# CSE 4020 - MACHINE LEARNING

# Lab 29+30

# Lab Assignment-3

# Submitted by: Alokam Nikhitha(19BCE2555)

# Support Vector Machine(SVM)

Ques: Train SVM classifier using sklearn digits dataset (i.e. from sklearn.datasets import load\_digits) and then

1. Measure accuracy of your model using different kernels such as rbf, poly and linear.

2. Tune your model further using regularization and gamma parameters and try to come up with highest accuracy score.

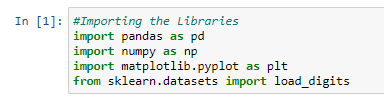
3. Use 80% of samples as training data size.

Dataset Used: load\_digits dataset from sklearn

**Procedure:**

* Using pandas, we first import the dataset into our workspace.
* The next step is to choose the independent and dependent variables that will be used in our regression model.
* After that, we divided our data into two sets: training and test.
* Then, using the 'rbf' kernel, we must initialise our Support Vector Machine classifier and fit it to the X\_train and y\_train attributes.
* Use the ‘linear' and ‘polynomial' kernels to repeat the previous process.
* Then, using the results predicted by X\_test on the 'rbf', 'linear', and 'polynomial' kernels, we establish three variables to store the X\_test result
* Finally, we compute evaluation metrics for each of the three kernels.

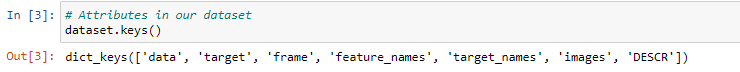
**CODE SNIPPETS AND EXPLAINATION**



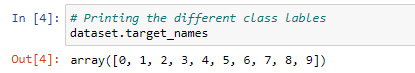
Importing the required Libraries



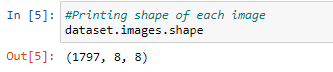
Importing the digits Dataset



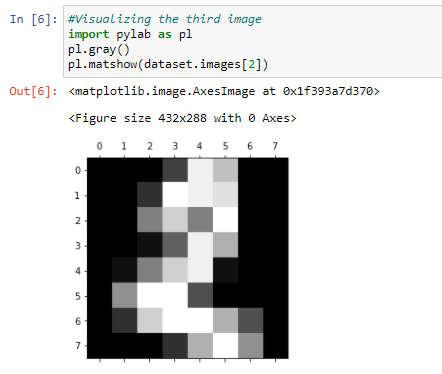
Listing the Attributes in our Dataset



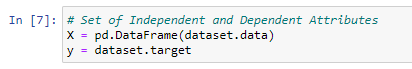
Printing the Different Class Labels



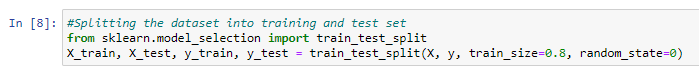
Print the image shape



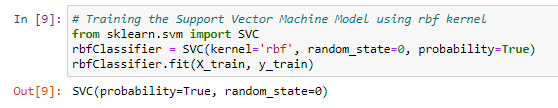
Visualizing the Third Image in the Dataset



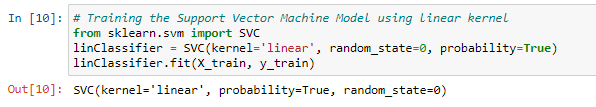
Taking the Independent and Depending Attributes



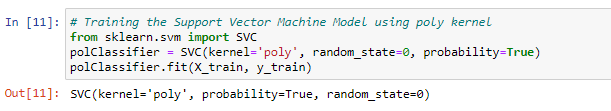
Splitting the dataset into Training set and Test set



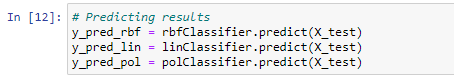
Training theSVM Model using rbf kernel



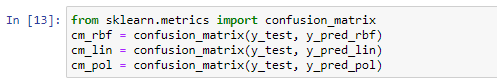
Training SVM model using Linear kernel



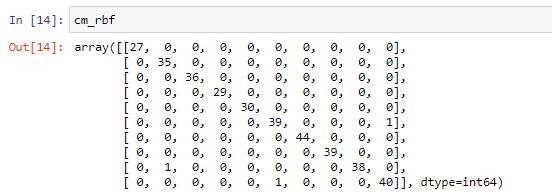
Training SVM model using poly kernel Model



