IE 7275 Data Mining in Engineering

Project Title: Disease Prediction

Final Project Report

Group: 12

Student 1: Ankur Pandey + 1857-3908-5740 pandey.anku@northeastern.edu

Student 2: Kajaal Shah +1857-390-5787 Shah.kaa@northeastern.edu

Student 3: Sneha Amin +1857-398-5938 amin.sn@ northeastern.edu

Percentage of Effort Contributed by Student 1: 100%

Percentage of Effort Contributed by Student 2: 100%

Percentage of Effort Contributed by Student 3: 100%

Signature of Student 1: Sneha Amin

Signature of Student 2: Kaajal Shah

Signature of Student 3: Ankur Pandey

Submission Date: April 23 2024

Problem Setting:

We deal with an often-overwhelming volume of medical data in the healthcare industry. It's not simple to go through this massive amount of data, find patterns, and use those patterns to make precise disease predictions. Currently, the process of diagnosing illnesses can be laborious and even inaccurate. Therefore, it's imperative to figure out how to streamline this procedure. Through the utilization of data analytics and predictive modeling, we have the opportunity to fundamentally alter the way in which diseases are diagnosed. Better patient outcomes will result from healthcare professionals being able to administer treatments more quickly and confidently.

Problem Definition:

We aim to analyze vast troves of medical data, pinpointing recurring trends, and leveraging this insight to anticipate and enhance the precision and efficiency of disease diagnosis. By doing so, we empower healthcare professionals to deliver superior care and treatment to patients. This approach enables us to substantially streamline the entire process, fostering improved outcomes for all involved.

Data Sources**:**

Few sources we would explore and use the data are:

1. Disease Prediction (kaggle.com)

Data Description:

The dataset includes 133 columns and 131 records with following features:

1. The Complete Dataset includes two CSV files - one for training your model and the other for testing it.
2. Each CSV file contains 133 columns, with 132 columns representing different symptoms experienced by a person and the last column indicating the prognosis.
3. These symptoms are linked to 42 different diseases, which can be classified based on the set of symptoms.

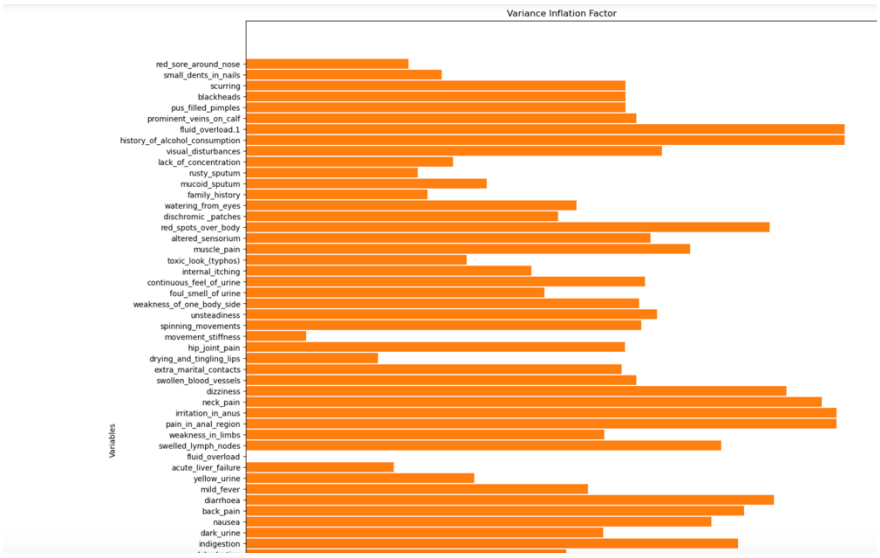
Data Exploration:

- Data information:

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_tongue	...	scurring	skin_p
0	1	1	1	0	0	0	0	0	0	0	...	0	
1	0	1	1	0	0	0	0	0	0	0	...	0	
2	1	0	1	0	0	0	0	0	0	0	...	0	
3	1	1	0	0	0	0	0	0	0	0	...	0	
4	1	1	1	0	0	0	0	0	0	0	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
4915	0	0	0	0	0	0	0	0	0	0	...	0	
4916	0	1	0	0	0	0	0	0	0	0	...	1	
4917	0	0	0	0	0	0	0	0	0	0	...	0	
4918	0	1	0	0	0	0	1	0	0	0	...	0	
4919	0	1	0	0	0	0	0	0	0	0	...	0	

4920 rows × 134 columns

- Computing the Variable Inflation Factor:



From the correlation matrix, We see a significant correlation between

- The code calculates the VIF values for each variable in a dataset.
- VIF helps assess the degree of multicollinearity in a linear regression model.
- Other variables with relatively high VIF values include "med_cons_in_stats", "no_sexual_cases", "med_sexual_funcs", "sexual_opening_cases", and "not_sure_sexual_times".
- Variables like "control_sexual_intrnc", "movement_dffcncs", "eye_and_hrtng_vhs", and "seizures" have relatively lower VIF values, as indicated by shorter bar lengths
- The VIF values range from approximately 1 (for variables with the shortest bars) to around 10 (for variables with the longest bars).

