Disease Diagnosis Using Machine Learning Algorithm in Python

Ranjith Ambeera
12018051
K20UT
Computer Science and Engineering
Jalandhar, Punjab, India

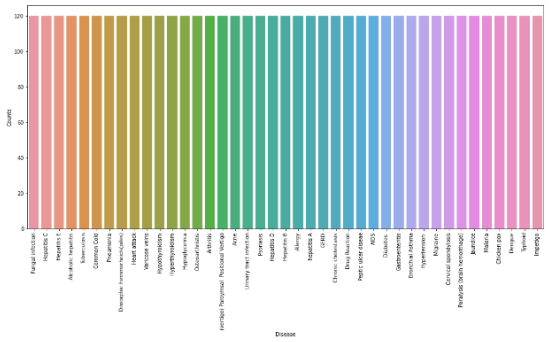
ABSTRACT - The wide adaptation of computer-based technology in the health care industry resulted in the accumulation of electronic data. Due to the substantial amounts of data, medical doctors are facing challenges to analyze symptoms accurately and identify diseases at an early stage. However, supervised machine learning (ML) algorithms have showcased significant potential in surpassing standard systems for disease diagnosis and aiding medical experts in the early detection of high-risk diseases. In this literature, the aim is to recognize trends across various types of supervised ML models in disease detection through the examination of performance metrics. The most prominently discussed supervised ML algorithms were Naive Bayes (NB), Decision Trees (DT). As per findings, Support Vector Machine (SVM) is the most adequate at detecting kidney diseases and Parkinson's disease. Finally, Random Forest (RF) predicted in precision breast diseases and common diseases, respectively.
Keywords—Health Care, Supervised Machine Learning, Dis-eases Prediction.

INTRODUCTION - It is projected that, every 2 months, over 70% of the population in India has a tendency toward general body ailments like viral influenza, cough, and cold. Etc. Since a lot of people don't realize that the symptoms of these regular illnesses may be symptoms into something more detrimental, 25% of the population succumbs to death due to ignorance of the early signs. Hence, the identification of the disease in the initial stages is crucial for the prevention of any unwarranted casualties. The current medical system is mainly devoted to very specific, known diseases and is largely unequipped to identify and accurately predict diseases based on early signs. The purpose of our system is to make predictions for the general and more commonly occurring disorder that when unchecked can become fatal diseases. The system applies data mining techniques, does pre-processing on the data and then implements the Machine Learning, decision tree algorithms. This system will predict the potential disorder based on the presented symptoms and precautionary measures required to avoid the aggravation of the condition, it will also aid the physicians analyze the patterns of current commonly occurring diseases. Within this project, the disease prediction system will execute data mining within its preliminary stages, the system will be trained and tested using decision trees, Naïve Bayes, Support Vector Machine.

DATA GATHERING - Data preparation is the primary step for any machine learning problem. I will be using Disease prediction dataset from Kaggle for this problem. This dataset consists of two CSV files one for training and one for testing. There is a total of 133 columns in the dataset out of which 132 columns represent the symptoms and the last column is the prognosis. Reading the dataset

Firstly, I will be loading the dataset from the folders using the Pandas library. While reading the dataset I will be dropping the null column. This dataset is a clean dataset with no null values and all the features consist of 0's and 1s. Whenever solving a classification task it is necessary to check whether our target column is balanced or not. I will be using a bar plot, to check whether the dataset is balanced or not.

```
# Reading the train.csv by removing the
# last column since it's an empty column
DATA_PATH = "C:/Users/ambee/OneDrive/Desktop/Training.csv"
data = pd.read_csv(DATA_PATH).dropna(axis = 1)

# Checking whether the dataset is balanced or not
disease_counts = data["prognosis"].value_counts()
temp_df = pd.DataFrame({
    "Disease": disease_counts.index,
    "Counts": disease_counts.values
})

plt.figure(figsize = (18,8))
sns.barplot(x = "Disease", y = "Counts", data = temp_df)
plt.xticks(rotation=90)
plt.show()
```



From the above plot, I observed that the dataset is a balanced dataset i.e. There are exactly 120 samples for each disease, and no further balancing is required. I can notice that target column i.e.prognosis column is of object datatype, this format is not suitable to train a machine learning model. So, I will be using a label encoder to convert the prognosis column to the numerical datatype. Label Encoder converts the labels into numerical form by assigning a unique index to the labels. If the total number of labels is n, then the numbers assigned to each label will be between 0 to n-1.

```
# Encoding the target value into numerical
# value using LabelEncoder
encoder = LabelEncoder()
data["prognosis"] = encoder.fit_transform(data["prognosis"])
```

