Queens' university school of computing

Fundamentals of machine learning

CSAI 801 Project: COVID-19 Outcome Prediction

Supervised by:

Prof. Hazem Abbas

T.A. Amr Zaki

T.A. Amira Gaber

T.A. Nour Zwawie

Team members:

Zeyad Tarek Mohamed

Sara Ahmed Elfetiany

Mohammed Mostafa Faidey

(Jan 2022, 2023)

CONTENTS

Introduction.. 3

Dataset view.. 3

Methodology.. 4

Algorithms.. 5

K-Nearest Neighbours.. 5

Logistic regression 27

Naive Bayes.. 8

Decision Trees.. 8

Support vector machines.. 11

Results.. 12

Test results 21

Introduction

Classification is a process of categorizing a given set of data into classes; it can be performed on both structured and unstructured data. The process starts with predicting the class of given data points. The classes are often referred to as target, label or categories.

The classification predictive modeling is the task of approximating the mapping function from input variables to discrete output variables. The main goal is to identify which class/category the new data will fall into.

Dataset view

So, in our project, we have a medical and geographical dataset, it's 862 rows * 15 columns but the first column (feature is just counter so we won't consider it as one of features)

1. **Country**: where the person lives.
2. **Location**: which part in the Country.
3. **Age**: Classification of the age group for each person, based on WHO Age Group Standard.
4. **Gender**: Male or Female or unknown.
5. **Visited_Wuhan**: whether the person has visited Wuhan, China or not.
6. **From_Wuhan**: whether the person is from Wuhan, China or not.
7. **Symptoms**: there are six families of symptoms that are coded in six fields.
8. **diff_sym_hos**: Time_before_symptoms_appear.
9. **Result**: death (1) or recovered (0).

All these features represented in numeric values.

Our goal here is create five classification algorithms to predict whether the person is going to recover then the result of the classifier will be 0 or he is going to die then the result of the classifier will be 1.

location	country	gender	age	…symptom4	symptom5	symptom6	diff_sym_hos	result
104	8	1	66	………12	3	1	8	1
101	8	0	56	………12	3	1	0	0
137	8	1	46	…. ….12	3	1	13	0
116	8	0	60	………12	3	1	0	0
116	8	1	58	………12	3	1	0	0
23	8	0	44	………12	3	1	0	0
105	8	1	34	………12	3	1	0	0

Methodology

We used object-oriented programming concept to organize our code and make it cleaner.

First, we created a class called `prepare_data,` it's responsible for preparing the data and transform some feature from numerical representation to one hot representation.

Note: transform from numerical datatype to one hot (binary) datatype will make our data not biased and it will not negatively affect the training process.

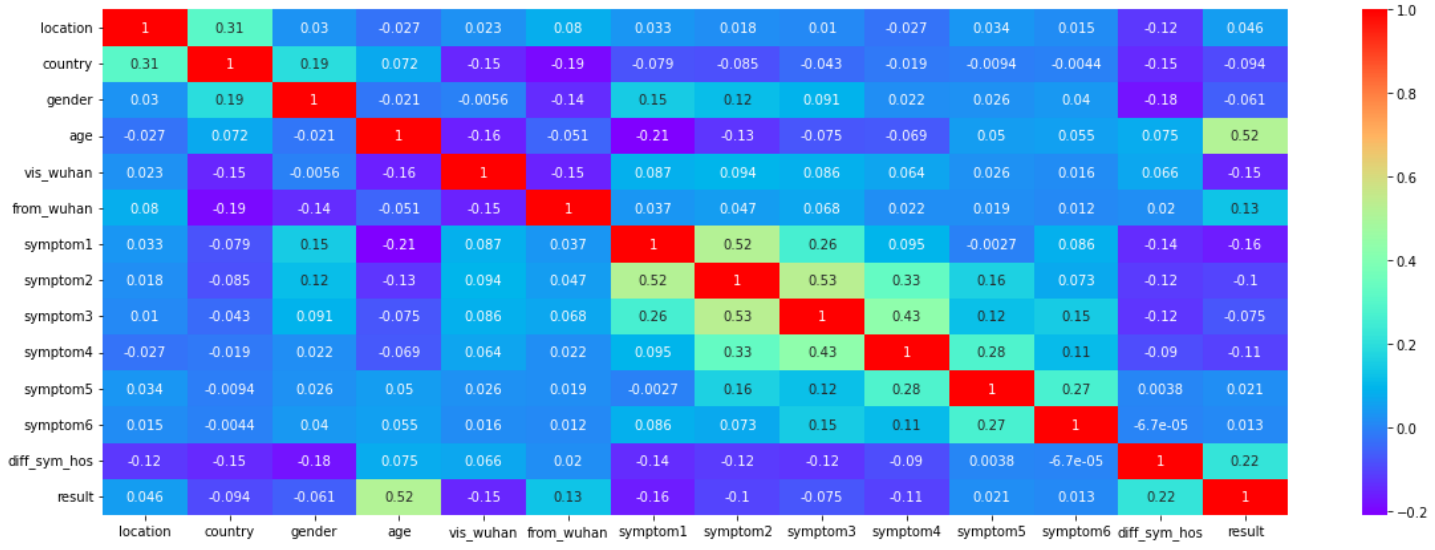
The method used for transformation the data type of features is called
`pd.get_dummies(features_values, features_names)`

The first feature, we've transformed is Gender to doesn't make one value in this feature preferable into another one in classifier model like decision tree classifier, also I have transformed symptoms features expect last two because they tend to be constant and to don't add more features unnecessarily.

Also, it will divide the whole dataset randomly after transforming it to training data and test data, and validation set to tune the hyperparameter of the dataset, but we didn't add it because we'll use `GridSearchCV(estimator = model, param_grid = classifier_parameters, cv = num_of_folds, scores = 'accuracy')` and it will have a validation set implicitly.

Note: the divided dataset is balanced (if `y_train` have 60% of 1s' then `y_test` will have 40% of 1s' and the same for zeros').

Also, it will plot the correlation of each feature with others.



	location	country	gender	age	vis_wuhan	from_wuhan	symptom1	symptom2	symptom3	symptom4	symptom5	symptom6	diff_sym_hos	result
location	1	0.31	0.03	-0.027	0.023	0.08	0.033	0.018	0.01	-0.027	0.034	0.015	-0.12	0.046
country	0.31	1	0.19	0.072	-0.15	-0.19	-0.079	-0.085	-0.043	-0.019	-0.0094	-0.0044	-0.15	-0.094
gender	0.03	0.19	1	-0.021	-0.0056	-0.14	0.15	0.12	0.091	0.022	0.026	0.04	-0.18	-0.061
age	-0.027	0.072	-0.021	1	-0.16	-0.051	-0.21	-0.13	-0.075	-0.069	0.05	0.055	0.075	0.52
vis_wuhan	0.023	-0.15	-0.0056	-0.16	1	-0.15	0.087	0.094	0.086	0.064	0.026	0.016	0.066	-0.15
from_wuhan	0.08	-0.19	-0.14	-0.051	-0.15	1	0.037	0.047	0.068	0.022	0.019	0.012	0.02	0.13
symptom1	0.033	-0.079	0.15	-0.21	0.087	0.037	1	0.52	0.26	0.095	-0.0027	0.086	-0.14	-0.16
symptom2	0.018	-0.085	0.12	-0.13	0.094	0.047	0.52	1	0.53	0.33	0.16	0.073	-0.12	-0.1
symptom3	0.01	-0.043	0.091	-0.075	0.086	0.068	0.26	0.53	1	0.43	0.12	0.15	-0.12	-0.075
symptom4	-0.027	-0.019	0.022	-0.069	0.064	0.022	0.095	0.33	0.43	1	0.28	0.11	-0.09	-0.11
symptom5	0.034	-0.0094	0.026	0.05	0.026	0.019	-0.0027	0.16	0.12	0.28	1	0.27	0.0038	0.021
symptom6	0.015	-0.0044	0.04	0.055	0.016	0.012	0.086	0.073	0.15	0.11	0.27	1	-6.7e-05	0.013
diff_sym_hos	-0.12	-0.15	-0.18	0.075	0.066	0.02	-0.14	-0.12	-0.12	-0.09	0.0038	-6.7e-05	1	0.22
result	0.046	-0.094	-0.061	0.52	-0.15	0.13	-0.16	-0.1	-0.075	-0.11	0.021	0.013	0.22	1

After we prepared our dataset, we'll start using our classifiers to train them on this clean data and see the results.

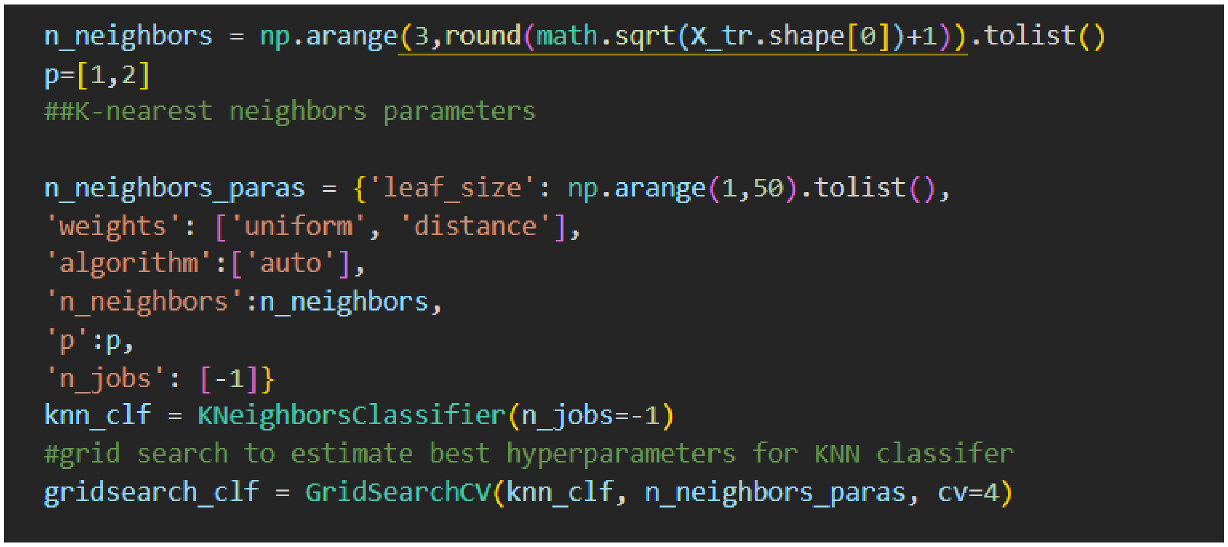
Algorithm

Note: please take care that the train and test step for all classifiers is exactly the same think so I illustrated it in the KNN classifier only to prevent more pages in the report.