Data Mining Tasks:

Our dataset consists of categorical data which was already encoded into binary format (0's and 1's), only few values were missing. Additionally, we had balanced data, which contributed to the robustness of our models during training and evaluation.

```
In [5]: training
```

Out[5]:

...	scurring	skin_peeling	silver_like_dusting	small_dents_in_nails	inflammatory_nails	blister	red_sore_around_nose	yellow_crust_ooze	prognosis	Unnamed: 133
...	0	0	0	0	0	0	0	0	Fungal infection	NaN
...	0	0	0	0	0	0	0	0	Fungal infection	NaN
...	0	0	0	0	0	0	0	0	Fungal infection	NaN
...	0	0	0	0	0	0	0	0	Fungal infection	NaN
...	0	0	0	0	0	0	0	0	Fungal infection	NaN
...	...	...	...	...	...	...	...	...	...	...
...	0	0	0	0	0	0	0	0	(vertigo) Paroymsal Positional Vertigo	NaN
...	1	0	0	0	0	0	0	0	Acne	NaN
...	0	0	0	0	0	0	0	0	Urinary tract infection	NaN
...	0	1	1	1	1	0	0	0	Psoriasis	NaN
...	0	0	0	0	0	1	1	1	Impetigo	NaN

```
In [6]: del training[training.columns[-1]]
        training.head()
```

Out[6]:

...	blackheads	scurring	skin_peeling	silver_like_dusting	small_dents_in_nails	inflammatory_nails	blister	red_sore_around_nose	yellow_crust_ooze	prognosis
...	0	0	0	0	0	0	0	0	0	Fungal infection
...	0	0	0	0	0	0	0	0	0	Fungal infection
...	0	0	0	0	0	0	0	0	0	Fungal infection
...	0	0	0	0	0	0	0	0	0	Fungal infection
...	0	0	0	0	0	0	0	0	0	Fungal infection

```
In [8]: training.describe()
```

Out[8]:

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_to
count	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.00
mean	0.137805	0.159756	0.021951	0.045122	0.021951	0.162195	0.139024	0.045122	0.045122	0.02
std	0.344730	0.366417	0.146539	0.207593	0.146539	0.368667	0.346007	0.207593	0.207593	0.14
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00

8 rows × 132 columns

```
In [9]: training.isnull().sum()
```

Out[9]:

```
itching                0
skin_rash              0
nodal_skin_eruptions   0
continuous_sneezing    0
shivering              0
                      ..
inflammatory_nails     0
blister                0
red_sore_around_nose   0
yellow_crust_ooze      0
prognosis              0
Length: 133, dtype: int64
```

```
In [10]: training.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4920 entries, 0 to 4919
Columns: 133 entries, itching to prognosis
dtypes: int64(132), object(1)
memory usage: 5.0+ MB
```

```
In [11]: training.dropna()
```

Out[11]:

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_tongue	...	blackheads	scu
0	1	1	1	0	0	0	0	0	0	0	...	0	
1	0	1	1	0	0	0	0	0	0	0	...	0	
2	1	0	1	0	0	0	0	0	0	0	...	0	
3	1	1	0	0	0	0	0	0	0	0	...	0	
4	1	1	1	0	0	0	0	0	0	0	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
4915	0	0	0	0	0	0	0	0	0	0	...	0	
4916	0	1	0	0	0	0	0	0	0	0	...	1	
4917	0	0	0	0	0	0	0	0	0	0	...	0	
4918	0	1	0	0	0	0	1	0	0	0	...	0	
4919	0	1	0	0	0	0	0	0	0	0	...	0	

4920 rows × 133 columns

Displaying the training dataset. Dropping the last column as it has Nan value.

Checking Null Values :

```
In [7]: training.isnull().any()

Out[7]: itching                 False
        skin_rash               False
        nodal_skin_eruptions    False
        continuous_sneezing     False
        shivering               False
                                ...
        inflammatory_nails      False
        blister                 False
        red_sore_around_nose    False
        yellow_crust_ooze       False
        prognosis               False
        Length: 133, dtype: bool
```

We don’t have any Null Values now.

```
In [13]: X=training.iloc[:,:-1]

In [14]: X
```

Out[14]:

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_tongue	...	pus_filled_pimple
0	1	1	1	0	0	0	0	0	0	0	...	
1	0	1	1	0	0	0	0	0	0	0	...	
2	1	0	1	0	0	0	0	0	0	0	...	
3	1	1	0	0	0	0	0	0	0	0	...	
4	1	1	1	0	0	0	0	0	0	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
4915	0	0	0	0	0	0	0	0	0	0	...	
4916	0	1	0	0	0	0	0	0	0	0	...	
4917	0	0	0	0	0	0	0	0	0	0	...	
4918	0	1	0	0	0	0	1	0	0	0	...	
4919	0	1	0	0	0	0	0	0	0	0	...	