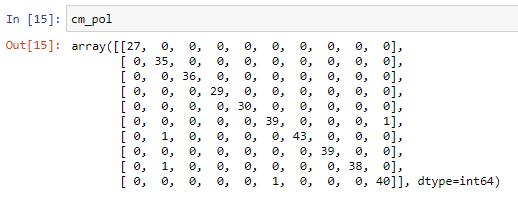
Predicting Results for various Kernels



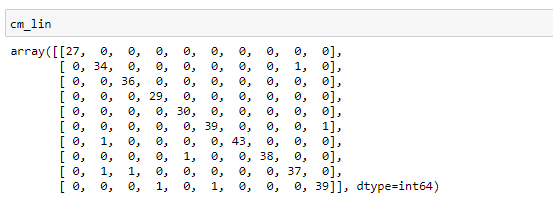
Confusion Matrix for various Kernels



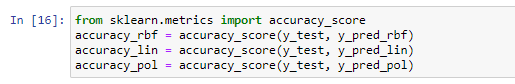
Confusion Matrix for rbf Kernel



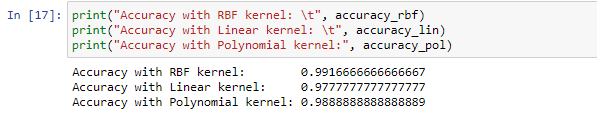
Confusion Matrix for poly Kernel



Confusion Matrix for linear Kernel



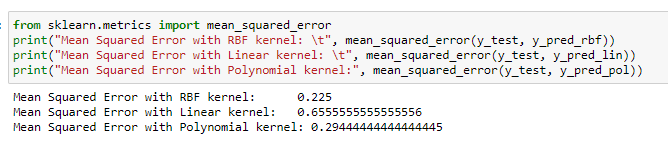
Calculating Accuracy for various Kernels



Printing Accuracy for Different Kernels

We can see here that the accuracy with rbf kernel is max and thus it is most suitable.

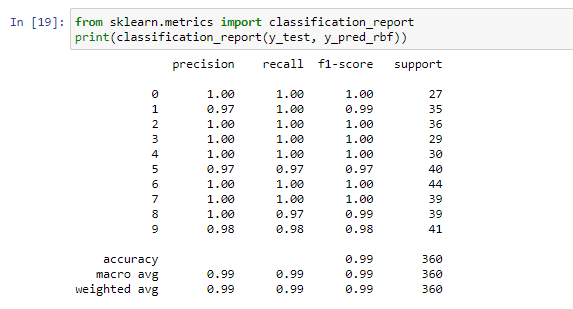
Accuracy(Linear) < Accuracy(Poly) < Accuracy(rbf)

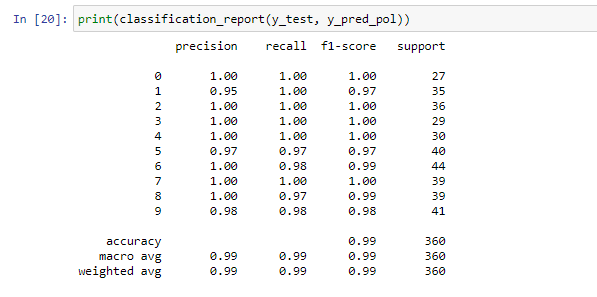


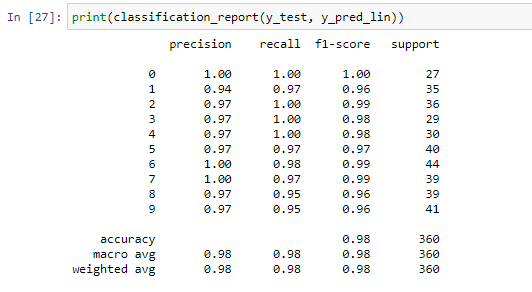
Mean Squared Errors with various Kernels

From here we can again infer that MSE in least for rbf kernel and hence it is the most suitable kernel for our dataset in Support Vector Classifier.

MSE(rbf) < MSE(Poly) < MSE(Linear)

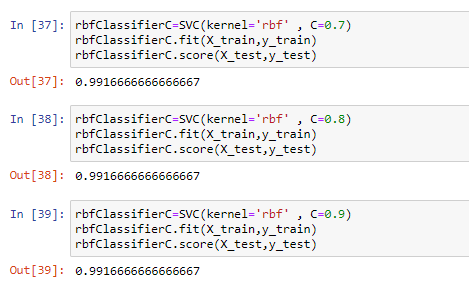






Here we have printed the classification report of Support Vector Classifier with all three kernels.

