ML report

IRIS dataset :

The Iris Dataset contains four features (length and width of sepals and petals) of 50 samples of three species of Iris (Iris setosa, Iris virginica, and Iris versicolor). These measures created a linear discriminant model to classify the species.

Algorithms we are using :

- KNN- K-nearest neighbors
- SVM- support vector machine
- Logistic regression

Splitting the data into train and test data:

Code:-

Here we are dividing the data into training and testing data where X contains the iris data except for the columns “target and species“ and Y contains the target column. Here the test size and the train size are divided into 50% each.

1)**KNN-K-nearest neighbors:**

The k-nearest neighbor's algorithm, sometimes referred to as KNN or k-NN, is a supervised learning classifier that employs proximity to producing classifications or predictions about the grouping of a single data point. Although it can be applied to classification or regression issues, it is commonly employed as a classification algorithm because it relies on the idea that comparable points can be discovered close to one another.

Code: - KNN

Training the model by giving the training data and also changing the n_neighbors values from 3 to 9 for each iteration the n_neighbors value increments every time training the model with a new n_neighbors value. We use metrics. classification _report and confusion_matrix for getting the classification report and the confusion matrix for y_test and knn_predictions. y_test is the values that are to be tested on and knn_predictions are the predicted values by using the train data. the n_neighbors are the data points how many are we taking and also the data points which are near to the points which we are finding

```
Precision, Recall, Confusion matrix, intraining
, 3
              precision    recall  f1-score   support

         0.0      1.000     1.000     1.000        29
         1.0      0.920     1.000     0.958        23
         2.0      1.000     0.913     0.955        23

    accuracy                          0.973        75
   macro avg      0.973     0.971     0.971        75
weighted avg      0.975     0.973     0.973        75

[[29  0  0]
 [ 0 23  0]
 [ 0  2 21]]
```

This is output for when n_neighbors =3 and confusion matrix . accuracy is 97.3%

```
Precision, Recall, Confusion matrix, intraining
, 5
              precision    recall  f1-score   support

         0.0      1.000     1.000     1.000        29
         1.0      0.852     1.000     0.920        23
         2.0      1.000     0.826     0.905        23

    accuracy                          0.947        75
   macro avg      0.951     0.942     0.942        75
weighted avg      0.955     0.947     0.946        75

[[29  0  0]
 [ 0 23  0]
 [ 0  4 19]]
```

This is output when n_neighbors =5 and confusion matrix accuracy is 94.7%

```
Precision, Recall, Confusion matrix, intraining
, 7
              precision    recall  f1-score   support

         0.0      1.000     1.000     1.000        29
         1.0      0.852     1.000     0.920        23
         2.0      1.000     0.826     0.905        23

    accuracy                          0.947        75
   macro avg      0.951     0.942     0.942        75
weighted avg      0.955     0.947     0.946        75

[[29  0  0]
 [ 0 23  0]
 [ 0  4 19]]
```

This is output when n_neighbors =7 and confusion matrix accuracy is 94.7%

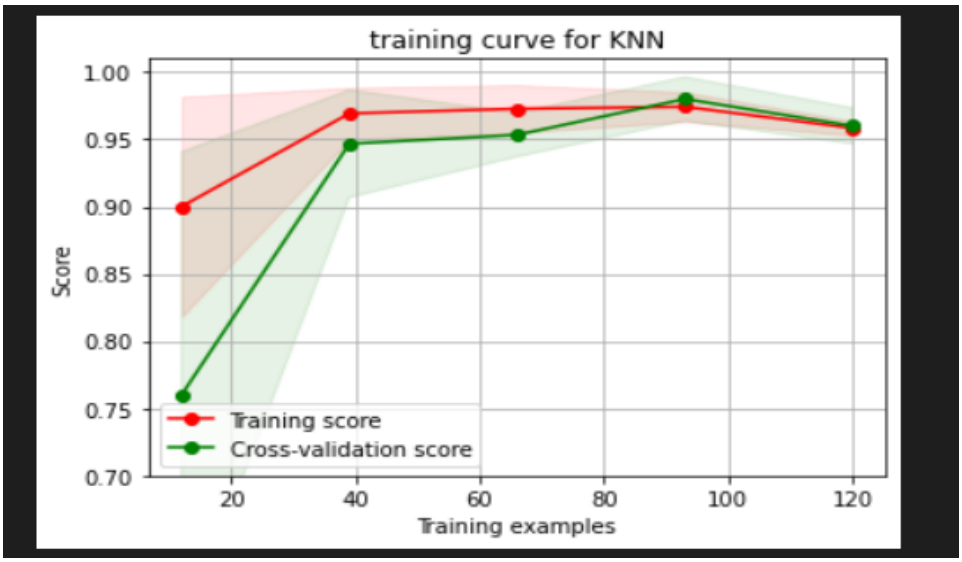
```
Precision, Recall, Confusion matrix, intraining
, 9
              precision    recall  f1-score   support

         0.0      1.000     1.000     1.000        29
         1.0      0.885     1.000     0.939        23
         2.0      1.000     0.870     0.930        23

    accuracy                          0.960        75
   macro avg      0.962     0.957     0.956        75
weighted avg      0.965     0.960     0.960        75

[[29  0  0]
 [ 0 23  0]
 [ 0  3 20]]
```

This is output when n_neighbors = 9 and confusion matrix accuracy is 96.0%
*So for KNNthe n_neighbors the best we can use is 3 because the accuracy is 97.3%

