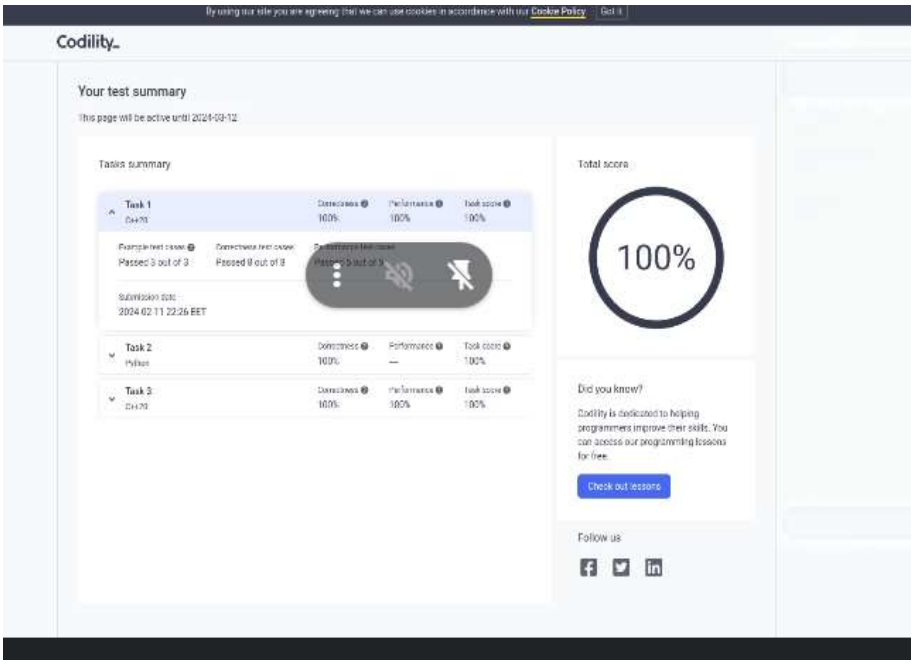
Artificial Intelligence and Machine Learning COVID-19 Outcome Prediction



Introduction:

The dataset that was used in this project is used to design classifiers that would help identify whether a person is going to recover from coronavirus or not based on 14 attributes including the person's country, age, gender and six different coronavirus symptoms.

Preprocessing:

-We first started by exploring the shape of the dataset (number of instances and features). The dataset contains **863** instances and **14** features.

-Next we viewed the names and data types for each column (feature). Two of the column names were not clear. One of them is "vis_wuhan" which indicated whether the person has visited Wuhan before or not. The name was changed to "visited_wuhan" for more clarity. The same was applied to the column "diff_sym_hos" which indicated the number of days that passed before symptoms began to appear on the patient. It was changed to "days_before_symptoms".

-We checked for null values in the dataset but none was found.

-We viewed the correlation between each independent variable (feature) and our target variable. The "age" was the feature with the highest correlation "0.515127".

-Since most of the features were categorical, one-hot-encoded was used through pandas' function "get_dummies" to convert their columns to categorical representations of values (0 and 1). Only the "age" and "days_before_symptoms" columns were left unchanged since they represent numerical data.

-We split the dataset into 3 sets: training, validation and test with ratios (0.6,0.2,0.2) respectively. The training set would be used to train the classifier, then the validation would be used to tune the hyperparameters and finally the test set would be used to test how good the classifier operates on unseen data.

Classifiers:

1-KNN:

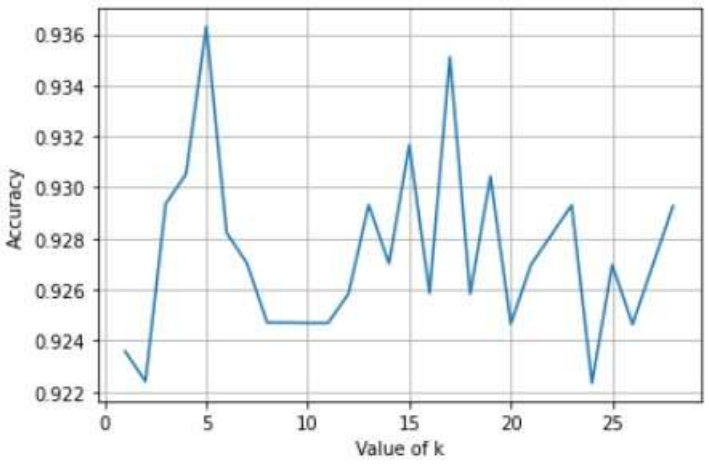
-We first started by using different values for K (3,5,7) and checking the accuracy score on the validation set.

-Since K=3 produced the highest accuracy (0.948) we used it for now till further optimization and used "MinMaxScaler" to normalize the numeric data columns so that their values lie between 0 and 1. However, the accuracy score dropped by 2% after the normalization step so we went with the un-normalized data.

-Based on the convention that K should be less than the square root of n, with n being the number of instances in the dataset. Therefore, K should be less than 29.

K-fold cross validation was used to find the best K for the KNN.

