CSE : 343 Machine Learning

Assignment 2

Arka Sarkar , 2018222

Question 1

(a) **Principal Component Analysis** (PCA) is an unsupervised learning method in machine learning which extracts new independent features from a set of old features. It is essentially a dimensionality reduction technique, it groups the features in such a way that we can drop the least significant feature for our model prediction.
PCA is calculated in the following manner :

1. Standardisation of data.
2. Generate correlation matrix for the desired dimensions.
3. Calculate the eigenvectors (i.e. the principal component) and the variance.
4. Eigen pairs are arranged in decreasing order with respect to the eigenvalues. The first principal component will have the maximum value, thus projects the maximum information of the original data.

PCA projects the data into a lower dimensional space while retaining the maximum variance in the data.
PCA can be used for multiple instances, some listed below :

1. To reduce the number of features in the dataset.
2. To check whether the features are independent of each other or not.

(b) **Singular Value Decomposition** (SVD) is dimensionality reduction technique in data science. It is an unsupervised learning method that decomposes a matrix (A) into 3 matrices namely **U, S, V** such as :

$$A = U S V^T$$

Where U,V are orthogonal matrices with orthonormal eigenvectors. S is a diagonal matrix with elements equal to the root of the positive eigenvalues of A^TA and AA^T in decreasing order.
This method decomposes a matrix into lower rank matrices which enables us to pick columns having higher singular values. Thus, we have reduced a high-rank matrix into a low rank matrix while preserving the most important information.

(c) **t-Distributed Stochastic Neighbor Embedding** (t-SNE) is an unsupervised learning method primarily used for visualizing high dimensional data. t-SNE is similar to PCA but is more flexible to non-linear data. While PCA preserves large pairwise distances to maximise variance, t-SNE on the other hand preserves only small pairwise changes or similarities, doing this would preserve the structure of a non-linear data as well. t-SNE algorithm calculates similarity between the pairs in a high dimensional space as well as in the low dimensional space and tries to optimise the two using a cost function.

(d) **Stratified Sampling** is a method of dividing data into sets having similar features. This method ensures that one label is not trained more than the other in the model.
Performing stratified sampling on 80-20 train-test split on the dataset gave us the following **results** :

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

For **Training** set :

#	class 0	class 1	class 2	class 3	class 4	class 5	class 6	class 7	class 8	class 9
Frequency	320	395	314	339	333	318	353	345	328	315
Ratio	0.095	0.118	0.093	0.101	0.099	0.095	0.105	0.103	0.098	0.094

For **Testing** set :

#	class 0	class 1	class 2	class 3	class 4	class 5	class 6	class 7	class 8	class 9
Frequency	80	99	79	85	83	80	88	86	82	78
Ratio	0.095	0.118	0.094	0.101	0.099	0.095	0.105	0.102	0.098	0.093

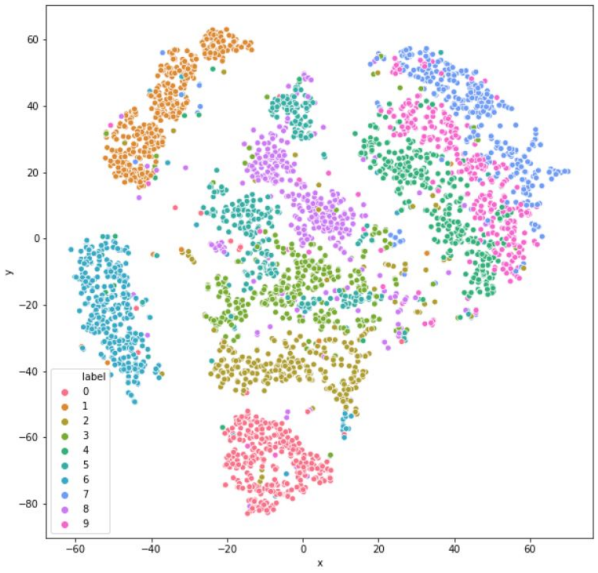
As it can be seen that

(e) Training sklearn Linear Regression model on dimensionally reduced **Dataset A** (Using PCA), we get an **accuracy = 0.879762 , precision score = 0.873893 , f1 score = 0.871003.**

```
C:\Users\Asus\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:
ged to 'auto' in 0.22. Specify the multi_class option to silence this warni
  "this warning.", FutureWarning)
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor

convergence after 2115 epochs took 4 seconds
convergence after 3224 epochs took 8 seconds
convergence after 1259 epochs took 3 seconds
convergence after 4729 epochs took 11 seconds
convergence after 2262 epochs took 5 seconds
convergence after 1687 epochs took 4 seconds
convergence after 2699 epochs took 6 seconds
convergence after 2475 epochs took 6 seconds
convergence after 1499 epochs took 3 seconds
convergence after 3360 epochs took 8 seconds



|   Accuracy score |   Precision Score |   f1 score |
|------------------+-------------------+------------|
|         0.879762 |          0.873893 |   0.871003 |

[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:   58.5s finished
```