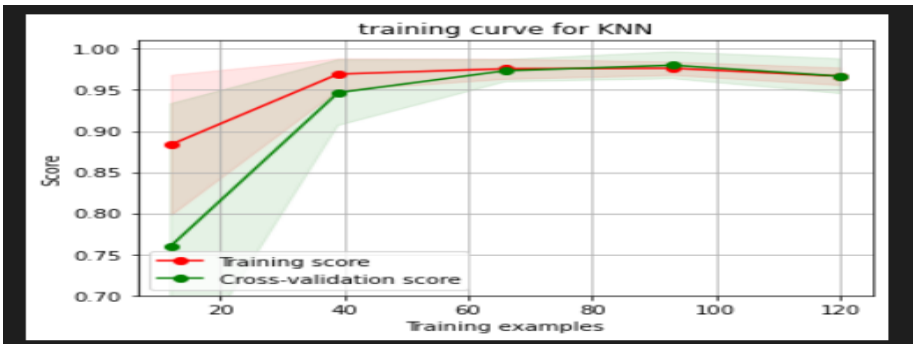
Learning curves for KNN:

Learning curves for n_neighbors for plotting the graphs

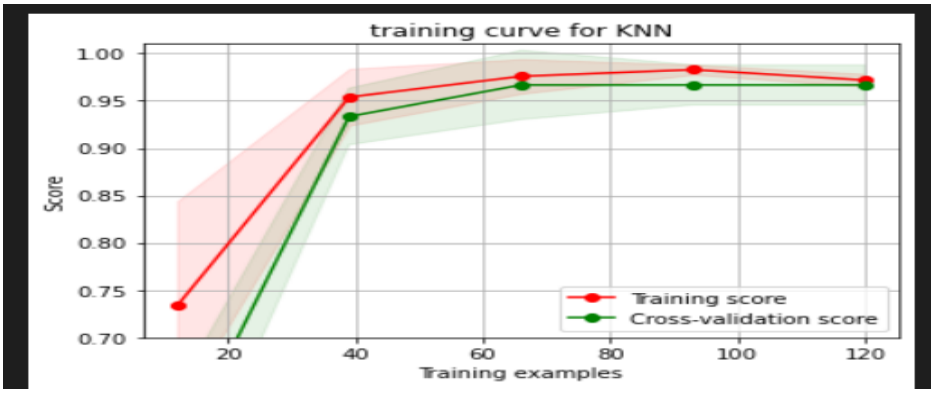
Training score: the score calculated from the training dataset that gives an idea of how well the model is learning
Cross-validation score: learning score calculated from a hold-out validation dataset that gives an idea of how well the model is generating



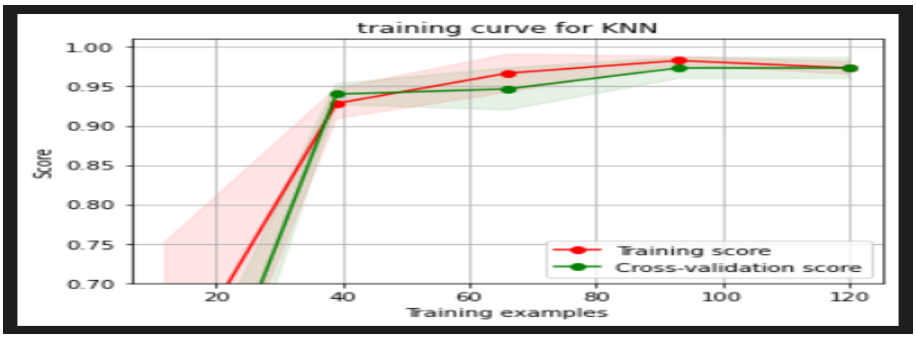
The graphs when n_neighbors =3



The graphs when n_neighbors =5



The graphs when n_neighbors =7



The graphs when N_neighbors =9

2) SVM- support vector machine:

Support vector machines (SVMs) are a group of supervised learning techniques for classifying data, performing regression analysis, and identifying outliers. Support vector machines' benefits include efficiency in high-dimensional environments. Still useful in situations where the number of dimensions exceeds the number of samples.

Training the model by giving the training data and changing the kernels for each iteration the kernels change every time giving the values for linear, poly, RBF, and sigmoid training the model with different kernels and checking the output. We use metrics. classification _report and confusion matrix for getting the classification report and the confusion matrix for y_test and

knn_predictions. y_test is the values that are to be tested on and knn_predictions are the predicted values by using the train data.
Kernel - It aids in transforming the data set used in triangulation in order for a non-linear decision surface to become a linear equation in more multidimensional spaces.

<u>There are 4 types of kernels;</u>
1)linear - it is used mainly for linearly separable data
2) poly-it represents the similarity of vectors in the training set of data in a feature space over polynomials of the original variables used in the kernel

3)rbf(Gaussian Kernel Radial Basis Function)- it is used to perform transformation when there is no prior knowledge about data and radial basis method to improve the transformation

4) sigmoid-this function is equivalent to a two-layer perceptron model of the neural network, which is used as an activation function

The output for SVM algorithm

```
Precision, Recall, Confusion matrix, intraining
, linear
              precision    recall  f1-score   support

         0.0      1.000     1.000     1.000        29
         1.0      1.000     1.000     1.000        23
         2.0      1.000     1.000     1.000        23

    accuracy                          1.000        75
   macro avg      1.000     1.000     1.000        75
weighted avg      1.000     1.000     1.000        75

[[29  0  0]
 [ 0 23  0]
 [ 0  3 20]]
```

By using linear kernel we got an accuracy of 100%

```
Precision, Recall, Confusion matrix, intraining
, poly
              precision    recall  f1-score   support

         0.0      1.000     1.000     1.000        29
         1.0      0.885     1.000     0.939        23
         2.0      1.000     0.870     0.930        23

    accuracy                          0.960        75
   macro avg      0.962     0.957     0.956        75
weighted avg      0.965     0.960     0.960        75

[[29  0  0]
 [ 0 23  0]
 [ 0  3 20]]
```