10 folds were used and K from values 1 to 29 were tested.



As we can see here, values of 5 and 17 gave the best scores. We settled on K=17 since picking a small value of K would cause the model to overfit.

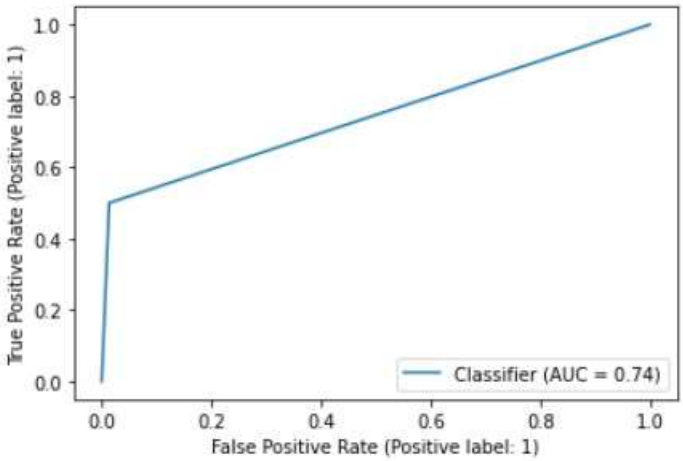
-The classifier produced the following values for accuracy, precision, recall and f1score.

```
print("Accuracy: ", accuracy_score(y_test,y_predict_test))
print("Precision: ", precision_score(y_test,y_predict_test))
print("Recall: ", recall_score(y_test,y_predict_test))
print("f1score: ", f1_score(y_test,y_predict_test))
```

```
Accuracy:  0.9190751445086706
Precision:  0.9166666666666666
Recall:  0.4583333333333333
f1score:  0.611111111111111
```

The model gave a much higher value of precision (0.917) as compared to the recall of (0.458). -

Finally we plotted the ROC curve and calculated the AUC score




```
AUC: 0.743288590604027
```

-We went on and implemented the KNN algorithm from scratch. The intuition here was to calculate the Euclidean distance between the new instance and all the instances in the training dataset using all the features for calculating the distance, then choosing the K instances with the smallest distances and using the mode of their label to predict the label for our new instance.

The KNN algorithm that we implemented from scratch gave the same accuracy as the one we used before from the Scikit-learn library (91%)

Using our implemented algorithm on the training and test sets with the optimal value of k that we got through cross validation

```
[ ] y_pred = knn_alg(X_train.values,y_train,X_test.values, 17)

    accuracy_score(y_test, y_pred)
```

```
0.9190751445086706
```

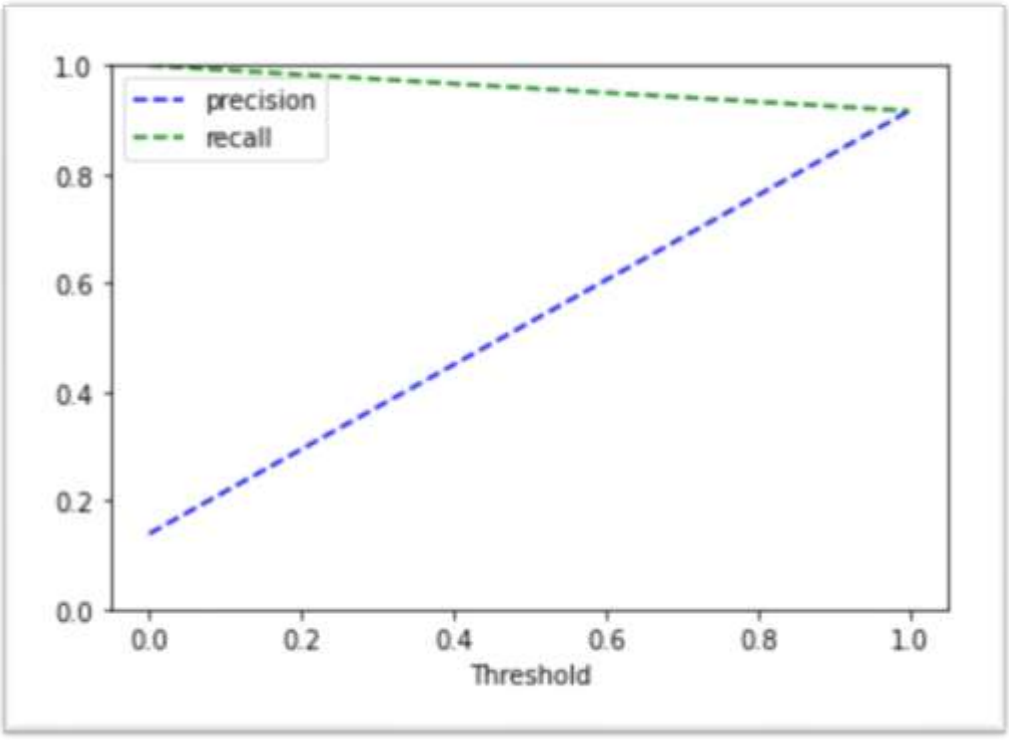
Logistic Regression

Logistic Regression is a classification model. It uses the sigmoid function to convert the input value between 0 and 1. The basic idea of logistic regression is to adapt linear regression so that it estimates the probability a new entry falls in a class. The linear decision boundary is simply a consequence of the structure of the regression function and the use of a threshold in the function to classify. Logistic Regression tries to maximize the conditional likelihood of the training data, it is highly prone to outliers.

