Machine Learning 545

Group Final Programming Project

Group Members:

Kirtan Patel, Parth Parashar, Manish Manchegowda, Reeya Kambhojkar,

Tarun Kumar

Team Contribution:

We were trying to compare the Test accuracy from 4 different algorithms. So we divided all 4 algorithms among us. Neural Network algorithm code was done by Kirtan, SVM was done by Parth, Manish did the Logistic Regression algorithm and Reeya and Tarun did Naïve bayes. Along with the coding everyone did the write-ups for their part in the project documentation.

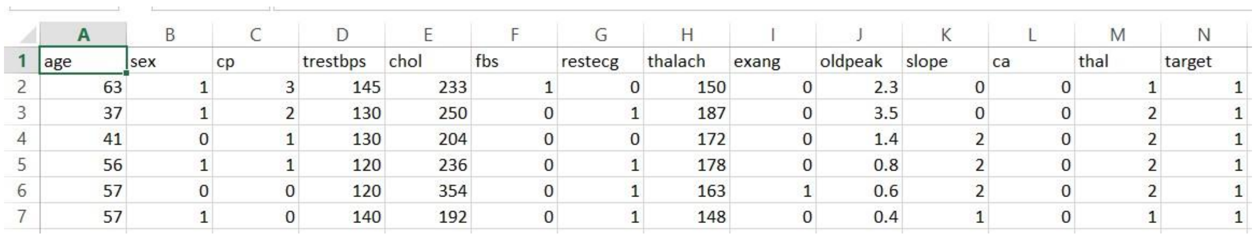
Abstract:

Using the dataset provided, we built simple Machine Learning models for predicting the accuracy and confusion matrices for the models which we have used.

Our objective for calculating these accuracies and confusion matrices is that we want to compare the accuracies and confusion matrices obtained using these models on the same “heart Disease” data set. This will allow us to find out which model best suits this dataset.

Choosing data:

One of the most important steps in machine learning and predictive modeling is gathering good data, performing the appropriate cleaning steps and normalize data. The heart disease data is used (source: kaggle). The snapshot of the data is listed below.

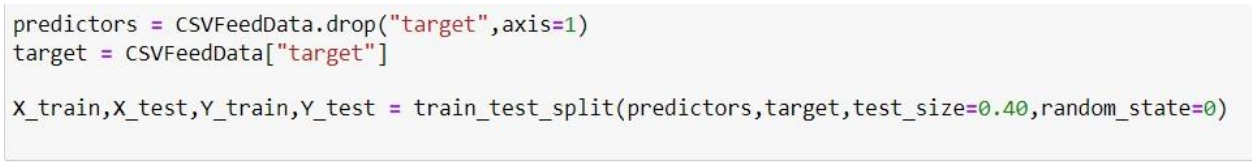


	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
2	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
3	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
4	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
5	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
6	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
7	57	1	0	140	192	0	1	148	0	0.4	1	0	1	1

Data Preprocessing:

The Dataset is divided into 2 parts i.e. Predictors and Targets. The predictors are all the features in the dataset which determine if the person is going to have disease or not and the Target is the predictions made according to the data if the person will have heart disease or not. The Target is informed of Binary numbers in which "0" means the person won't have heart disease and "1" indicating the person will have heart disease.

We will be splitting the dataset into 60% training set which we will be using to train the model and remaining 40% will be used for testing and calculating the accuracy.



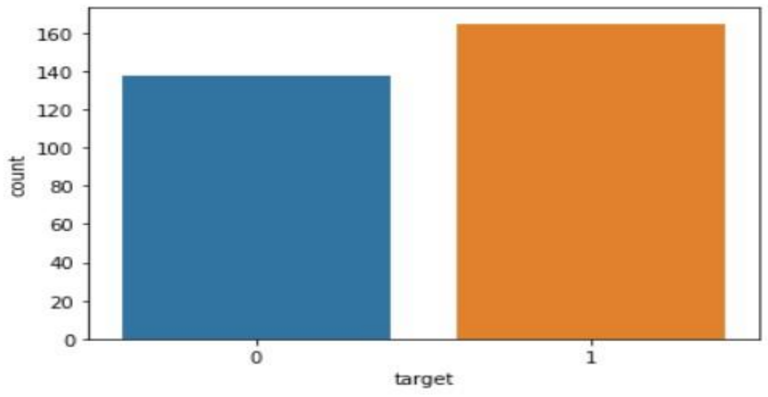
```
predictors = CSVFeedData.drop("target",axis=1)
target = CSVFeedData["target"]

X_train,X_test,Y_train,Y_test = train_test_split(predictors,target,test_size=0.40,random_state=0)
```

