

COVID-19 Outcome Prediction

By

Ahmed Waheed Ahmed Mohamed

CONTENTS

Introduction.....	3
Dataset view.....	3
Methodology	4
Algorithms	5
K-Nearest Neighbours	21
Logistic regression	27
Naive Bayes	33
Decision Trees	35
Support vector machines.....	35
Results.....	56
Test results	58
.....	59
.....	61
Histograms.....	63
Discussion.	66
Market Analysis	68
Region X.....	71
Country Y	73
Male vs Female.....	75
Age Groups.....	78
Conclusions	81
Recommendations	83
Group Meeting.....	86
Regulatory Bodies	87
Agents.....	88
	91

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	6612	3	1	8	1
101	8	0	5612	3	1	0	0
137	8	1	4612	3	1	13	0
116	8	0	6012	3	1	0	0
116	8	1	5812	3	1	0	0
23	8	0	4412	3	1	0	0
105	8	1	3412	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.

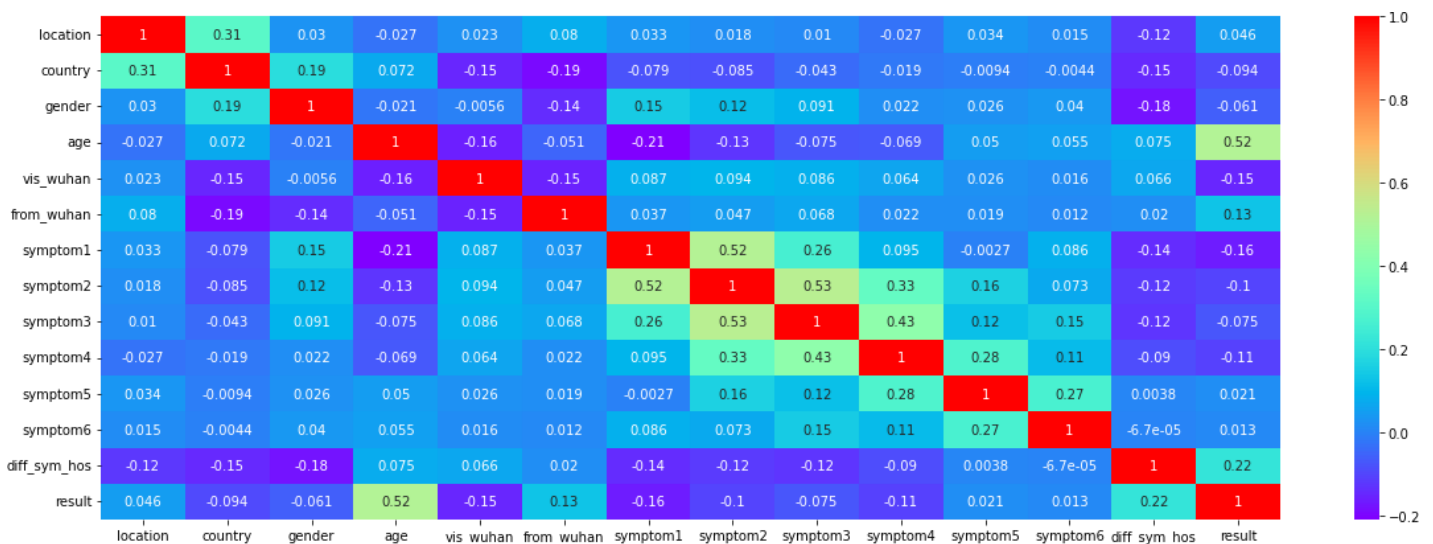
The method used for transformation the data type of features is called `pd.get_dummies(features_values, features_names)`

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 except 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.



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

Algorithm

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_neighbours' = $\sqrt{\text{number of samples}}$, p=[1,2].

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'

estimated run time: 1.5 minute

KNN implementation

Step 1: Calculate Euclidean Distance

The first step is to calculate the distance between two rows in a dataset.

Step 2: Get Nearest Neighbors

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

Step 3: Make Predictions

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

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- $Solver = ['liblinear']$

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.