1- KNN

It is a lazy learning algorithm that **stores all instances corresponding to training data in n-dimensional space**. It is a **lazy learning algorithm** as it does not focus on constructing a general internal model, instead, it works on storing instances of training data.

Hyperparameters for knn: leaf_size: [1-50], weights = [‘uniform’,’distance’], ‘algorithm’ = [‘auto’], ‘n_neighbors’ = from 3 to $\sqrt{num\ of\ samples} + 1$, p=[‘1’,’2’].

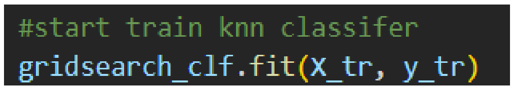


```
n_neighbors = np.arange(3,round(math.sqrt(X_tr.shape[0])+1)).tolist()
p=[1,2]
##K-nearest neighbors parameters

n_neighbors_paras = {'leaf_size': np.arange(1,50).tolist(),
'weights': ['uniform', 'distance'],
'algorithm':['auto'],
'n_neighbors':n_neighbors,
'p':p,
'n_jobs': [-1]}
knn_clf = KNeighborsClassifier(n_jobs=-1)
#grid search to estimate best hyperparameters for KNN classifer
gridsearch_clf = GridSearchCV(knn_clf, n_neighbors_paras, cv=4)
```

KNN best hyperparameter after using grid search

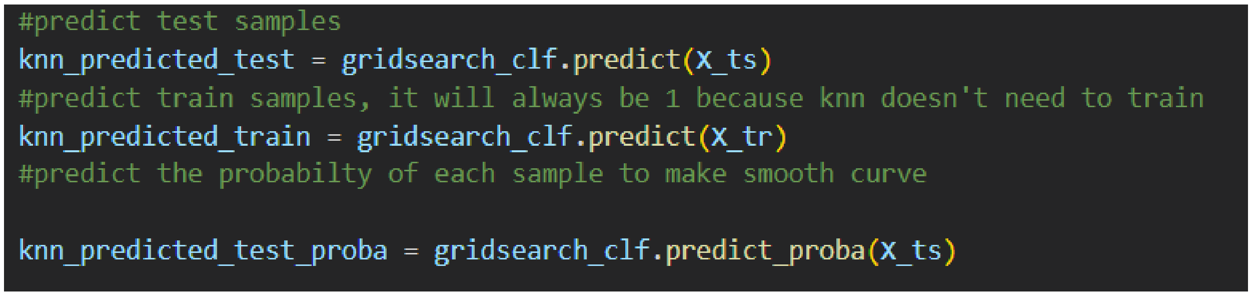
1- Leaf size = 1
2- n_jobs = -1 will run algorithm in parallel
3- n_neighbours = 25
4- weights = ‘distance

train the model using fit method for train set and test set using:



```
#start train knn classifer
gridsearch_clf.fit(X_tr, y_tr)
```

Test the model using:



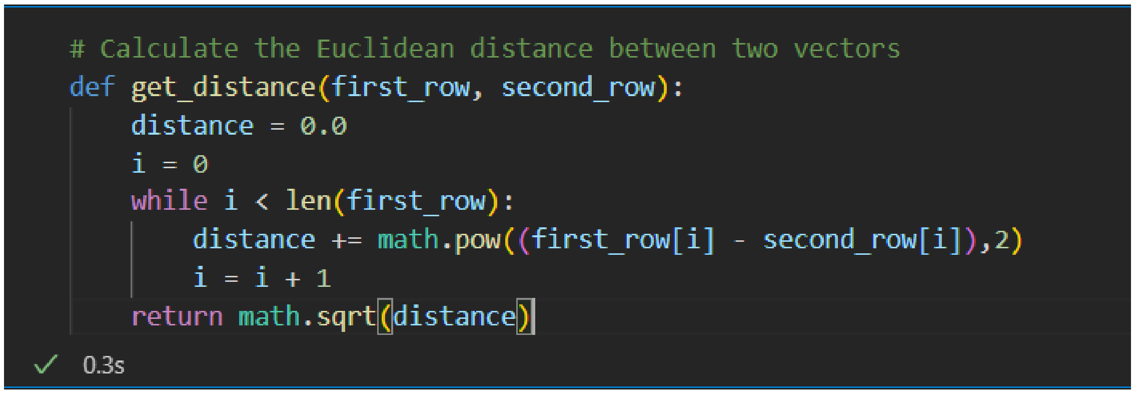
```
#predict test samples
knn_predicted_test = gridsearch_clf.predict(X_ts)
#predict train samples, it will always be 1 because knn doesn't need to train
knn_predicted_train = gridsearch_clf.predict(X_tr)
#predict the probabilty of each sample to make smooth curve

knn_predicted_test_proba = gridsearch_clf.predict_proba(X_ts)
```

estimated run time: 6.5 minute

KNN implementation from scratch (by Zeyad tarek)

Step 1: Calculate Euclidean Distance

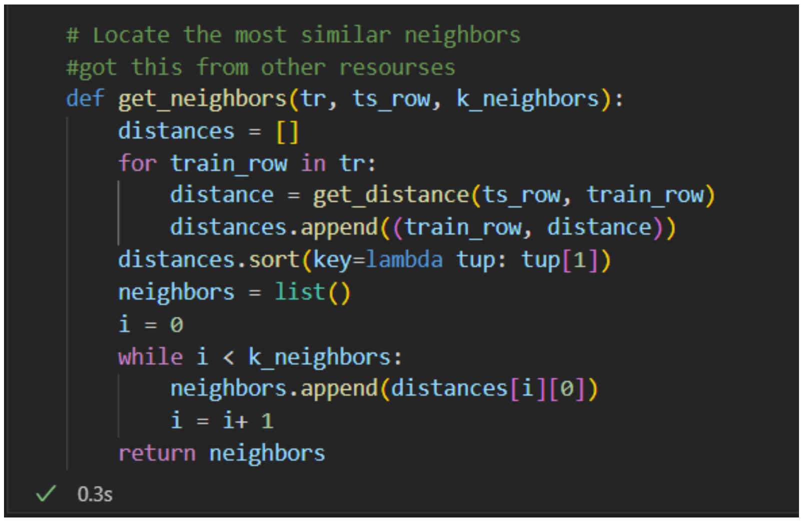
The first step is to calculate the distance between two samples in a data frame.



```
# Calculate the Euclidean distance between two vectors
def get_distance(first_row, second_row):
    distance = 0.0
    i = 0
    while i < len(first_row):
        distance += math.pow((first_row[i] - second_row[i]),2)
        i = i + 1
    return math.sqrt(distance)
```

Step 2: Get Nearest Neighbors

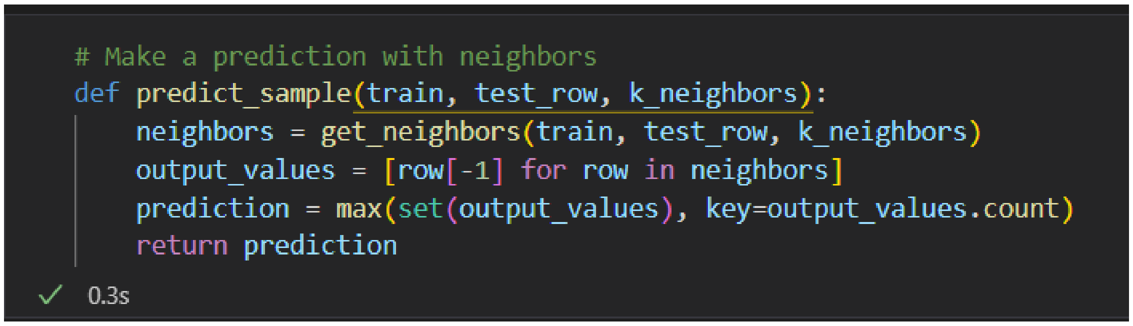
Neighbors for a new sample of data in the data frame are the *k* closest instances, as defined by our distance measure.



```
# Locate the most similar neighbors
#got this from other resourses
def get_neighbors(tr, ts_row, k_neighbors):
    distances = []
    for train_row in tr:
        distance = get_distance(ts_row, train_row)
        distances.append((train_row, distance))
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
    i = 0
    while i < k_neighbors:
        neighbors.append(distances[i][0])
        i = i+ 1
    return neighbors
✓ 0.3s
```

Step 3: Make Predictions

The most similar neighbors collected from the training dataset can be used to make predictions.



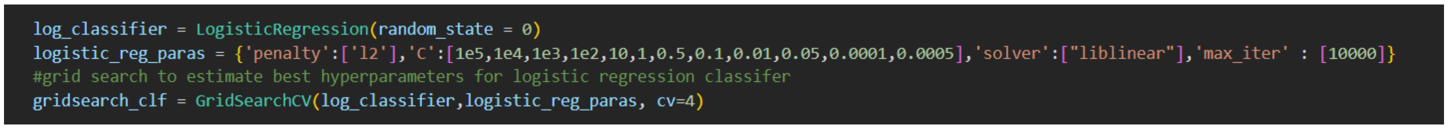
```
# Make a prediction with neighbors
def predict_sample(train, test_row, k_neighbors):
    neighbors = get_neighbors(train, test_row, k_neighbors)
    output_values = [row[-1] for row in neighbors]
    prediction = max(set(output_values), key=output_values.count)
    return prediction
✓ 0.3s
```

2- Logistic regression

It is a classification algorithm in machine learning that uses one or more independent variables to determine an outcome. The outcome is measured with a dichotomous variable meaning **it will have only two possible outcomes**.

