Disease Classification and Prediction using Python

Devika Chandnani
Masters of Science in Data Science
Fordham University, LC
Brooklyn, NY, USA
dchandnani@fordham.edu

***Abstract*—Clinical decisions are often made on a doctor's experience and intuition rather than on knowledge rich data. This has the potential of leading to errors in diagnosis and opens the possibility of patients questioning the quality of medical services they are being provided. Many countries require years of education to become a doctor. It is a challenging field with high stakes, long hours and a high barrier to entry. Prediction based Machine Learning models can play a significant role in lessening doctor's workload and increasing the overall reliability of health care. The Naïve Bayes Classifier, Support Vector Classifier and Random Forest Classifier can be employed on patient data to predict their illness. This implementation can be done through the Python programming language. I will be using Jupyter Notebooks. Disease prediction using patient health data and treatment history with a combination of machine learning and data mining has been a continuous struggle for the past few decades, but early detection can shapeshift patient care by identifying the risk of illnesses beforehand. The following paper explores this approach. The research topic tests the algorithms based on their accuracy, which is determined based on the model's performance on the chosen data set, which is tested using a confusion matrix.**

Keywords— Machine Learning, Disease Prediction, Classification, Healthcare, Python, Naïve Bayes, Random Forest, Decision Trees, Support Vector Classifier, Big Data Analysis.

I. Introduction and Background

Globally, there is a huge unmet need for effective treatments of diseases. The complexity of disease diagnosis and the heterogeneity of the patient population present massive challenges to the development of early diagnostic tools and effective treatments for these diseases. Machine learning, a subfield of artificial intelligence, is enabling scientists, clinicians and patients to address some of these challenges. Machine learning can aid early diagnosis and interpretation of medical images as well as the discovery and development of new therapies. Although this paper does not dive into that area of study, a unifying theme of the different applications of machine learning in healthcare is the integration of multiple high-dimensional sources of data, which all provide a different view on diseases, and the automated derivation of actionable insights. The following paper uses a sample data set found on Kaggle with a sample size of 120 per illness. Working with large collections of curated datasets and robust assessments of the machine learning models used will allow for full integration of machine learning into diagnostic fields and allow for it to become a norm in the practice.

II. Summary

A. Basis

Machine learning in healthcare is becoming more widely used and helping patients and clinicians in different ways. Artificial Intelligence reduces the need for human intervention and is of high interest in the field now because healthcare workers have been overworked severely since the beginning of the COVID-19 pandemic. Computers can now be programmed to learn information without human intervention. The most common use cases of machine learning in healthcare are automating medical billing, clinician decision support and development of clinical care guidelines [4]. Clinical decision support, although may run on similar systems to the scope of this project, it is different in its purpose and design. In the following project, I will discuss a project where I worked on predicting patient illnesses using Machine Learning algorithms.

B. Design

My project was implemented in JupyterLab and requires the following libraries:

1. **Numpy:** This allows us to work with Arrays.
2. **Pandas:** This makes it easier to work with CSV data files and data frames.
3. **Matplotlib:** To create charts using plyplot, define parameters using rcParams.
4. **Scipy.stats:** To conduct statistical tests and use the function mode.
5. **Seaborn**: To create statistical graphics and visualize our data for better exploration and understanding.
6. **Scikit-learn:** For classification, regression and model selection

Using scikit-learn, I was able to import the three machine learning models I implied on the data, the Support Vector Classifier as *"from sklearn.svm import SVC"*, the Naive Bayes Classifier as *"from sklearn.naive_bayes import GaussianNB"*, the Random Forest classifier as "*from sklearn.ensemble import RandomForestClassifier"*.

C. Process

I used the following 4-step approach for my project. (A) Gathering the Data: After importing all the Machine Learning algorithms, I downloaded the data set from Kaggle. My data set consists of two CSV files, one for training and one for testing. There is a total of 133 columns in my dataset out of which 132 columns represent the symptoms, the last column is the prognosis. (B) Next, we moved onto cleaning the data: this is an important step because the quality of the data determines the quality of the

machine learning model. Most of the data is numerical, except the disease prognosis, which is a string type. It was encoded into numerical form using a label encoder. (C) Model Building: After gathering and cleaning the data, the data is ready and can be used to train a machine learning model. I used the clean data to train the Support Vector Classifier, Naïve Bayes Classifier and Random Forest Classifier. I then used a confusion matrix to determine the quality of the three models and test them toward each other. (D) Inference: After training the three models, we will see how I predicted the disease for the input symptoms by combining the predictions of all three models. This allows the overall prediction to be more robust and accurate. Finally, I defined a python function that takes the symptoms separated by commas as input, and predicts the disease based on the symptoms by using the trained models. It returns the predictions in JSON format.