Gaussian Naïve bayes has no best hyperparameter, it's **generative model**.

4- Decision Tree

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']

best Hyperparameters for Decision tree

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

Decision tree implementation

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 GridSearchCV() with different hyperparameters as shown below

Hyperparameters for SVM.

- 1- C = [0.001,0.01,0.1, 1, 10, 100]
- 2- Kernel = ['linear', 'rbf', 'poly']
- 3- Degree = [1,2,3,4,5]
- 4- Gamma = [0.1, 0.5, 0.7]

Best hyperparameters for svm

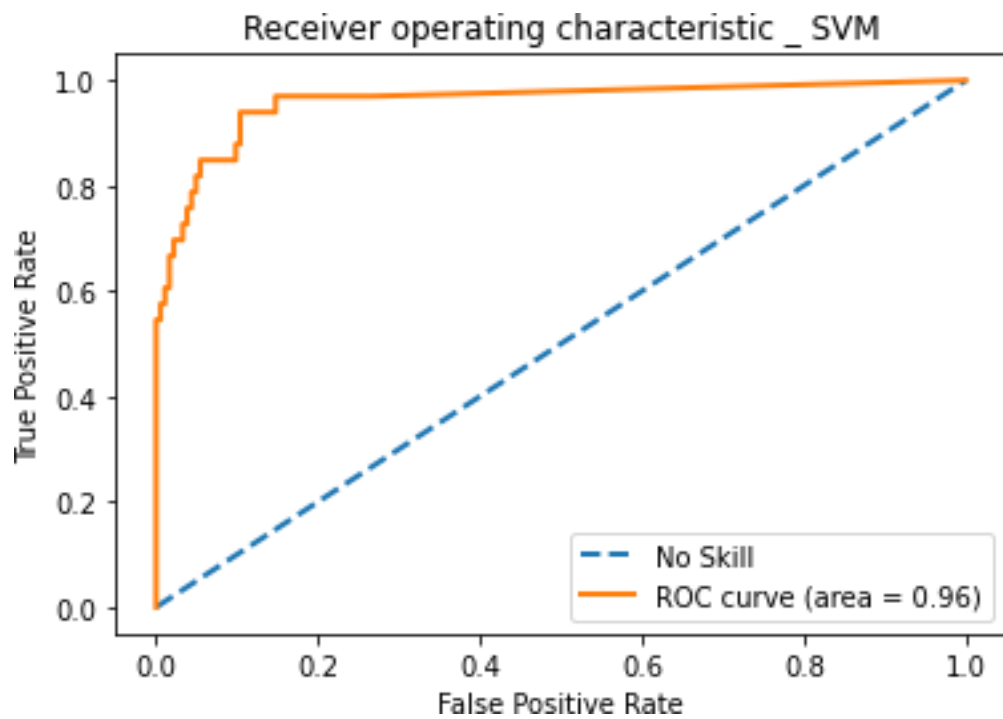
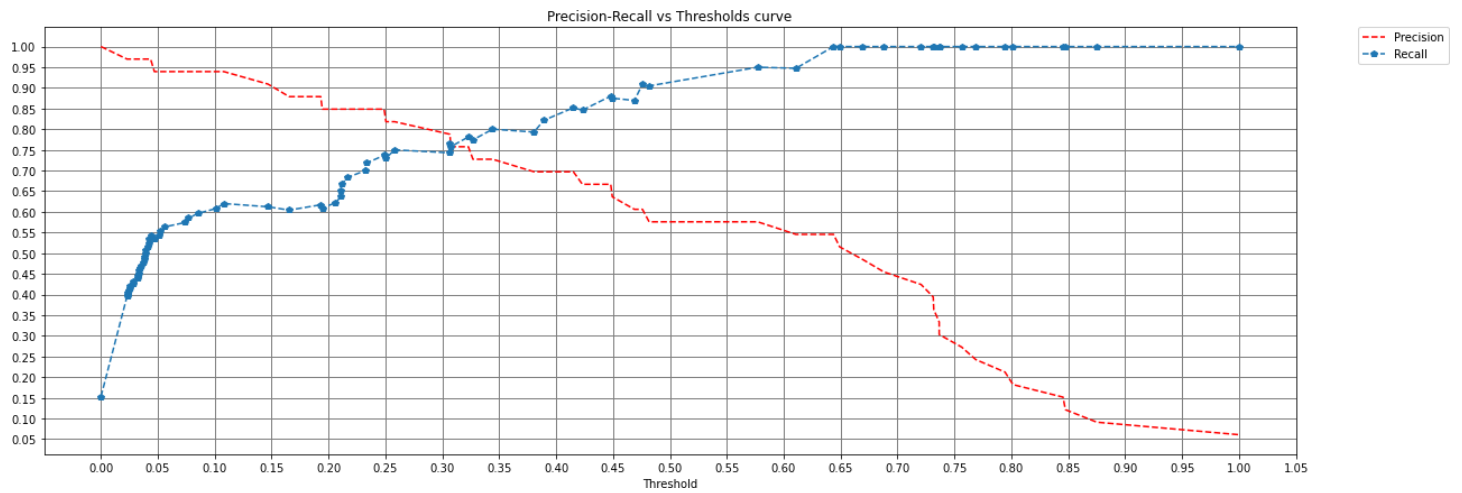
- 1- C =0.1
- 2- Gamma = 0.1
- 3- Kernel = "linear"
- 4- Probability= True