Hyperparameter for logistic regression:

1- C = [1e5, 1e4, 1e3, 1e2, 10, 1, 0.5, 0.1, 0.005, 0.005, 0.0001,0.0005]
2- Penalty = [‘l2’],
3- Solver = [liblinear’]



```
log_classifier = LogisticRegression(random_state = 0)
logistic_reg_paras = {'penalty':['l2'],'C':[1e5,1e4,1e3,1e2,10,1,0.5,0.1,0.01,0.05,0.0001,0.0005],'solver':["liblinear"],'max_iter' : [10000]}
#grid search to estimate best hyperparameters for logistic regression classifer
gridsearch_clf = GridSearchCV(log_classifier,logistic_reg_paras, cv=4)
```

best hyperparameters for logistic regression after using grid search is:

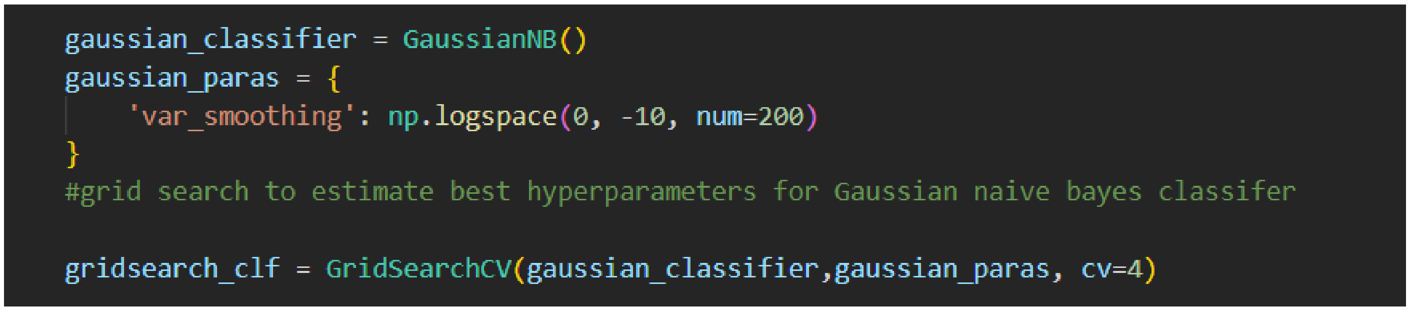
1- C=1
2- Max_iter = 10000
3- Solver = ‘liblinear’

3- Naïve Bayes

It is a classification algorithm based on **Bayes’s theorem** which gives an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. (Generative model).

Hyperparameter for logistic regression:

1- var_smoothing = np.logspace(0,-10)



```
gaussian_classifier = GaussianNB()
gaussian_paras = {
    'var_smoothing': np.logspace(0, -10, num=200)
}
#grid search to estimate best hyperparameters for Gaussian naive bayes classifer

gridsearch_clf = GridSearchCV(gaussian_classifier,gaussian_paras, cv=4)
```

Best var smoothing is = 0.0037211.

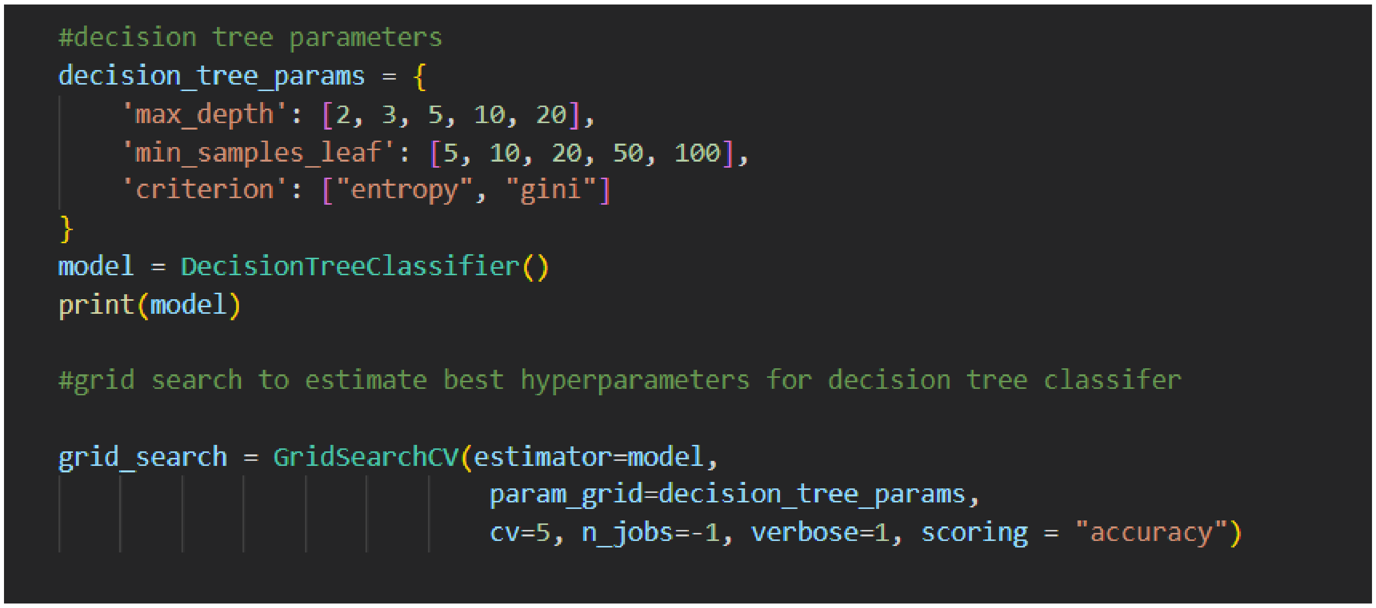
It may change because the data is split randomly.

4- Decision Tree (sklearn implementation by Zeyad Mohamed)

The decision tree algorithm builds the classification model in the form of a **tree structure**. It utilizes the if-then rules which are equally exhaustive and mutually exclusive in classification. The process goes on with breaking down the data into smaller structures and eventually associating it with an incremental decision tree. The final structure looks like a tree with nodes and leaves. The **rules are learned sequentially** using the training data one at a time. Each time a rule is learned, the tuples covering the rules are removed. The process continues the training set until the termination point is met.

Hyperparameters for Decision Tree.

1- Max_depth= [2,3,5,10,20],
2- Min_samples_leaf = [5,10,20,50,100]
3- n_neighbours = 25
4- criterion = [‘entropy’,gini’]



```
#decision tree parameters
decision_tree_params = {
    'max_depth': [2, 3, 5, 10, 20],
    'min_samples_leaf': [5, 10, 20, 50, 100],
    'criterion': ["entropy", "gini"]
}
model = DecisionTreeClassifier()
print(model)

#grid search to estimate best hyperparameters for decision tree classifer

grid_search = GridSearchCV(estimator=model,
                           param_grid=decision_tree_params,
                           cv=5, n_jobs=-1, verbose=1, scoring = "accuracy")
```

best Hyperparameters for Decision tree

1- max_depth = 5 or 10
2- min_samples_leaf = 5
3- criterion = gini

Decision tree implementation from scratch (most of it from outside resources)

1. step 1: create Gini index method

The Gini index is the name of the cost function used to evaluate splits in the dataset.

A split in the dataset involves one input attribute and one value for that attribute. It can be used to divide training patterns into two groups of rows.

2. step 2: Create Split

A split is comprised of an attribute in the dataset and a value.

We can summarize this as the index of an attribute to split and the value by which to split rows on that attribute. This is just a useful shorthand for indexing into rows of data.

Creating a split involves three parts, the first we have already looked at which is calculating the Gini score. The remaining two parts are:

1. Splitting a Dataset.
2. Evaluating All Splits

3. step 3: Evaluating all splits

With the Gini function above and the test split function we now have everything we need to evaluate splits.

Given a dataset, we must check every value on each attribute as a candidate split, evaluate the cost of the split and find the best possible split we could make.

Once the best split is found, we can use it as a node in our decision tree.

4. step 4: Build a Tree

Creating the root node of the tree is easy.

We call the above **get_split()** function using the entire dataset.
Adding more nodes to our tree is more interesting.

Building a tree may be divided into 3 main parts:

1. Terminal Nodes.
2. Recursive Splitting.
3. Building a Tree.

3.1. Terminal Nodes

We need to decide when to stop growing a tree.

We can do that using the depth and the number of rows that the node is responsible for in the training dataset.

- **Maximum Tree Depth**. This is the maximum number of nodes from the root node of the tree. Once a maximum depth of the tree is met, we must stop splitting adding new nodes. Deeper trees are more complex and are more likely to overfit the training data.
- **Minimum Node Records**. This is the minimum number of training patterns that a given node is responsible for. Once at or below this minimum, we must stop splitting and adding new nodes. Nodes that account for too few training patterns are expected to be too specific and are likely to overfit the training data.

These two approaches will be user-specified arguments to our tree building procedure.

5. Step 5: Make a Prediction

Making predictions with a decision tree involves navigating the tree with the specifically provided row of data.

5- SVM

Support Vector Machines are machine learning algorithms that are used for classification and regression purposes. SVMs are one of the powerful machine learning algorithms for classification, regression and outlier detection purposes. That aim to maximize the margin between the support vectors.

SVM algorithm is implemented using a kernel. It uses a technique called the kernel trick. The kernel is just a function that maps the data to a higher dimension where data is separable. A kernel transforms a low-dimensional input data space into a higher dimensional space.