Applying **t-SNE** to analyse the training data :

```
tsne_em = TSNE(n_components=2, perplexity=30.0, n_iter=1000, verbose=1).fit_transform(X_train)

[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 3360 samples in 0.006s...
[t-SNE] Computed neighbors for 3360 samples in 0.892s...
[t-SNE] Computed conditional probabilities for sample 1000 / 3360
[t-SNE] Computed conditional probabilities for sample 2000 / 3360
[t-SNE] Computed conditional probabilities for sample 3000 / 3360
[t-SNE] Computed conditional probabilities for sample 3360 / 3360
[t-SNE] Mean sigma: 4.441751
[t-SNE] KL divergence after 250 iterations with early exaggeration: 78.592880
[t-SNE] KL divergence after 1000 iterations: 1.285138
```



Analysis :

We can clearly see the individual clusters pertaining to the different labels.
Hence PCA has selected very good principal components which is also evident by our model accuracy.

(f) Training sklearn Linear Regression model on dimensionally reduced **Dataset A** (Using SVD), we get an **accuracy = 0.880952 , precision score = 0.874865,f1 score =0.872404.**

```
C:\Users\Asus\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:469: F
ged to 'auto' in 0.22. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

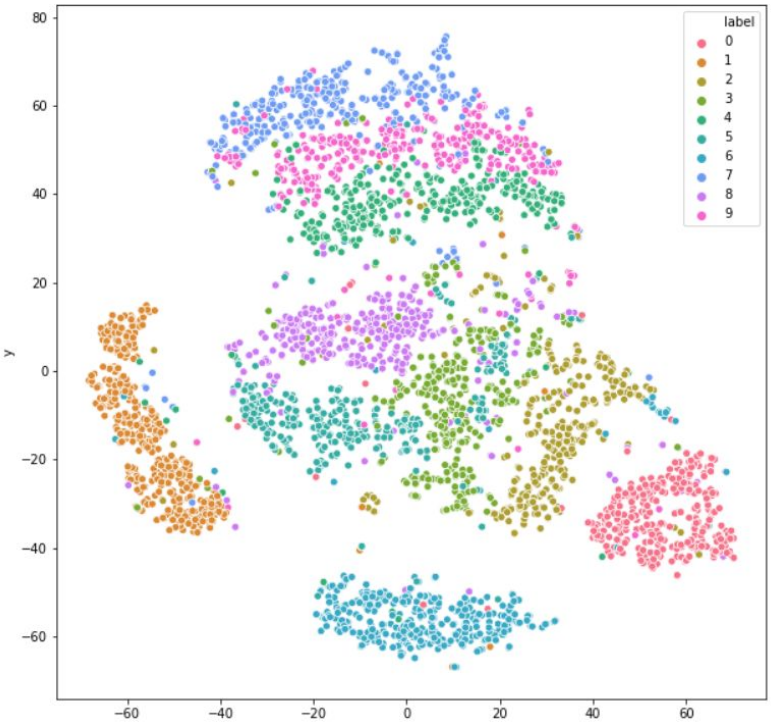
```
convergence after 2329 epochs took 6 seconds
convergence after 3261 epochs took 8 seconds
convergence after 1254 epochs took 4 seconds
convergence after 4757 epochs took 11 seconds
convergence after 2230 epochs took 6 seconds
convergence after 1774 epochs took 4 seconds
convergence after 2716 epochs took 7 seconds
convergence after 2291 epochs took 5 seconds
convergence after 1546 epochs took 4 seconds
convergence after 2039 epochs took 5 seconds
```

```
|   Accuracy score |   Precision Score |   f1 score |
|------------------+-------------------+------------|
|         0.880952 |          0.874865 |   0.872404 |
```

```
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:   59.5s finished
```

Applying **t-SNE** on the training set :

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 3360 samples in 0.007s...
[t-SNE] Computed neighbors for 3360 samples in 1.015s...
[t-SNE] Computed conditional probabilities for sample 1000 / 3360
[t-SNE] Computed conditional probabilities for sample 2000 / 3360
[t-SNE] Computed conditional probabilities for sample 3000 / 3360
[t-SNE] Computed conditional probabilities for sample 3360 / 3360
[t-SNE] Mean sigma: 4.446044
[t-SNE] KL divergence after 250 iterations with early exaggeration: 78.589691
[t-SNE] KL divergence after 1000 iterations: 1.274800
```



Analysis :

We can clearly see the individual clusters pertaining to the different labels.
Hence SVD has selected very good principal components which is also evident by our model accuracy.