Results

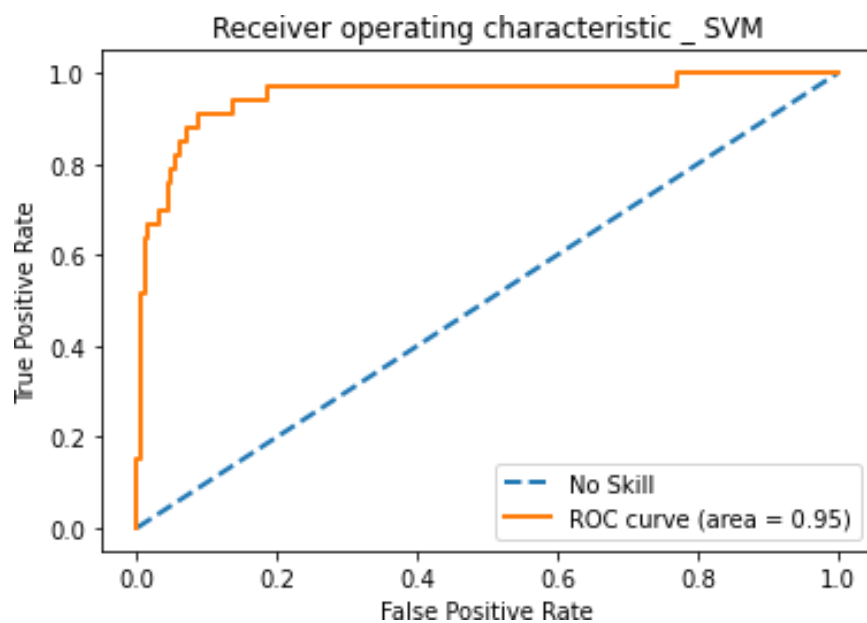
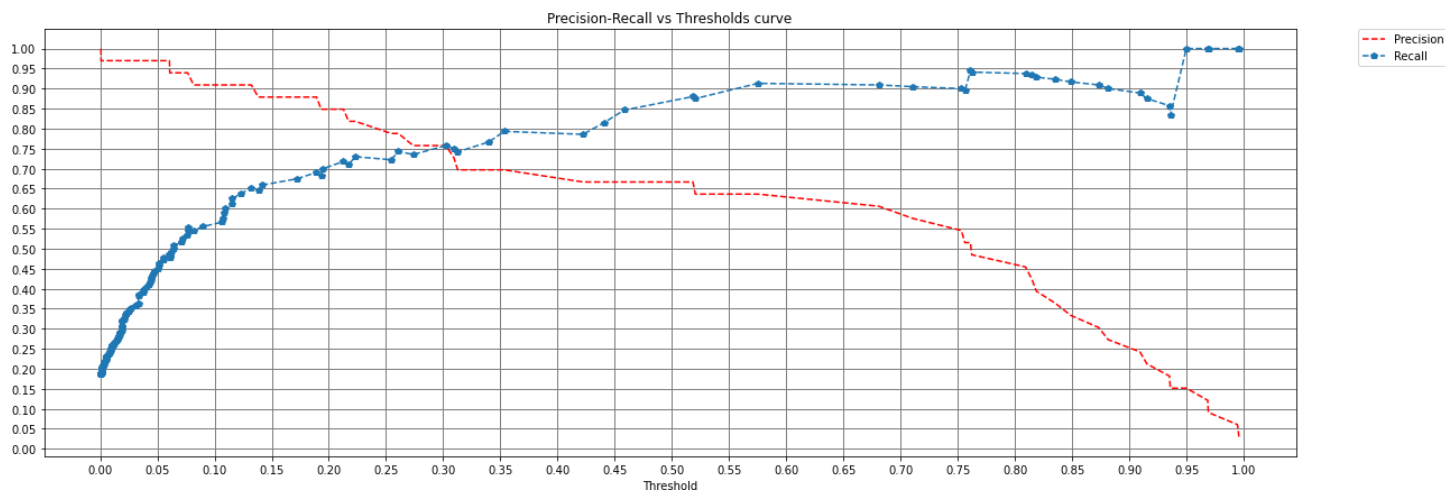
All classifiers got good result with one-hot encoded data except for Naïve bayes.