Splitting the data for training and testing the model -
Now that I have cleaned my data by removing the Null values and converting the labels to numerical format, It's time to split the data to train and test the model. I splitted the data into 80:20 format i.e. 80% of the dataset used for training the model and 20% of the data used to evaluate the performance of the models.

```
X = data.iloc[:,:-1]
y = data.iloc[:, -1]
X_train, X_test, y_train, y_test =train_test_split(
X, y, test_size = 0.2, random_state = 24)

print(f"Train: {X_train.shape}, {y_train.shape}")
print(f"Test: {X_test.shape}, {y_test.shape}")
```

```
Train: (3936, 132), (3936,)
Test: (984, 132), (984,)
```

Model Building - After splitting the data, I worked on the modeling part. I used K-Fold cross-validation to evaluate the machine-learning models. I used Support Vector Classifier, Gaussian Naive Bayes Classifier, and Random Forest Classifier for cross-validation.

K-Fold Cross-Validation:
K-Fold cross-validation is one of the cross-validation techniques in which the whole dataset is split into k number of subsets, also known as folds, then training of the model is performed on the k-1 subsets and the remaining one subset is used to evaluate the model performance.

Support Vector Classifier:
Support Vector Classifier is a discriminative classifier i.e. when given a labeled training data, the algorithm tries to find an optimal hyperplane that accurately separates the samples into different categories in hyperspace.

Gaussian Naive Bayes Classifier:
It is a probabilistic machine learning algorithm that internally uses Bayes Theorem to classify the data points.
Random Forest Classifier: Random Forest is an ensemble learning-based supervised machine learning classification algorithm that internally uses multiple decision trees

to make the classification. In a random forest classifier, all the internal decision trees are weak learners, and the outputs of these weak decision trees are combined i.e. mode of all the predictions is as the final prediction.

K-Fold Cross-Validation for model selection :

```
# 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(),
    "Gaussian NB":GaussianNB(),
    "Random Forest":RandomForestClassifier(random_state=18)
}

# Producing cross validation score for the models
for model_name in models:
    model = models[model_name]
    scores = cross_val_score(model, X, y, cv = 10,
                             n_jobs = -1,
                             scoring = cv_scoring)
    print("=="*30)
    print(model_name)
    print(f"Scores: {scores}")
    print(f"Mean Score: {np.mean(scores)}")
```

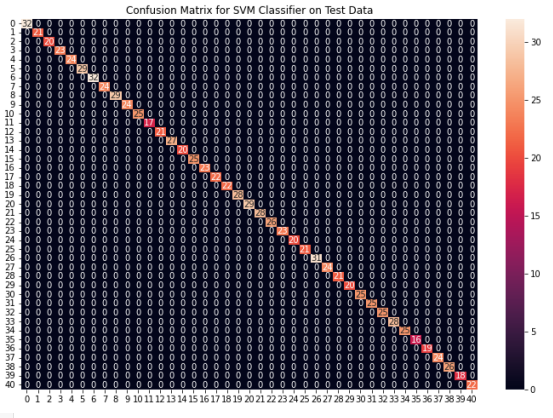
```
============================================================
SVC
Scores: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Mean Score: 1.0
============================================================
Gaussian NB
Scores: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Mean Score: 1.0
============================================================
Random Forest
Scores: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Mean Score: 1.0
```

From the above output, I noticed that all my machine learning algorithms are performed very well and the mean scores after k fold cross-validation are also very high. To build a robust model I combined i.e. take the mode of the predictions of all three models so that even one of the models makes wrong predictions and the other two make correct predictions then the final output would be the correct one. This approach helped me to keep the predictions much more accurate on completely unseen data. In the below code I trained all the three models on the train data, checking the quality of my models using a confusion matrix, and then combined the predictions of all three models.