Calculating The targets

Here we are checking the number of “0” and “1” in the target and using the countplot to plot the graph for the same.

```
CSVFeedData = pd.read_csv("heart.csv")
type(CSVFeedData)
y = CSVFeedData["target"]


sns.countplot(y)

target_temp = CSVFeedData.target.value_counts()

print(target_temp)
```

```
1    165
0    138
Name: target, dtype: int64
```



Machine Learning Algorithms:

1) Logistic Regression

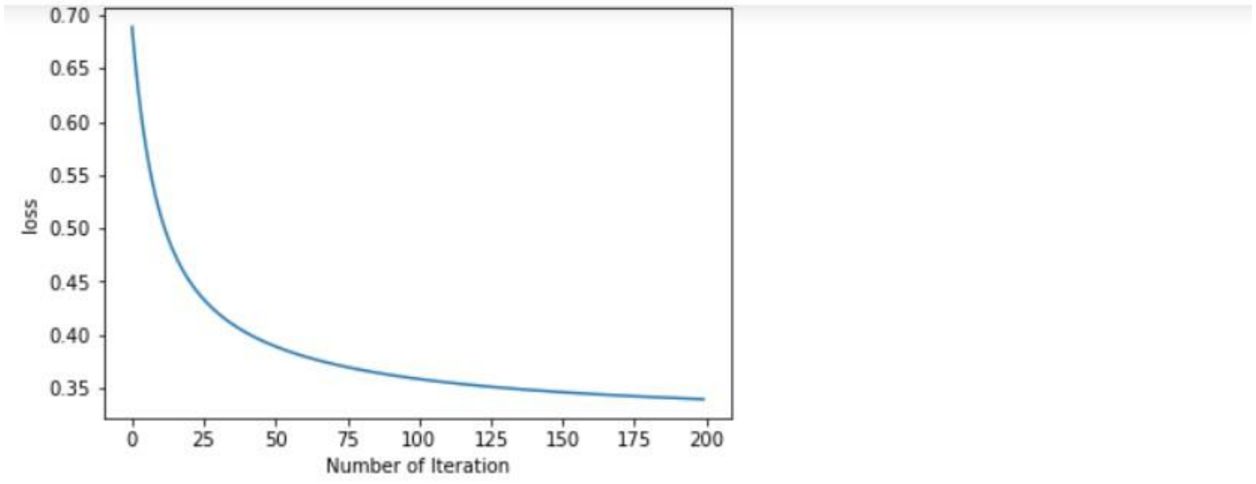
Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Some of the examples of classification problems are Email spam or not spam, Online transactions Fraud or not Fraud, Tumor Malignant or Benign. Logistic regression transforms its output using the logistic sigmoid function to return a probability value.

We can call a Logistic Regression a Linear Regression model but the Logistic Regression uses a more complex cost function, this cost function can be defined as the '**Sigmoid function**' or also known as the 'logistic function' instead of a linear function.