By using poly kernel we got an accuracy of 96%

```
Precision, Recall, Confusion matrix, intraining
, rbf
              precision    recall  f1-score   support

         0.0      1.000     1.000     1.000        29
         1.0      1.000     1.000     1.000        23
         2.0      1.000     1.000     1.000        23

    accuracy                          1.000        75
   macro avg      1.000     1.000     1.000        75
weighted avg      1.000     1.000     1.000        75

[[29  0  0]
 [ 0 23  0]
 [ 0  3 20]]
```

By using rbf kernel we got an accuracy of 100%

```
Precision, Recall, Confusion matrix, intraining
, sigmoid
              precision    recall  f1-score   support

         0.0      0.000     0.000     0.000        29
         1.0      0.361     0.565     0.441        23
         2.0      0.000     0.000     0.000        23

    accuracy                          0.173        75
   macro avg      0.120     0.188     0.147        75
weighted avg      0.111     0.173     0.135        75

[[29  0  0]
 [ 0 23  0]
 [ 0  3 20]]
```

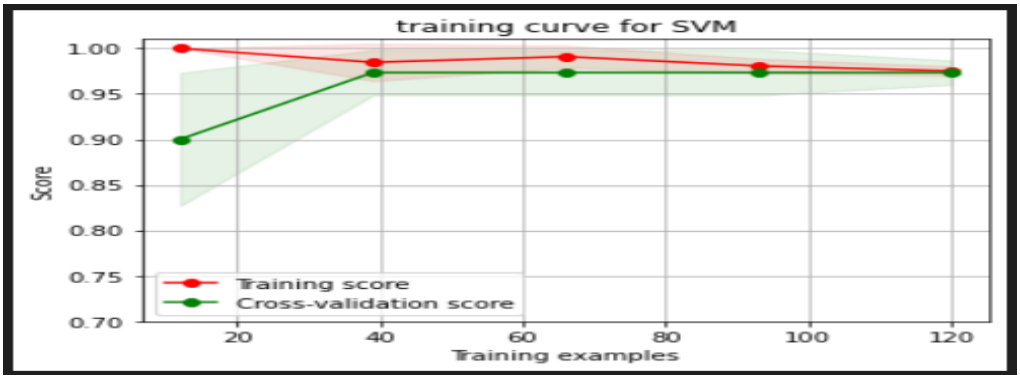
By using sigmoid kernel we got an accuracy of 17.1%

*So for SVM the best kernels we can use are RBF and Linear because both accuracies are 100%

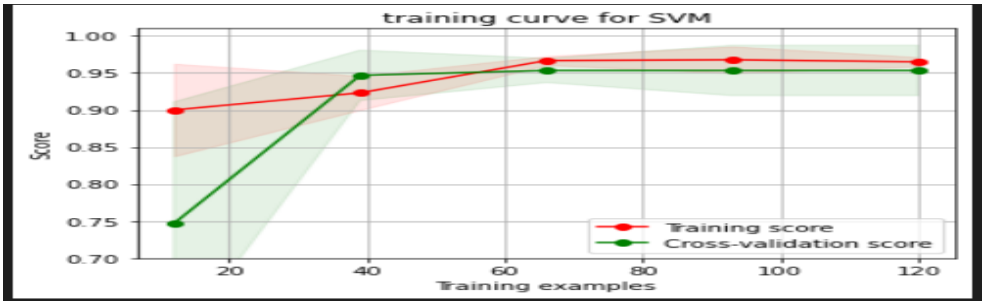
Learning curves for SVM:

Learning curves for SVM for plotting the graphs

Graph for “linear” kernel



Graph for “poly” kernel



Graph for “rbf” kernel

Logistic regression:

The method of modeling the likelihood of a discrete result given an input variable is known as logistic regression. The most popular type of logistic regression model is a binary result, i.e. True or False, Yes or No. Using multinomial logistic regression, events with more than two distinct possible outcomes can be modeled.

Training the model by giving the training data and also changing the Regularization methods like l1and l2 for each iteration, the Regularization methods increments every time the model is trained with a new penalty value. We use metrics classification _report and confusion_matrix for getting the classification report and the confusion matrix for y_test and knn_predictions. y_test is the values that are to be tested on and knn_predictions are the predicted values by using the train data.

Penalty - Penalizes the logistic model for having excessively numerous variables. This process, known as regularization, causes the coefficients of the less significant variables to decrease toward zero.

L1-this regularization penalizes the sum of values of weights

L2--this regularization penalizes the sum of squares of the weights

Output for logistic regression

```
Precision, Recall, Confusion matrix, intraining
, l1
              precision    recall  f1-score   support

         0.0      1.000     1.000     1.000        29
         1.0      1.000     0.957     0.978        23
         2.0      0.958     1.000     0.979        23

    accuracy                          0.987        75
   macro avg      0.986     0.986     0.986        75
weighted avg      0.987     0.987     0.987        75

[[29  0  0]
 [ 0 23  0]
 [ 0  3 20]]
```

By using l1 penalty we get an accuracy of 98.7%

```
Precision, Recall, Confusion matrix, intraining
, l2
              precision    recall  f1-score   support

         0.0      1.000     1.000     1.000        29
         1.0      1.000     0.913     0.955        23
         2.0      0.920     1.000     0.958        23

    accuracy                          0.973        75
   macro avg      0.973     0.971     0.971        75
weighted avg      0.975     0.973     0.973        75

[[29  0  0]
 [ 0 23  0]
 [ 0  3 20]]
```

By using l2 penalty we get an accuracy of 97.3%

**So for Logistic Regression the best Regularization method we can use is l1 and its accuracy is 98.7%

Learning curves for Logistic Regression:

Graph for penalty l1

Graph for penalty l2

2nd Dataset SVHN:

The Street View House Numbers (SVHN) is a dataset that includes bounding boxes for each unique digit and roughly 200k street numbers, for a total of about 600 k digits.
To our knowledge, all previously published research attempted to identify individual numbers by cropping them. Instead, we focus on concurrently recognizing all the digits in original photos with multiple digits.
Modern map production relies heavily on the ability to recognize multi-digit digits in images taken at street level. Google's Street View imagery, which consists of hundreds of millions of geo-located 360-degree panoramic photos, is a classic example of a corpus of such street-level shots. The ability to automatically translate an address number from a geo-located patch of pixels and link the translated number with a recognized street address makes it possible to locate a building with a high degree of precision.

