PCA followed by classification of MNIST dataset using decision trees

MTH552A

Submitted by:

Ananyae Kumar Bhartari(190129)

Deepak Kumar Ram (180222)

Acknowledgement

We feel highly obliged to pay acknowledgment to Dr. Amit Mitra, Department of Mathematics and Statistics, IIT Kanpur, for giving us an opportunity, valuable guidance, and inspiration to complete this project.

We also thank the authors and publishers of the various books and sites without which this project would not have been completed.

At last, we would like to thank our seniors and batchmates for their cooperation throughout the project

• INTRODUCTION:

This report shows the application of the classification tree for classifying and the Principal component analysis for reduction of dimensions.

Our goal is to classify a given image of a handwritten digit into one of 10 classes representing integer values from 0 to 9. We will build a decision tree modal and compare their result and performance using the MNIST data set.

This report will show the following work:

1. Reshaping and splitting the dataset.
2. Reduce the dimension of the dataset using the Principal Component.
3. Find the best optimal height of the decision tree which give the best f1 score.
4. Predict the class label of images from optimal decision tree models.

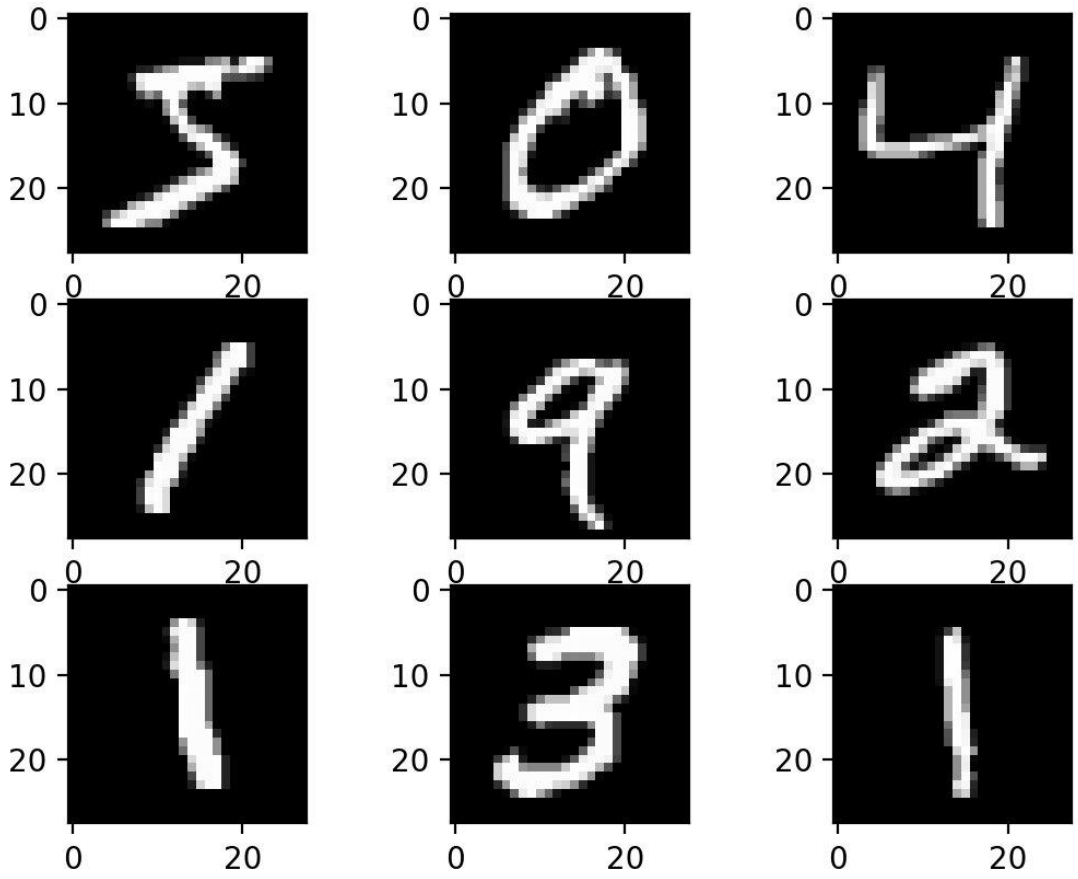
➢ ABOUT THE DATA:

The MNIST data set is well known public dataset that consists of handwritten single digits from zero to nine. It has a training set that has 60,000 examples and test set of 10,000 examples. And that image are indeed square of 28*28 pixels.

