In [1]:
```
import multilabel_knn as mlk # Multilabel KNN
import evaluation
import pandas as pd
import numpy as np
import re # regex
import nltk # Tokenizing, Stopwords
from sklearn.feature_extraction.text import TfidfVectorizer #TF-IDF
from sklearn.preprocessing import MultiLabelBinarizer #One Hot Encoding
from sklearn.model_selection import train_test_split # Train test split
```

In [2]:
```
#Disease Summary contains the following columns:
# Disease: Name of the Disease
# Symptoms: comma seperated symptoms text
# Treatment: comma seperated treatment text
# Summary: Mayo Clinic small paragraph of disease
disease_summary = pd.read_excel(open('respiratory_symptoms_and_treatment.xls
              sheet_name='Data1')

conversations = pd.read_csv("RES_Conversations.csv", index_col=0)
```

In [3]:
```
# Function to clean summary text
# Removed parenthesis text ()
# Removed commas, periods
# Lower Case
# Remove stop words
def clean_summary(text):
    text = text.lower()
    text=re.sub("\(.*?\)","",text)
    text=re.sub("\.","",text)
    text=re.sub(",","",text)
    text = text.lower()
    tokens = wpt.tokenize(text)
    filtered_tokens = [token for token in tokens if token not in stop_words]
    text = " ".join(filtered_tokens)
    return(text)
```

In [14]:
```
# Creating list of stop words
# Clinical stop: List from txt file that is a list of clinical stop words
clinical_stop = open('/Users/rickytrujillo/Desktop/School Files/CPP/SP24/CIS
clinical_stop = clinical_stop.read()
clinical_stop = clinical_stop.split("\n")
del clinical_stop[0]

# Adding extra list of stop words found in conversations that can be removed
extra_words = ["uhm","um", "yeah", "ah", "hi", "need", "uh","'", "okay", "OK
clinical_stop.extend(extra_words)

# Stop words from nltk package
stop_words = nltk.corpus.stopwords.words("english")
stop_words.extend(clinical_stop)

#Defining Tokenizer
wpt = nltk.WordPunctTokenizer()
```

In [11]:
```
#Pre-Processing Disease Summary Document
# Lowercasing Disease, Symptoms, and Treatment columns
# Changing Summary Column into a tokenized version after its been cleaned fr
# above function
for i in range(len(disease_summary)):
    disease_summary.loc[i,"Disease"] = disease_summary.loc[i,"Disease"].lowe
    disease_summary.loc[i,"Symptoms"] = disease_summary.loc[i,"Symptoms"].lo
    disease_summary.loc[i,"Treatment "] = disease_summary.loc[i,"Treatment "
    disease_summary.loc[i,"Summary"] = clean_summary(disease_summary.loc[i,"
```

In [16]:
```
# This is the TF-IDF vectorizer with the following parameters
# max_features: capping # of final features to ouput
# min_df: excluses anything less than this value (cannot be lower than 0.0)
# min_df: exclused anything greater than this value (cannot be greater than
# Note: min_df < max_df
#use_idf = to do the idf portion in process
# ngram_range: allows us to explore unigrams and bigrams
tfv = TfidfVectorizer(max_features=200, min_df=0.3, max_df=0.8, use_idf=True
tfv_matrix = tfv.fit_transform(conversations["Conversation"].values.astype('
tfv_matrix = tfv_matrix.toarray()
vocab = tfv.get_feature_names_out().tolist()
tfv_df = pd.DataFrame(np.round(tfv_matrix,5), columns=vocab)
```