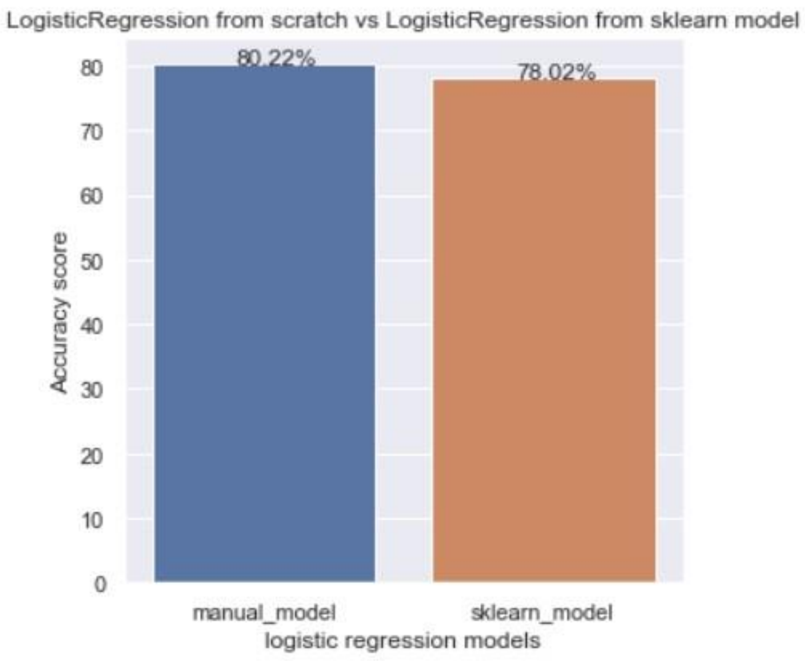
The hypothesis of logistic regression tends it to limit the cost function between 0 and 1. Therefore linear functions fail to represent it as it can have a value greater than 1 or less than 0 which is not possible as per the hypothesis of logistic regression.

In order to map predicted values to probabilities, we use the Sigmoid function. The function maps any real value into another value between 0 and 1. In machine learning, we use sigmoid to map predictions to probabilities.

For this algorithm we are getting the Test Accuracy of 80.22% in Logistic Regression from scratch. And we are getting Test accuracy of 78.02% by using Logistic Regression from Sklearn.



```
Test Accuracy of Logistic regression from scratch: 80.22%
The accuracy score achieved using Logistic Regression model from sklearn is: 78.02 %
```



2) Neural Network

Neural nets are a means of doing machine learning, in which a computer learns to perform some tasks by analysing training examples. Usually, the examples have been hand-labelled in advance. An object recognition system, for instance, might be fed thousands of labelled images of cars, houses, coffee cups, and so on, and it would find visual patterns in the images that consistently correlate with particular labels.