(g) The accuracy obtained on using SVD is **0.880952** which is slightly higher than accuracy obtained on using PCA (i.e. **0.879762),** though this difference lies in the margin of error but we can still claim that SVD have selected better features as compared to PCA.
The main difference between SVD and PCA is that, SVD gives us a diagonalisation matrix of the singular values which are easy to manipulate and analyse. PCA skips the less significant components while SVD keeps them. We can use SVD to find PCA by truncating the less important basis vectors in the SVD matrix decomposition.
We cannot concretely comment on the performance of both the models as they are quite similar and hence should give similar results.

Question 2

The columns were swapped in the dataset to produce a more realistic model.

(a) **100** bootstrap samples were created and are of training size = **8500** and were tested on **1500** test examples. The results are :

```
Bias :  1.1703697213874578
Variance : 0.000514523938981076
MSE 2.201868871503514
MSE - bias**2 - Variance :  0.8315890628239774
```

(b) The value MSE - bias^2 - variance is **0.8315890628239774.** This value indicates the noise in out dataset. Higher this value more would be the noise.

Question 3

Dataset A

The dataset was split into **60-20-20 train-test-val** splits.

```
X,Y = load_dataset(0)
train_ratio = 0.6
validation_ratio = 0.2
test_ratio = 0.2
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=1 - train_ratio)
x_val, x_test, y_val, y_test = train_test_split(x_test, y_test, test_size=test_ratio/(test_ratio + validation_ratio))

print(" x_train shape : ", x_train.shape, "\n", "x_val shape : ", x_val.shape,"\n","x_test shape :", x_test.shape)
```

```
(4200, 784) (4200, 10)
The class frequencies are :
The frequency of class 0 is 400.0 / 4200
The frequency of class 1 is 494.0 / 4200
The frequency of class 2 is 393.0 / 4200
The frequency of class 3 is 424.0 / 4200
The frequency of class 4 is 416.0 / 4200
The frequency of class 5 is 398.0 / 4200
The frequency of class 6 is 441.0 / 4200
The frequency of class 7 is 431.0 / 4200
The frequency of class 8 is 410.0 / 4200
The frequency of class 9 is 393.0 / 4200
 x_train shape :  (2520, 784)
 x_val shape :  (840, 784)
 x_test shape : (840, 784)
```

(a) To find the optimal depth in case of **Decision Tree (DT)** using **MyGridSearch** implemented in Q3.py file

```
X,Y = load_dataset(0)
std_slc = StandardScaler()

X = std_slc.fit_transform(X)

dec_tree = tree.DecisionTreeClassifier()

max_depth = list([i for i in range(1,30,1)])

parameters = dict( max_depth=max_depth)

clf_GS = MyGridSearchCV(dec_tree, parameters)
clf_GS.fit(X, Y)
```

The parameters were :

Depth : 1 to 30
Cv = 5 (default)

```
Best  max_depth  :  10
DecisionTreeClassifier ( {'class_weight': None, 'criterion': 'gini', 'max_depth': 29, 'max_features': None, 'max_leaf_nodes': N
one, 'min_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fract
ion_leaf': 0.0, 'presort': False, 'random_state': None, 'splitter': 'best'} )
```

The optimal depth achieved was **10.**

The accuracy achieved on **depth = 10** was **Best validation score achieved : 0.763095238095238**
Average validation score achieved : 0.75

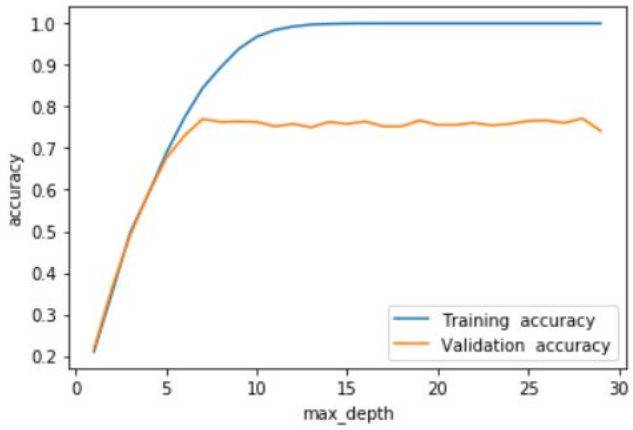
```
Current parameters :  {'max_depth': 10}
Best validation score achieved :  0.763095238095238  Average validation score achieved :  0.75
```