III. ARCHITECHTURE

A substantial number of machine learning algorithms exist and choosing the correct algorithms to apply to this data type was crucial to obtain reliable results. The following are the three models I used.

A. Support Vector Classifier

A supervised machine learning method, the support vector machine algorithm has demonstrated high performance in solving classification problems in the biomedical fields [1]. In contrast to logistic regression, which depends on a predetermined model to predict the occurrence or not of a binary event by fitting data to a logistic curve, SVM discriminates between two classes by generating a hyperplane that optimally separates classes after the input data have been transformed mathematically into a high-dimensional space. Because the SVM approach is data-driven and model-free, it may have important discriminative power for classification, especially in cases where sample sizes are small and a large number of variables are involved (high-dimensionality space). This technique has recently been used to develop automated classification of diseases and to improve methods for detecting disease in clinical settings. The basic idea behind the SVM technique is to construct an n-1 dimensional separating hyperplane to discriminate two classes in an n-dimensional space. A data point is viewed as an n-dimensional vector. For example, two variables in a dataset will create a two-dimensional space; the separating hyperplane would be a straight line (one dimensional) dividing the space in half. When more dimensions are involved, SVM searches for an optimal separating hyperplane called the maximum-margin separating hyperplane. The SVM approach tends to classify entities without providing estimates of the probabilities of class membership in the dataset, which is a fundamental difference from multiple logistic regression [3].

B. Naïve Bayes Classifier

In probability theory, Bayes' theorem relates the conditional and marginal probabilities of two random events. It is often used to compute posterior probabilities given observations. For example, a patient may be observed to have certain symptoms. Bayes' theorem can be used to compute the probability that a proposed diagnosis is correct, given that observation. A naive Bayes classifier is a term dealing with a simple probabilistic classification based on applying Bayes' theorem. In simple terms, a naive Bayes classifier assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature. Despite its oversimplified design, this model works much better in complex real-world scenarios than one might expect. Another advantage of the naïve Bayes classifier is that requires a small amount of training data to estimate the parameters (means and variances) necessary for the classification.

C. Random Forest Classifier

Random Forest is an ensemble learner, a method that generates many classifiers and aggregates their results. Random Forest will create multiple classification and regression trees, each trained on a bootstrap sample of the original data and searches across a randomly selected subset of input variables to determine the split [2]. Classification and Regression trees are binary decision trees that are constructed by splitting the data in a node into child nodes repeatedly, starting with the root note that contains the whole learning sample, in this case, the training data. Each tree in Random Forest will then cast a vote for some input x, then the output of the classifier is determined by majoring voting of the trees. Random Forest can handle high dimensional data, which is why it is such a good fit for our data set. We are dealing with the possibility of multiple diseases, each with overlapping symptoms. It can use many trees in the ensemble to bring us to the right conclusion. Radom Forest is also helpful when dealing with imbalanced data [5], and missing data, although as we will see in the next section, our data set is evenly balanced. I was also able to get rid of null values.