In the context of SVMs, there are 4 popular kernels – Linear kernel, Polynomial kernel and Radial Basis Function (RBF) kernel (also called Gaussian kernel) and Sigmoid kernel. Through this project we will test Linear, Polynomial, and RBF with different hyperparameters to choose the best model

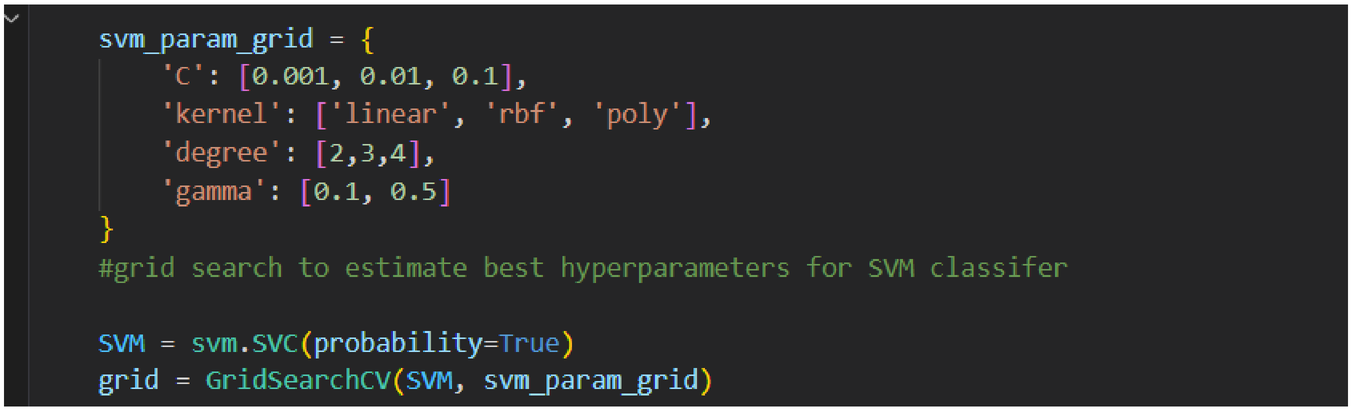
2. Implementing SVM using sklearn.svm

A python library 'Scikit Learn' provides 'SVC' class, which is used to implement SVM. So, in the beginning we try to get a high-level insight by using SVM classifier with simplest hyperparameter which is the linear kernel and measuring its score that resulting accuracy = 0.953667

In the second step, we build our model using GrideSearchCV() with different hyperparameters as shown below

Hyperparameters for SVM.

1- C = [0.001,0.01,0.1]
2- Kernel = [‘linear’, ‘rbf’, ‘poly’]
3- Degree = [2,3,4,5]
4- Gamma = [0.1, 0.5]



```python
svm_param_grid = {
    'C': [0.001, 0.01, 0.1],
    'kernel': ['linear', 'rbf', 'poly'],
    'degree': [2,3,4],
    'gamma': [0.1, 0.5]
}
#grid search to estimate best hyperparameters for SVM classifer

SVM = svm.SVC(probability=True)
grid = GridSearchCV(SVM, svm_param_grid)
```

Best hyperparameters for svm

1- C =0.01

2- Degree = 2
3- Gamma = 0.5
4- Kernel = “poly”
5- Probability= True

SVM implementation explanation:

General required for algorithm

- xi.w + b <= -1 if yi = -1 (belongs to -ve class)
- xi.w + b >= +1if yi = +1 (belongs to +ve class)

o We need to solve this optimization problem

$$\begin{aligned} \max_\alpha \quad & W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t.} \quad & 0 \le \alpha_i \le C, \quad i = 1, \ldots, n \\ & \sum_{i=1}^{n} \alpha_i y^{(i)} = 0, \end{aligned}$$

- We will solve this optimization problem by using the CVXOPT library in python that solves the quadratic programming problems. According to:

$$\begin{aligned} & \min \frac{1}{2} x^T P x + q^T x \\ s.t. \quad & Gx \le h \\ & Ax = b \end{aligned}$$

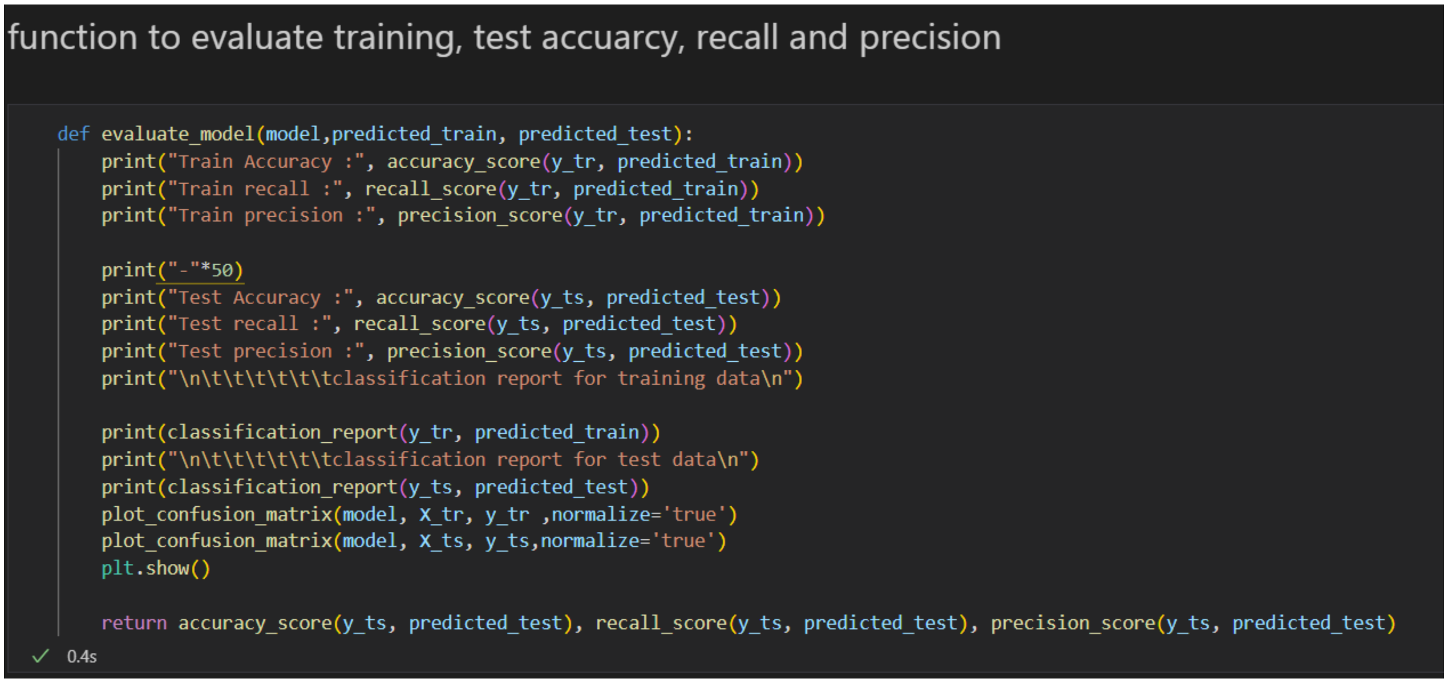
- So, we will get the values of alpha which will help us to get the values for weights and bias
- And by using the values of w and b we can make our prediction according to: sign(x*w + b)

Results

Let’s find the accuracy, recall, precision for each classifier.

Note, f1-score values illustrated in the code for each classifier.

Code to get the accuracy, recall and precision for each classifier.

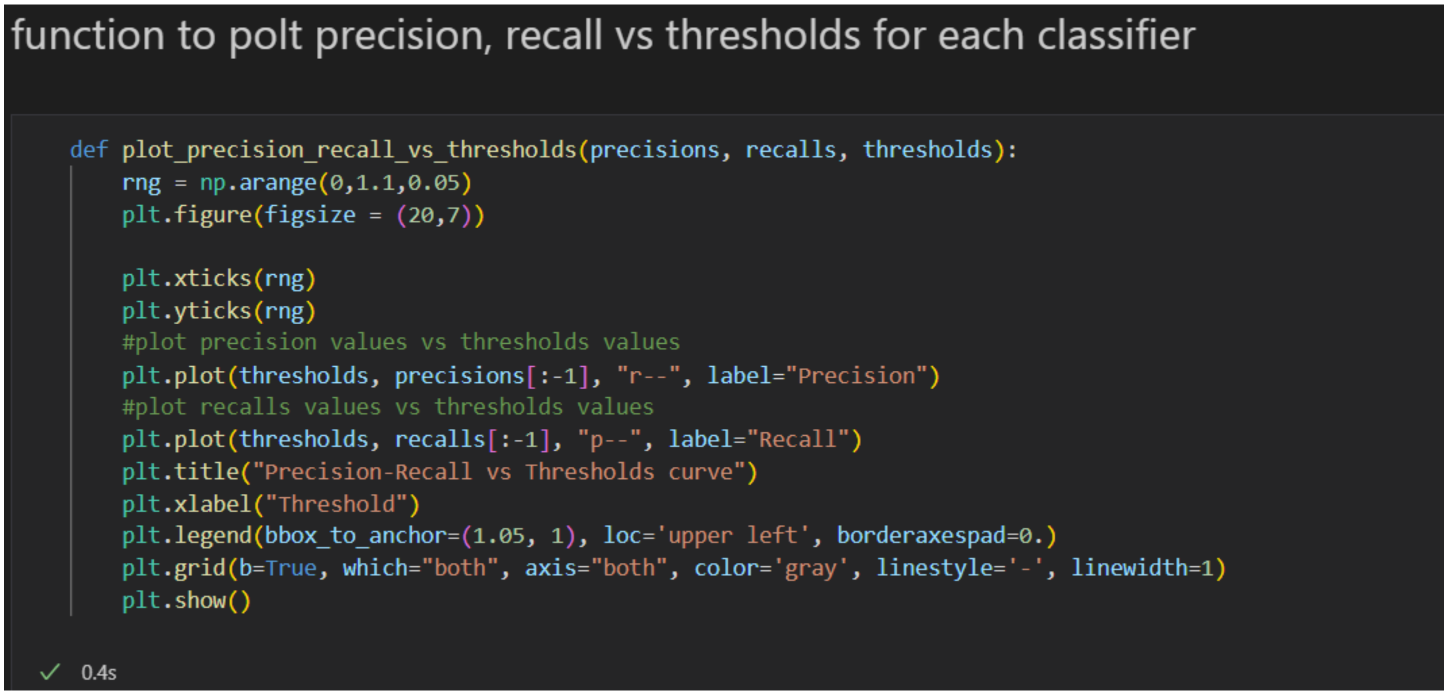


function to evaluate training, test accuarcy, recall and precision

```python
def evaluate_model(model,predicted_train, predicted_test):
    print("Train Accuracy :", accuracy_score(y_tr, predicted_train))
    print("Train recall :", recall_score(y_tr, predicted_train))
    print("Train precision :", precision_score(y_tr, predicted_train))

    print("-"*50)
    print("Test Accuracy :", accuracy_score(y_ts, predicted_test))
    print("Test recall :", recall_score(y_ts, predicted_test))
    print("Test precision :", precision_score(y_ts, predicted_test))
    print("\n\t\t\t\t\t\tclassification report for training data\n")

    print(classification_report(y_tr, predicted_train))
    print("\n\t\t\t\t\t\tclassification report for test data\n")
    print(classification_report(y_ts, predicted_test))
    plot_confusion_matrix(model, X_tr, y_tr ,normalize='true')
    plot_confusion_matrix(model, X_ts, y_ts,normalize='true')
    plt.show()

    return accuracy_score(y_ts, predicted_test), recall_score(y_ts, predicted_test), precision_score(y_ts, predicted_test)
```