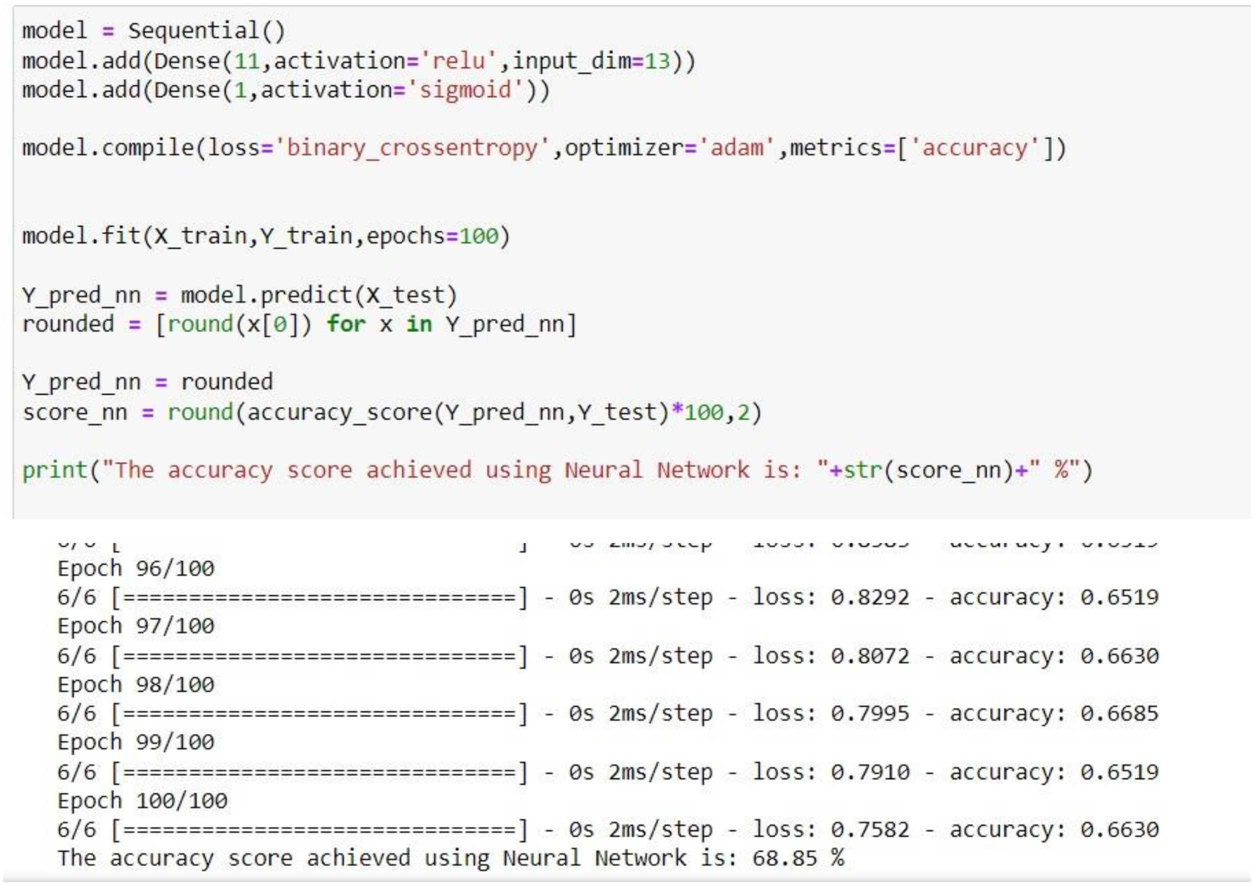
Modelled loosely on the human brain, a neural net consists of thousands or even millions of simple processing nodes that are densely interconnected. Most of today's neural nets are organized into layers of nodes, and they're "feed-forward," meaning that data moves through them in only one direction. An individual node might be connected to several nodes in the layer beneath it, from which it receives data, and several nodes in the layer above it, to which it sends data.

To each of its incoming connections, a node will assign a number known as a "weight." When the network is active, the node receives a different data item — a different number — over each of its connections and multiplies it by the associated weight. It then adds the resulting products together, yielding a single number. If that number is below a threshold value, the node passes no data to the next layer. If the number exceeds the threshold value, the node "fires," which in today's neural nets generally means sending the number — the sum of the weighted inputs — along all its outgoing connections.

When a neural net is being trained, all of its weights and thresholds are initially set to random values. Training data is fed to the bottom layer — the input layer — and it passes through the succeeding layers, getting multiplied and added together in complex ways, until it finally arrives, radically transformed, at the output layer.

During training, the weights and thresholds are continually adjusted until training data with the same labels consistently yield similar outputs.

In this algorithm we are getting only 68.85% accuracy after running 100 epochs.



```
model = Sequential()
model.add(Dense(11,activation='relu',input_dim=13))
model.add(Dense(1,activation='sigmoid'))

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])


model.fit(X_train,Y_train,epochs=100)

Y_pred_nn = model.predict(X_test)
rounded = [round(x[0]) for x in Y_pred_nn]

Y_pred_nn = rounded
score_nn = round(accuracy_score(Y_pred_nn,Y_test)*100,2)

print("The accuracy score achieved using Neural Network is: "+str(score_nn)+" %")
```

```
Epoch 96/100
6/6 [==============================] - 0s 2ms/step - loss: 0.8292 - accuracy: 0.6519
Epoch 97/100
6/6 [==============================] - 0s 2ms/step - loss: 0.8072 - accuracy: 0.6630
Epoch 98/100
6/6 [==============================] - 0s 2ms/step - loss: 0.7995 - accuracy: 0.6685
Epoch 99/100
6/6 [==============================] - 0s 2ms/step - loss: 0.7910 - accuracy: 0.6519
Epoch 100/100
6/6 [==============================] - 0s 2ms/step - loss: 0.7582 - accuracy: 0.6630
The accuracy score achieved using Neural Network is: 68.85 %
```

3) Support Vector Machine (SVM)

Support vector machine is another simple algorithm that every machine learning expert should have in his/her arsenal. Support vector machine is highly preferred by many as it produces significant accuracy with less computation power. Support Vector Machine, abbreviated as SVM can be used for both regression and classification tasks. But it is widely used in classification objectives.