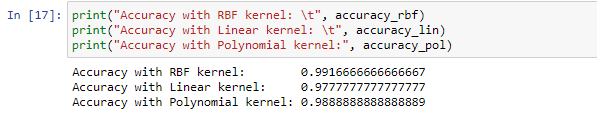




Here we have tried to tune in the C value for rbf kernel. Initially we have used 0.3 as we increase the C value and we can see that the accuracy increases till C=0.7, after that C remains constant and there is no significant increase in models accuracy and hence the C value can be taken as C=0.7.

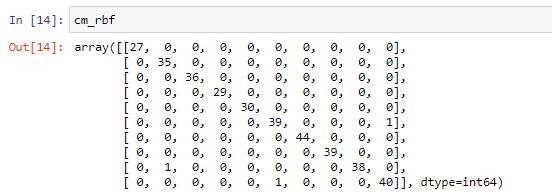
**Result and Conclusion:**

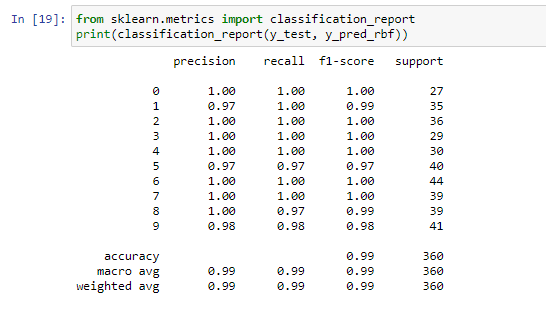
**Accuracy**

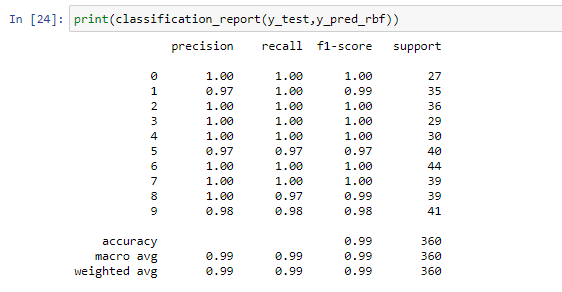


RBF Kernel-

• Model Accuracy = 99.1667%



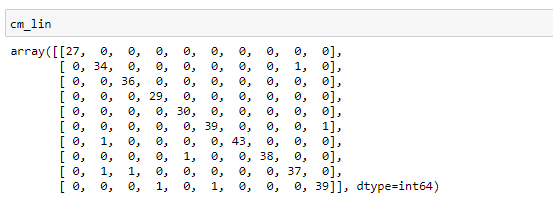


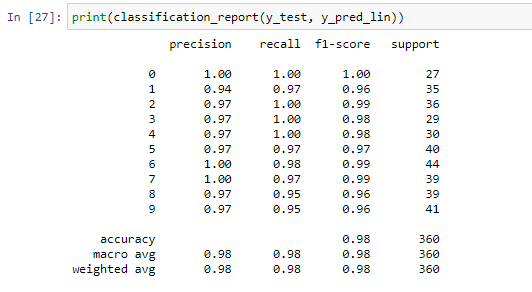


* Identified 0 = 27
* True 0 = 27
* Identified 1 = 34
* True 1 = 35
* Identified 2 = 36
* True 2 = 36
* Identified 3 = 29
* True 3 = 29
* Identified 4 = 30
* True 4 = 30
* Identified 5 = 39
* True 5 = 40
* Identified 6 = 44
* True 6 = 44
* Identified 7 = 39
* True 7 = 39
* Identified 8 = 39
* True 8 = 39
* Identified 9 = 40
* True 9 = 41
* Precision of 0 = 1.00
* Precision of 1 = 0.97
* Precision of 2 = 1.00
* Precision of 3 = 1.00
* Precision of 4 = 1.00
* Precision of 5 = 0.97
* Precision of 6 = 1.00
* Precision of 7 = 1.00
* Precision of 8 = 1.00
* Precision of 9 = 0.98
* Recall of 0 = 1.00
* Recall of 1 = 1.00
* Recall of 2 = 1.00
* Recall of 3 = 1.00
* Recall of 4 = 1.00
* Recall of 5 = 0.97
* Recall of 6 = 1.00
* Recall of 7 = 1.00
* Recall of 8 = 0.97
* Recall of 9 = 0.98