✓ 0.4s

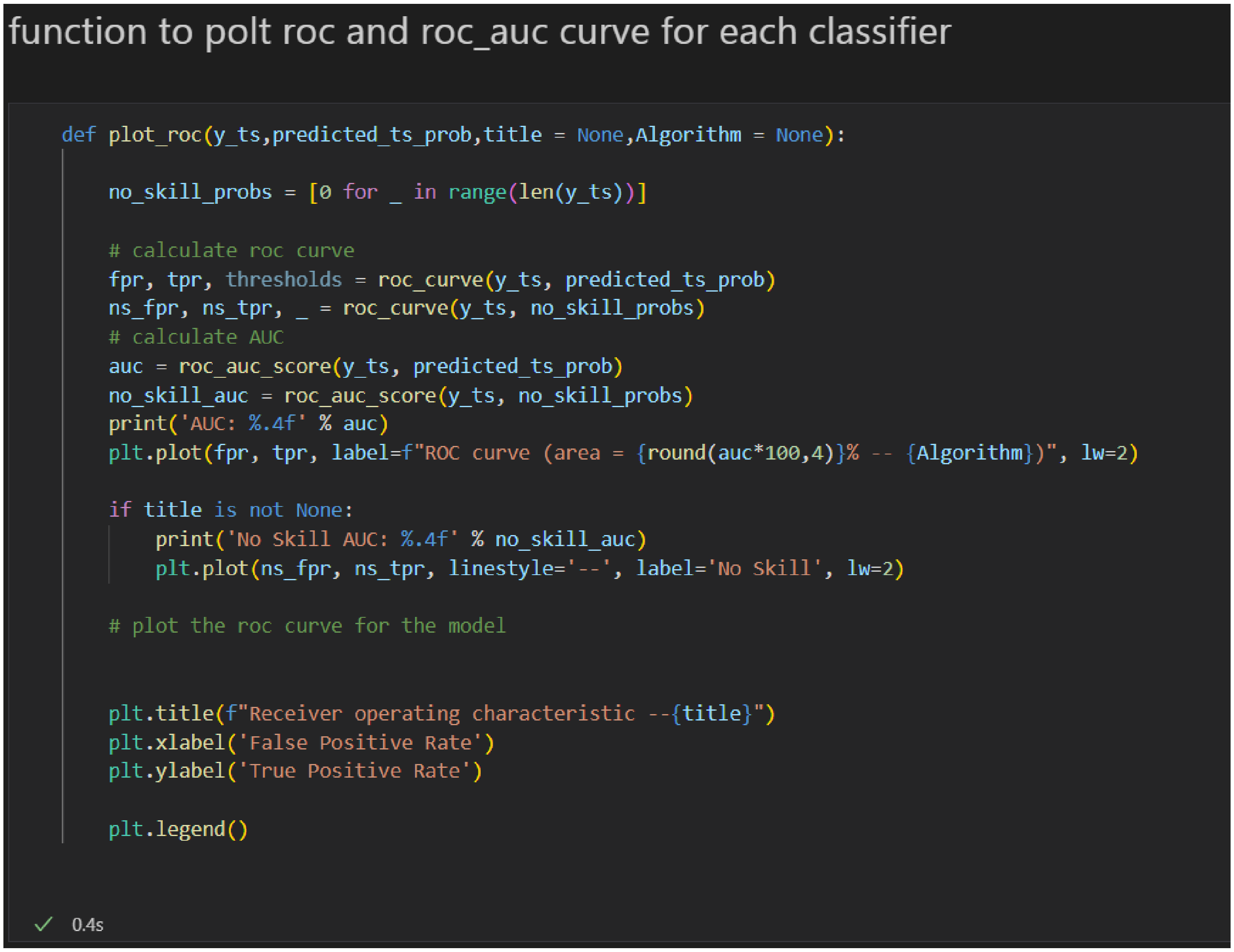
Code to plot recalls and precisions vs thresholds for each classifier.


function to polt precision, recall vs thresholds for each classifier

```
def plot_precision_recall_vs_thresholds(precisions, recalls, thresholds):
    rng = np.arange(0,1.1,0.05)
    plt.figure(figsize = (20,7))

    plt.xticks(rng)
    plt.yticks(rng)
    #plot precision values vs thresholds values
    plt.plot(thresholds, precisions[:-1], "r--", label="Precision")
    #plot recalls values vs thresholds values
    plt.plot(thresholds, recalls[:-1], "p--", label="Recall")
    plt.title("Precision-Recall vs Thresholds curve")
    plt.xlabel("Threshold")
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
    plt.grid(b=True, which="both", axis="both", color='gray', linestyle='-', linewidth=1)
    plt.show()
```

✓ 0.4s

Code to plot roc and roc_auc curve for each classifier.

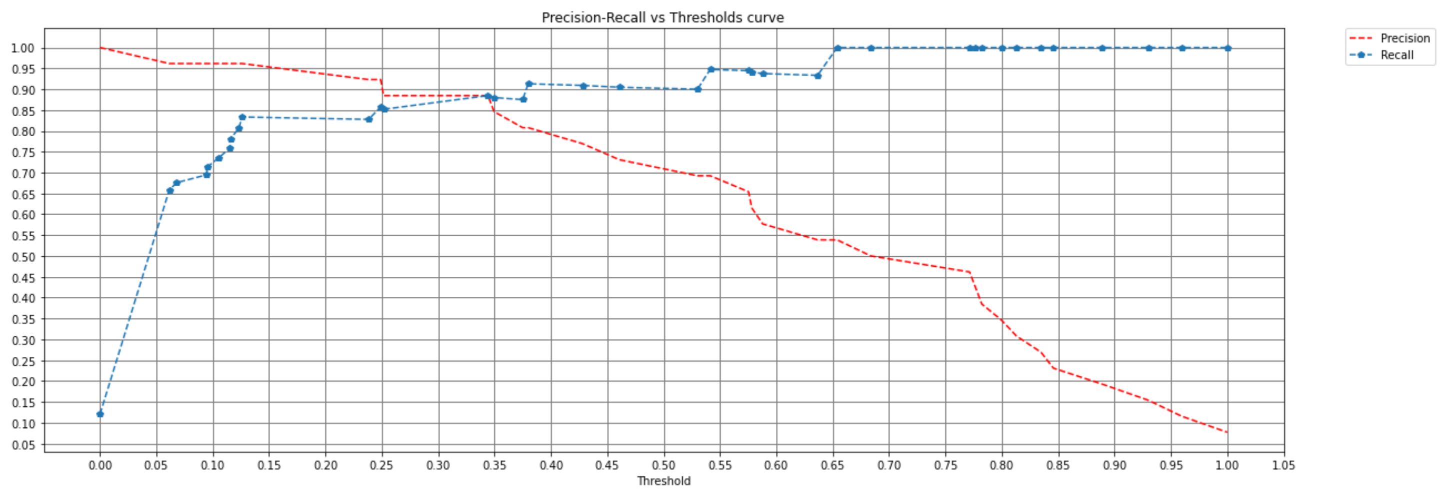

function to polt roc and roc_auc curve for each classifier

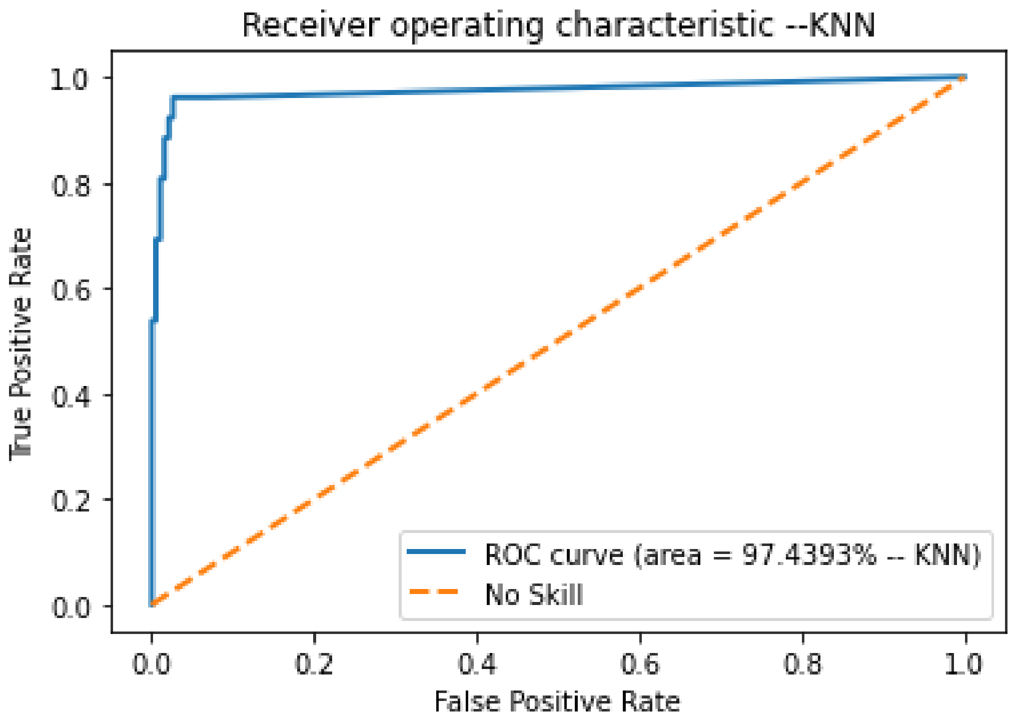
```
def plot_roc(y_ts,predicted_ts_prob,title = None,Algorithm = None):

    no_skill_probs = [0 for _ in range(len(y_ts))]

    # calculate roc curve
    fpr, tpr, thresholds = roc_curve(y_ts, predicted_ts_prob)
    ns_fpr, ns_tpr, _ = roc_curve(y_ts, no_skill_probs)
    # calculate AUC
    auc = roc_auc_score(y_ts, predicted_ts_prob)
    no_skill_auc = roc_auc_score(y_ts, no_skill_probs)
    print('AUC: %.4f' % auc)
    plt.plot(fpr, tpr, label=f"ROC curve (area = {round(auc*100,4)}% -- {Algorithm})", lw=2)

    if title is not None:
        print('No Skill AUC: %.4f' % no_skill_auc)
        plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill', lw=2)

    # plot the roc curve for the model


    plt.title(f"Receiver operating characteristic --{title}")
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')

    plt.legend()
```