IV. DATASET PREPERATIONS

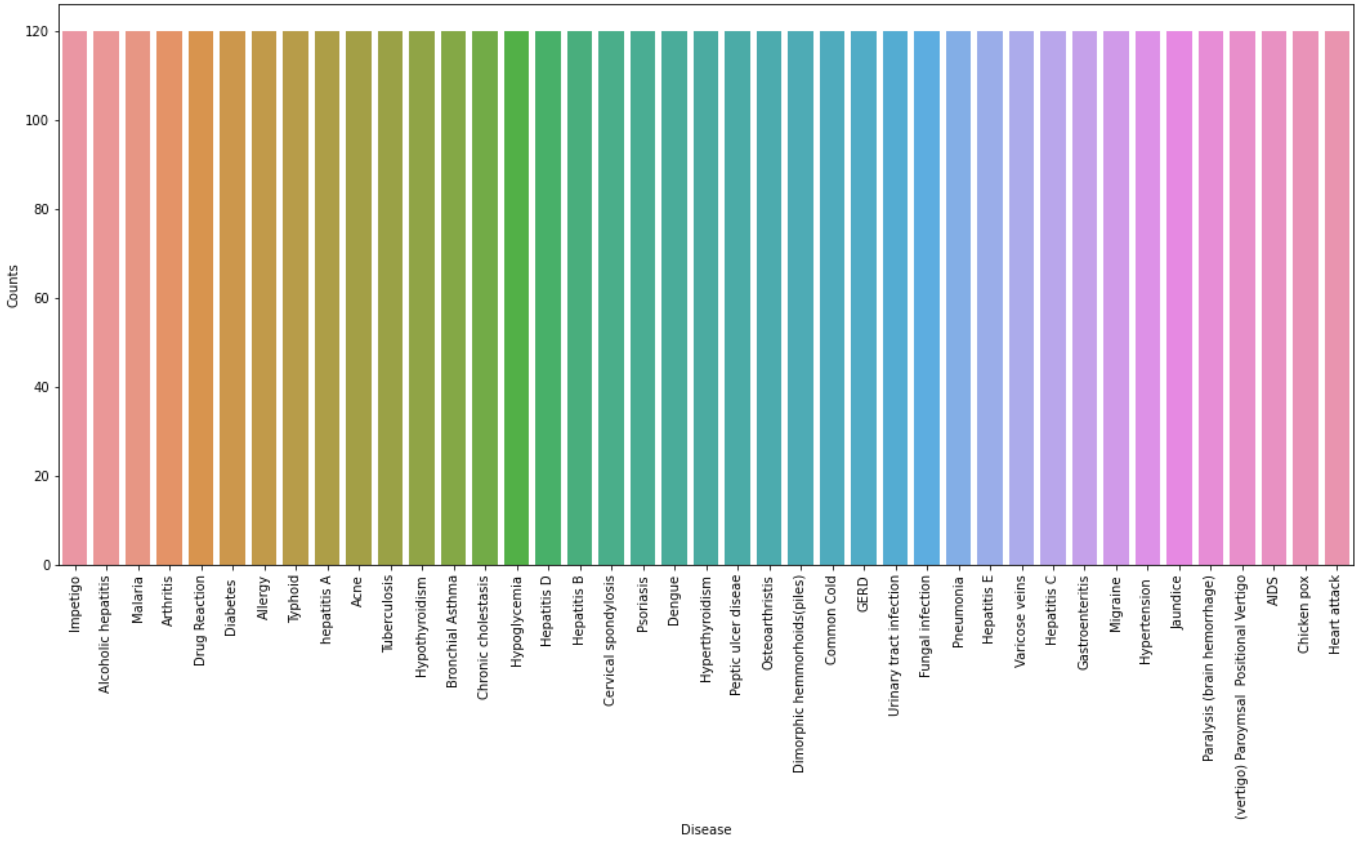
After importing the required libraries and importing the data step, the next step was to make sure to read the data set appropriately. I dropped the null column and made sure all the features consist of 0's and 1's.

A. Reading and Cleaning

The last column in the training data set was an empty column, so I imported the dataset into my Jupyter Lab by using "*pd.read"* and "*dropna"* to ensure the data was prepared for the next steps.

B. Testing Process

I used a bar graph to check if the data set was balanced or not. From the below plot, we can observe that the dataset is a balanced dataset i.e. there are exactly 120 samples for each disease, and no further balancing is required. We can notice that our target column i.e. prognosis column is of object datatype, this format is not suitable to train a machine learning model. So, we 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.



C. *Creating the Training and Testing Data*

Now that we have cleaned our 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. We will be splitting the data into 80:20 format i.e., 80% of the dataset will be used for training the model and 20% of the data will be used to evaluate the performance of the models. The output result of splitting the data was as follows:

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

V. MODEL BUILDING

After splitting the data, I worked on the modeling part. As discussed above (III), I used the Support Vector Classifier, Gaussian Naive Bayes Classifier, and Random Forest Classifier for cross-validation. I also utilized K-Fold cross-validation to evaluate the machine learning models.

