix-----------------------
    cm = confusion_matrix(y_test, predictions)
    cm_percent = cm / cm.sum(axis=1, keepdims=True) * 100
    class_names = y_test.unique()
    plt.figure(figsize=(10, 10))
    sns.heatmap(cm_percent, annot=True, fmt=".1f", cmap="Blues")
    plt.xlabel("Predicted Labels")
    plt.ylabel("True Labels")
    plt.title("Confusion Matrix Heatmap")
    plt.xticks(np.arange(len(class_names)) + 0.5, class_names)
    plt.yticks(np.arange(len(class_names)) + 0.5, class_names)
    plt.show()
```

2) Load Models PK1 files and pass them to accuracy calc function to calculate accuracy and plot the confusion matrix

Naive Bayes Accuracy ¶

```
naive_bayes_model = joblib.load('pk1/Naive Bayes Multinominal.pk1')
accuracy_calc(naive_bayes_model ,"Naive Bayes Multinominal" )
#Accuracy For Naive Bayes Multinominal is = 0.8661710037174721
```

```
Accuracy For Naive Bayes Multinominal is = 0.8661710037174721
```

Logistic Regression Accuracy

```
logistic_regression_model = joblib.load('pk1/Logistic Regression.pk1')
accuracy_calc(logistic_regression_model ,"Logistic Regression" )
#Accuracy For Logistic Regression is = 0.8629845990440786
```

```
Accuracy For Logistic Regression is = 0.8629845990440786
```

Support Vector Machine Accuracy

```
svm_model = joblib.load('pk1/Support Vector Classifier.pk1')
accuracy_calc(svm_model ,"Support Vector Classifier" )
#Accuracy For Support Vector Classifier is = 0.8757302177376527
```

```
Accuracy For Support Vector Classifier is = 0.8757302177376527
```

Random Forest Accuracy

```
RNF_model = joblib.load('pk1/Random Forest Classifier.pk1')
accuracy_calc(RNF_model ,"Random Forest Classifier" )
#Accuracy For Random Forest Classifier is = 0.8066914498141264
```

```
Accuracy For Random Forest Classifier is = 0.8066914498141264
```

3) Use neural network weights saved in H5 file to evaluate the neural network on test data

Neural network Accuracy

```
NN_model = load_model("pk1/NN_model.h5")
x_test_reshaped = x_test.toarray()
loss, accuracy = NN_model.evaluate(x_test_reshaped, y_test)
print(f"Loss is  {loss}")
print(f"accuracy is {accuracy}")
#accuracy is 0.8903345465660095 (BEST)
```

Predictions :

1) Save any article to predict.txt file
2) Load the predict.txt file and save its content to string

```
def load_file_to_string(file_path):
    with open(file_path, 'r') as file:
        text = file.read()
    return text

file_path = 'predict.txt'
file_contents = load_file_to_string(file_path)

```

3) Tokenize the text and remove stop words

```
stop = list(stopwords.words('english'))
def tokenize(text):
    text = text.lower()
    text = text.replace("\\n", "")
    text = text.replace("\\", "")
    text = text.replace("'", "")
    pattern = r"^[a-z]+(-[a-z]+)*.?$"
    tokens = word_tokenize(text)
    tokens = [token for token in tokens if  re.match(pattern, token)]
    tokens = [x for x in tokens if x not in stop]

    return list(tokens)

text_token = tokenize(file_contents)
```

4) apply pos tagging and lemmatizing

```
tag_map = {
    "N": "n",
    "V": "v",
    "R": "r",
    "J": "a"
}

def pos_lemma(tokens):
    tags=pos_tag(tokens)
    lemmatizer=WordNetLemmatizer()
    w_net_lemma=[]
    for token ,tag in tags:
        if(tag_map.get(tag[0]) is not None ):
            w_net_lemma.append(lemmatizer.lemmatize(token,pos=tag_map[tag[0]]))
        else:
            w_net_lemma.append(lemmatizer.lemmatize(token,pos='n'))

    return list(w_net_lemma)

text_lemma = pos_lemma(text_token)
```

5) Load the tf-idf of the dataset and transform the text based on it