✓ 0.4s

First:- k-nearest neighbor

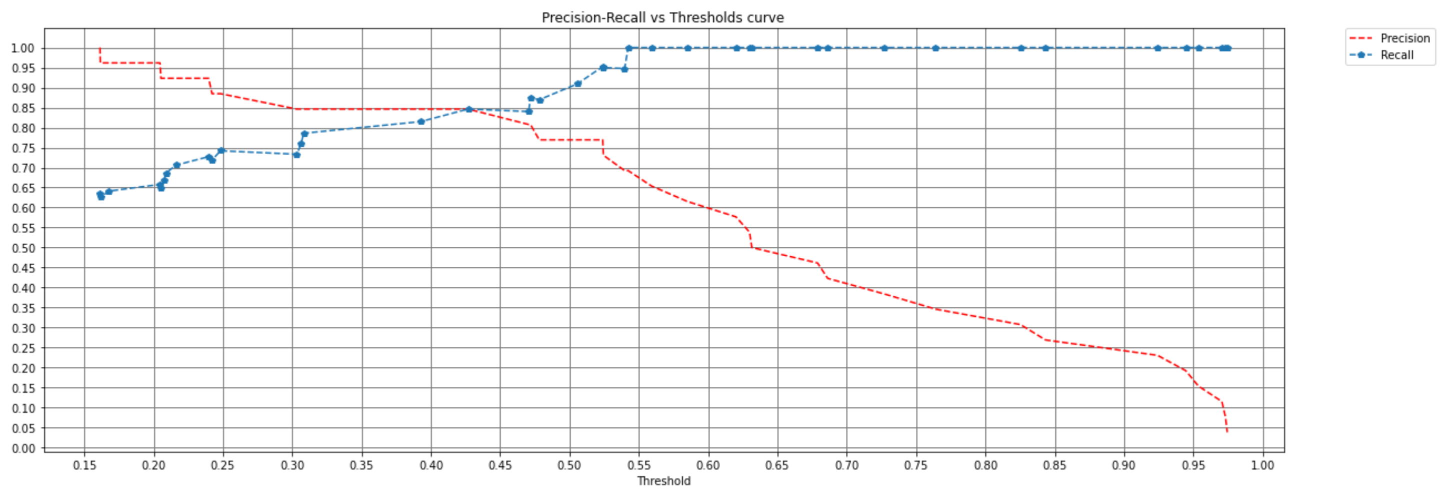
	K- nearest neighbor scikit-learn	K- nearest neighbor implementation
Accuracy	97.9%	94.2 %
Recall	63.50%	85.7%
precision	90%	77.15%

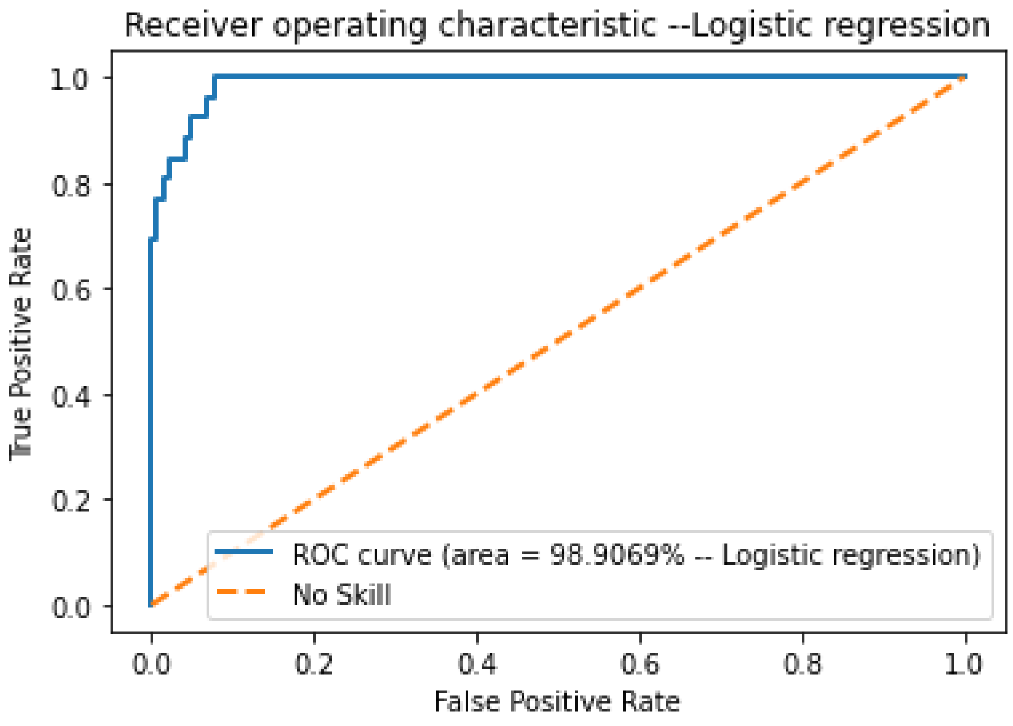




Second:- Logistic regression

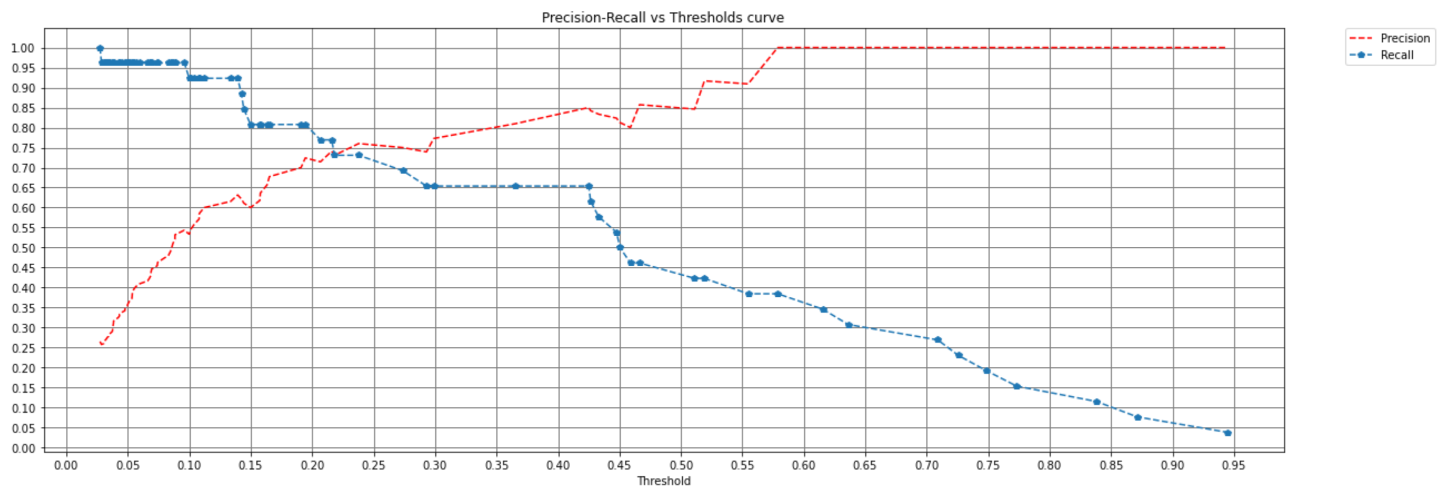
	Logistic regression scikit learn	Logistic regression implementation
Accuracy	96.4444%	14.00%
recall	67%	30%
precision	90%	90%