Importing the dataset:

we have loaded the data we are converting image format data into arrays here we took the training length as 6000 and the testing length as 3000

Using SVHN dataset for the below algorithms.

1) SVM- support vector machine :

Training the model by giving the training data and also changing the kernels for each iteration the kernels change every time giving the values for linear, poly, rbf, and sigmoid training the model with different kernels and checking the output. We use metrics. classification _report and confusion_matrix for getting the classification report and the confusion matrix for y_test and y_pred. y_test is the values that are to be tested on and y_pred are the predicted values by using the train data.

The output for the SVM algorithm using different kernel:

accuracy			0.570	1000
macro avg	0.526	0.576	0.539	1000
weighted avg	0.620	0.570	0.577	1000

By using “poly kernel” we got an accuracy of 57%

accuracy			0.456	1000
macro avg	0.376	0.569	0.400	1000
weighted avg	0.673	0.456	0.483	1000

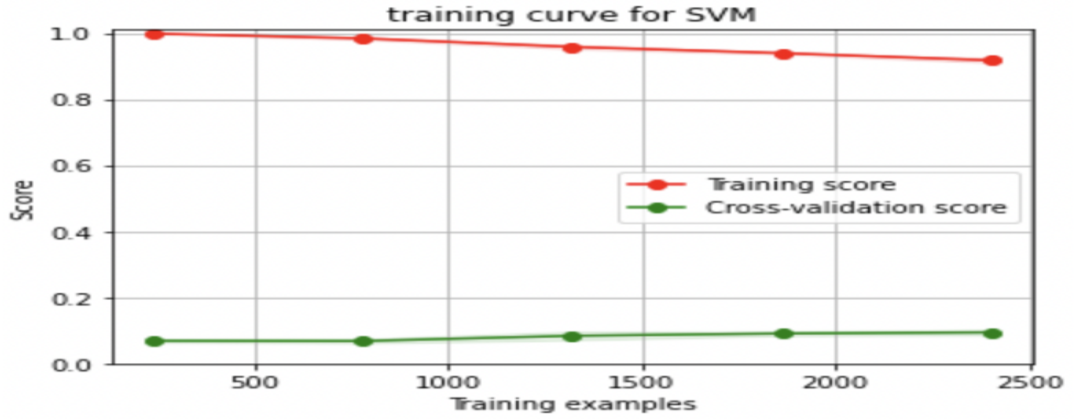
By using “rbf kernel” we got an accuracy of 45%

By using” linear kernel” for the SVHN dataset the dataset is linearly separable and it is time taking process to check all the 5000 data points so in this case, we are omitting it
By using sigmoid kernel we got an accuracy of 10%

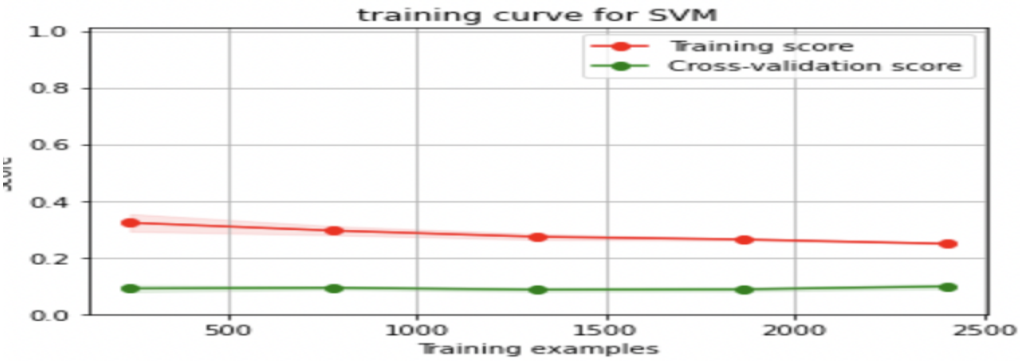
accuracy			0.100	1000
macro avg	0.087	0.029	0.042	1000
weighted avg	0.292	0.100	0.145	1000

*So for SVM the best kernel we can use are poly because the accuracy is 57%

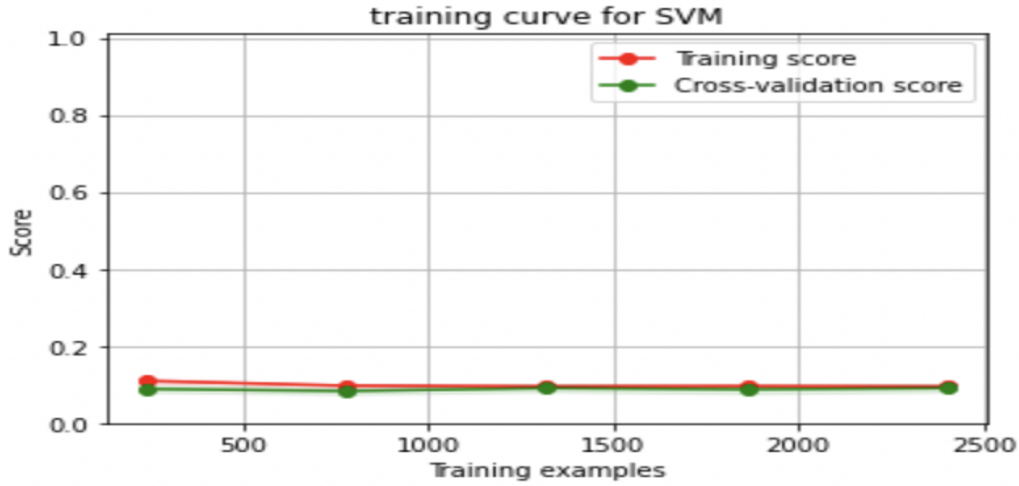
Learning Curves



Graph for poly kernel



Graph for rbf kernel



Graph for sigmoid kernel

2)KNN-K-nearest neighbors:

The output for the KNN algorithm using different neighbor values:

We took the n_neighbors as “3,5,7,9”
Training the model by giving the training data and also changing the n_neighbors values from 3 to 9 . for each iteration the n_neighbors value increments every time training the model with a new n_neighbors value. We use metrics. classification _report and confusion_matrix for getting the classification report and the confusion matrix for y_test and y_pred. y_test is the values that are to be tested on and y_pred are the predicted values by using the train data.the n_neighbors are the data points how many are we taking and also the data points which are near to the points which we are finding

Output for KNN:

```
              precision    recall  f1-score   support

         1.0      0.778     0.355     0.487      1274
         2.0      0.392     0.353     0.371       519
         3.0      0.238     0.255     0.246       322
         4.0      0.405     0.346     0.373       338
         5.0      0.192     0.310     0.237       168
         6.0      0.138     0.333     0.196        93
         7.0      0.097     0.361     0.153        61
         8.0      0.090     0.310     0.139        58
         9.0      0.106     0.303     0.157        66
        10.0      0.217     0.446     0.292       101

    accuracy                          0.341      3000
   macro avg      0.265     0.337     0.265      3000
weighted avg      0.498     0.341     0.378      3000

[[452 181 160 104  92  48 103  46  52  36]
 [ 44 183  46  31  32  25  61  26  29  42]
 [ 32  34  82  20  52  19  19  20  21  23]
 [ 33  22  17 117  20  52  10  25  15  27]
 [  8  10  17   3  52  23   4  24  14  13]
 [  2   3   3   5   9  31   3  17   8  12]
 [  3  11   6   1   1   2  22   4   8   3]
 [  3   8   4   2   4   8   1  18   8   2]
 [  3   8   6   0   5   9   4   7  20   4]
 [  1   7   4   6   4   7   0  14  13  45]]
```

This is output when n_neighbors =3 and confusion matrix accuracy is 34.1%

```
              precision    recall  f1-score   support

         1.0      0.780     0.368     0.500      1232
         2.0      0.398     0.403     0.400       462
         3.0      0.194     0.238     0.214       281
         4.0      0.433     0.393     0.412       318
         5.0      0.177     0.298     0.222       161
         6.0      0.201     0.388     0.265       116
         7.0      0.128     0.305     0.180        95
         8.0      0.114     0.287     0.164        80
         9.0      0.186     0.273     0.222       128
        10.0      0.222     0.362     0.275       127

    accuracy                          0.352      3000
   macro avg      0.283     0.332     0.285      3000
weighted avg      0.487     0.352     0.384      3000

[[453 165 145  92  81  50  94  48  57  47]
 [ 26 186  49  23  35  20  56  25  19  23]
 [ 32  35  67  17  43  12  18  19  18  20]
 [ 32  14  18 125  21  40   9  21  16  22]
 [ 10  10  27   3  48  24   6  15   9   9]
 [  5   2   7   8  11  45   2  18   3  15]
 [ 11  17  11   4   3   1  29   5   6   8]
 [  5   8   9   3   9   9   2  23   7   5]
 [  3  19   6  10  11  12   9  11  35  12]
 [  4  11   6   4   9  11   2  16  18  46]]
```

This is output when n_neighbors =5 and confusion matrix accuracy is 35.2%

```
              precision    recall  f1-score   support

         1.0      0.809     0.379     0.516      1239
         2.0      0.420     0.426     0.423       460
         3.0      0.220     0.265     0.241       287
         4.0      0.436     0.403     0.419       313
         5.0      0.207     0.311     0.248       180
         6.0      0.174     0.336     0.229       116
         7.0      0.132     0.312     0.186        96
         8.0      0.114     0.303     0.166        76
         9.0      0.170     0.256     0.204       125
        10.0      0.203     0.389     0.267       108

    accuracy                          0.363      3000
   macro avg      0.289     0.338     0.290      3000
weighted avg      0.506     0.363     0.397      3000

[[470 173 150  99  77  49  95  39  50  37]
 [ 23 196  47  17  30  16  56  23  23  29]
 [ 39  32  76  17  35  15  18  24  19  12]
 [ 18  13  14 126  20  49   9  18  16  30]
 [  9  12  20   6  56  27   4  20  11  15]
 [  4   1   9   8  16  39   0  20   4  15]
 [  8  17  11   1   5   3  30   6   8   7]
 [  2   6   5   2  11   8   4  23   9   6]
 [  4  11   7   9  14   9   9  16  32  14]
 [  4   6   6   4   7   9   2  12  16  42]]
```