4920 rows × 132 columns

```
In [16]: Y = training.iloc[:,-1]

In [17]: Y.value_counts()

Out[17]: prognosis
         Fungal infection                          120
         Hepatitis C                               120
         Hepatitis E                               120
         Alcoholic hepatitis                       120
         Tuberculosis                              120
         Common Cold                               120
         Pneumonia                                 120
         Dimorphic hemmorhoids(piles)              120
         Heart attack                              120
         Varicose veins                            120
         Hypothyroidism                            120
         Hyperthyroidism                           120
         Hypoglycemia                              120
         Osteoarthristis                           120
         Arthritis                                 120
         (vertigo) Paroymsal  Positional Vertigo   120
         Acne                                      120
         Urinary tract infection                   120
         Psoriasis                                 120
         Hepatitis D                               120
         Hepatitis B                               120
         Allergy                                   120
         hepatitis A                               120
         GERD                                      120
         Chronic cholestasis                       120
         Drug Reaction                             120
         Peptic ulcer diseae                       120
         AIDS                                      120
         Diabetes                                  120
         Gastroenteritis                           120
         Bronchial Asthma                          120
         Hypertension                              120
         Migraine                                  120
         Cervical spondylosis                      120
         Paralysis (brain hemorrhage)              120
         Jaundice                                  120
         Malaria                                   120
         Chicken pox                               120
         Dengue                                    120
         Typhoid                                   120
         Impetigo                                  120
         Name: count, dtype: int64

In [18]: label_encoder = LabelEncoder()
         Y_encoded = label_encoder.fit_transform(Y)

In [19]: Y_encoded

Out[19]: array([15, 15, 15, ..., 38, 35, 27])
```

- Checking the null values per row
- Check the description of DataFrame
- Checking if any value is missing per variable (column)
- Displaying basic information of DataFrame
- Remove the rows with null values
- Separating the Dataset and Results as X & Y
- Encoding Object type in Y

FEATURE SELECTION:

Train-Test Split:

- The code splits the dataset into training, validation, and test sets.
- It ensures that the model is trained on one portion of the data, validated on another, and tested on a separate portion.
- This helps evaluate the model's performance and generalization ability.

Feature Selection with SelectKBest:

- The code uses the SelectKBest method to select the top 40 features.
- These features are likely the most relevant for the classification task.
- Feature selection reduces dimensionality and improves model efficiency.

```
In [24]: from sklearn.model_selection import train_test_split
         X_train, X_temp, Y_train, Y_temp = train_test_split(X, Y, test_size=0.4, random_state=42)
         X_validation, X_test, Y_validation, Y_test = train_test_split(X_temp, Y_temp, test_size=0.5, random_state=42)
```

```
In [25]: # https://lifewithdata.com/2022/03/19/feature-selection-with-selectkbest-in-scikit-learn/

         from sklearn.feature_selection import SelectKBest, f_classif

         # create an instance of SelectKBest with f_classif as the score function and k value
         selector = SelectKBest(score_func=f_classif, k=40)

         # fit the selector to the training data and transform the data to select the top k features
         X_train_selected = selector.fit_transform(X_train, Y_train)
         X_test_selected = selector.transform(X_test)
         # X_train_selected.columns
```

Data Mining Models/Methods:

KNN:

KNN

```
In [28]: from sklearn.neighbors import KNeighborsClassifier
         from matplotlib.colors import ListedColormap

         # create a KNN classifier
         knn = KNeighborsClassifier(n_neighbors=7, p=2, metric='minkowski')

         # fit the classifier to the selected training data
         knn.fit(X_train_selected, Y_train)

         # evaluate the classifier on the selected test data
         knn_acc = knn.score(X_test_selected, Y_test)
         print(knn_acc)

         Accuracy of the classifier: 0.9105691056910569
```

The K-Nearest Neighbors (KNN) classifier, when applied to the test data, exhibited commendable performance, achieving an accuracy of around 91.06%. Notably, this accuracy was achieved using a feature selection process, whereby the top 40 features were meticulously chosen to optimize model performance. This outcome underscores the efficacy of the KNN algorithm in effectively discerning patterns and making accurate classifications, particularly when informed by a focused selection of pertinent features.

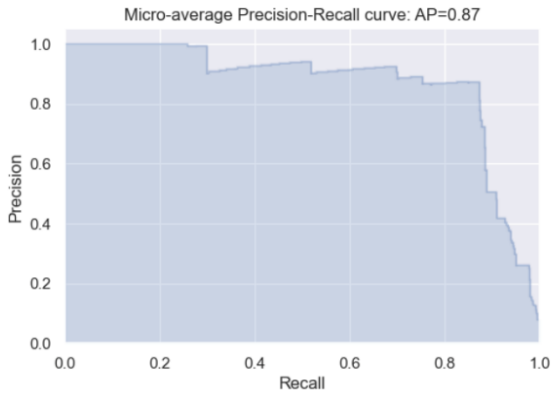
Multinomial Naive Bayes classifier:

Naive Bayes

In [50]:
```
from sklearn.naive_bayes import MultinomialNB

# Create a Multinomial Naive Bayes classifier
nb = MultinomialNB()
X_train_shifted = X_train_selected - np.min(X_train_selected) + 1  # Shift all values by 1
X_test_shifted = X_test_selected - np.min(X_test_selected) + 1
# Train the classifier using the training data
nb.fit(X_train_shifted, Y_train)

# Predict on the test data
y_pred1 = nb.predict(X_test_shifted)

# Print the predicted values
# print(y_pred1)

# Calculate the accuracy score
from sklearn.metrics import accuracy_score
nb_accuracy = accuracy_score(Y_test, y_pred1)
print("Accuracy:", nb_accuracy)
```

```
Accuracy: 0.9085365853658537
```