In [18]:
```
# i iterates through the conversations
# j iterates through the diseases
# k iterates through the list of ordered words
# So for each text document, we are going through all the diseases in our li
# many of the 200 words are present

text_disease_mat = pd.DataFrame()
for i in range(len(tfv_df)): #len(tfv_df)
    disease_score = [] #appends the scores for each of the 16 diseases and r
    ordered_list = tfv_df.iloc[i].sort_values(ascending=False)
    for j in range(len(disease_summary)):
        score1=0 #when words match the disease name
        score2=0 #when words match the symptoms list
        score3=0 #when words match the treatment list
        score4=0 #when words match the normalized summary
        for k in range(len(ordered_list)):
            # Adds the weight/score associated to each word if it matches th
            if ordered_list.index[k] in disease_summary.loc[j,"Disease"]:
                score1 = score1 + ordered_list.values[k]
            if ordered_list.index[k] in disease_summary.loc[j,"Symptoms"]:
                score2 = score2 + ordered_list.values[k]
            if ordered_list.index[k] in disease_summary.loc[j,"Treatment "]:
                score3 = score3 + ordered_list.values[k]
            if ordered_list.index[k] in disease_summary.loc[j,"Summary"]:
                score4 = score4 + ordered_list.values[k]
            # Sum of all the scores for the jth disease
            score= score1 +score2+score3+score4
        disease_score.append(score)
    disease_score = pd.DataFrame(disease_score).T
    # concatenate the empty dataframe with the newly developed disease_score
    text_disease_mat = pd.concat([text_disease_mat, disease_score])
# rename columns to be the names of the diseases
text_disease_mat.rename(columns = disease_summary["Disease"], inplace = True
text_disease_mat= text_disease_mat.reset_index(drop=True) #reset index
```

In [20]:
```
#Initializing all entries in the Disease
tfv_df["Disease_Label"] = ""
# Setting the column as an object type so that I can make each cell in the c
tfv_df["Disease_Label"] = tfv_df["Disease_Label"].astype("object")
# For each text, I am going to grab the disease names corresponding to the t
# in the Disease_Label column corresponding to that text
for i in range(len(tfv_df)):
    max_3 = text_disease_mat.iloc[i].sort_values(ascending=False)[0:3].index
    tfv_df.at[i,"Disease_Label"] = max_3
```

In [22]:
```
#Defining the Multi Label One Hot Encoder object
# One-Hot Encoding allows me to set a 1 for the disease name if it is presen
# or 0 if it is not present in the top 3 list
mlb = MultiLabelBinarizer(classes= disease_summary["Disease"].tolist())
dummy_df = pd.DataFrame(mlb.fit_transform(tfv_df['Disease_Label']),columns=m
final_df = pd.concat([tfv_df, dummy_df], axis=1)
```

In [24]:
```
# Doing an 80-20 split
# 80% of the data is dedicated to training the model
# 20% of the data is dedicated to evaluating the model on unseen data
X_train,X_test, y_train, y_test = train_test_split(tfv_matrix,dummy_df, test
```

In [32]:
```
# Checking shapes of datasets
print("Original Size: ", tfv_matrix.shape, dummy_df.shape) #original: size 2
print("Training Size: ", X_train.shape, y_train.shape) #train: size 170
print("Testing Size: ", X_test.shape, y_test.shape) #test: size 43
```

```
Original Size:  (213, 186) (213, 19)
Training Size:  (170, 186) (170, 19)
Testing Size:  (43, 186) (43, 19)
```

In [26]:
```
# Model is a Multi-Label KNN
# Rather than a normal KNN where it is fed in a single label, here we feed i
model = mlk.multilabel_kNN(k=10, metric = "euclidean")
```

In [27]:
```
# Changing the format of labels to be compatible with function
y_train = y_train.to_numpy()

#fitting the model with the TF_IDF features, and disease labels
model.fit(X_train, y_train)
```

Out[27]:
```
<multilabel_knn.multilabel_kNN at 0x138630a50>
```

In [29]:
```
# Using the model to predict unseen data (test set)
# The first one just gives us the predictions
diseases= disease_summary["Disease"]
Y_pred = model.predict(X_test)
Y_pred = pd.DataFrame.sparse.from_spmatrix(Y_pred, columns=diseases)
# This second one gives us the associated probability of the respective dise
# Not really needed unless interested in seeing the probabilities
# requires knn.py python file
Y_prob = model.predict(X_test,return_prob=True)
```