We load the MNIST dataset from keras dataset

```
(dataset_X, dataset_y), (test_X, test_y) = mnist.load_data()
```

This is the plot of the first nine images in the dataset:



Each image is in 28*28 matrix, for fit into the decision tree we need to flattering them into one dimension array

```
X = dataset_X.reshape((dataset_X.shape[0], 28*28))
Y=dataset_y
testX = test_X.reshape((test_X.shape[0], 28* 28))
testY=test_y
```

After flattering X shape are change to (60000,784)
And testX change to (10000,784)

Than we split the test dataset (X , Y) into 80 percent for train the model and 20 percent for test the model.

```
X_train, X_test, y_train, y_test = train_test_split(X,
Y, test_size = 0.2, random_state = 0)
```

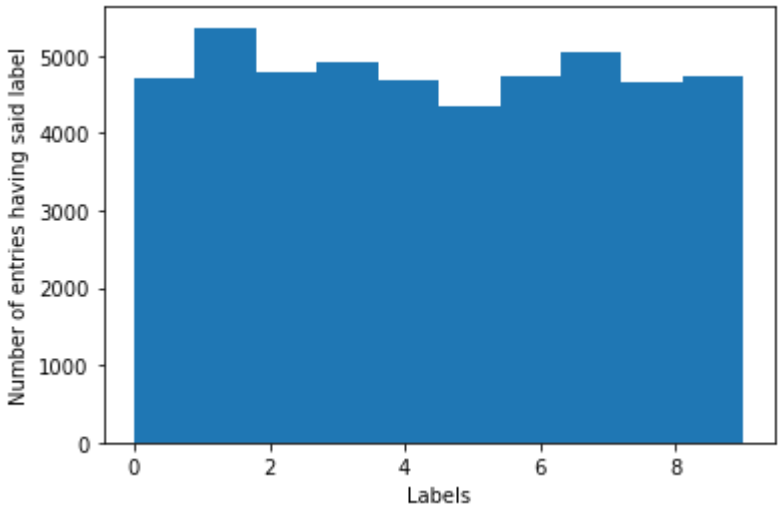
From dataset (X-train, y_train) we build the tree model of different height and find the best optimal height of tree.

And from dataset (X_test, y_test) predict the class label and compare from it.

Plotting histogram to check that Train dataset is well balanced:

```
fig, ax = plt.subplots()
plt.xlabel("Labels")
```

```
plt.ylabel("Number of entries having said label")
ax.hist(y_train)
plt.show()
```



We use the Principal Component Analysis(PCA) technique for reducing the dimensionality of the dataset.

```
pca = PCA(.95)

X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

X_train and X_test reduce from (48000, 784) to (48000, 154) and (12000, 784) to (12000, 154) respectively.

Metrics to evaluate the performances of different model

We discuss the main metric to evaluate our model .
We used Recall, Precision and F1 score to evaluate the best model.
First some definition:-
$\mathbf{TC_C}$ -#when the predicted class is C and the actual class the same class C.
$\mathbf{FP_C}$ -#when the actual class is C but predicted class is classes other than C.
$\mathbf{FN_C}$ -#when the predicted class is C but the actual class is other than C.

Recall:-

This is a measure that is given by the formula:-

$$\mathbf{R_C=TC_C /(TC_C+ FN_C)}$$

This intuitively means that percentage of predicting a class correctly.(example the model predicted 500 observation correctly belonging to C1 but actually 1000 observation from C1 class hece there were Recall is .5)

Actual

Precision:-

This is a measure that is given by the formula:-

$$\mathbf{P_C=TC_C /(TC_C+ FP_C)}$$

This intuitively means that percentage of times the class was predicted correctly. (For example the model said that 1000 observations belong to C1 but actually only 400 belong to C1 class . in this case Precision will be .4)

F1 score:-

F1-score is the Harmonic mean of the Precision and Recall:-

F1=2*Precision*Recall/(Precision+Recall)

NOTE:- in our case we are going to use weighted average of F1 score.

$$\textbf{Weighted F1}=\left(\sum_{i=1}^{n} \#\text{sample of Ci}*\mathbf{F1_{Ci}}\right)/\left(\sum_{i=1}^{n} \#\text{sample of Ci}\right)$$

Example of above metric using a example Confusion matrix

Predicted

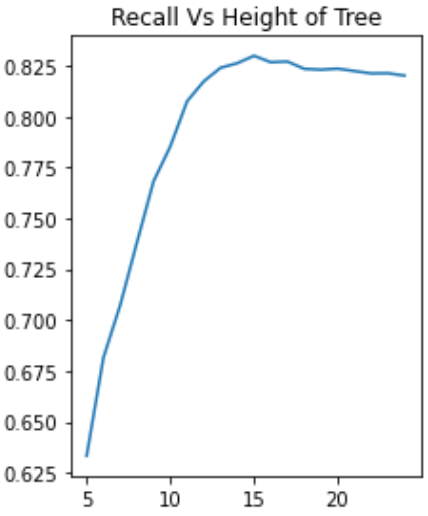
10	**2**	**3**
5	**10**	**4**
5	**1**	**10**

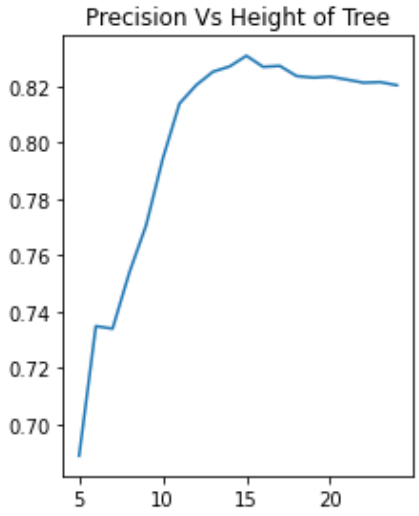
Recall for C1:-10/(10+2+3)=10/15=0.667
Precision for C1:-10/(10+5+5)=10/20=0.5
F1 score for C1:-2*.5*.667/(.5+.667)=0.571

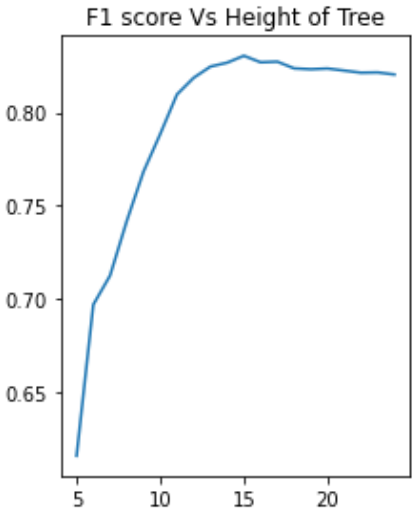
Making a series of models to get the best height of tree

Build a series of decision tree model of tree height range from 5 to 24 and for each model fit the (X_train , y_train) after that predict the class label for every X_test.
Also find the F1 score, precision and recall of each models.