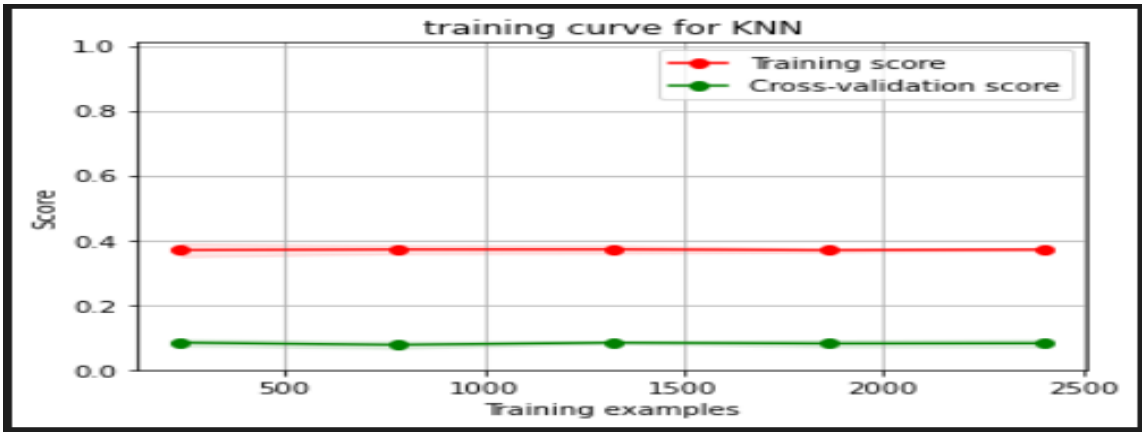
This is output when n_neighbors =7 and confusion matrix accuracy is 36.3%

```
              precision    recall  f1-score   support

         1.0      0.811     0.366     0.504      1288
         2.0      0.407     0.402     0.404       473
         3.0      0.214     0.263     0.236       281
         4.0      0.446     0.403     0.424       320
         5.0      0.196     0.298     0.236       178
         6.0      0.170     0.342     0.227       111
         7.0      0.128     0.354     0.188        82
         8.0      0.100     0.385     0.158        52
         9.0      0.154     0.284     0.200       102
        10.0      0.237     0.434     0.306       113

    accuracy                          0.361      3000
   macro avg      0.286     0.353     0.288      3000
weighted avg      0.517     0.361     0.396      3000

[[471 181 159  96  85  53 100  45  57  41]
 [ 31 190  52  22  37  16  57  23  21  24]
 [ 39  32  74  14  30  18  15  20  22  17]
 [ 16  15  10 129  26  43  11  21  16  33]
 [  8  15  18   6  53  26   4  19  10  19]
 [  3   4   8   9  13  38   3  20   3  10]
 [  5  12   8   1   4   4  29   6   6   7]
 [  2   2   2   1   4  10   1  20   7   3]
 [  4  11  10   6  12   7   4  15  29   4]
 [  2   5   4   5   7   9   3  12  17  49]]
```

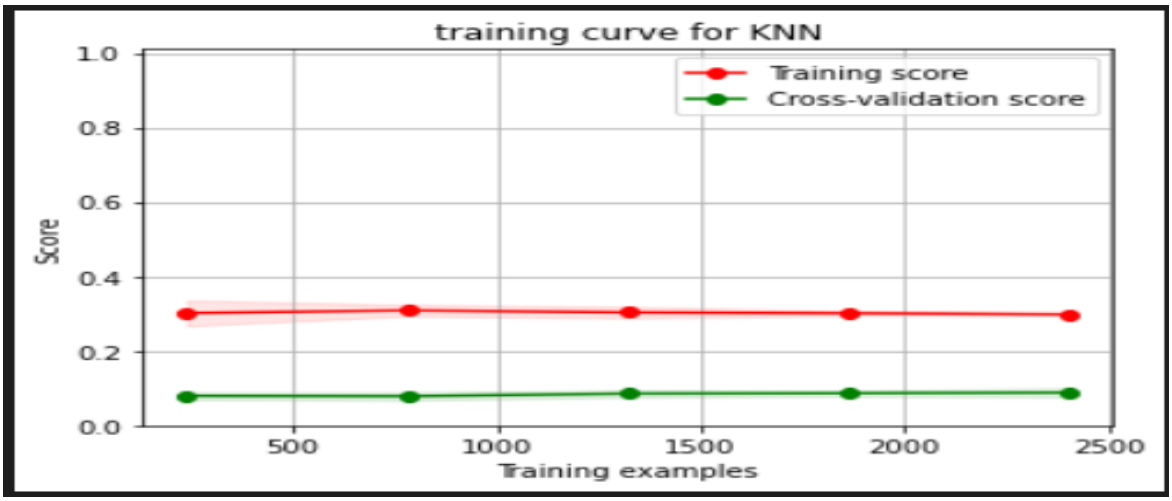
This is output when n_neighbors =9 and confusion matrix accuracy is 36.1%

*So for KNN the n_neighbors the best we can use is 7 because the accuracy is 36.3**%**

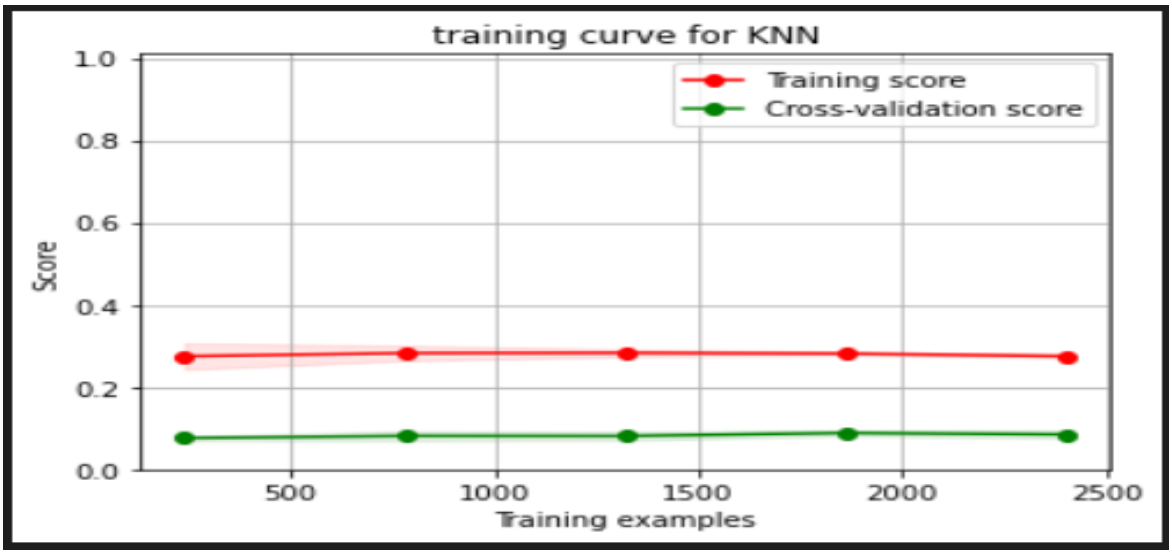
Learning curves:



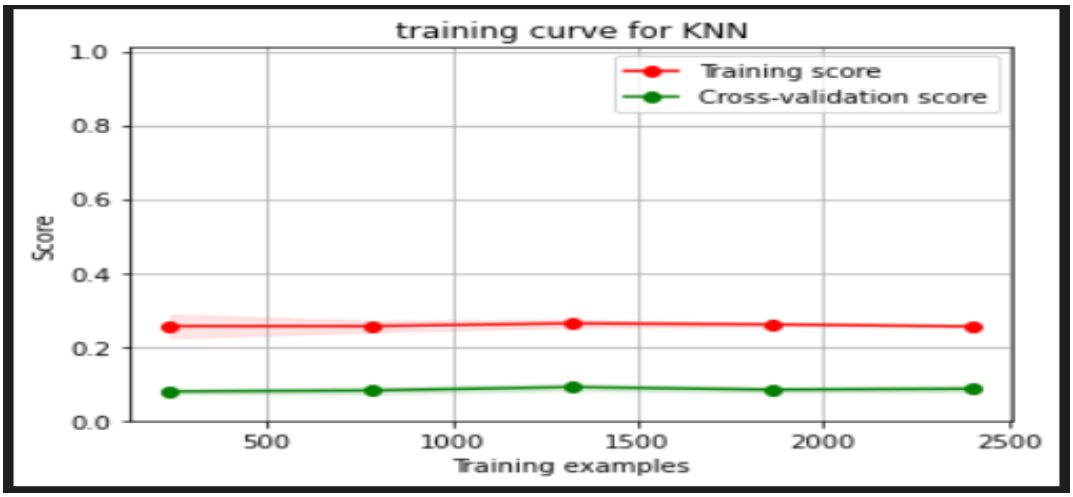
Graph for n_neighbors=3



Graph for n_neighbors=5



Graph for n_neighbors=7



Graph for n_neighbors=9

Logistic regression:

Training the model by giving the training data and also changing the Regularization methods like l2 and none for each iteration the Regularization methods increments every time training the model with a new penalty value. We use metrics classification _report and confusion_matrix for getting the classification report and the confusion matrix for y_test and test_preds2. y_test is the values that are to be tested on and test_preds2 are the predicted values by using the train data.

Output for Logistic Regression:

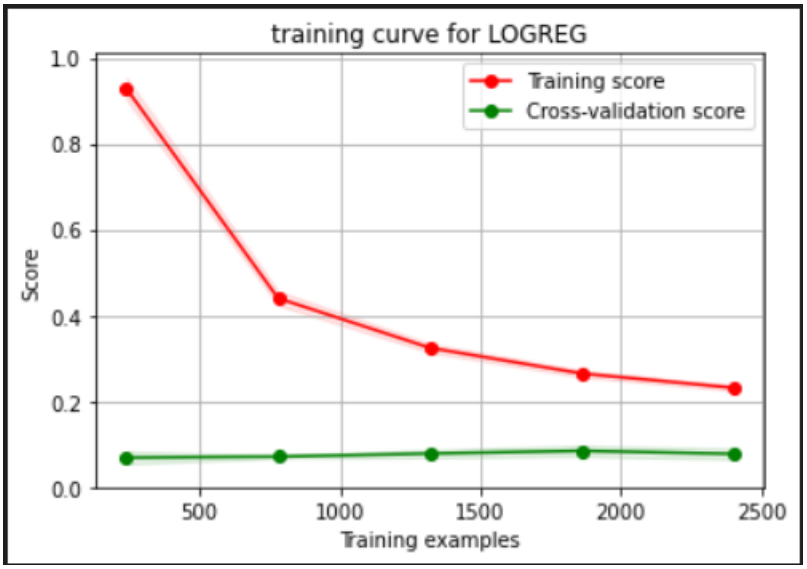
```
classification              precision    recall  f1-score   support

         1.0       0.24      0.55      0.33       581
         2.0       0.22      0.27      0.24       467
         3.0       0.18      0.10      0.13       345
         4.0       0.16      0.10      0.12       289
         5.0       0.17      0.10      0.13       271
         6.0       0.20      0.11      0.14       224
         7.0       0.11      0.05      0.07       227
         8.0       0.13      0.05      0.07       201
         9.0       0.19      0.11      0.14       188
        10.0       0.13      0.06      0.08       207

    accuracy                           0.21      3000
   macro avg       0.17      0.15      0.15      3000
weighted avg       0.18      0.21      0.18      3000

confusion matirx [[322  98  34  38  23  11  26   4  14  11]
 [187 125  28  24  16  15  29  16  14  13]
 [154  67  36  13  21   9  10  10  14  11]
 [123  55  13  30  17  12   7   5  12  15]
 [109  42  27  16  27  12   8   9  15   6]
 [ 92  44  11  18  10  24   7   2   6  10]
 [ 98  47   9  19  15  10  12   8   3   6]
 [ 83  32  14   8  15  16   4  10   4  15]
 [ 89  35  14  11   4   3   3   5  21   3]
 [104  29  11  14  11  10   2   7   6  13]]
```