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points.

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e., the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3.

Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.

In the SVM algorithm, we are looking to maximize the margin between the data points and the hyperplane. The loss function that helps maximize the margin is hinge loss.

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$$

The cost is 0 if the predicted value and the actual value are of the same sign. If they are not, we then calculate the loss value. We also add a regularization parameter the cost function. The objective of the regularization parameter is to balance the margin maximization and loss. After adding the regularization parameter, the cost functions look as below.

Loss function for SVM -

$$min_w \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} (1 - y_i \langle x_i, w \rangle)_+$$

Now that we have the loss function, we take partial derivatives with respect to the weights to find the gradients. Using the gradients, we can update our weights.

Gradients -

$$\frac{\delta}{\delta w_k} \lambda \parallel w \parallel^2 \quad = 2\lambda w_k$$

$$\frac{\delta}{\delta w_k} \big(1 - y_i \langle x_i, w \rangle\big)_+ \quad = \begin{cases} 0, & \text{if } y_i \langle x_i, w \rangle \geq 1 \\ -y_i x_{ik}, & \text{else} \end{cases}$$

When there is no misclassification, i.e., our model correctly predicts the class of our data point, we only have to update the gradient from the regularization parameter.

Gradient Update — No misclassification →

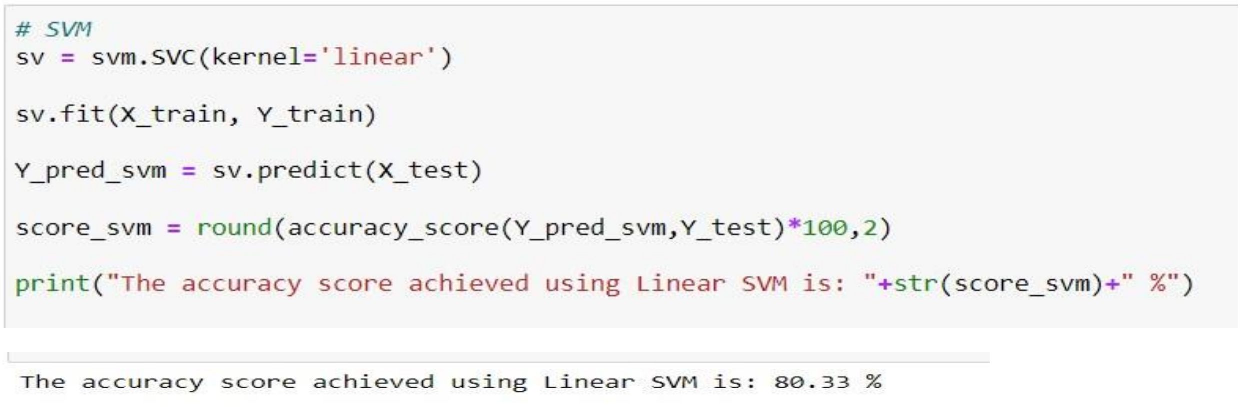
$$w = w - \alpha \cdot (2\lambda w)$$

When there is a misclassification, i.e., our model makes a mistake on the prediction of the class of our data point, we include the loss along with the regularization parameter to perform gradient update.