Running **Gaussian Naive Bayes** Classification with 5 fold cross validation, the results obtained were :

Best Model	Average Validation Score	Best Validation Score	Best training Score
GaussianNB(priors=None, var_smoothing=1e-09)	0.550238	0.586905	0.607324

Analysis:
The Gaussian Naive Bayes model performed relatively poorly as compared to the Decision Tree model. This can be due to the fact that Dataset A has a lot of **0 columns** which makes the gaussian estimate to fluctuate a lot and potentially reach an underflow.

(b) Training vs Validation accuracy with respect to tree depth plot.



Analysis:
The curve clearly shows us that the model overfits at higher depth. The optimal depth chosen from our grid search was 10, which is very correct as after 10 depth the model begins to overfit.

(c) Saving the **best model** to disk.

```
#best model save
dec_tree = tree.DecisionTreeClassifier(max_depth = 10)
dec_tree.fit(x_train, y_train)
filename = 'models/DT_datasetA.sav'
pickle.dump(dec_tree, open(filename, 'wb'))
```

Loading the best model from disk and **predicting** on the test set.

```
loaded_model = pickle.load(open(filename, 'rb'))
score = loaded_model.score(x_test, y_test)
print(score)
```

```
0.7416666666666667
```

The accuracy received on the test set is **0.7416666666666667.**

(d)

For Decision Tree classifier

The model was evaluated using **MyEvaluationMetric** implemented in Q3.py.

```
metrics = MyEvaluationMetric()
confusion_matrix = metrics.confusion_matrix(y_test,y_pred)
accuracy_score = metrics.accuracy_score(y_test,y_pred)
pres_score_micro = metrics.precision_score(y_test,y_pred,average = "micro")
pres_score_macro = metrics.precision_score(y_test,y_pred,average = "macro")
recall_score_micro = metrics.recall_score(y_test,y_pred,average = "micro")
recall_score_macro = metrics.recall_score(y_test,y_pred,average = "macro")
f1_score_micro = metrics.f1_score(y_test,y_pred,average = "micro")
f1_score_macro = metrics.f1_score(y_test,y_pred,average = "macro")
```

The results obtained are :

1. **Confusion Matrix**

```
Confusion matrix
 [[75  0  0  1  0  5  2  2  1  2]
 [ 0 82  1  2  2  1  0  1  2  0]
 [ 0  0 56  5  8  1  5  2  4  1]
 [ 2  2  4 64  0  1  1  1  3  3]
 [ 1  0  0  1 69  2  6  3  1  8]
 [ 0  1  3  7  3 47  3  3  1  2]
 [ 2  0  4  1  2  7 64  2  5  4]
 [ 1  2  0  0  0  0  0 64  2  6]
 [ 4  6  6 11  1  2  2  0 49  6]
 [ 0  2  4  1  7  4  2  7  4 53]]
```

2. **Other metrics**

   This being a multiclass dataset, Micro and Macro averages were calculated for all the metrics :

   Micro Average :
   In the micro average method, individual components such as true positive, false negative, false positive, true negative are summed up and applied to get the required statistic.
   For Example :

   Micro - Average for precision $$P = \frac{\sum_c TP_c}{\sum_c TP_c + \sum_c FP_c}$$

   Macro Average :
   In the macro average method, the averages of all the precisions for each class is taken.
   For Example :

   Macro-Average for precision **P =** $$\frac{\sum_{i=1}^{l} \frac{tp_i}{tp_i + fp_i}}{l}$$

   The results obtained on the dataset were :