Linear Kernel:

• Model Accuracy = 97.77%

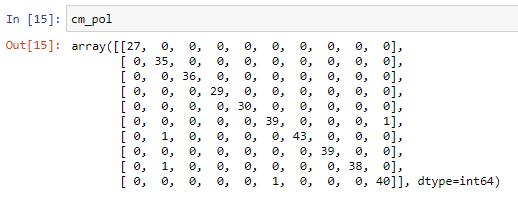


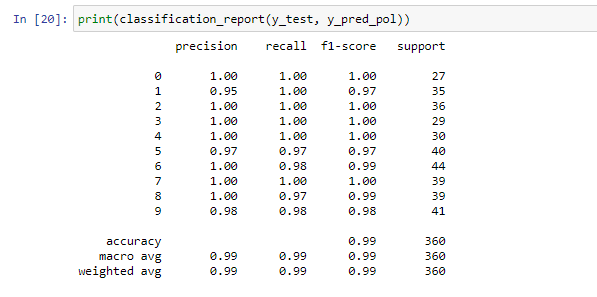


* Identified 0 = 27
* True 0 = 27
* Identified 1 = 33
* True 1 = 35
* Identified 2 = 35
* True 2 = 36
* Identified 3 = 28
* True 3 = 29
* Identified 4 = 29
* True 4 = 30
* Identified 5 = 39
* True 5 = 40
* Identified 6 = 44
* True 6 = 44
* Identified 7 = 39
* True 7 = 39
* Identified 8 = 38
* True 8 = 39
* Identified 9 = 40
* True 9 = 41
* Precision of 0 = 1.00
* Precision of 1 = 0.97
* Precision of 2 = 1.00
* Precision of 3 = 1.00
* Precision of 4 = 1.00
* Precision of 5 = 0.97
* Precision of 6 = 0.98
* Precision of 7 = 0.97
* Precision of 8 = 0.95
* Precision of 9 = 0.95
* Recall of 0 = 1.00
* Recall of 1 = 0.96
* Recall of 2 = 0.99
* Recall of 3 = 0.98
* Recall of 4 = 0.98
* Recall of 5 = 0.97
* Recall of 6 = 0.99
* Recall of 7 = 0.99
* Recall of 8 = 0.96
* Recall of 9 = 0.96

Poly Kernel:

• Model Accuracy = 98.88%





* Identified 0 = 27
* True 0 = 27
* Identified 1 = 35
* True 1 = 35
* Identified 2 = 36
* True 2 = 36
* Identified 3 = 29
* True 3 = 29
* Identified 4 = 30
* True 4 = 30
* Identified 5 = 39
* True 5 = 40
* Identified 6 = 44
* True 6 = 44
* Identified 7 = 39
* True 7 = 39
* Identified 8 = 39
* True 8 = 39
* Identified 9 = 40
* True 9 = 41
* Precision of 0 = 1.00
* Precision of 1 = 0.95
* Precision of 2 = 1.00
* Precision of 3 = 1.00
* Precision of 4 = 1.00
* Precision of 5 = 0.97
* Precision of 6 = 1.00
* Precision of 7 = 1.00
* Precision of 8 = 1.00
* Precision of 9 = 0.98
* Recall of 0 = 1.00
* Recall of 1 = 1.00
* Recall of 2 = 1.00
* Recall of 3 = 1.00
* Recall of 4 = 1.00
* Recall of 5 = 0.97
* Recall of 6 = 0.98
* Recall of 7 = 1.00
* Recall of 8 = 0.97
* Recall of 9 = 0.98

**KNN**

**Question:**

1. Load the data

2. Initialize K to your chosen number of neighbors

3. For each example in the data

3.1 Calculate the distance between the query example and the current example from the data.

3.2 Add the distance and the index of the example to an ordered collection

4. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances

5. Pick the first K entries from the sorted collection

6. Get the labels of the selected K entries

7. If regression, return the mean of the K labels

8. If classification, return the mode of the K labels

**Dataset Used:**

diabetes dataset from https://www.kaggle.com/uciml/pima-indians-diabetesdatabase/version/1

**Procedure:**

-Using pandas, we first import the dataset into our workspace.

-The independent and dependent attributes to be employed in our classification model must then be decided.

- After that, we divided our data into two sets: training and test.

- After that, we must Feature Scale our dataset.