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 [1].

To apply this technique to my three models, I first started by defining a scoring metric for K-Fold cross validation, then I initialized the models and created a cross validation score. Below are the results of the cross verification.

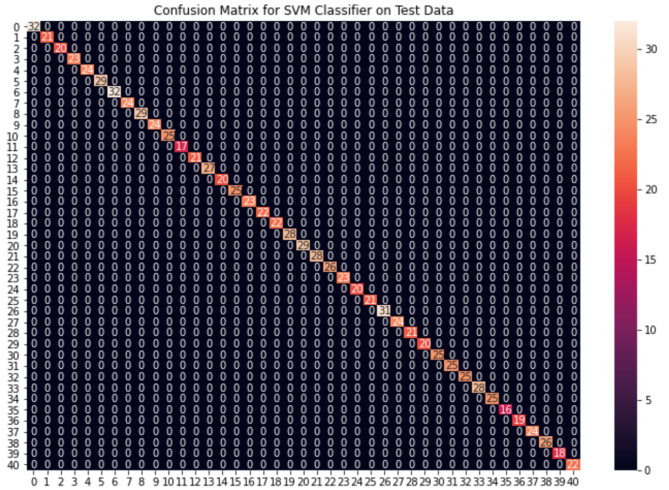
```
============================================================
SVC
Scores: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Mean Score: 1.0
============================================================
Gaussian NB
Scores: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Mean Score: 1.0
============================================================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

From the above output, we can notice that all my machine learning algorithms are performing fine and the mean scores after k fold cross-validation are high. To build a robust model I combined i.e. took the mode of the predictions of all three models so that even if one of the models makes a wrong prediction 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. Moving forward, I next trained all the three models on the train data, checked the quality of the models using a confusion matrix, and then combine the predictions of all the three models as discussed in section II.C.

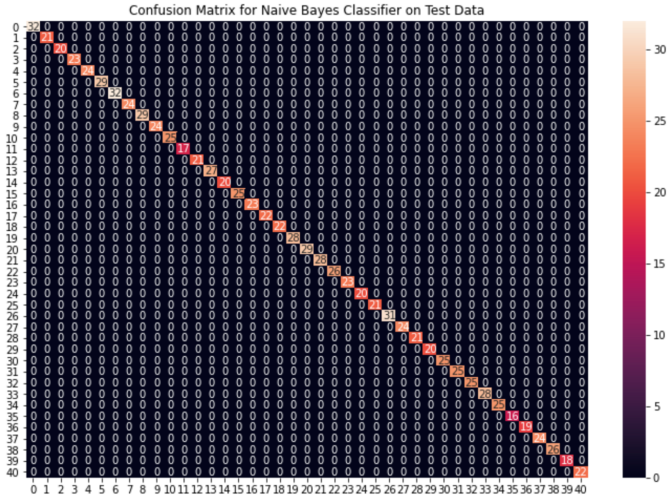
VI. Confusion Matrix and Model Testing

In a confusion matrix, we can summarize the performance of a classification model. The number of correct and incorrect predictions are summarized with count values and broken down by each class. Below are the results of the confusion matrix after model building.

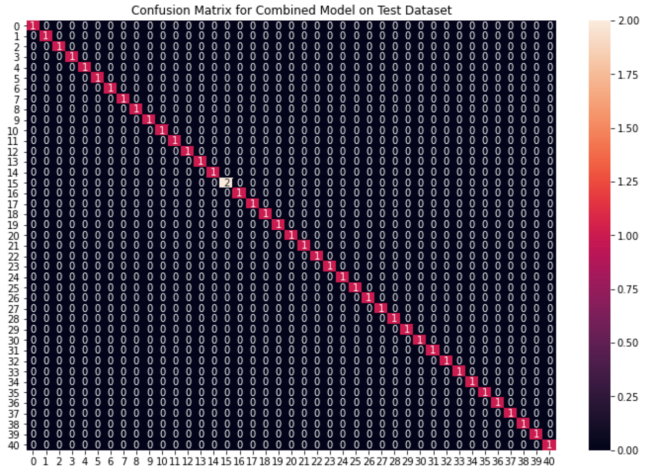
```
Accuracy on train data by SVM Classifier: 100.0
Accuracy on test data by SVM Classifier: 100.0
```



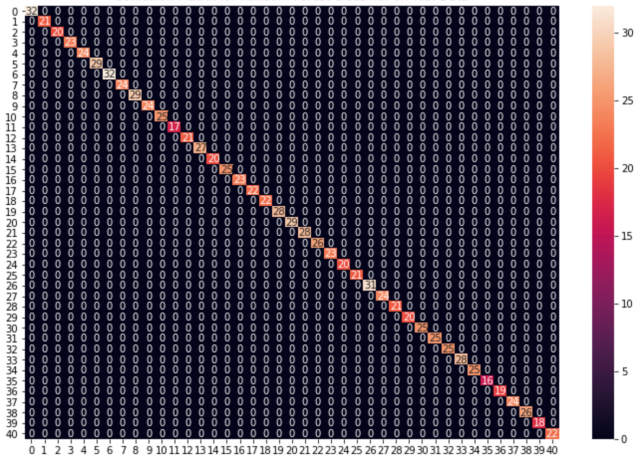