```
tfidf_vect = joblib.load("pk1/general_tfidf.joblib")
text_tfidf =  tfidf_vect.transform(text_lemma)

```

6) Initialize dictionary that have every class name and encoded number from the preprocessing phase

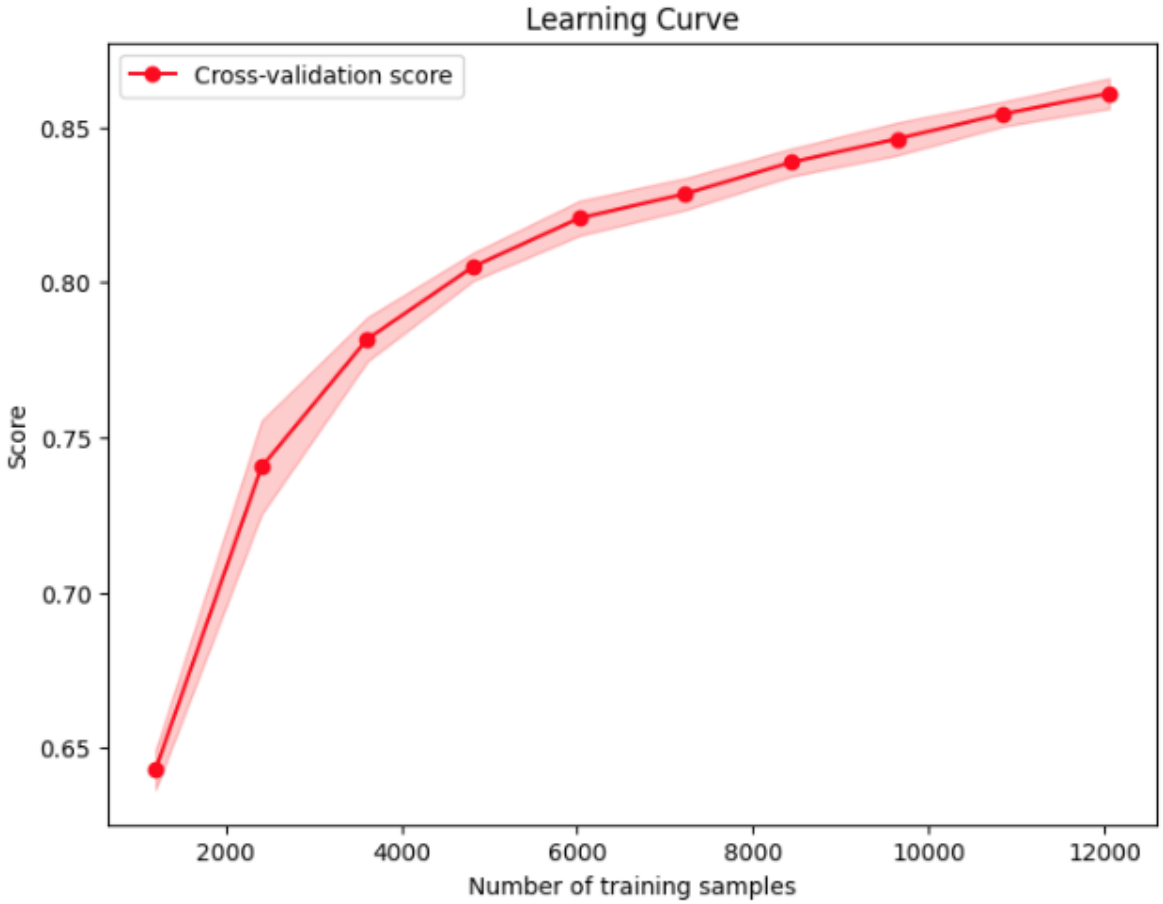
```
class_dict = {
    0: 'alt.atheism',
    1: 'comp.graphics',
    2: 'comp.os.ms-windows.misc',
    3: 'comp.sys.ibm.pc.hardware',
    4: 'comp.sys.mac.hardware',
    5: 'comp.windows.x',
    6: 'misc.forsale',
    7: 'rec.autos',
    8: 'rec.motorcycles',
    9: 'rec.sport.baseball',
    10: 'rec.sport.hockey',
    11: 'sci.crypt',
    12: 'sci.electronics',
    13: 'sci.med',
    14: 'sci.space',
    15: 'soc.religion.christian',
    16: 'talk.politics.guns',
    17: 'talk.politics.mideast',
    18: 'talk.politics.misc',
    19: 'talk.religion.misc'
}
```

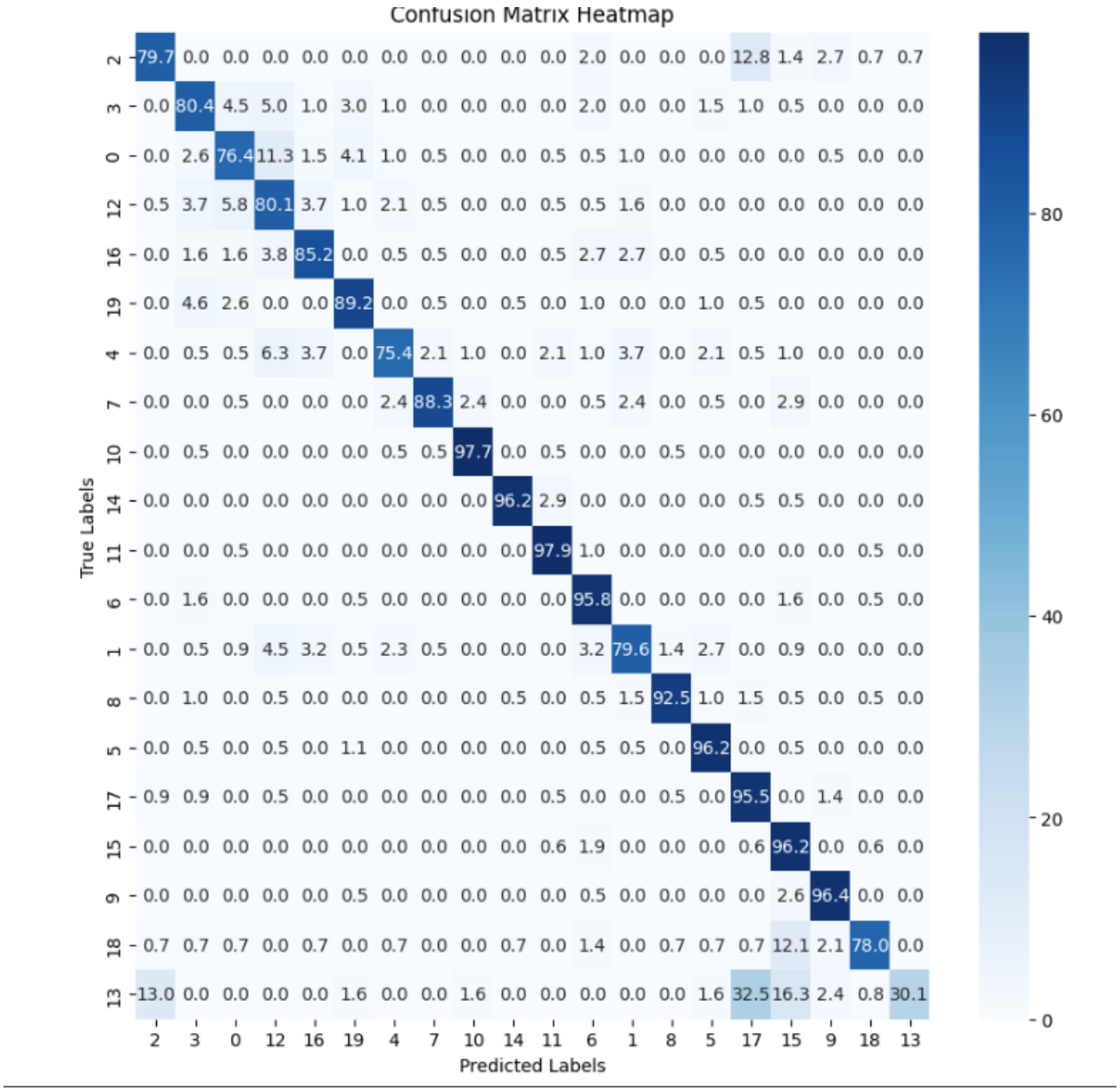
7) Load NN weights and predict the class based on it