-Scaling concerns should be accounted for in many attributes.

- Next, we determine the k value for which the classifier has the lowest error.

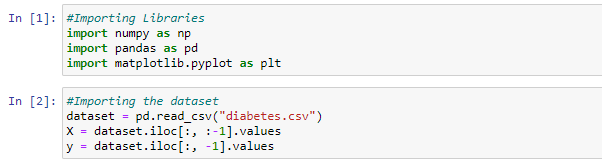
- After that, we use the best k value to fit our classifier model.

- After that, we construct a variable to record our expected result.

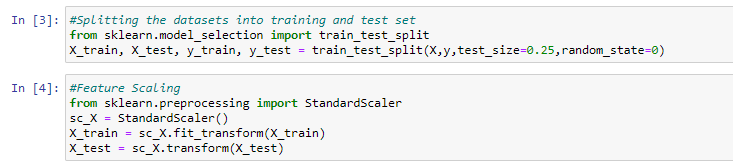
-the X test set's classifier

- Last, we compute our assessment metrics.

**Code Snippets and Explanation:**

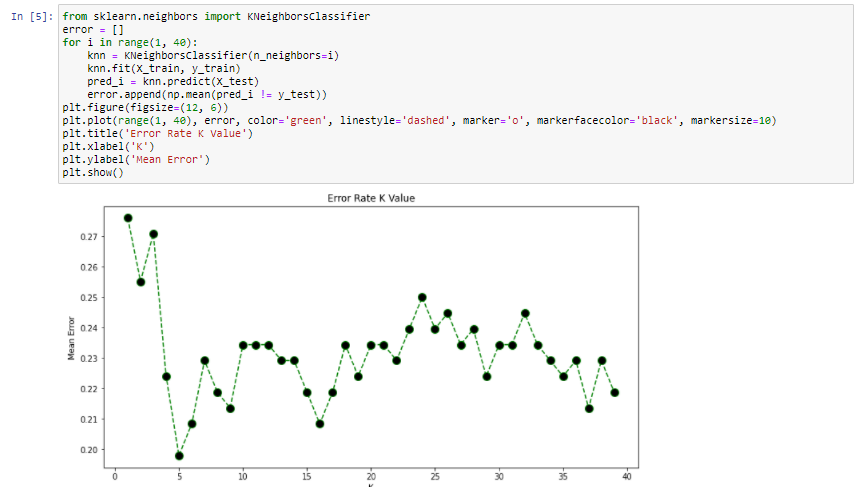


Importing the Libraries and Dataset

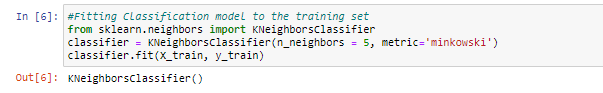


Splitting the dataset into Training and testing sets and Feature scaling

Here we are splitting our dataset into training set and test set with 25% of our dataset values in test set and remaining 75% in training set.



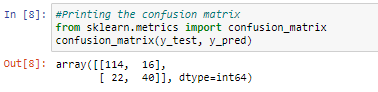
We're trying to figure out what the best value for K is. When we choose K as 5, we can observe that the error value is the lowest.

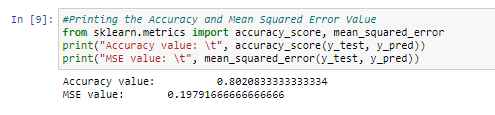


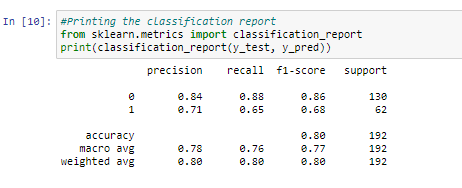
We're using training sets to fit our KNN classifier. Because of the previous outcome, we chose K value of 5.



On the test set, we're creating a list of predictions based on the classifier's predictions. Our Confusion Matrix has also been created.







