FCIS
AIN SHAMS
UNIVERSITY

Sentiment analysis of Movie reviews

Team Members:

ID	Name	Dep	Level
20201700556	عمرو سامي محمد عبد السلام	SC	4
20201700375	سهيل أبي محمد	SC	4
20191700203	حازم محمد أحمد عطية	SC	4
20201701261	حازم عبد الكريم سعد	SC	4
20201700036	أحمد خالد محمد عطا	SC	4
20201700312	زياد عصام الدين مصطفي	SC	4

Data Reading

-we use function ”read data” that reads all text files from multiple folders, assigns the folder name as the label for each file, and returns a pandas Data Frame with the text content and label for each file.

```
def read_data(folders):
    data = []
    for folder in folders:
        folder_path = os.path.join('C:\\Users\\AMR SAMI\\Desktop\\nlp project\\nlp project\\review_polarity\\txt_sentoken', folder)
        for file in os.listdir(folder_path):
            file_path = os.path.join(folder_path, file)
            with open(file_path, 'r') as f:
                text = f.read()
            label = folder
            data.append((text, label))
    df = pd.DataFrame(data, columns=['text', 'label'])
    return df
```

Data Preprocessing

- ***Convert-Text-to-Lower :***

  we decided to lower all the text

```
df['text'] = df['text'].apply(lambda x: x.lower())
```

- ***Removing punctuations :***

  We decided to remove all non-word characters,
  Such as Punctuation Marks.

```
df['text'] = df['text'].str.replace('[^\w\s]', '', regex=True)
```

- ***Stopword Removal :***

  Removes Common words that don’t carry much meaning like “and” , “The” , etc.

```
stop = stopwords.words('english')
df['text'] = df['text'].apply(lambda x: " ".join(x for x in x.split() if x not in stop))
```

- ***Stemming :***

  Reduces words to their root form. For example, ”running” And “runs” would both be reduced to “run”.

```
st = SnowballStemmer('english')
df['text'] = df['text'].apply(lambda x: " ".join([st.stem(word) for word in x.split()]))
```

- ***Lemmatization :***

  Similar to Stemming but more advanced. It reduces words their base or dictionary form. For example, ”am”, “are”, “is” all become “be”.

```
lemmatizer = WordNetLemmatizer()
df['text'] = df['text'].apply(lambda x: " ".join([lemmatizer.lemmatize(word) for word in x.split()]))
```

- ***Shuffling ,Splitting and Encoding Data :***

  -Randomizes the order of the data

  -Splits the dataset into Training and Validation sets

  -Converts categorical labels into numerical values

```
df = df.sample(frac=1, random_state=42)
train_x, valid_x, train_y, valid_y = model_selection.train_test_split( *arrays: df['text'], df['label'], test_size=test_size, random_state=42)
encoder = preprocessing.LabelEncoder()
train_y = encoder.fit_transform(train_y)
valid_y = encoder.transform(valid_y)
return train_x, valid_x, train_y, valid_y
```

Feature Extraction

- *We create the TF-IDF Vectorizer Object and set its parameters then we fit the vectorizer on the DataFrame text Column then define our function feature extraction, this function extract features as it takes input the dataframe "df" along with the train and validation text data and transform both of them into tf-idf feature matrices then returned for subsequent model training and evaluation then call it.*

```
# create a TF-IDF vectorizer object
tfidf_vect = TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}', max_features=5000)

# fit the vectorizer on the dataframe text column
tfidf_vect.fit(df['text'])

# function to extract features
1 usage
def extract_features(df, train_x, valid_x):
    xtrain_tfidf = tfidf_vect.transform(train_x)
    xvalid_tfidf = tfidf_vect.transform(valid_x)
    return xtrain_tfidf, xvalid_tfidf

# call the function with the desired arguments
xtrain_tfidf, xvalid_tfidf = extract_features(df, train_x, valid_x)
```

Training and Testing Models

- *We train many classifiers to find the best of them, the code include Naïve Bayes, SVM, Random Forest, Gradient Boosting, and Decision Tree.*