```
NN_model = load_model("pk1/NN_model.h5")
text_tfidf_reshaped = text_tfidf.toarray()
pred = NN_model.predict(text_tfidf_reshaped)
print(f"Predictions for NN is {pred}")
```

Results

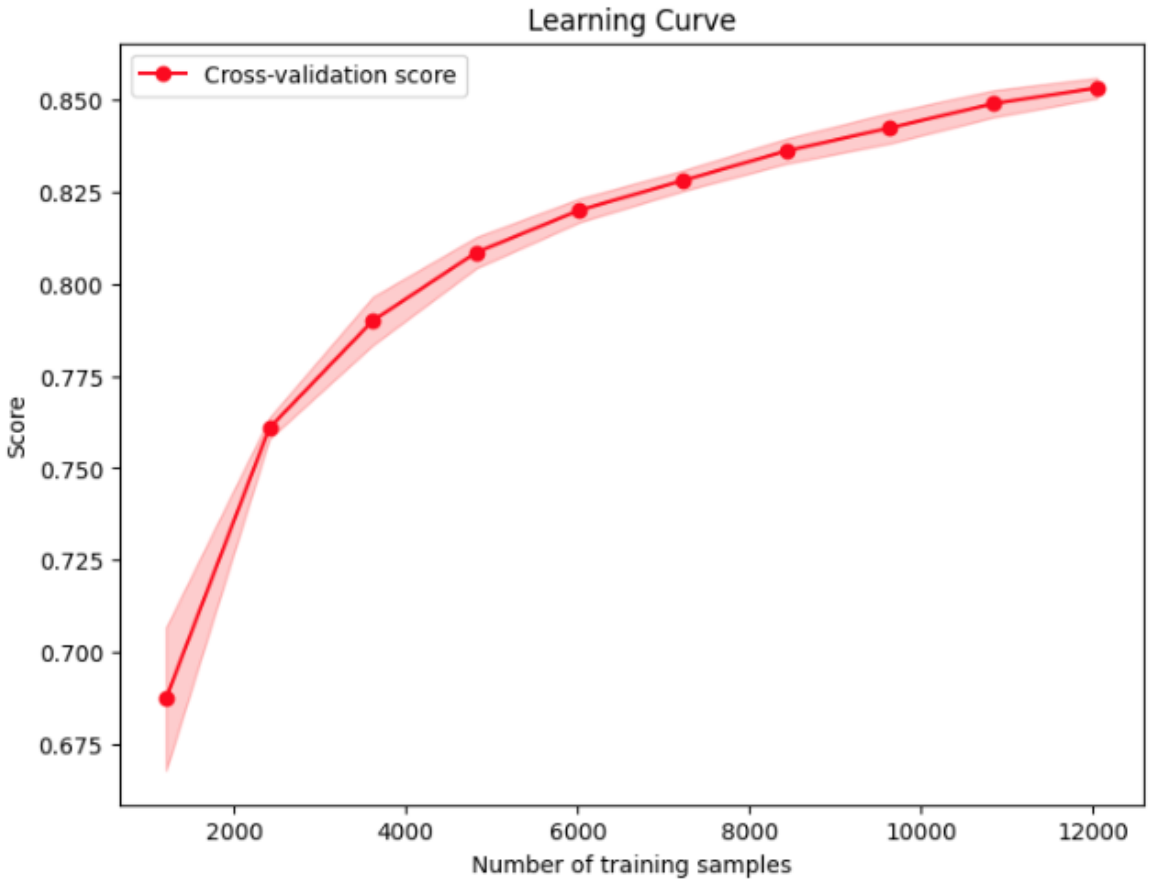
1) Naive bayes
   Accuracy is 86%

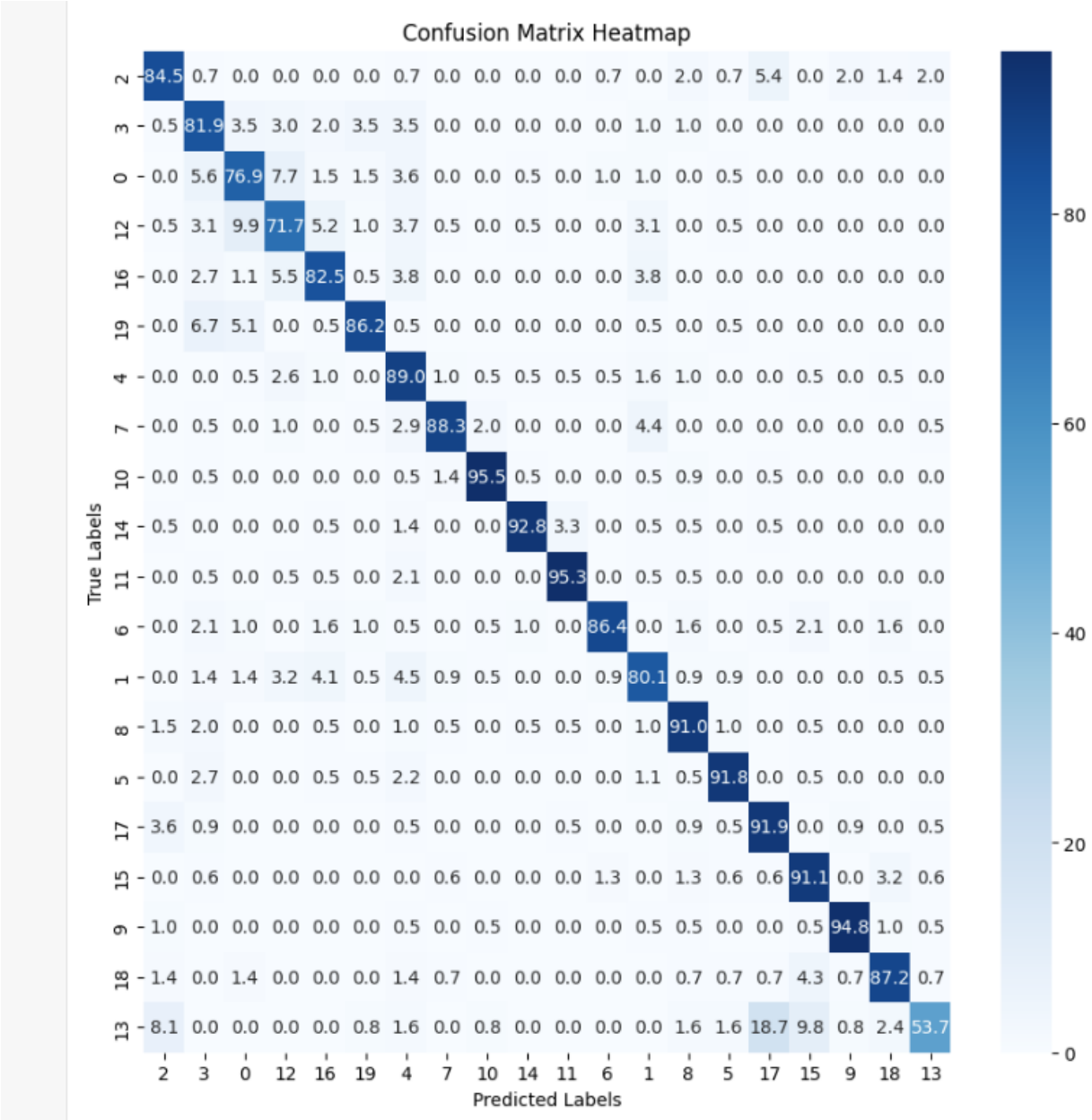

Learning Curve
Cross-validation score
0.85
0.80
0.75
0.70
0.65
Score
2000
4000
6000
8000
10000
12000
Number of training samples



2) Logistic Regression

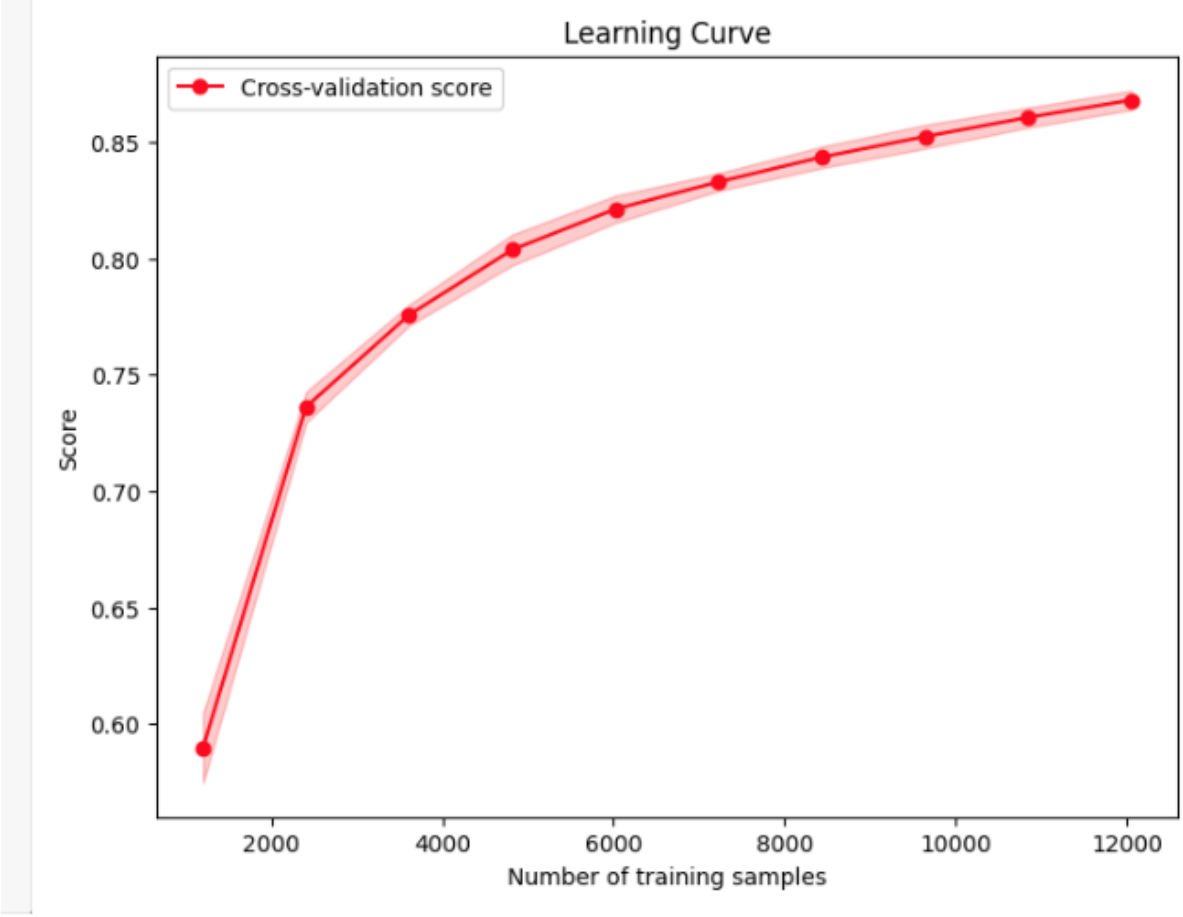
Accuracy is 86%

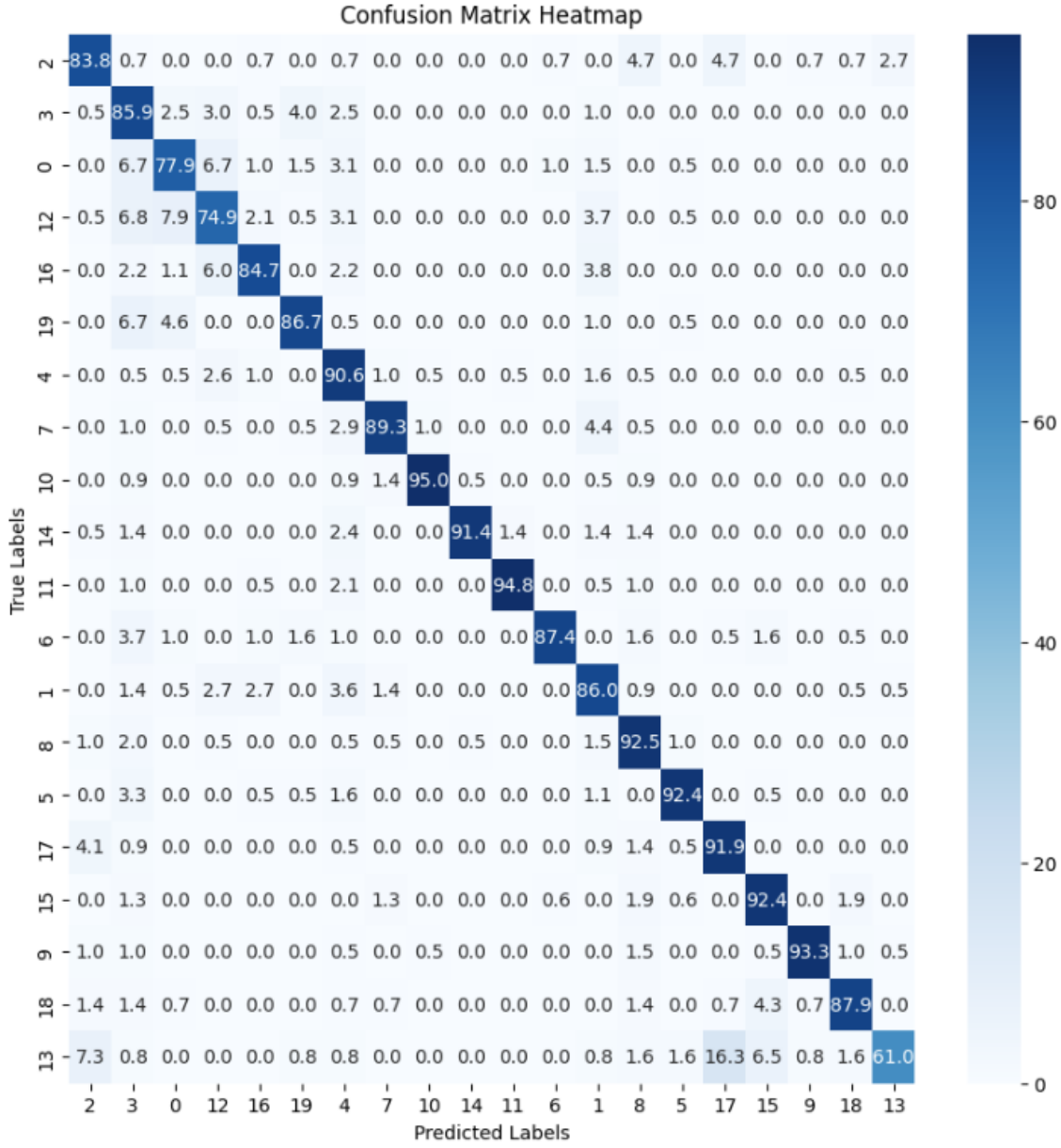