We've created our numerous evaluation matrixes here. Precision, recall, and f1 score are all included, as well as accuracy and mean squared error value. Our model has an accuracy of 80 percent and an MSE score of 0.1979.

def knn(data, query, k, distance\_fn, choice\_fn):

neighbor\_distances\_and\_indices = []

    # 3. For each example in the data

    for index, example in enumerate(data):

        # 3.1 Calculate the distance between the query example and the current

        # example from the data.

        distance = distance\_fn(example[:-1], query)

        # 3.2 Add the distance and the index of the example to an ordered collection

        neighbor\_distances\_and\_indices.append((distance, index))

    # 4. Sort the ordered collection of distances and indices from

    # smallest to largest (in ascending order) by the distances

    sorted\_neighbor\_distances\_and\_indices = sorted(neighbor\_distances\_and\_indices)

    # 5. Pick the first K entries from the sorted collection

    k\_nearest\_distances\_and\_indices = sorted\_neighbor\_distances\_and\_indices[:k]

    # 6. Get the labels of the selected K entries

    k\_nearest\_labels = [data[i][1] for distance, i in k\_nearest\_distances\_and\_indices]

    # 7. If regression (choice\_fn = mean), return the average of the K labels

    # 8. If classification (choice\_fn = mode), return the mode of the K labels

    return k\_nearest\_distances\_and\_indices , choice\_fn(k\_nearest\_labels)

#function to calculate the mean used in case of regression

def mean(labels):

    return sum(labels) / len(labels)

#function to calculate the mode used in case of classification

def mode(labels):

    return Counter(labels).most\_common(1)[0][0]

#function to calculate the distance between two data points

def euclidean\_distance(point1, point2):

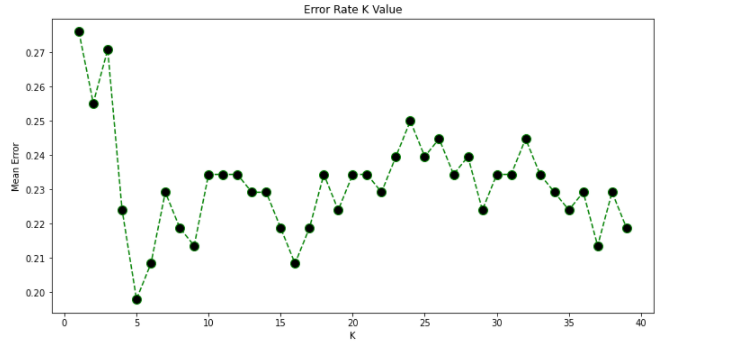
    sum\_squared\_distance = 0

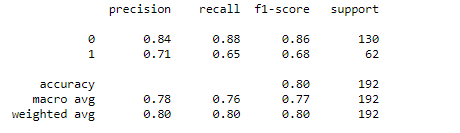
    for i in range(len(point1)):

        sum\_squared\_distance += math.pow(point1[i] - point2[i], 2)

    return math.sqrt(sum\_squared\_distance)

**Result and Conclusion:**

****

****

**Modal Accuracy:80%**

**MLP**

**Question:**

To train a Multi-Layer Perceptron (MLP) model to classify the network traffic record whether it is a normal or attack…

1. Read and parse the dataset.

2. Create Multi-Layer Perceptron Model (MLP)

3. Train and evaluate a Multi-Layer Perceptron (MLP) model

**Dataset Used:**

NSL KDD – Intrusion Detection Dataset <https://www.unb.ca/cic/datasets/nsl.html>

**Procedure:**

-Using pandas, we first import the dataset into our workspace.

-Assign the column names to our dataset as it doesn’t have one.

- Pick out and encode our specific variable.

- After encoding the specific variable, we want to dummy encode them on the way to keep away from ordinality among nominal information.

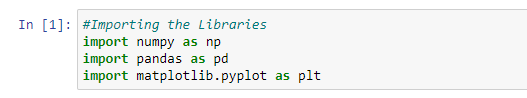
- We then want to re-assign our label information. All labels different than ordinary are assigned as attacks.

- We then want to divide the schooling set and check set information into set of structured attributes and impartial attributes.

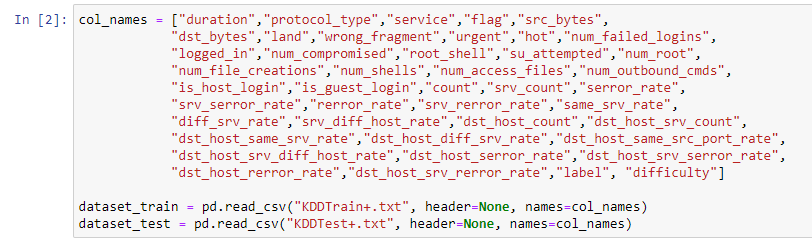
- Next, we lay down the Multi-Layer Perceptron and byskip our enter records into enter layer of our neural network.

- Finally, we generate our check set consequences and evaluation metrices.