In [51]:
```
from sklearn.metrics import precision_recall_curve, average_precision_score
from sklearn.preprocessing import label_binarize

# Binarize the target variable
Y_test_bin = label_binarize(Y_test, classes=np.unique(Y_train))

# Train the classifier using the training data
nb.fit(X_train_shifted, Y_train)

# Predict on the test data
y_score = nb.predict_proba(X_test_shifted)

# Compute precision and recall for each class
precision = dict()
recall = dict()
average_precision = dict()
n_classes = Y_test_bin.shape[1]
for i in range(n_classes):
    precision[i], recall[i], _ = precision_recall_curve(Y_test_bin[:, i], y_score[:, i])
    average_precision[i] = average_precision_score(Y_test_bin[:, i], y_score[:, i])

# Compute micro-average precision-recall curve and score
precision["micro"], recall["micro"], _ = precision_recall_curve(Y_test_bin.ravel(), y_score.ravel())
average_precision["micro"] = average_precision_score(Y_test_bin, y_score, average="micro")

# Plot the micro-average precision-recall curve
import matplotlib.pyplot as plt
plt.figure(figsize=(6, 4))
plt.step(recall['micro'], precision['micro'], color='b', alpha=0.2,
         where='post')
plt.fill_between(recall["micro"], precision["micro"], step='post', alpha=0.2,
                 color='b')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Micro-average Precision-Recall curve: AP={0:0.2f}'
          .format(average_precision["micro"]))
plt.show()
```

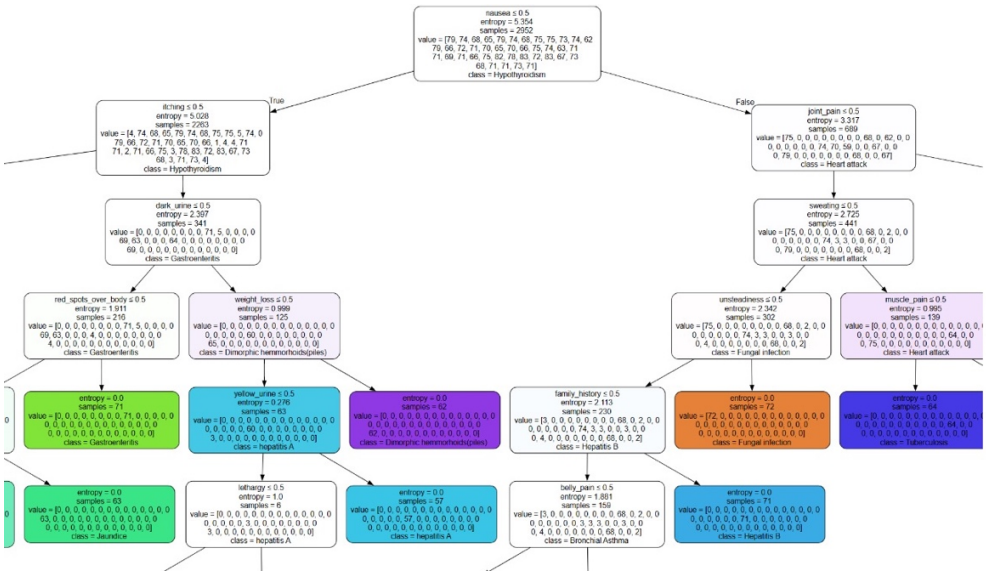


The code initializes a Multinomial Naive Bayes classifier (nb). This classifier is commonly used for text classification and other discrete data. It assumes that features follow a multinomial distribution . The features (X_train_selected and X_test_selected) are shifted by adding 1 to all values. Shifting ensures that all feature values are positive. The classifier is trained using the shifted training data (X_train_shifted) and corresponding labels (Y_train). Predictions are made on the shifted test data (X_test_shifted), resulting in y_pred1. The accuracy score of the classifier on the test data is approximately 90.85%. The micro-average precision-recall curve is calculated by considering all classes together. It combines precision and recall across all classes, providing an overall performance measure. The area under this curve (AP) is 0.87.

DECISION TREE:

Decision Tree

```
In [53]: from sklearn import tree

         ent = tree.DecisionTreeClassifier(criterion = 'entropy')
         ent = ent.fit(X_train_selected, Y_train)
         # Predicting on the test data
         y_pred = ent.predict(X_test_selected)
         y_pred
         result_entropy=pd.DataFrame()
         dt_accuracy = accuracy_score(Y_test, y_pred)
         #Calculates the accuracy score
         from sklearn.metrics import accuracy_score
         print(dt_accuracy)

         0.915650406504065

In [54]: ent.feature_importances_

Out[54]: array([0.08887095, 0.08931189, 0.03074004, 0.00193714, 0.03135681,
                0.00051126, 0.0450132 , 0.02005156, 0.00881803, 0.01508694,
                0.09219277, 0.02522902, 0.1433965 , 0.00292911, 0.01840856,
                0.00230729, 0.00000705, 0.00160364, 0.01713913, 0.00161508,
                0.027423  , 0.0020309 , 0.00135263, 0.02490761, 0.01685744,
                0.01919796, 0.02602264, 0.0543075 , 0.01816671, 0.01358244,
                0.01081086, 0.03471584, 0.00143409, 0.01051454, 0.02158403,
                0.00063972, 0.02078968, 0.02417843, 0.01069527, 0.01626273])

In [57]: from sklearn.feature_selection import SelectKBest, f_classif

         # create an instance of SelectKBest with f_classif as the score function and k value
         selector = SelectKBest(score_func=f_classif, k=40)

         # fit the selector to the training data and transform the data to select the top k features
         X_train_selected = selector.fit_transform(X_train, Y_train)
         X_test_selected = selector.transform(X_test)
         # X_train_selected.columns

         C:\Users\ankur\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\feature_selection\_univariate_sele
         ction.py:112: UserWarning: Features [33 59] are constant.
           warnings.warn("Features %s are constant." % constant_features_idx, UserWarning)
         C:\Users\ankur\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\feature_selection\_univariate_sele
         ction.py:113: RuntimeWarning: divide by zero encountered in divide
           f = msb / msw
         C:\Users\ankur\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\feature_selection\_univariate_sele
         ction.py:113: RuntimeWarning: invalid value encountered in divide
           f = msb / msw

In [58]: # Get the mask of selected features
         mask = selector.get_support()
```