Learning Curve
Cross-validation score
0.850
0.825
0.800
0.775
0.750
0.725
0.700
0.675
Score
2000
4000
6000
8000
10000
12000
Number of training samples



3) Support Vector Machine

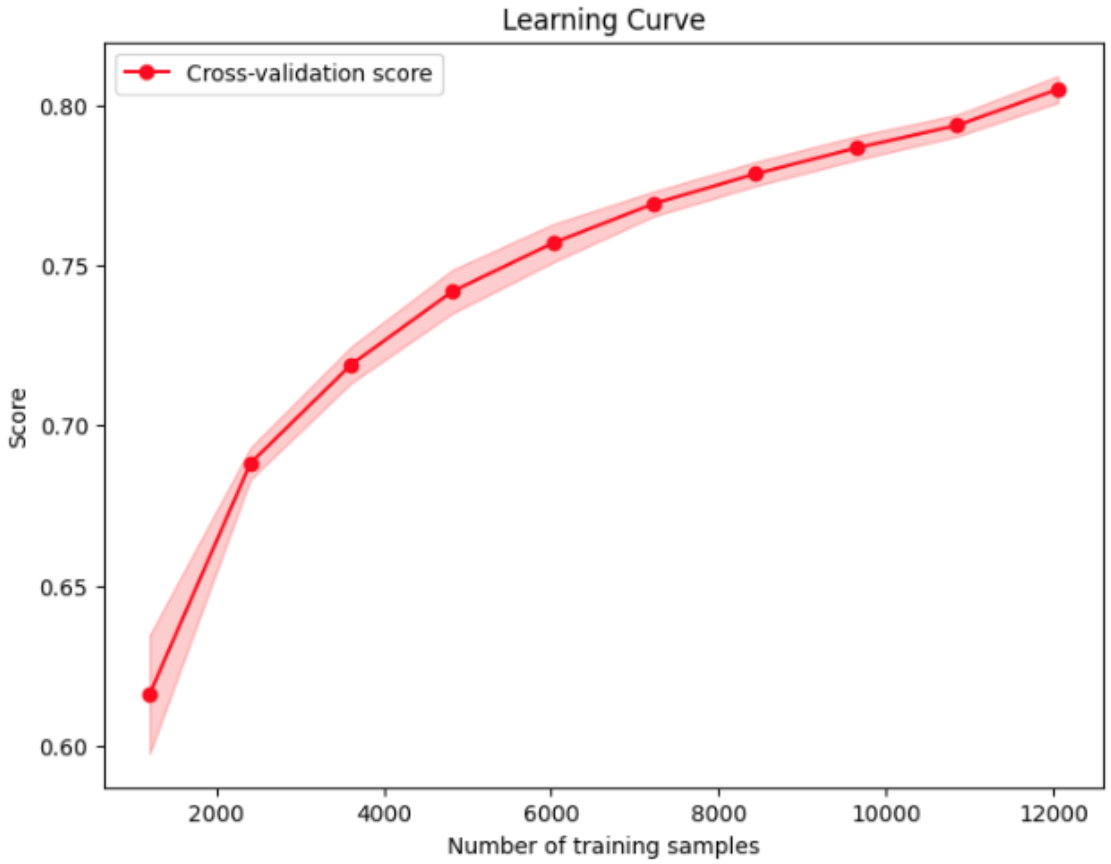
Accuracy is 87%



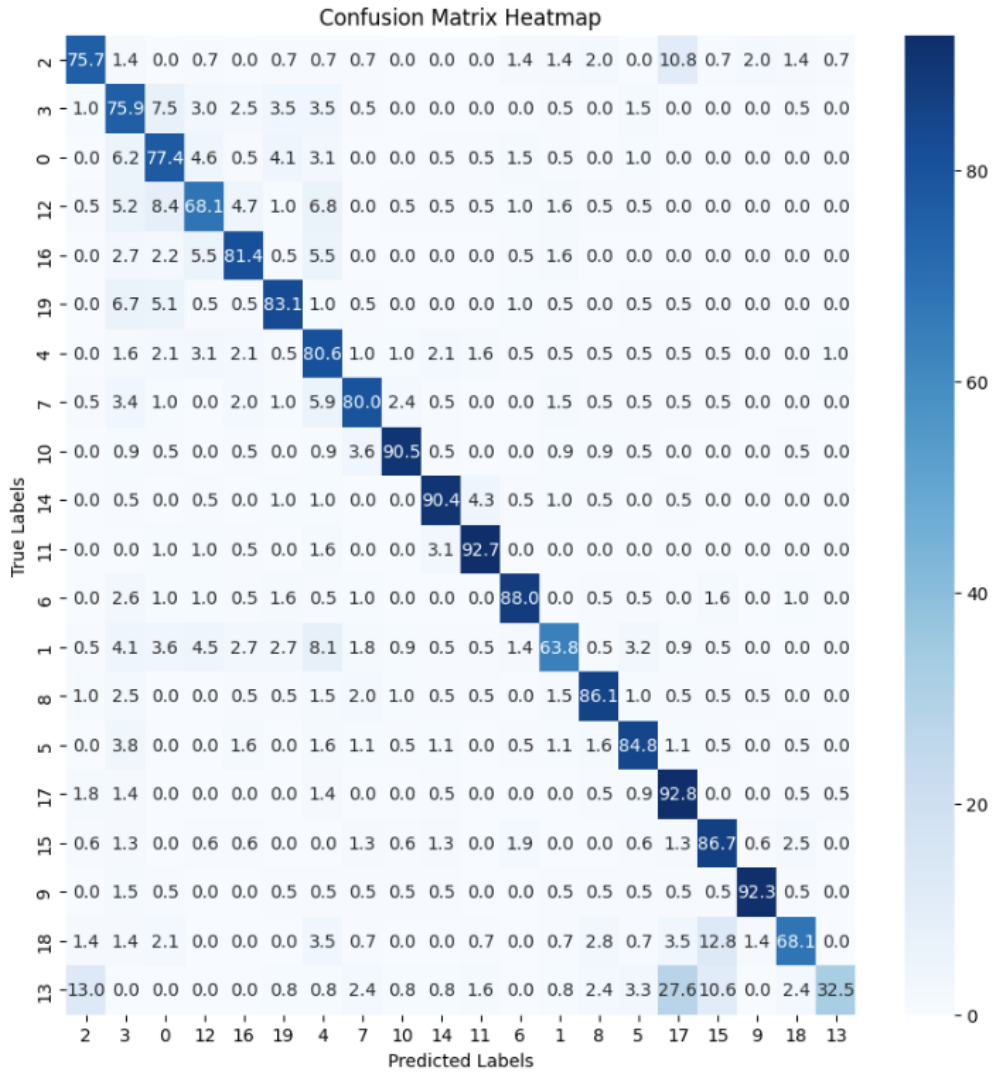

Confusion Matrix Heatmap
True Labels
Predicted Labels
2 3 0 12 16 19 4 7 10 14 11 6 1 8 5 17 15 9 18 13
2 83.8 0.7 0.0 0.0 0.7 0.0 0.7 0.0 0.0 0.0 0.0 0.7 0.0 4.7 0.0 4.7 0.0 0.7 0.7 2.7