```
models=list()
for i in range(5,25):
   clf = tree.DecisionTreeClassifier(random_state=0, max_depth=i)
   clf = clf.fit(X_train, y_train)
   y_pred=clf.predict(X_test)
   conf_matr=confusion_matrix(y_test, y_pred)
   f1_p_r=cal_weighted(conf_matr,y_test)
   dic=dict()
   dic["model"]=clf
   dic['pred']=y_pred
   dic["conf"]=conf_matr
   dic['F1']=f1_p_r[0]
   dic["Prec"]=f1_p_r[1]
   dic["Recall"]=f1_p_r[2]
   dic["Height"]=i
   models.append(dic)
```







Here we see that the best F1 score are heights between 10 and 15.
So we select the model with the best F1 score.

```
f1=list()
p=list()
r=list()
for i in range(len(models)):
   f1.append(models[i]['F1'])
   p.append(models[i]["Prec"])
   r.append(models[i]['Recall'])
Max_value = max(f1)
Model_selected = model[f1.index(max_value)
print("Height of tree ",model_selected["Height"])
```

Optimal height of tree is 15 which give the best prediction.

Now predict the image with training dataset `(testX, test_y)`.

```
y_pred=model_selected["model"].predict(testX)
```

Performance of models are:

	precision	recall	f1-score	support
0	0.88	0.89	0.89	980
1	0.97	0.96	0.96	1135
2	0.86	0.81	0.84	1032
3	0.79	0.81	0.80	1010
4	0.79	0.79	0.79	982
5	0.74	0.74	0.74	892
6	0.89	0.89	0.89	958
7	0.86	0.84	0.85	1028
8	0.72	0.78	0.75	974
9	0.76	0.76	0.76	1009
accuracy			0.83	10000
macro avg	0.83	0.83	0.83	10000
weighted avg	0.83	0.83	0.83	10000

- **Confusion Matrix**: A confusion matrix(also known as error matrix) is a summary of prediction results on a classification problem. Each row of the matrix represents the number of values in the predicted class whereas each column represents the number of values in the actual class.

  Confusion matrix of models are:

```
[[ 872    5   18    9    5   28   19    6   10    8]
 [   5 1090    5    7    5    2    7    1   13    0]
 [  21    2  838   38   12   17   13   20   63    8]
 [  16    3   14  814    7   54   11   14   66   11]
 [   1    4   13    8  772   16   16   20   27  105]
 [  18    4   13   45   25  662   26   12   65   22]
 [  25    2   18    9   22   18  848    3    8    5]
 [   2    9   14   18   23   12    6  864   20   60]
 [  15    0   30   61   13   63    8    9  755   20]
 [  12    6    6   18   98   25    4   61   17  762]]
```