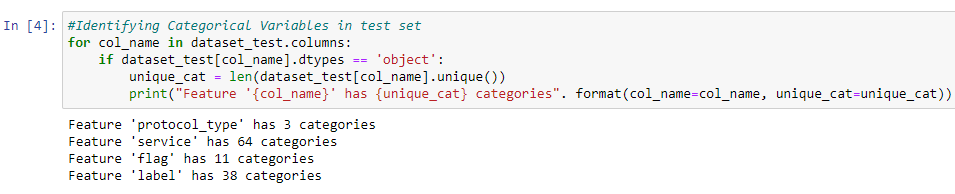
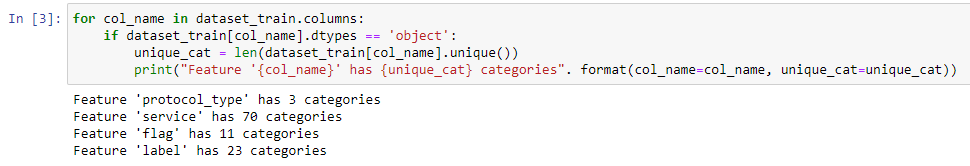
**Code Snippets and Explanation:**

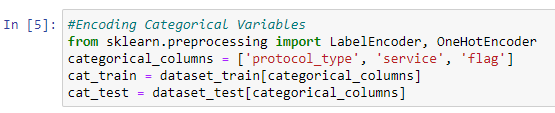


Here we are importing the necessary libraries in our workspace.

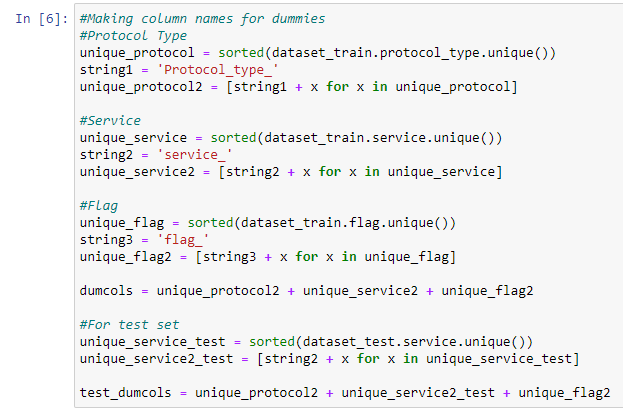


Here we're uploading the dataset into our workspace and are assigning them with the column names because it isn't always pre-blanketed in the given dataset.

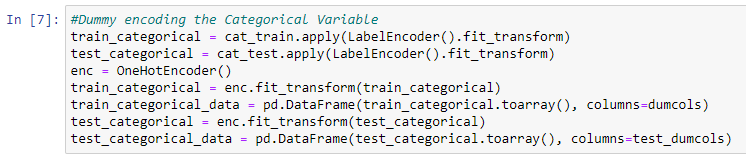


Here we've identified all of the express attributes in our training set and take a look at set. We have additionally identified the range of classes inculcating inside every attribute

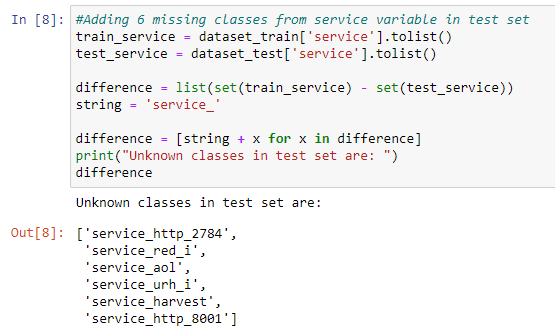
Here we have created 2 dummy data frames to include the categorical attributes in them



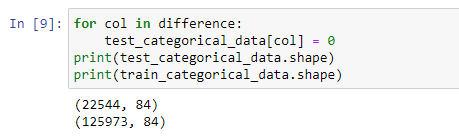
Here we've created the dummy attributes to keep away from the ordinal introduction among those nominal specific attributes.



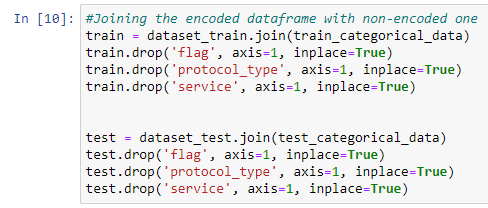
Here we've used the label encoder to fill withinside the dummy attributes in every of the specific attributes.