Supervised machine learning algorithm Consists of many decision trees that operate as an ensemble. Each individual tree predicts a class. The class with maximum number of votes becomes the model's prediction. Individual trees protect each other from their individual errors i.e., low correlation. Therefore, yields better results Model Accuracy = 91%. The decision tree is classified according to the entropy.

Support Vector Classifier:

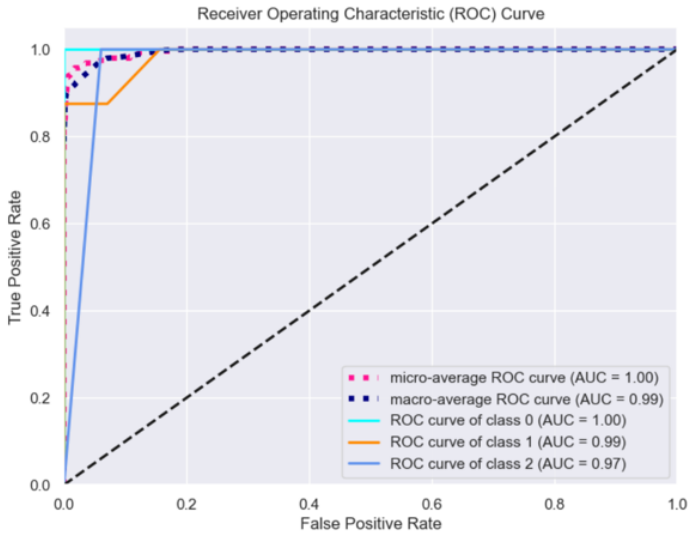
Support Vector Classifier (SVC)

In [62]:
```
from sklearn.svm import SVC

# create an SVC classifier
svcl = SVC(kernel='linear', probability = True)

# fit the classifier to the selected training data
svcl.fit(X_train_selected, Y_train)

# evaluate the classifier on the selected test data
svc_accuracyl = svcl.score(X_test_selected, Y_test)
print('For Linear Kernel - ',svc_accuracyl)

# create an SVC classifier
svcr = SVC(kernel='rbf', probability = True)

# fit the classifier to the selected training data
svcr.fit(X_train_selected, Y_train)

# evaluate the classifier on the selected test data
svc_accuracyr = svcr.score(X_test_selected, Y_test)
print('For RBF Kernel - ',svc_accuracyr)

# create an SVC classifier
svcp = SVC(kernel='poly', probability = True)

# fit the classifier to the selected training data
svcp.fit(X_train_selected, Y_train)

# evaluate the classifier on the selected test data
svc_accuracyp = svcp.score(X_test_selected, Y_test)
print('For Polynomial Kernel - ',svc_accuracyp)
```

```
For Linear Kernel -  0.9146341463414634
For RBF Kernel -  0.9176829268292683
For Polynomial Kernel -  0.8922764227642277
```