Let's the accuracy for each classifier

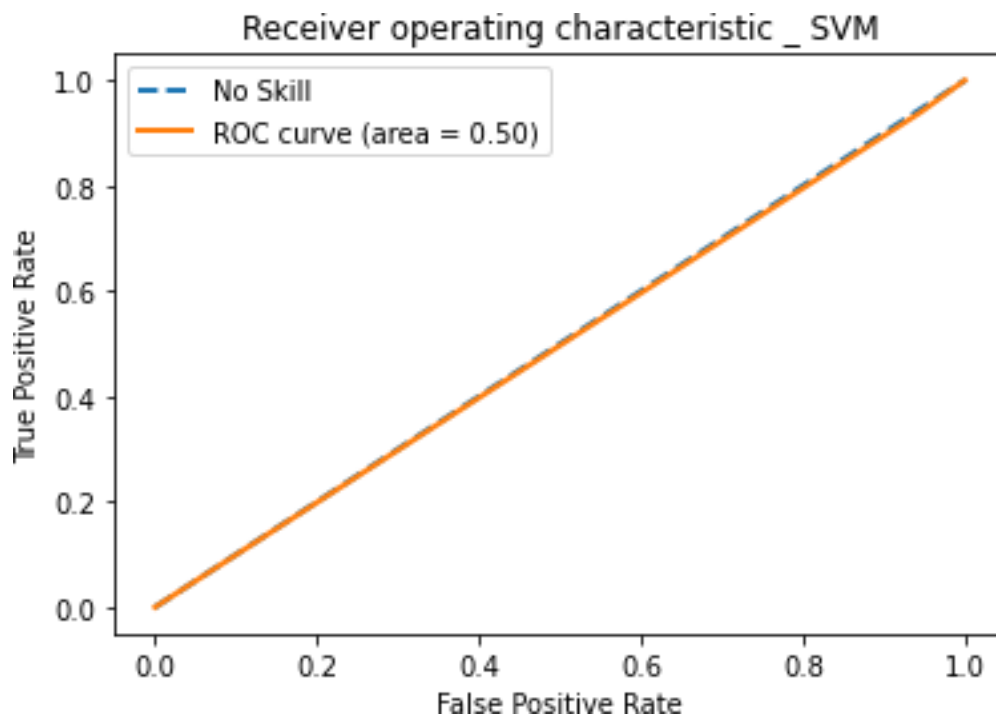
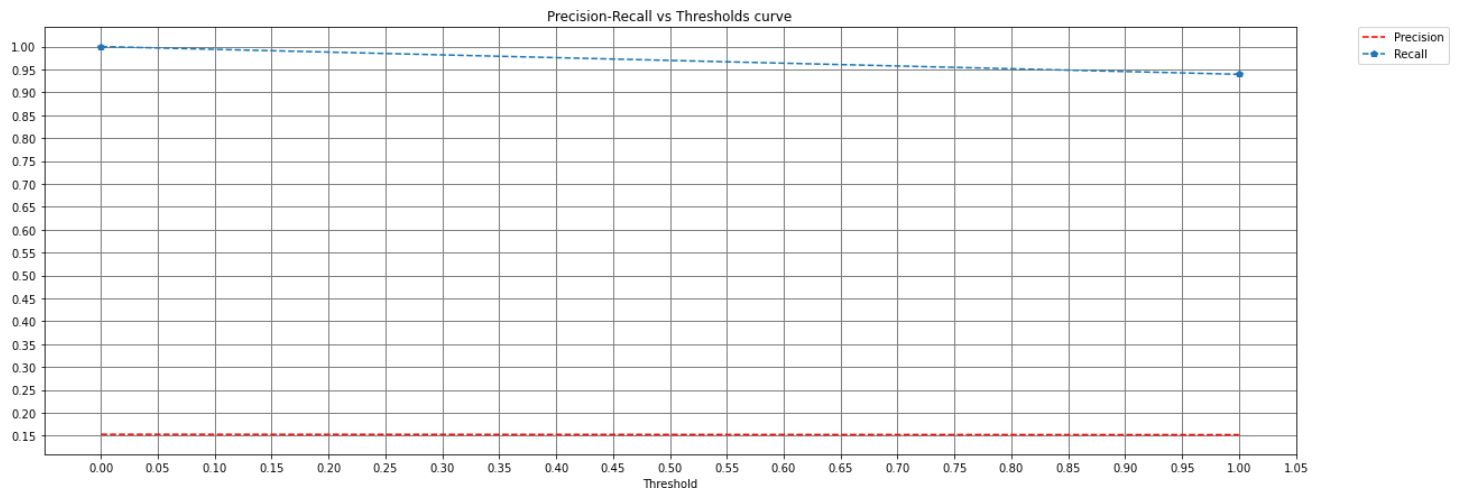
	K- nearest neighbor scikit-learn	K- nearest neighbor implementation
Accuracy	94.9%	94.2 %
Recall	63.50%	85.7%
precision	0.75%	51.15%