3 0.5 85.9 2.5 3.0 0.5 4.0 2.5 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0 0.0 6.7 77.9 6.7 1.0 1.5 3.1 0.0 0.0 0.0 0.0 1.0 1.5 0.0 0.5 0.0 0.0 0.0 0.0 0.0
12 0.5 6.8 7.9 74.9 2.1 0.5 3.1 0.0 0.0 0.0 0.0 0.0 3.7 0.0 0.5 0.0 0.0 0.0 0.0 0.0
16 0.0 2.2 1.1 6.0 84.7 0.0 2.2 0.0 0.0 0.0 0.0 0.0 3.8 0.0 0.0 0.0 0.0 0.0 0.0 0.0
19 0.0 6.7 4.6 0.0 0.0 86.7 0.5 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.5 0.0 0.0 0.0 0.0 0.0
4 0.0 0.5 0.5 2.6 1.0 0.0 90.6 1.0 0.5 0.0 0.5 0.0 1.6 0.5 0.0 0.0 0.0 0.0 0.5 0.0
7 0.0 1.0 0.0 0.5 0.0 0.5 2.9 89.3 1.0 0.0 0.0 0.0 4.4 0.5 0.0 0.0 0.0 0.0 0.0 0.0
10 0.0 0.9 0.0 0.0 0.0 0.0 0.9 1.4 95.0 0.5 0.0 0.0 0.5 0.9 0.0 0.0 0.0 0.0 0.0 0.0
14 0.5 1.4 0.0 0.0 0.0 0.0 2.4 0.0 0.0 91.4 1.4 0.0 1.4 1.4 0.0 0.0 0.0 0.0 0.0 0.0
11 0.0 1.0 0.0 0.0 0.5 0.0 2.1 0.0 0.0 0.0 94.8 0.0 0.5 1.0 0.0 0.0 0.0 0.0 0.0 0.0
6 0.0 3.7 1.0 0.0 1.0 1.6 1.0 0.0 0.0 0.0 0.0 87.4 0.0 1.6 0.0 0.5 1.6 0.0 0.5 0.0
1 0.0 1.4 0.5 2.7 2.7 0.0 3.6 1.4 0.0 0.0 0.0 0.0 86.0 0.9 0.0 0.0 0.0 0.0 0.5 0.5
8 1.0 2.0 0.0 0.5 0.0 0.0 0.5 0.5 0.0 0.5 0.0 0.0 1.5 92.5 1.0 0.0 0.0 0.0 0.0 0.0
5 0.0 3.3 0.0 0.0 0.5 0.5 1.6 0.0 0.0 0.0 0.0 0.0 1.1 0.0 92.4 0.0 0.5 0.0 0.0 0.0