In [63]:
```
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.preprocessing import LabelBinarizer
import matplotlib.pyplot as plt

# fit the classifier to the selected training data
svcp.fit(X_train_selected, Y_train)

# predict class probabilities for the selected test data
Y_prob = svcp.predict_proba(X_test_selected)

# binarize the true labels to plot the ROC curves
lb = LabelBinarizer()
lb.fit(Y_test)
Y_test_bin = lb.transform(Y_test)

# compute the AUC score for each class
n_classes = Y_test_bin.shape[1]
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(Y_test_bin[:, i], Y_prob[:, i])
    roc_auc[i] = roc_auc_score(Y_test_bin[:, i], Y_prob[:, i])

# compute micro-average ROC curve and AUC score
fpr["micro"], tpr["micro"], _ = roc_curve(Y_test_bin.ravel(), Y_prob.ravel())
roc_auc["micro"] = roc_auc_score(Y_test_bin, Y_prob, average="micro")

# compute macro-average ROC curve and AUC score
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = roc_auc_score(Y_test_bin, Y_prob, average="macro")

# plot the ROC curves
plt.figure(figsize=(8,6))
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (AUC = {0:0.2f})'
               ''.format(roc_auc["micro"]),
         color='deeppink', linestyle=':', linewidth=4)

plt.plot(fpr["macro"], tpr["macro"],
         label='macro-average ROC curve (AUC = {0:0.2f})'
               ''.format(roc_auc["macro"]),
         color='navy', linestyle=':', linewidth=4)

colors = ['aqua', 'darkorange', 'cornflowerblue']
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label='ROC curve of class {0} (AUC = {1:0.2f})'
             ''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=2)
```

```
plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```



This analysis pertains to a supervised machine learning model employing classification algorithms, particularly Support Vector Machines (SVM). SVM is chosen for its ability to accurately classify data points, particularly in scenarios with relatively small datasets, typically in the order of thousands. Given pairs of (x, y) coordinates, SVM efficiently categorizes them into distinct groups by establishing a hyperplane, known as the decision boundary. The SVM algorithm utilizes existing data points to determine the optimal hyperplane that effectively separates the given data points. This optimal hyperplane is defined as the one with the maximum distance to each nearest element, ensuring robust classification performance. Upon assessing different kernel functions for SVM, the following accuracies were obtained:

For Linear Kernel: 91.46%

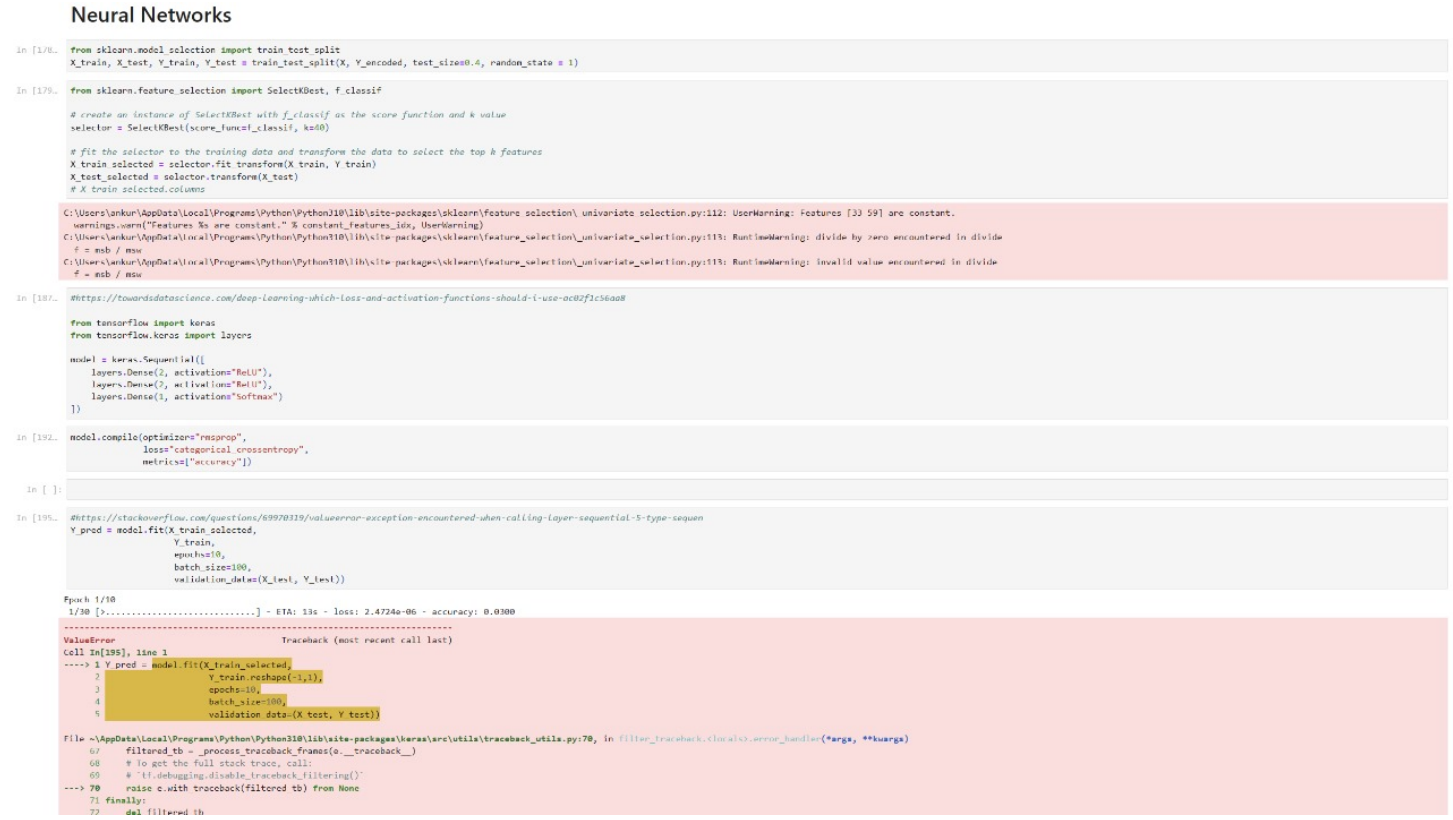
For RBF Kernel: 91.77%

For Polynomial Kernel: 89.23%

These results highlight the efficacy of SVM across various kernel functions, demonstrating its versatility and applicability in accurately classifying data points across diverse problem domains.

Neural Netwroks:

Our attempt to implement a Neural Network was hindered by the nature of our dataset, characterized by categorical data and a large number of classes in the target variable. This complexity exceeded the capability of the Neural Network to effectively handle.



Neural Networks

```
In [178...
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y_encoded, test_size=0.4, random_state = 1)
```

```
In [179...
from sklearn.feature_selection import SelectKBest, f_classif

# create an instance of SelectKBest with f_classif as the score function and k value
selector = SelectKBest(score_func=f_classif, k=40)