In [30]:
```
from sklearn.metrics import multilabel_confusion_matrix, classification_repo
diseases = disease_summary["Disease"].tolist()
print(classification_report(y_test, Y_pred,target_names=diseases))
```

	precision	recall	f1-score	support
acute respiratory distress syndrome	0.67	0.29	0.40	7
asbestosis	1.00	0.28	0.43	18
aspergillosis	0.00	0.00	0.00	0
asthma	0.78	0.58	0.67	12
bronchiolitis	0.00	0.00	0.00	0
bronchitis	0.00	0.00	0.00	0
chronic bronchitis	0.00	0.00	0.00	0
chronic obstructive pulmonary disease	0.75	0.18	0.29	17
influenza	0.50	0.11	0.18	9
pneumonia	1.00	0.20	0.33	5
pneumothorax	0.00	0.00	0.00	1
respiratory syncytial virus	0.00	0.00	0.00	1
tuberculosis	0.00	0.00	0.00	3
bronchopulmonary dysplasia (bpd)	0.00	0.00	0.00	1
cystic fibrosis	0.00	0.00	0.00	0
pulmonary fibrosis	0.53	1.00	0.70	23
sleep apnea	0.00	0.00	0.00	3
covid	0.61	1.00	0.76	22
strep throat	1.00	0.71	0.83	7
micro avg	0.64	0.53	0.58	129
macro avg	0.36	0.23	0.24	129
weighted avg	0.67	0.53	0.51	129
samples avg	0.65	0.53	0.58	129

```
/Users/rickytrujillo/anaconda3/lib/python3.11/site-packages/sklearn/metrics/
_classification.py:1469: UndefinedMetricWarning: Precision and F-score are i
ll-defined and being set to 0.0 in labels with no predicted samples. Use `ze
ro_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/rickytrujillo/anaconda3/lib/python3.11/site-packages/sklearn/metrics/
_classification.py:1469: UndefinedMetricWarning: Recall and F-score are ill-
defined and being set to 0.0 in labels with no true samples. Use `zero_divis
ion` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [31]:

```
from sklearn.metrics import hamming_loss
from statistics import mean

y_test = y_test.reset_index(drop=True)

# The Hamming loss is the fraction of labels that are incorrectly predicted
hamming_loss_list = []
for i in range(len(Y_pred)):
    #print(y_test.loc[i].tolist(), "\n", Y_pred.loc[i].tolist())
    hamming_loss_val = hamming_loss(y_test.loc[i], Y_pred.loc[i])
    hamming_loss_list.append(hamming_loss_val)

print("Individual Hamming Loss:","\n", hamming_loss_list, "\n")
print("Average Hamming Loss:","\n", mean(hamming_loss_list))
```

```
Individual Hamming Loss:
 [0.21052631578947367, 0.15789473684210525, 0.05263157894736842, 0.157894736
84210525, 0.10526315789473684, 0.15789473684210525, 0.21052631578947367, 0.1
5789473684210525, 0.15789473684210525, 0.15789473684210525, 0.21052631578947
367, 0.05263157894736842, 0.21052631578947367, 0.10526315789473684, 0.052631
57894736842, 0.10526315789473684, 0.10526315789473684, 0.10526315789473684,
0.15789473684210525, 0.0, 0.2631578947368421, 0.10526315789473684, 0.1578947
3684210525, 0.15789473684210525, 0.05263157894736842, 0.05263157894736842,
0.15789473684210525, 0.10526315789473684, 0.10526315789473684, 0.05263157894
736842, 0.10526315789473684, 0.10526315789473684, 0.10526315789473684, 0.105
26315789473684, 0.05263157894736842, 0.10526315789473684, 0.0526315789473684
2, 0.05263157894736842, 0.15789473684210525, 0.2631578947368421, 0.157894736
84210525, 0.05263157894736842, 0.05263157894736842]

Average Hamming Loss:
 0.12117503059975519
```