17 4.1 0.9 0.0 0.0 0.0 0.0 0.5 0.0 0.0 0.0 0.0 0.0 0.9 1.4 0.5 91.9 0.0 0.0 0.0 0.0
15 0.0 1.3 0.0 0.0 0.0 0.0 0.0 1.3 0.0 0.0 0.0 0.6 0.0 1.9 0.6 0.0 92.4 0.0 1.9 0.0
9 1.0 1.0 0.0 0.0 0.0 0.0 0.5 0.0 0.5 0.0 0.0 0.0 0.0 1.5 0.0 0.0 0.5 93.3 1.0 0.5
18 1.4 1.4 0.7 0.0 0.0 0.0 0.7 0.7 0.0 0.0 0.0 0.0 0.0 1.4 0.0 0.7 4.3 0.7 87.9 0.0
13 7.3 0.8 0.0 0.0 0.0 0.8 0.8 0.0 0.0 0.0 0.0 0.0 0.8 1.6 1.6 16.3 6.5 0.8 1.6 61.0
80
60
40
20
0

4) Random Forest

Accuracy is 80%



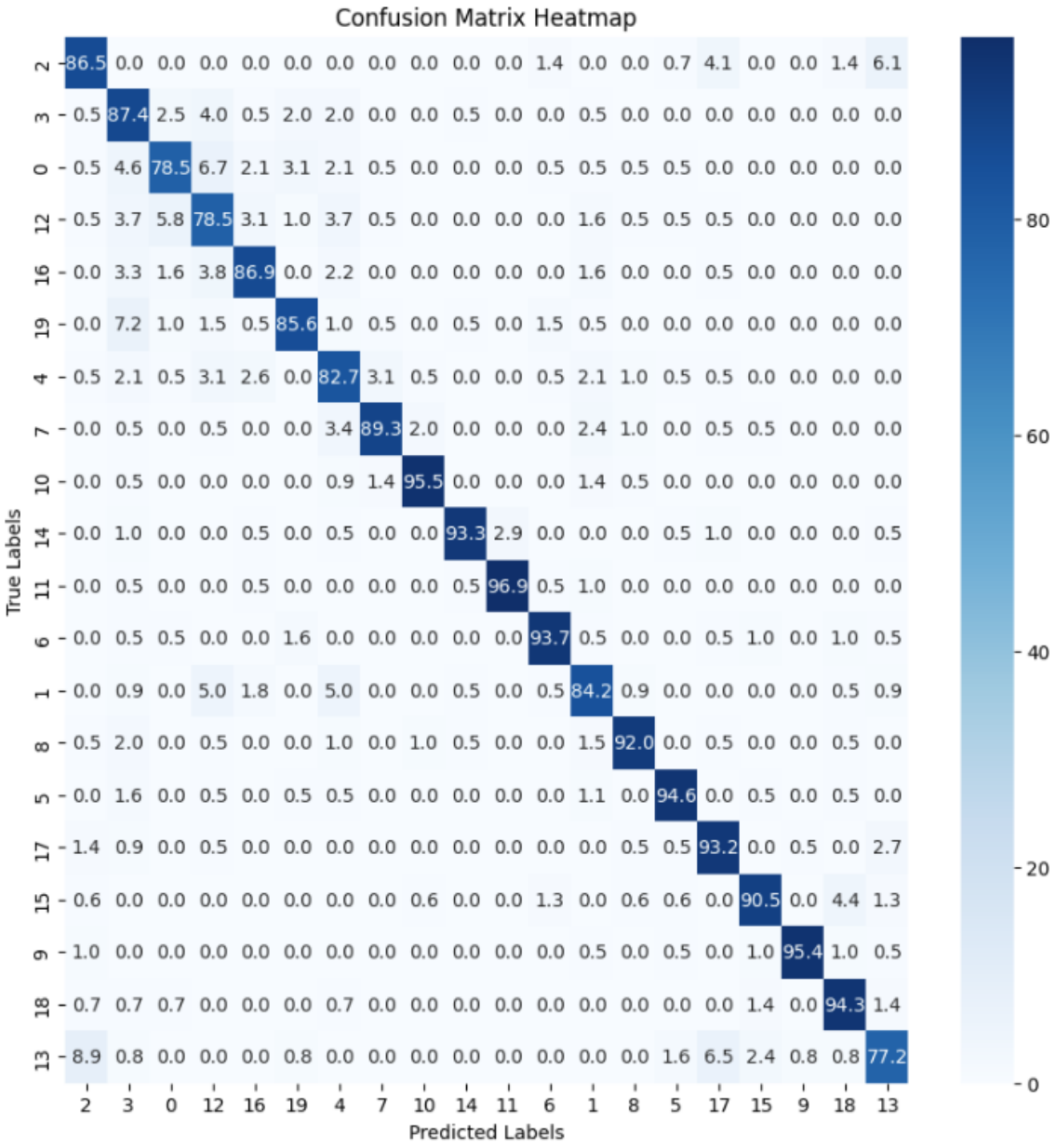
Accuracy For Random Forest Classifier is = 0.8066914498141264



5) Neural Network

Train Accuracy is 98%
Test Accuracy is 89% (BEST Accuracy)

```
118/118 [==============================] - 2s 5ms/step - loss: 0.6431 - accuracy: 0.8903
Loss is  0.6430635452270508
accuracy is 0.8903345465660095
```



Experiment

Using text = “computer consists of electrical components. I use windows 10 because i don't like windows 11”

The predictions from each model is :

```
1/1 [==============================] - 1s 544ms/step
Predictions for NN is comp.sys.ibm.pc.hardware
```