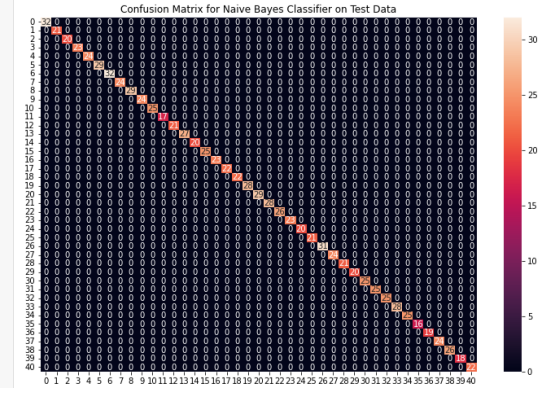
Building robust classifier by combining all models:

```
# Training and testing SVM Classifier
svm_model = SVC()
svm_model.fit(X_train, y_train)
preds = svm_model.predict(X_test)

print(f"Accuracy on train data by SVM Classifier\
: {accuracy_score(y_train, svm_model.predict(X_train))*100}")

print(f"Accuracy on test data by SVM Classifier\
: {accuracy_score(y_test, preds)*100}")
cf_matrix = confusion_matrix(y_test, preds)
plt.figure(figsize=(12,8))
sns.heatmap(cf_matrix, annot=True)
plt.title("Confusion Matrix for SVM Classifier on Test Data")
plt.show()

# Training and testing Naive Bayes Classifier
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
preds = nb_model.predict(X_test)
print(f"Accuracy on train data by Naive Bayes Classifier\
: {accuracy_score(y_train, nb_model.predict(X_train))*100}")

print(f"Accuracy on test data by Naive Bayes Classifier\
: {accuracy_score(y_test, preds)*100}")
cf_matrix = confusion_matrix(y_test, preds)
plt.figure(figsize=(12,8))
sns.heatmap(cf_matrix, annot=True)
plt.title("Confusion Matrix for Naive Bayes Classifier on Test Data")
plt.show()

# Training and testing Random Forest Classifier
rf_model = RandomForestClassifier(random_state=18)
rf_model.fit(X_train, y_train)
preds = rf_model.predict(X_test)
print(f"Accuracy on train data by Random Forest Classifier\
: {accuracy_score(y_train, rf_model.predict(X_train))*100}")

print(f"Accuracy on test data by Random Forest Classifier\
: {accuracy_score(y_test, preds)*100}")

cf_matrix = confusion_matrix(y_test, preds)
plt.figure(figsize=(12,8))
sns.heatmap(cf_matrix, annot=True)
plt.title("Confusion Matrix for Random Forest Classifier on Test Data")
plt.show()
```

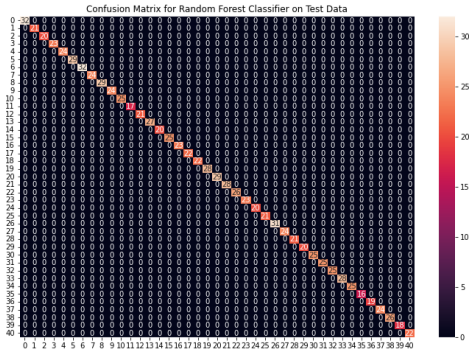
```
Accuracy on train data by SVM Classifier: 100.0
Accuracy on test data by SVM Classifier: 100.0
```




```
Accuracy on train data by Naive Bayes Classifier: 100.0
Accuracy on test data by Naive Bayes Classifier: 100.0
```



```
Accuracy on train data by Random Forest Classifier: 100.0
Accuracy on test data by Random Forest Classifier: 100.0
```

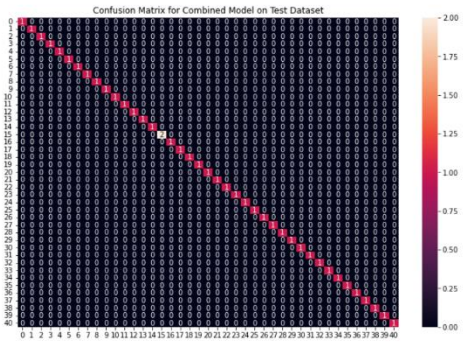


From the above confusion matrices, I saw that the models are performed very well on the unseen data. Next I trained the models on the whole train data present in the dataset that I downloaded and then tested my combined model on test data present in the dataset.