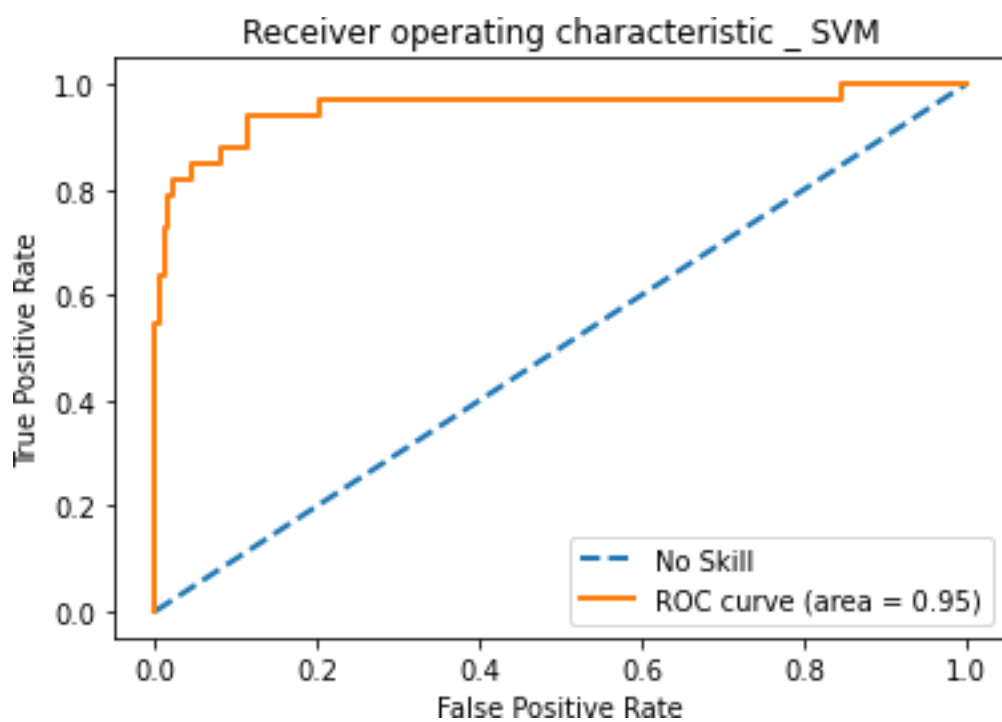
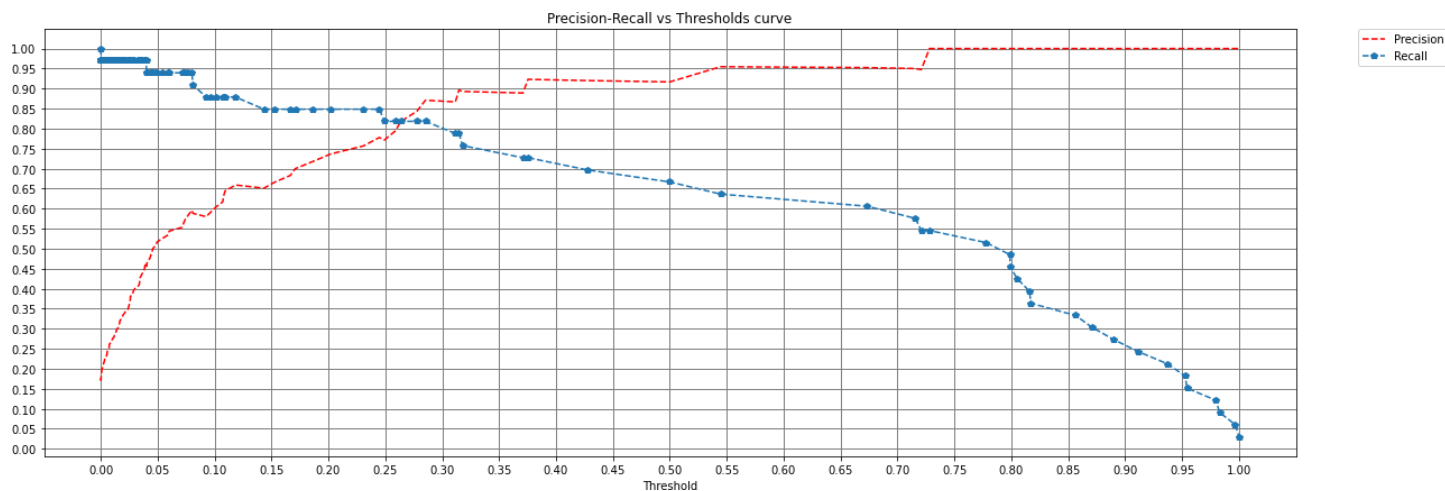
	Logistic regression scikit learn	Logistic regression implementation
Accuracy	94.4444%	14.00%
recall	63%	0.30%
precision	70%	90%