Gradient Update — Misclassification →

$$w = w + \alpha \cdot (y_i \cdot x_i - 2\lambda w)$$

For this algorithm we are getting accuracy of 80.33%



```
# SVM
sv = svm.SVC(kernel='linear')

sv.fit(X_train, Y_train)

Y_pred_svm = sv.predict(X_test)

score_svm = round(accuracy_score(Y_pred_svm,Y_test)*100,2)

print("The accuracy score achieved using Linear SVM is: "+str(score_svm)+" %")
```

```
The accuracy score achieved using Linear SVM is: 80.33 %
```

4) Naïve Bayes

Naïve Bayes theorem assumes every feature to be independent without any correlation with other attributes, but this assumption has limitations, as in real life it’s not possible to get complete independence.

It is a classification technique based on Bayes’ Theorem with 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.

For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as ‘Naive’.

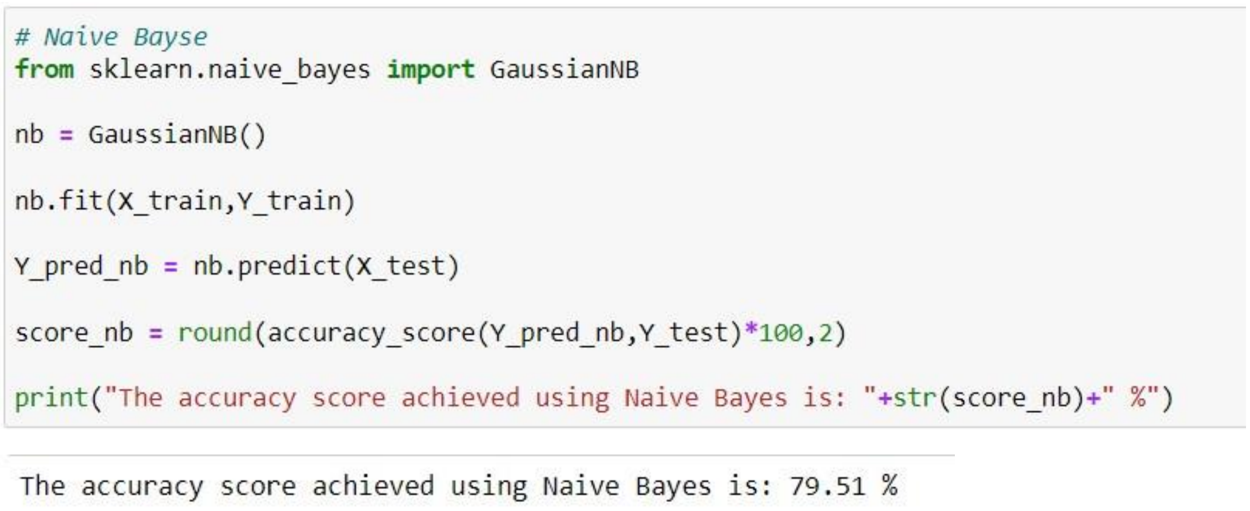
Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Step 1: Convert the data set into a frequency table

Step 2: Create Likelihood table by finding the probabilities

Step 3: Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

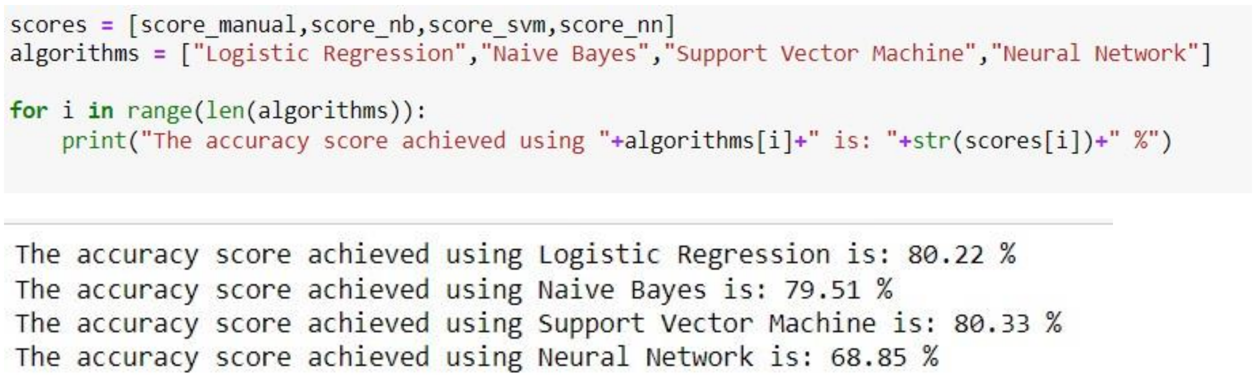
The accuracy we are getting from this is around 79.51%.



```python
# Naive Bayse
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()

nb.fit(X_train,Y_train)

Y_pred_nb = nb.predict(X_test)

score_nb = round(accuracy_score(Y_pred_nb,Y_test)*100,2)

print("The accuracy score achieved using Naive Bayes is: "+str(score_nb)+" %")
```