Fitting the model on whole data and validating on the Test dataset:

```
# Training the models on whole data
final_svm_model = SVC()
final_nb_model = GaussianNB()
final_rf_model = RandomForestClassifier(random_state=18)
final_svm_model.fit(X, y)
final_nb_model.fit(X, y)
final_rf_model.fit(X, y)

# Reading the test data
test_data = pd.read_csv("C:/Users/ambee/OneDrive/Desktop/Testing.csv").dropna(axis=1)

test_X = test_data.iloc[:, :-1]
test_Y = encoder.transform(test_data.iloc[:, -1])

# Making prediction by take mode of predictions
# made by all the classifiers
svm_preds = final_svm_model.predict(test_X)
nb_preds = final_nb_model.predict(test_X)
rf_preds = final_rf_model.predict(test_X)

final_preds = [mode([i,j,k])[0][0] for i,j,
               k in zip(svm_preds, nb_preds, rf_preds)]

print(f"Accuracy on Test dataset by the combined model\
: {accuracy_score(test_Y, final_preds)*100}")

cf_matrix = confusion_matrix(test_Y, final_preds)
plt.figure(figsize=(12,8))

sns.heatmap(cf_matrix, annot = True)
plt.title("Confusion Matrix for Combined Model on Test Dataset")
plt.show()
```

```
Accuracy on Test dataset by the combined model: 100.0
```



I saw that my combined model classified all the data points accurately. I came to the final part of this whole implementation, and created a function that takes symptoms separated by commas as input and outputs the predicted disease using the combined model based on the input symptoms.

Created a function that can take symptoms as input and generate predictions for disease:

```
]: symptoms = X.columns.values

    # Creating a symptom index dictionary to encode the
    # input symptoms into numerical form
    symptom_index = {}
    for index, value in enumerate(symptoms):
        symptom = " ".join([i.capitalize() for i in value.split("_")])
        symptom_index[symptom] = index

    data_dict = {
        "symptom_index":symptom_index,
        "predictions_classes":encoder.classes_
    }

    # Defining the Function
    # Input: string containing symptoms separated by commas
    # Output: Generated predictions by models
    def predictDisease(symptoms):
        symptoms = symptoms.split(",")

        # creating input data for the models
        input_data = [0] * len(data_dict["symptom_index"])
        for symptom in symptoms:
            index = data_dict["symptom_index"][symptom]
            input_data[index] = 1

        # reshaping the input data and converting it
        # into suitable format for model predictions
        input_data = np.array(input_data).reshape(1,-1)

        # generating individual outputs
        rf_prediction = data_dict["predictions_classes"][final_rf_model.predict(input_data)[0]]
        nb_prediction = data_dict["predictions_classes"][final_nb_model.predict(input_data)[0]]
        svm_prediction = data_dict["predictions_classes"][final_svm_model.predict(input_data)[0]]

        # making final prediction by taking mode of all predictions
        final_prediction = mode([rf_prediction, nb_prediction, svm_prediction])[0][0]
        predictions = {
            "rf_model_prediction": rf_prediction,
            "naive_bayes_prediction": nb_prediction,
            "svm_model_prediction": svm_prediction,
            "final_prediction":final_prediction
        }
        return predictions

    # Testing the function
    print(predictDisease("Itching,Skin Rash,Nodal Skin Eruptions"))
```

Out put :
{'rf_model_prediction': 'Fungal infection', 'naive_bayes_prediction': 'Fungal infection', 'svm_model_prediction': 'Fungal infection', 'final_prediction': 'Fungal infection'}

Conclusion - The use of different ML algorithms enabled the early detection of many maladies such as heart, kidney, breast, and brain diseases. Throughout the literature, SVM, RF and Naïve Bayes algorithms were the most widely used at prediction, while accuracy was the most used performance metric.
In future work, the creation of more complex ML algorithms is much needed to increase the efficiency of disease prediction. In addition, learning models should be calibrated more often after the training phase for potentially a better performance. Moreover, datasets should be expanded on different demo-graphics to avoid overfitting and increase the accuracy

of the deployed models. Finally, more relevant feature selection methods should be used to enhance the performance of the learning models.

REFERENCES –

1] A. Gavhane, G. Kokkula, I. Pandya, and K. Devadkar, “Prediction of heart disease using machine learning,” in 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA), 2018, pp. 1275–1278.
[2] Y. Hasija, N. Garg, and S. Sourav, “Automated detection of dermatological disorders through image-processing and machine learning,” in 2017International Conference on Intelligent Sustainable Systems (ICISS),2017, pp. 1047–1051.
[3] S. Uddin, A. Khan, M. E. Hossain, and M. A. Moni, “Comparing different supervised machine learning algorithms for disease prediction,” BMC Medical Informatics and Decision Making, vol. 19, no. 1, pp. 1–16, 2019.
[4] R. Katarya and P. Srinivas, “Predicting heart disease at early stages using machine learning: A survey,” in 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC), 2020,pp. 302–305.
[5] P. S. Kohli and S. Arora, “Application of machine learning in disease prediction,” in 2018 4th International Conference on Computing Communication and Automation (ICCCA), 2018, pp. 1–4.