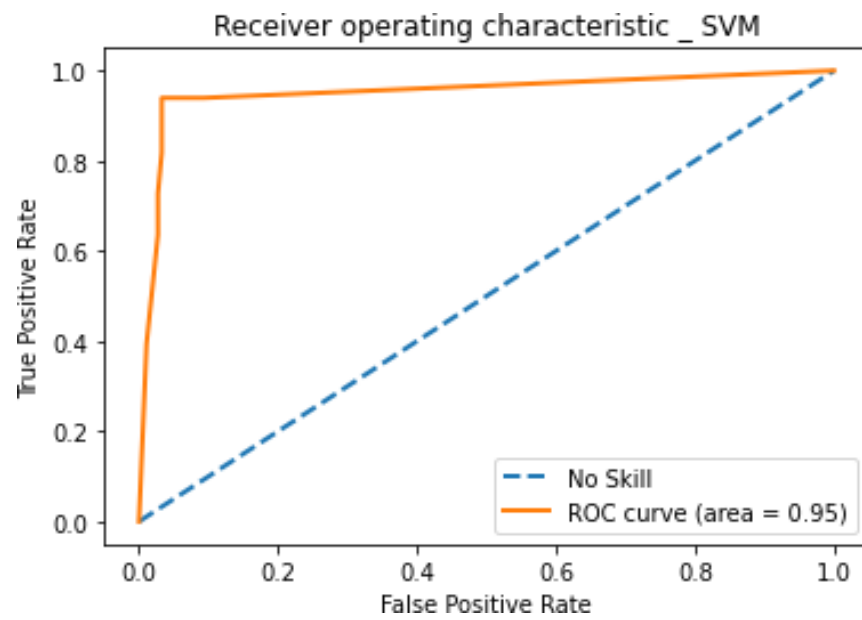
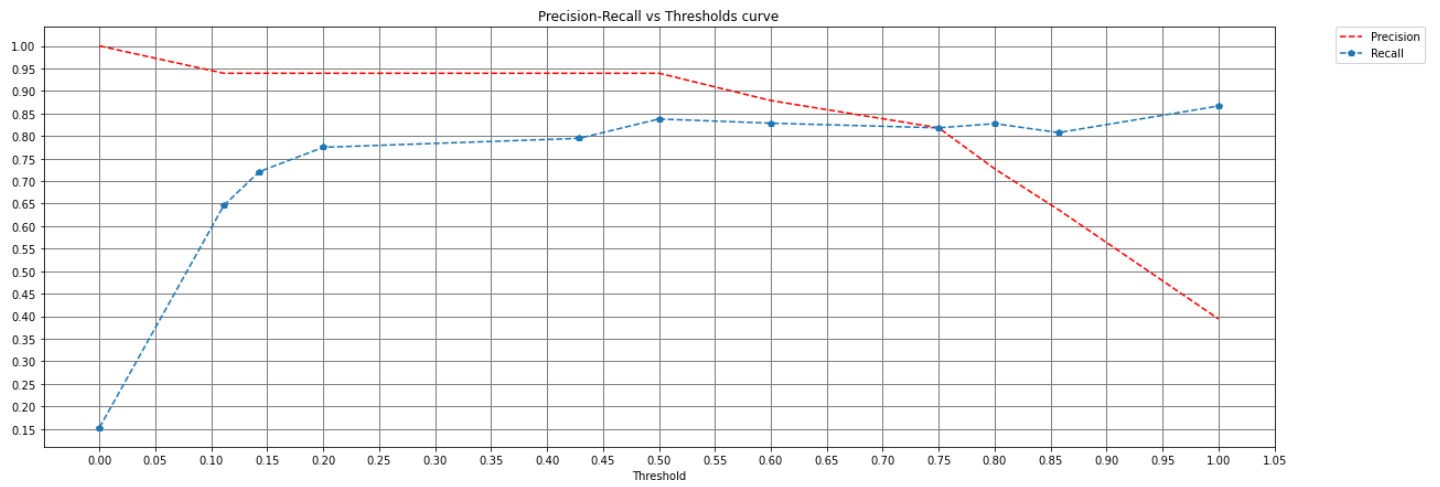
	Naïve bayes scikit learn	Naïve bayes implementation
Accuracy	11.54%	10.5%
recall	63%	0.20%
precision	70%	100%