After applying grid search to find the best estimator the result was: LogisticRegression(C=10, solver='newton-cg')

And these parameters give us these results with a good relation between recall and precession:

Model	Accuracy	Recall	Precision	F1
Logistic Regression	0.98	0.92	0.92	0.92



Support-Vector Machine

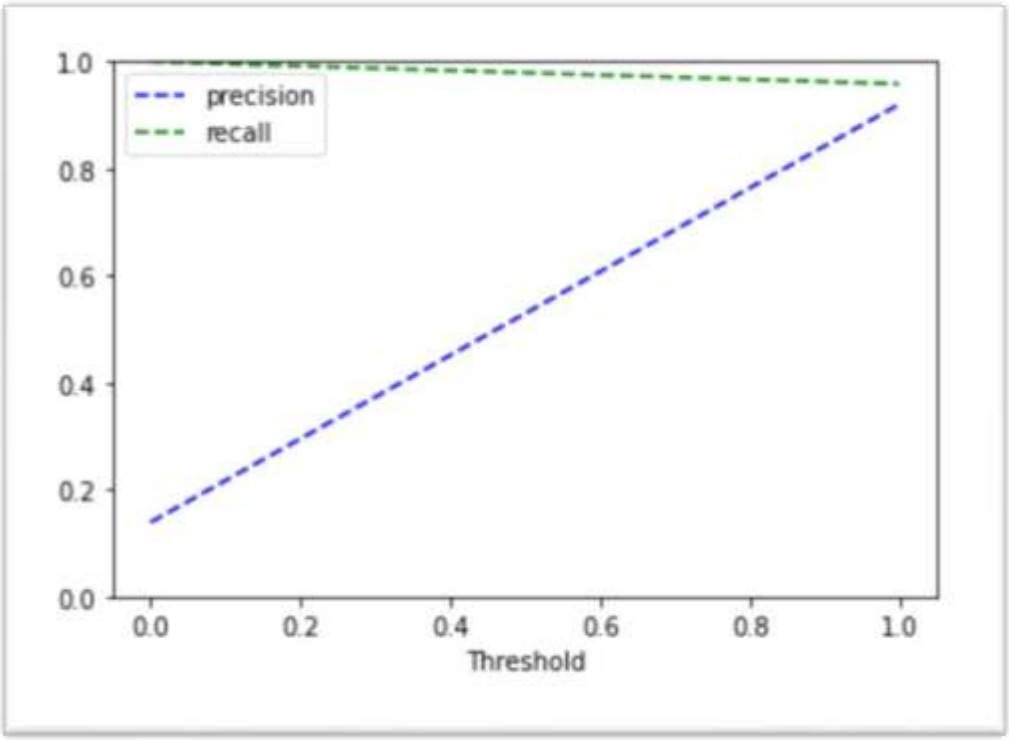
Support Vector Machine is a powerful classification algorithm to maximize the margin among class variables. This margin (support vector) represents the distance between the separating hyperplanes (decision boundary). The reason to have decision boundaries with large margin is to separate positive and negative hyperplanes with adjustable bias-variance proportion. The goal is to separate so that negative samples would fall under negative hyperplane and positive samples would fall under positive hyperplane. SVM is not as prone to outliers as it only cares about the points closest to the decision boundary. It changes its decision boundary depending on the placement of the new positive or negative events.

After applying grid search to find the best estimator the result was: SVC(C=100, degree=2, gamma=1, kernel='linear')

And these parameters give us these results with a good relation between recall and precession:

Model	Accuracy	Recall	Precision	F1

Logistic Regression	0.98	0.96	0.92	0.94



Decision tree:

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a treestructured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

In the implemented model, GridsearchCV was applied to find the best estimators for the decision tree classifier, after implementing it, it came up with the criterion being based on gini index, the maximum depth for our tree is 2 and with no attributes.

The classification metrics are:

```
Accuracy 0.953757225433526
F1 score: 0.8260869565217391
Recall: 0.7916666666666666
Precision: 0.8636363636363636
```

Naïve Bayes:

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

At first, Bernoulli Naïve Bayes was introduced to the data, the output accuracy was high, but the recall was very low, since we have medical data and the recall needs to be high, we tried Gaussian Naïve Bayes altering the hyperparameters using GridsearchCV and scaling the data, the accuracy was low comparing to Bernoulli but the recall was very high which is a good thing in medical datasets.

The metric results were:

```
Accuracy 0.8323699421965318
F1 score: 0.32558139534883723
Recall: 0.2916666666666667
Precision: 0.3684210526315789
```

```
Accuracy 0.5260115606936416
F1 score: 0.33870967741935487
Recall: 0.875
Precision: 0.21
```