# fit the selector to the training data and transform the data to select the top k features
X_train_selected = selector.fit_transform(X_train, Y_train)
X_test_selected = selector.transform(X_test)
# X_train_selected.columns
```

```
C:\Users\ankur\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\feature_selection\_univariate_selection.py:112: UserWarning: Features [33 59] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx, UserWarning)
C:\Users\ankur\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\feature_selection\_univariate_selection.py:113: RuntimeWarning: divide by zero encountered in divide
  f = msb / msw
C:\Users\ankur\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\feature_selection\_univariate_selection.py:113: RuntimeWarning: invalid value encountered in divide
  f = msb / msw
```

```
In [187...
#https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8

from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Dense(2, activation="ReLU"),
    layers.Dense(7, activation="ReLU"),
    layers.Dense(1, activation="Softmax")
])
```

```
In [192...
model.compile(optimizer="rmsprop",
              loss="categorical_crossentropy",
              metrics=["accuracy"])
```

```
In [ ]:
```

```
In [195...
#https://stackoverflow.com/questions/69970319/valueerror-exception-encountered-when-calling-layer-sequential-5-type-sequen
Y_pred = model.fit(X_train_selected,
                   Y_train,
                   epochs=10,
                   batch_size=100,
                   validation_data=(X_test, Y_test))
```

```
Epoch 1/10
 1/30 [>.............................] - ETA: 13s - loss: 2.4724e-06 - accuracy: 0.0300
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[195], line 1
----> 1 Y_pred = model.fit(X_train_selected,
      2                    Y_train.reshape(-1,1),
      3                    epochs=10,
      4                    batch_size=100,
      5                    validation_data=(X_test, Y_test))

File ~\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\utils\traceback_utils.py:70, in filter_traceback.<locals>.error_handler(*args, **kwargs)
     67     filtered_tb = _process_traceback_frames(e.__traceback__)
     68     # To get the full stack trace, call:
     69     # `tf.debugging.disable_traceback_filtering()`
---> 70     raise e.with_traceback(filtered_tb) from None
     71 finally:
     72     del filtered_tb
```

Performance Evaluation:

In our analysis, we employed accuracy as the primary metric for evaluating the performance of the K-Nearest Neighbors (KNN), Support Vector Classifier (SVC), and Naïve Bayes models. Additionally, during the cross-validation phase, we adopted both error rate and accuracy as our key benchmarks for assessing the robustness and effectiveness of the models. By considering multiple metrics, we aimed to gain a comprehensive understanding of each model's performance across different evaluation criteria, thereby facilitating informed decision-making in model selection and refinement processes.

```
# Defining scoring metric for k-fold cross validation
def cv_scoring(estimator, X, y):
    return accuracy_score(y, estimator.predict(X))

# Initializing Models
models = {
    "SVC":SVC(kernel='linear', probability = True),
    "KNN": KNeighborsClassifier(n_neighbors=7, p=2, metric='minkowski'),
    "Naive bayes":MultinomialNB()
}