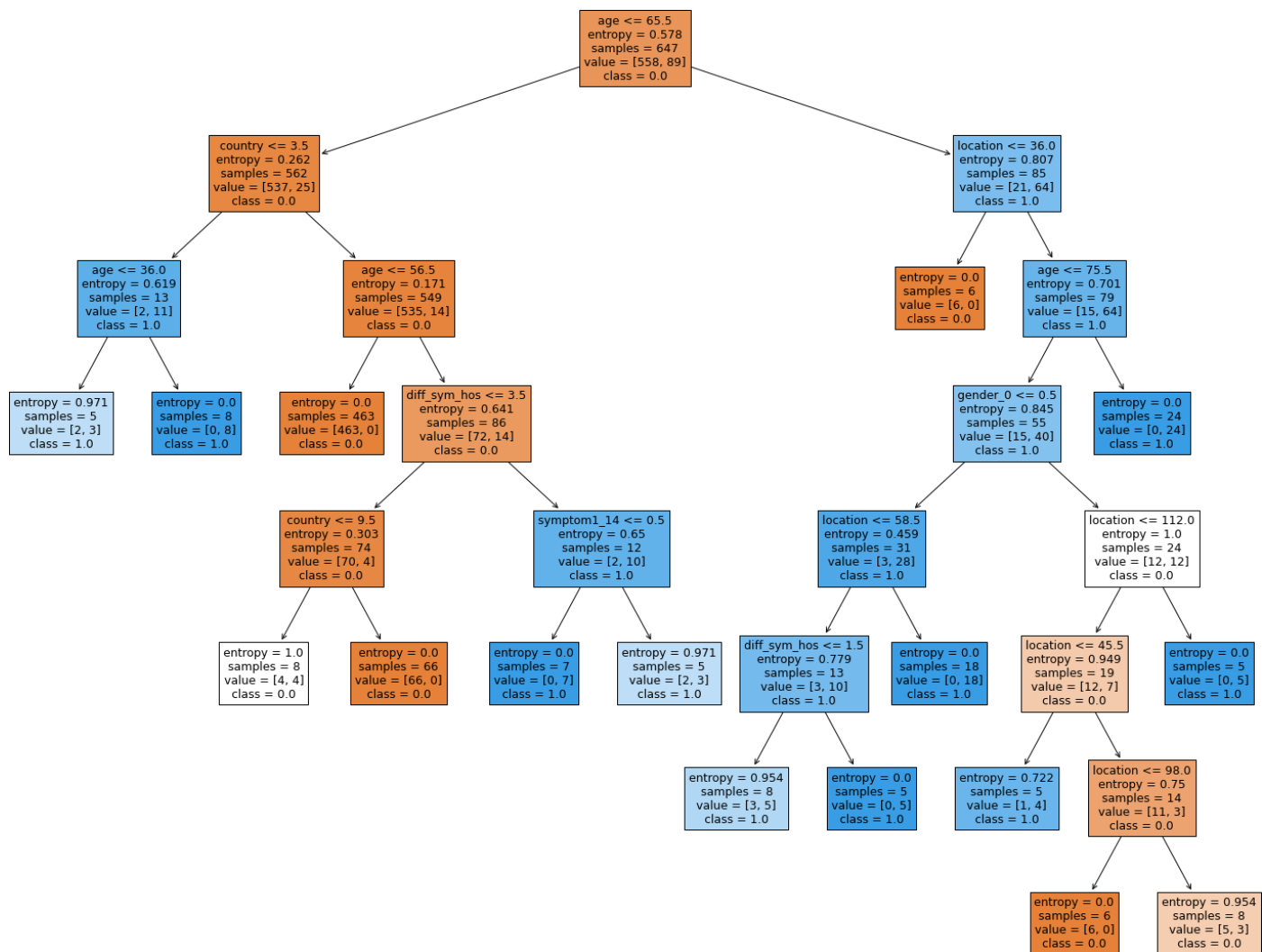


	SVM scikit learn	SVM implementation
Accuracy	96.5%	95.6%
recall	66.0%	75.33%
precision	91.2%	60.4%



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





Result:

From above result we that svm and decision tree are the best classifiers for such that problem, they have the highest accuracy, precision and recall.