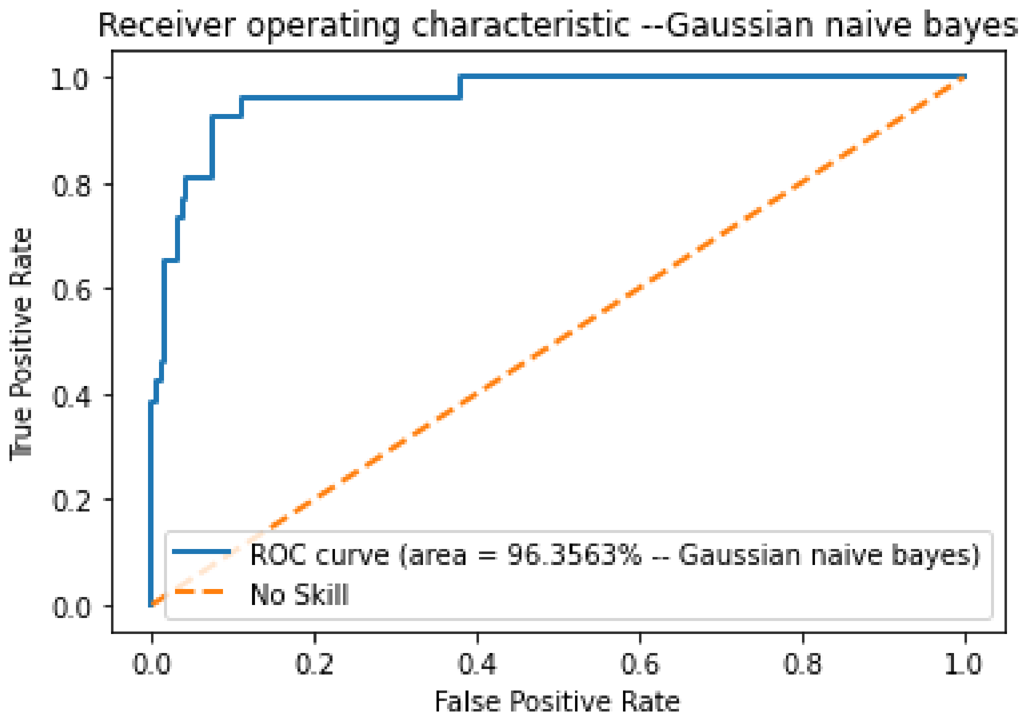




Third: Naïve bayes classifier

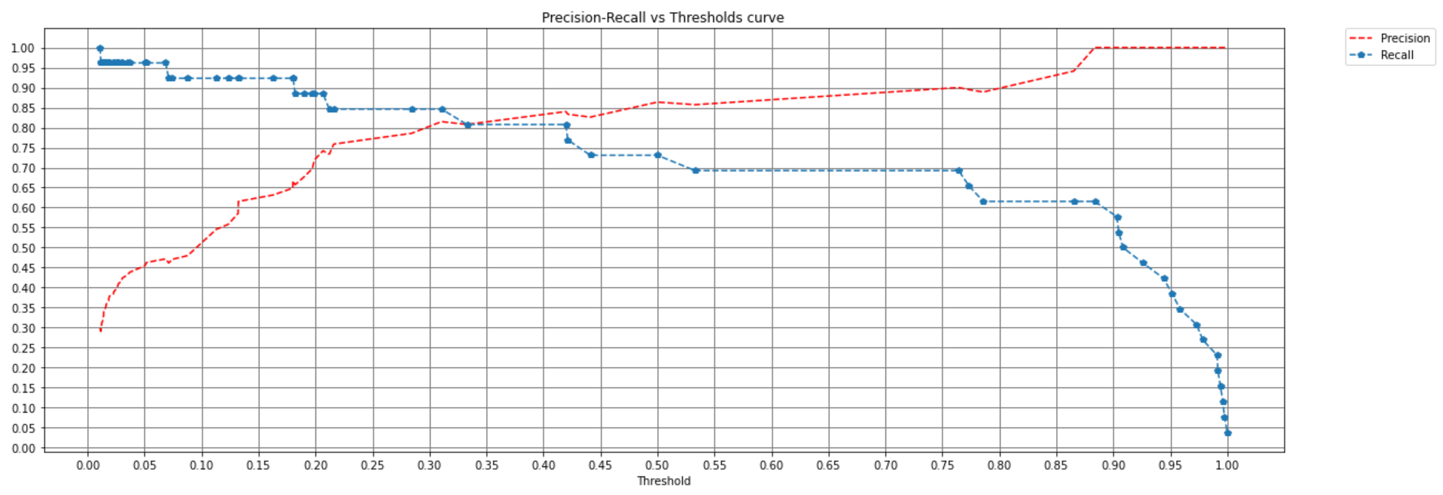
	Naïve bayes scikit learn	Naïve bayes implementation
Accuracy	92.54%	10.5%
recall	42%	20%
precision	84%	85.66%

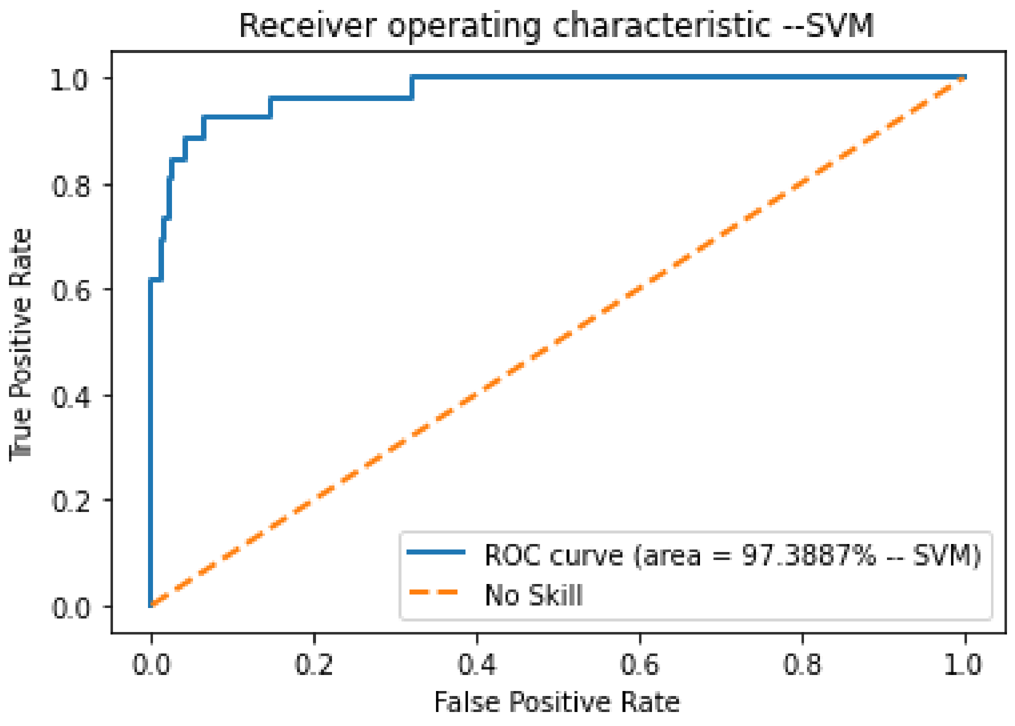




Fourth:- Support vector machines:

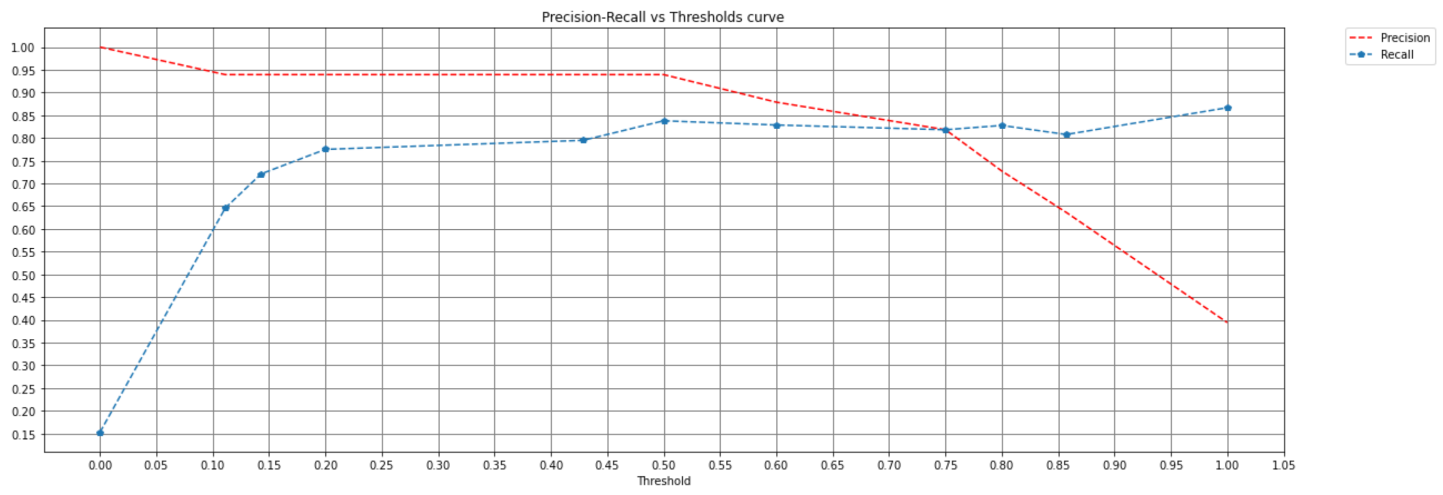
	SVM scikit learn	SVM implementation
Accuracy	96.5%	95.6%
recall	66.0%	75.33%
precision	91.2%	60.4%

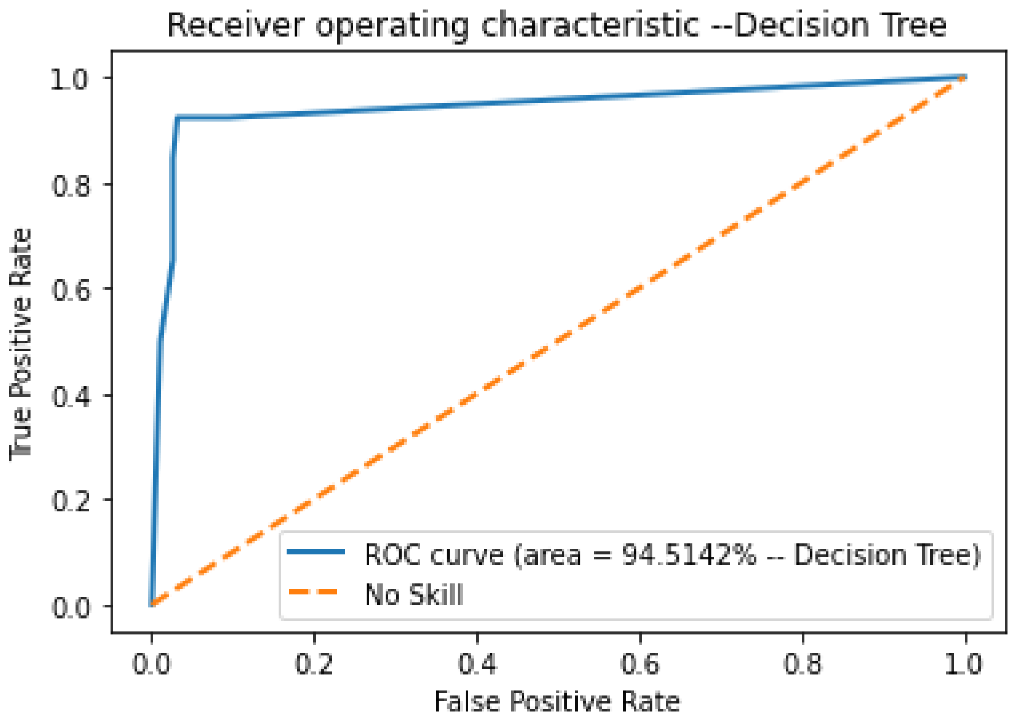

Precision-Recall vs Thresholds curve
Precision
Recall
Threshold
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
0.55
0.60
0.65
0.70
0.75
0.80
0.85
0.90
0.95
1.00