```
Accuracy on train data by Naive Bayes Classifier: 100.0
Accuracy on test data by Naive Bayes Classifier: 100.0
```




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




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



Now, we will see the same confusion matrix on the test data, not on the training data, here we have taken the mode of the predictions made by all the classifiers.

VII. Defining the Function and Final Output

Next, I defined a new function that would take a string of comma separated symptoms as input and output the combined result predicted by the three models. I assigned input data for the models and then reshaped the data into a suitable format for model predictions. The function first generates individual outputs. The last step is to take the mode of all predictions and return the final output.

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

```
{'rf_model_prediction': 'Fungal infection', 'naive_bayes_prediction': 'Fungal infection'
, 'svm_model_prediction': 'Fungal infection', 'final_prediction': 'Fungal infection'}
```

```
# Testing the function
print(predictDisease("Abdominal Pain,Dark Urine,Yellowish Skin"))
```

```
{'rf_model_prediction': 'Jaundice', 'naive_bayes_prediction': 'Jaundice',
'svm_model_prediction': 'Jaundice', 'final_prediction': 'Jaundice'}
```

```
# Testing the function
print(predictDisease("Pus Filled Pimples,Blackheads,Scurring"))
```

```
{'rf_model_prediction': 'Acne', 'naive_bayes_prediction': 'Acne',
'svm_model_prediction': 'Acne', 'final_prediction': 'Acne'}
```

The key requirement however for this model to work efficiently is that one provides at least 3 symptoms. Testing on a single symptom resulted in the algorithm providing the first disease in the data as the result. However, a disease prediction based on one symptom will likely not be an efficient prediction regardless of if it was made by man or by machine. The K-Fold Cross Validation shows that the model is has a good performance measure.

VIII. REFERENCES

[1] Kumar, Rahul. “Cross-Validation and Model Selection.” In Machine Learning Quick Reference: Quick and Essential Machine Learning Hacks for Training Smart Data Models, 27–30. Packt, 2019.

[2] Rossant, Cyrille. “8.1. Getting Started with Scikit-Learn.” In *IPython Interactive Computing and Visualization Cookbook - Second Edition*, 285–99. Packt Publishing, 2018.

[3] Yu, Wei, Tiebin Liu, Rodolfo Valdez, Marta Gwinn, and Muin J Khoury. “Application of Support Vector Machine Modeling for Prediction of Common Diseases: The Case of Diabetes and Pre-Diabetes - BMC Medical Informatics and Decision Making.” BioMed Central. BioMed Central, March 22, 2010.

https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/1472-6947-10-16.

[4] Barth, Steve. “Machine Learning in Healthcare – Benefits & Use Cases.” Forsee Medical, September 8, 20201.

https://www.foreseemed.com/blog/machine-learning-in-healthcare.

[5] Khalilia, Mohammed, Sounak Chakraborty, and Mihail Popescu. “Predicting Disease Risks from Highly Imbalanced Data Using Random Forest - BMC Medical Informatics and Decision Making.” SpringerLink. BioMed Central, July 29, 2011. https://link.springer.com/article/10.1186/1472-6947-11-51.