# Producing cross validation score for the models
for model_name in models:
    model = models[model_name]
    scores = cross_val_score(model, X_validation, Y_validation, cv = 5,
                             n_jobs = -1,
                             scoring = cv_scoring)
    print("=="*30)
    print(model_name)
    print(f"Scores: {scores}")
    print(f"Mean Score: {np.mean(scores)}")
```

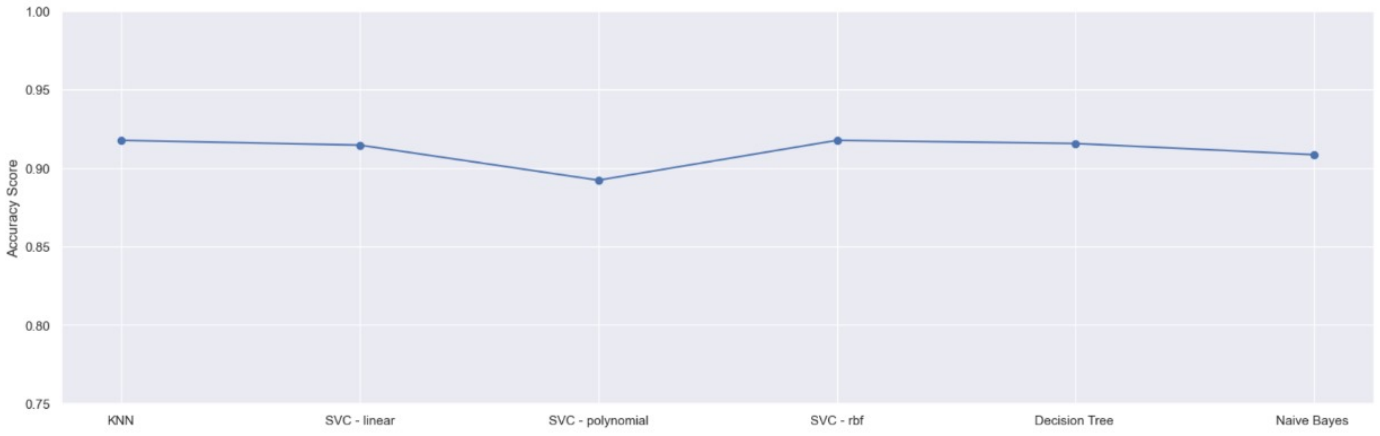
```
============================================================
SVC
Scores: [0.99492386 0.98477157 0.98477157 0.98984772 1.        ]
Mean Score: 0.9908629441624365
============================================================
KNN
Scores: [0.98984772 1.         1.         1.         1.        ]
Mean Score: 0.9979695431472081
============================================================
Naive bayes
Scores: [1.         0.98477157 0.98477157 0.99492386 0.99489796]
Mean Score: 0.9918729928519632
```

Project Results:

Upon comparing these results, we observed that the K-Nearest Neighbors (KNN) algorithm exhibited strong performance, benefitting from the dataset's simplicity. However, it's worth noting that KNN entails a higher computational cost compared to algorithms like Naïve Bayes, Decision Trees, and Support Vector Classifier (SVC). Therefore, for scenarios involving relatively small datasets akin to our current dataset, KNN emerges as a favorable choice. Conversely, if the dataset were to scale up in size, our preference would shift towards SVC due to its scalability and efficiency in handling larger datasets. It's important to highlight that post-cross-validation, our model's accuracy observed a substantial increase from 91% to 98%. However, this improvement in accuracy is accompanied by a higher computational cost.

Conclusion

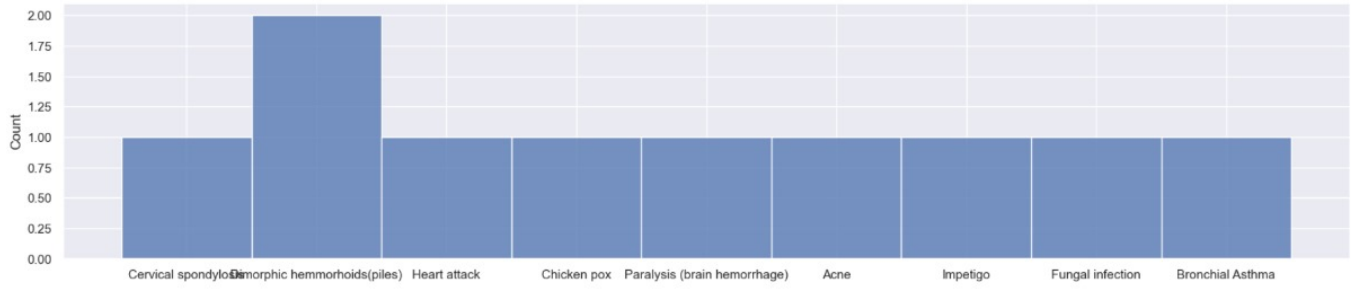
[201]:
```
#plt.style.use('whitegrid')
plt.figure(figsize=(20, 6))
models = ['KNN', 'SVC - linear', 'SVC - polynomial', 'SVC - rbf', 'Decision Tree ', 'Naive Bayes ']
test_accuracy = [knn_accuracy, svc_accuracyl, svc_accuracyp, svc_accuracyr, dt_accuracy, nb_accuracy]
plt.plot(models, test_accuracy, '-o')
plt.ylim(0.75, 1.00)
plt.ylabel("Accuracy Score")
plt.show()
```



Impact of the Project Outcomes:

Our model can be used as a backend processor for an application which is focused on suggesting treatments, doctors etc. We can use our predictions to analyze severity of the disease and then accordingly we can suggest best next steps. Below is a sample application of our application:

```
[156]: #To simulate user input
       Y_pred_sample = knn.predict([X_test_selected[0]])
       Y_pred_sample

[156]: array(['Fungal infection'], dtype=object)

[157]: Medicine_suggestions = {
         "Fungal infection": ["ketoconazole-oral-tablet", "Clotrimazole", "Miconazole"]
       }

       if Y_pred_sample[0] in Medicine_suggestions:
           print("We would recommend booking a appointement with a doctor but to treat the immidiate symptoms you might try the following recommended medicines:")
           print(Medicine_suggestions[Y_pred_sample[0]])
       else:
           print("No suggestions available")

We would recommend booking a appointement with a doctor but to treat the immidiate symptoms you might try the following recommended medicines:
['ketoconazole-oral-tablet', 'Clotrimazole', 'Miconazole']

[101]: import seaborn as sns
       sns.set(rc={"figure.figsize":(20, 4)})
       sns.histplot(Y_pred[np.random.choice(len(Y_pred), size=10, replace=False)],bins=10)          #Using seaborn alongwith matplotlib to display overall distribution plot

[101]: <Axes: ylabel='Count'>
```



This model will result in early detection and prevention of diseases which might become severe over time.

References:

- https://www.baeldung.com/cs/svm-choose-kernel - SVC Kernel selection.
- https://datascience.stackexchange.com/questions/15989/micro-average-vs-macro-average-performance-in-a-multiclass-classification-settin - Why Micro and macro average
- https://analyticsindiamag.com/a-hands-on-guide-to-ridge-regression-for-feature-selection/ - Ridge regression for feature selection (We did not use this method as it is not suitable for categorical data)
- https://www.statology.org/how-to-calculate-vif-in-python/ - VIF for feature importance
- https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8 - Selecting loss function for Neural networks
- https://stackoverflow.com/questions/69970319/valueerror-exception-encountered-when-calling-layer-sequential-5-type-sequen - Input data for neural networks is expected to be 2D error.