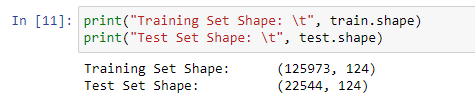
While checking the specific variables we noticed that service characteristic in check set has 70 training whilst schooling set has sixty four training. Hence, we want to encompass the ones 6 dummy attributes with zero fee in every our schooling set. This is what we've got diagnosed and achieved here.



Here we have finalised our data frames with the dummy values of categorical attributes.

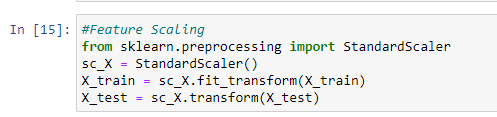
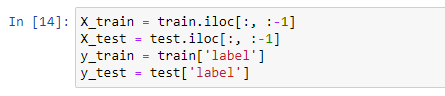
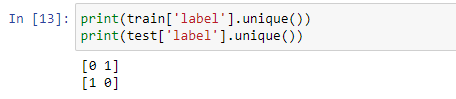


Next, we've got combined our original dataset with dummy attributes that we acquired in our specific assignment. Also right here we've got dropped the original specific attributes for you to inculcate only the non-specific attributes.

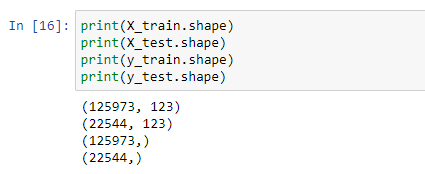


Here we have checked the number of attributes in both the training set and test set to see if they are equal… we can see that they have 124 attributes each and hence are compatible.

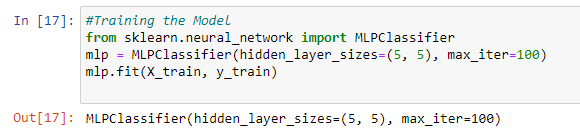
Here we have categorised each of the label attribute either as “normal” or an “attack”. All the labels which are normal are given a label of 0 and all those that indicate an attack are labelled as 1.



Since all the attributes in our dataset don’t follow a common scale, we need to feature scale the dataset in order to avoid any preassumed weight amongst them. We have used standard scalar to do this and it scales down each attribute to a range in -1 to 1.

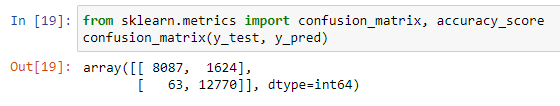


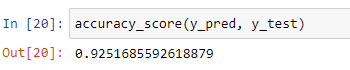
Here we have assigned the set of dependent and independent attributes. Also, we have printed the shape of each category that we have in order to check if they are compatible with each other.



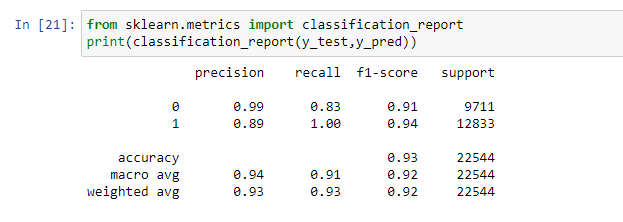
Here we have laid our neural network and then passed our input and output set to it in-order for it to adjust the weight biases.





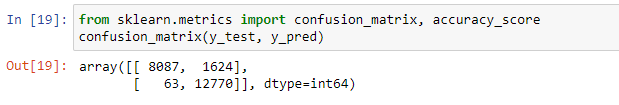
Here we have generated a vector y\_pred that stores the result as predicted by our mlp classifier on test set. We have also generated the confusion matrix to check the performance of our classifier.

we have printed the accuracy of our model and printed the classification reported to finally check the performance of our model. We can see that the accuracy of the model is 92.51%.

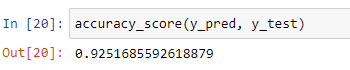


**Result and Conclusion:**

Confusion Matrix:



Accuracy:



Accuracy:92.51%.

* Identified Normal = 9711
* Actual Normal = 9362
* Identified Attack = 12833
* Actual Attack = 13182
* True Normal = 8087
* True Attack = 12770
* False Normal = 1624
* False Attack = 63
* Precision Normal = 0.99
* Precision Attack = 0.89
* Recall Normal = 0.83
* Recall Attack = 1.00