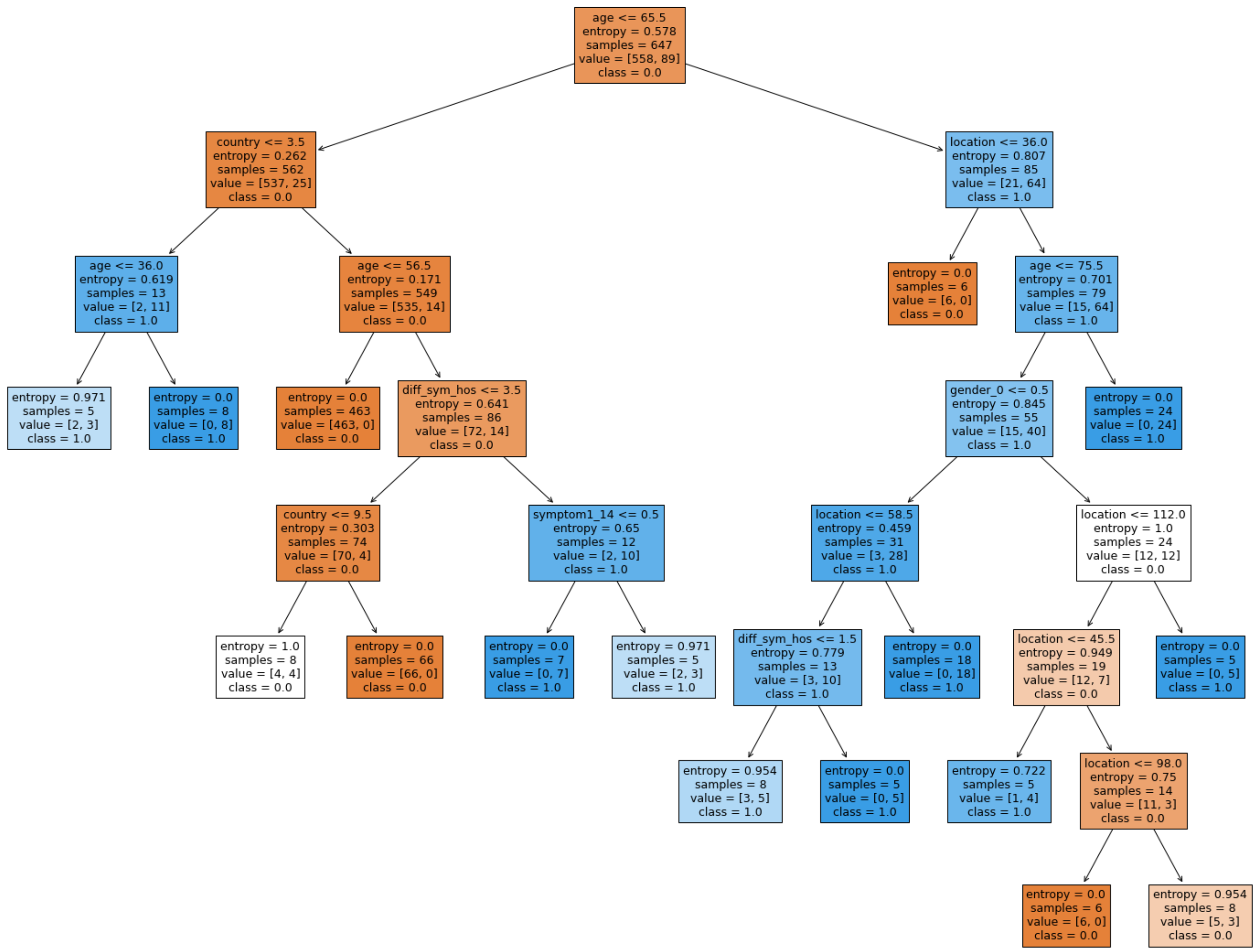

Receiver operating characteristic --SVM
True Positive Rate
False Positive Rate
ROC curve (area = 97.3887% -- SVM)
No Skill
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

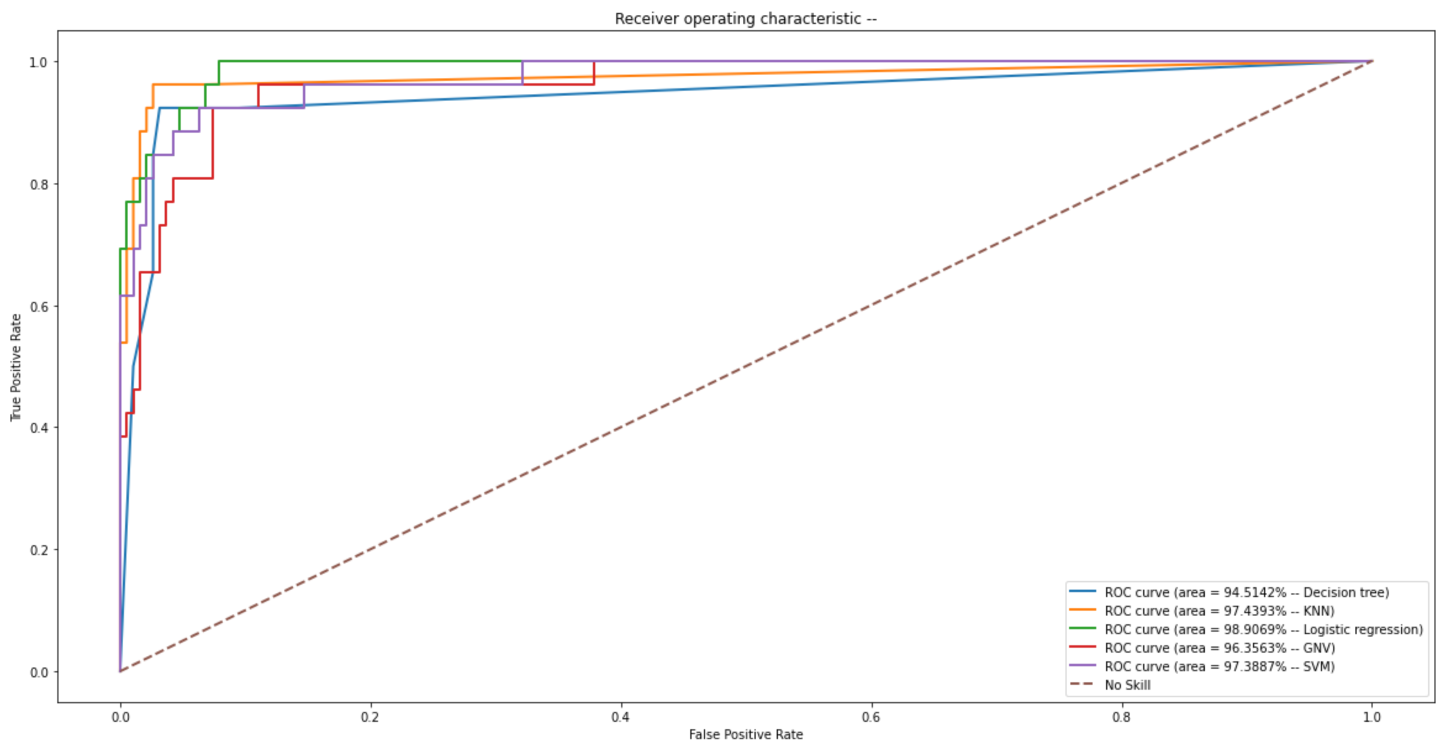
Fifth:- Decision tree classifier

	Decision tree scikit learn	Decision tree implementation
Accuracy	97.5%	95.6%
recall	84.0%	93.33%
precision	64.2%	75.4%






age <= 65.5 entropy = 0.578 samples = 647 value = [558, 89] class = 0.0
country <= 3.5 entropy = 0.262 samples = 562 value = [537, 25] class = 0.0
location <= 36.0 entropy = 0.807 samples = 85 value = [21, 64] class = 1.0
age <= 36.0 entropy = 0.619 samples = 13 value = [2, 11] class = 1.0
age <= 56.5 entropy = 0.171 samples = 549 value = [535, 14] class = 0.0
entropy = 0.0 samples = 6 value = [6, 0] class = 0.0
age <= 75.5 entropy = 0.701 samples = 79 value = [15, 64] class = 1.0
entropy = 0.971 samples = 5 value = [2, 3] class = 1.0
entropy = 0.0 samples = 8 value = [0, 8] class = 1.0
entropy = 0.0 samples = 463 value = [463, 0] class = 0.0
diff_sym_hos <= 3.5 entropy = 0.641 samples = 86 value = [72, 14] class = 0.0
gender_0 <= 0.5 entropy = 0.845 samples = 55 value = [15, 40] class = 1.0
entropy = 0.0 samples = 24 value = [0, 24] class = 1.0
country <= 9.5 entropy = 0.303 samples = 74 value = [70, 4] class = 0.0
symptom1_14 <= 0.5 entropy = 0.65 samples = 12 value = [2, 10] class = 1.0
location <= 58.5 entropy = 0.459 samples = 31 value = [3, 28] class = 1.0
location <= 112.0 entropy = 1.0 samples = 24 value = [12, 12] class = 0.0
entropy = 1.0 samples = 8 value = [4, 4] class = 0.0
entropy = 0.0 samples = 66 value = [66, 0] class = 0.0
entropy = 0.0 samples = 7 value = [0, 7] class = 1.0
entropy = 0.971 samples = 5 value = [2, 3] class = 1.0
diff_sym_hos <= 1.5 entropy = 0.779 samples = 13 value = [3, 10] class = 1.0
entropy = 0.0 samples = 18 value = [0, 18] class = 1.0
location <= 45.5 entropy = 0.949 samples = 19 value = [12, 7] class = 0.0
entropy = 0.0 samples = 5 value = [0, 5] class = 1.0
entropy = 0.954 samples = 8 value = [3, 5] class = 1.0
entropy = 0.0 samples = 5 value = [0, 5] class = 1.0
entropy = 0.722 samples = 5 value = [1, 4] class = 1.0
location <= 98.0 entropy = 0.75 samples = 14 value = [11, 3] class = 0.0
entropy = 0.0 samples = 6 value = [6, 0] class = 0.0
entropy = 0.954 samples = 8 value = [5, 3] class = 0.0



Result:

From above result we that logistic regression and SVM are the best classifiers for such that problem, they have the highest accuracy, precision and recall.

But due to that the data is split randomly the result may change.