By using “l2” penalty we get an accuracy of 21%

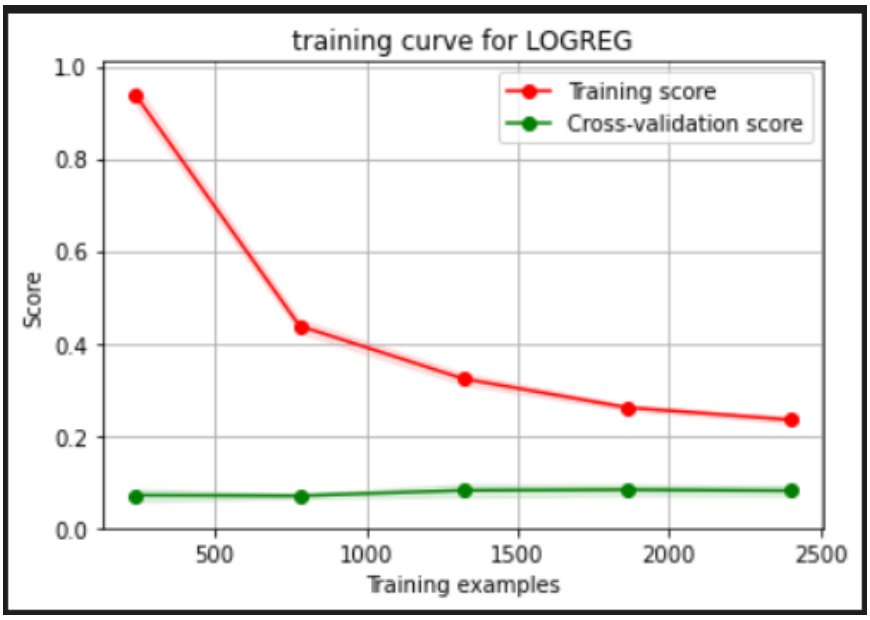
```
classification              precision    recall  f1-score   support

         1.0       0.24      0.56      0.33       581
         2.0       0.21      0.26      0.24       467
         3.0       0.18      0.10      0.13       345
         4.0       0.15      0.10      0.12       289
         5.0       0.17      0.10      0.13       271
         6.0       0.20      0.11      0.14       224
         7.0       0.10      0.05      0.07       227
         8.0       0.13      0.05      0.07       201
         9.0       0.19      0.11      0.14       188
        10.0       0.11      0.05      0.07       207

    accuracy                           0.21      3000
   macro avg       0.17      0.15      0.14      3000
weighted avg       0.18      0.21      0.17      3000

confusion matirx [[327  97  32  38  23  11  25   4  13  11]
 [186 122  30  25  17  15  29  16  14  13]
 [160  65  35  12  19  10  10   8  15  11]
 [123  55  13  29  19  12   7   5  11  15]
 [110  42  26  17  27  12   7   9  15   6]
 [ 90  47  12  17   9  24   7   2   6  10]
 [ 98  47  10  19  15  10  11   8   3   6]
 [ 83  33  14   8  15  15   4  10   4  15]
 [ 89  35  14  11   4   3   3   5  21   3]
 [107  28  11  12  11  10   2   8   7  11]]
```

By using “none” penalty we get an accuracy 21%

Learning Curves:



Graph for penalty “l2”



Graph for penalty “none”

Discussion:

KNN :

Iris dataset:-

the default distance metric is euclidean as the distance between 2 points is calculated using this euclidean distance

Here, we are changing n_neighbours to find the best parameter

cannot consider n = 1,2 or any less n value might misclassify the information due to closeness of data points, due to overfitting of data we cannot consider n =1 or 2, accuracy will be always highest

perhaps better not to consider it accurate

But we can see n = 9, accuracy again rises to 97.3, with this problem many points are widely separated will get wrongly classified due to distance between them is high as in underfitting

we can see n = 3 and 5 at it raises twice these values might be the optimal points and can be the best if n values among them

SVHN Dataset:-

here dataset is very large it might result in same level of the accuracy or any scores for different parameters around n =3,4,5,7, but if n. is large it might change due to underfitting of data

However, here we observed that score n = 7. as multinomial dataset we can't observe optimal k as accurately as possible

SVM :- param (linear, poly, rbf , sigmoid)

Iris dataset: -

Kernel linear models have linear decision boundaries and non-linear kernel models have more flexible on linear boundaries among all the kernels
linear kernel accuracy is highest as we observe by looking at the classification report and confusion matrix, we can say that iris dataset has data which can be linearly parted ways more
Poly and rbf deals with iris data in much more complex way, but as data is more simple and linear, svm picks up linear to be most accurate

SVHN Dataset: -

we found that poly kernel has more score that rest two rbf n sigmoid
as we vomited linear cause its more time taking for multinomial data sets divide on linear boundaries. sigmoid has least accuracy score compared to other 2
whereas poly and rbf will set nonlinear boundaries hence for larger data with higher degree data these are usually better in giving out the score

Logistic Regression :-

Iris dataset: -

here we use solver - liblinear as we have smaller dataset (iris)
now we found Penalty L1 is bit more accurate than L2
L1 calculation method - finds score using median estimate
L2 calculation method - finds score using mean of the data L2 normally avoids overfitting based on its calculation strategy considering iris dataset both models l1 n l2 does good,,
l1 does better as it finds the median - considering the data set iris we can visualize that, this dataset is spread across a line in 2d plane and if more towards one end of the plane then its better to calculate the data on less side
as L1 has median calculation methodology it might get a little edge over L2 which calculates over mean strategy which might be be the idle method when data tends slightly towards one side

SVHN dataset:

In this dataset we have to use default solver lbfgs as our data set is not small we can't use liblinear and solver - lbfgs is more appropriate
we found that both L2 and 'none 'Penalties have same accuracy when on predicting the data