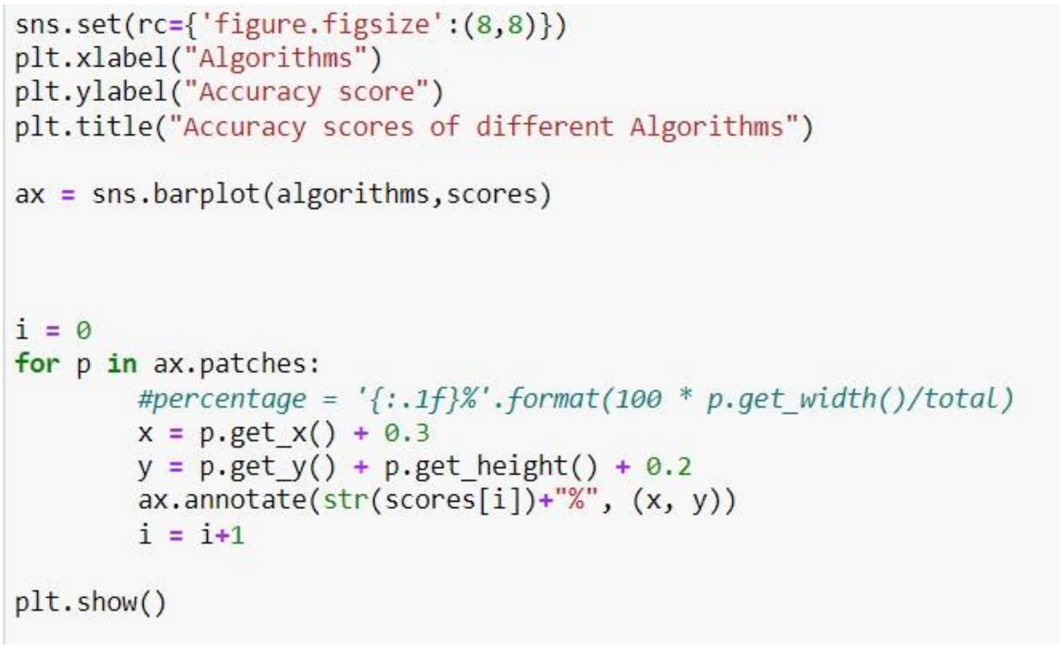
```
The accuracy score achieved using Naive Bayes is: 79.51 %
```

Comparing the accuracies of All 4 algorithms:

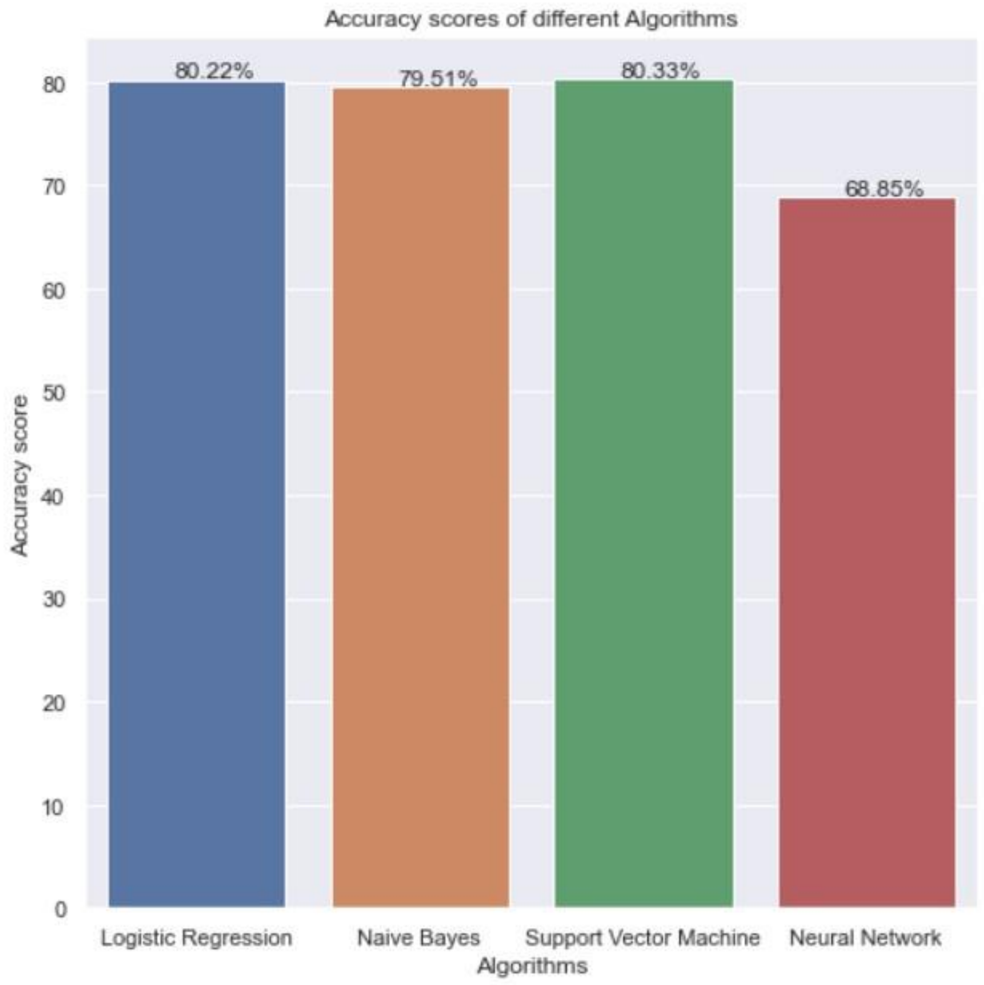


```python
scores = [score_manual,score_nb,score_svm,score_nn]
algorithms = ["Logistic Regression","Naive Bayes","Support Vector Machine","Neural Network"]

for i in range(len(algorithms)):
    print("The accuracy score achieved using "+algorithms[i]+" is: "+str(scores[i])+" %")
```

```
The accuracy score achieved using Logistic Regression is: 80.22 %
The accuracy score achieved using Naive Bayes is: 79.51 %
The accuracy score achieved using Support Vector Machine is: 80.33 %
The accuracy score achieved using Neural Network is: 68.85 %
```



```
sns.set(rc={'figure.figsize':(8,8)})
plt.xlabel("Algorithms")
plt.ylabel("Accuracy score")
plt.title("Accuracy scores of different Algorithms")

ax = sns.barplot(algorithms,scores)



i = 0
for p in ax.patches:
        #percentage = '{:.1f}%'.format(100 * p.get_width()/total)
        x = p.get_x() + 0.3
        y = p.get_y() + p.get_height() + 0.2
        ax.annotate(str(scores[i])+"%", (x, y))
        i = i+1

plt.show()
```



Conclusion:

After running the dataset with all different algorithms we can see that SVM algorithms performs slightly better than Logistic Regression model by getting Test accuracy of 80.33%. In this dataset the accuracy of Neural Network is the lowest which is possible to change by increasing the hidden layers and running more epochs. So for this dataset it is preferred to use SVM to find out accuracies but it doesn’t matter that every time SVM will outperform other algorithms. It completely depends on the type of dataset.