- *We train these Models on the transformed data using the "train_model" and prints the accuracy score of each classifier and Visualize our data to define the best performance by using "print_best_model" function.*

```
def train_model(classifier, feature_vector_train, label, feature_vector_valid, is_neural_net=False):
    classifier.fit(feature_vector_train, label)
    predictions = classifier.predict(feature_vector_valid)
    return metrics.accuracy_score(predictions, valid_y), classifier

# Naive Bayes training
accuracy1, naiveclf = train_model(naive_bayes.MultinomialNB(alpha=0.6), xtrain_tfidf, train_y, xvalid_tfidf)
print("Accuracy of Naive Bayes classifier is: ", accuracy1)

# SVM training
accuracy2, svmclf = train_model(svm.SVC(kernel='linear'), xtrain_tfidf, train_y, xvalid_tfidf)
print("Accuracy of SVM classifier is: ", accuracy2)

# Random Forest training
accuracy3, randomclf = train_model(RandomForestClassifier(n_estimators=100), xtrain_tfidf, train_y, xvalid_tfidf)
print("Accuracy of Random Forest classifier is: ", accuracy3)

# Gradient Boosting training
accuracy4, gradclf = train_model(GradientBoostingClassifier(), xtrain_tfidf, train_y, xvalid_tfidf)
print("Accuracy of Gradient Boosting classifier is: ", accuracy4)

# Decision Tree training
accuracy5, desclf = train_model(DecisionTreeClassifier(), xtrain_tfidf, train_y, xvalid_tfidf)
print("Accuracy of Decision Tree classifier is: ", accuracy5)
```

```
def print_best_model(models, accuracies):
    best_index = accuracies.index(max(accuracies))
    best_model = models[best_index]
    best_accuracy = accuracies[best_index]
    print(f"The best model is {best_model} with an accuracy of {best_accuracy}")
```

```
Accuracy of naive_bayes classifier is:  0.82
Accuracy of svm classifier is:  0.86
Accuracy of RandomForest classifier is:  0.8025
Accuracy of GradientBoosting classifier is:  0.7725
Accuracy of DecisionTree classifier is:  0.5975
The best model is SVM with an accuracy of 0.86
```

Data Visualization

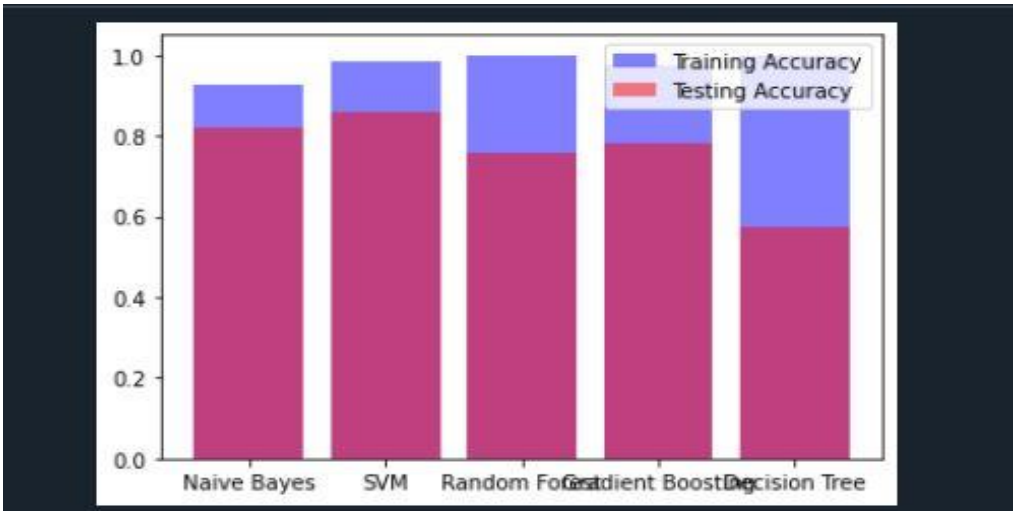
- *Then We Store our Scores and iterate over them and Visualize the data*

```
classifiers = [
    ('Naive Bayes', naiveclf),
    ('SVM', svmclf),
    ('Random Forest', randomclf),
    ('Gradient Boosting', gradclf),
    ('Decision Tree', desclf)
]

# initialize lists to store the accuracies
train_accs = []
test_accs = []

# loop through the classifiers
for name, clf in classifiers:
    # calculate the training accuracy
    train_acc = clf.score(xtrain_tfidf, train_y)
    train_accs.append(train_acc)
    # calculate the testing accuracy
    test_acc = clf.score(xvalid_tfidf, valid_y)
    test_accs.append(test_acc)

# create a bar chart to visualize the accuracies
x_pos = [i for i, _ in enumerate(classifiers)]
plt.bar(x_pos, train_accs, color='blue', alpha=0.5, label='Training Accuracy')
plt.bar(x_pos, test_accs, color='red', alpha=0.5, label='Testing Accuracy')
plt.xticks(x_pos, [name for name, _ in classifiers])
plt.legend()
plt.show()
```



Confusion Matrix

- *From the Function "print_best_model", We find that SVM is the best performing classifier to predict labels for the validation set*
- *We calculate our confusion matrix using scikit-learn's confusion matrix function and plots the confusion matrix using plot_confusion_matrix function.*

```
def plot_confusion_matrix(cm, classes, normalize=False, title='Confusion matrix', cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

# get the predictions for the validation set using the best model (SVM)
y_pred = svmclf.predict(xvalid_tfidf)

# calculate the confusion matrix
cm = confusion_matrix(valid_y, y_pred)

# plot the confusion matrix
plot_confusion_matrix(cm, classes=['neg', 'pos'], normalize=True, title='Confusion Matrix')
```