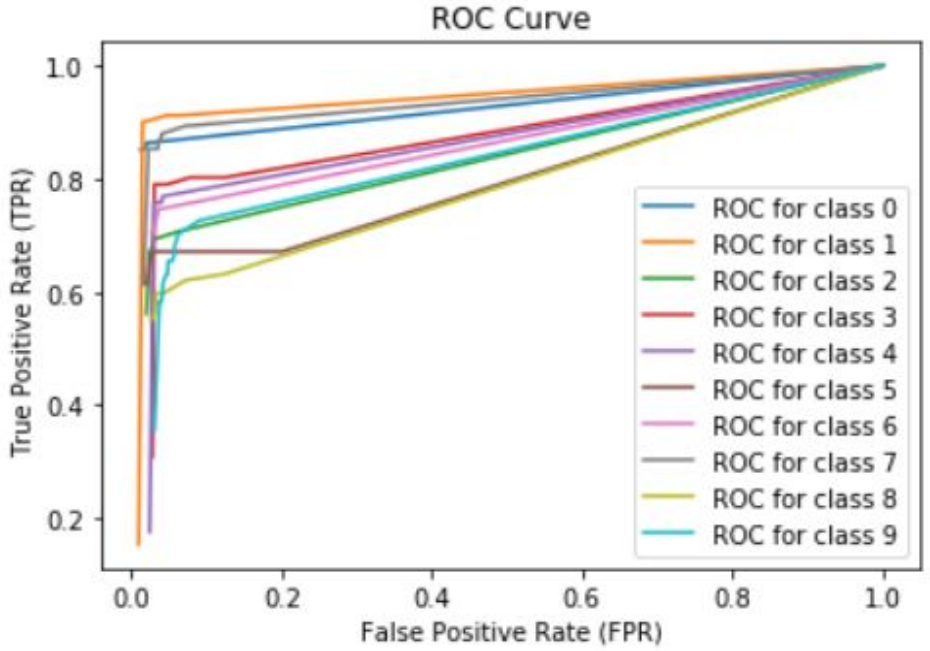
```
l = [[accuracy_score,pres_score_micro,pres_score_macro,recall_score_micro,recall_score_macro,f1_score_micro,f1_score_macr
table = tabulate(l, headers=["Accuracy","Micro Precision ", "Macro precision ", "Micro Recall ", "Macro Recall ", "Micro
print("\n")
print(table)
```

Accuracy	Micro Precision	Macro precision	Micro Recall	Macro Recall	Micro f1	Macro f1
0.741667	0.741667	0.738303	0.741667	0.740689	0.741667	0.739494

3. **ROC curve**

   Since this is a multiclass dataset, the roc curves are plotted for all the classes for this dataset.

```
prob = loaded_model.predict_proba(x_test)
metrics.plot_roc_curve(y_test, prob)
```



For Gaussian Naive Bayes classifier

The model was evaluated using **MyEvaluationMetric** implemented in Q3.py.

```
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
```

```
metrics = MyEvaluationMetric()
confusion_matrix = metrics.confusion_matrix(y_test,y_pred)
accuracy_score = metrics.accuracy_score(y_test,y_pred)
pres_score_micro = metrics.precision_score(y_test,y_pred,average = "micro")
pres_score_macro = metrics.precision_score(y_test,y_pred,average = "macro")
recall_score_micro = metrics.recall_score(y_test,y_pred,average = "micro")
recall_score_macro = metrics.recall_score(y_test,y_pred,average = "macro")
f1_score_micro = metrics.f1_score(y_test,y_pred,average = "micro")
f1_score_macro = metrics.f1_score(y_test,y_pred,average = "macro")
print("Confusion matrix \n", confusion_matrix)
```

The results obtained are :

1. **Confusion Matrix**

```
Confusion matrix
 [[77  0  1  0  1  1  4  0  1  3]
 [ 2 84  0  1  0  0  1  0  0  3]
 [18  3 19  8  0  4 17  0 12  1]
 [10  6  4 40  0  0  6  0  4 11]
 [ 2  1  2  0 29  3  6  5  3 40]
 [14  1  1  5  1 12  2  1 21 12]
 [ 0  2  1  0  0  1 86  0  1  0]
 [ 1  0  0  2  2  0  0 30  0 40]
 [ 5 16  1  1  0  0  2  0 44 18]
 [ 1  1  2  0  3  0  0  4  1 72]]
```

2. **Other metrics**

   This being a multiclass dataset, Micro and Macro averages were calculated for all the metrics :

   Micro Average :

   In the micro average method, individual components such as true positive, false negative, false positive, true negative are summed up and applied to get the required statistic.

   For Example :

   Micro - Average for precision $$P = \frac{\sum_c TP_c}{\sum_c TP_c + \sum_c FP_c}$$

   Macro Average :

   In the macro average method, the averages of all the precisions for each class is taken.

   For Example :

   Macro-Average for precision **P =** $$\frac{\sum_{i=1}^{l} \frac{tp_i}{tp_i + fp_i}}{l}$$

   The results obtained on the dataset were :

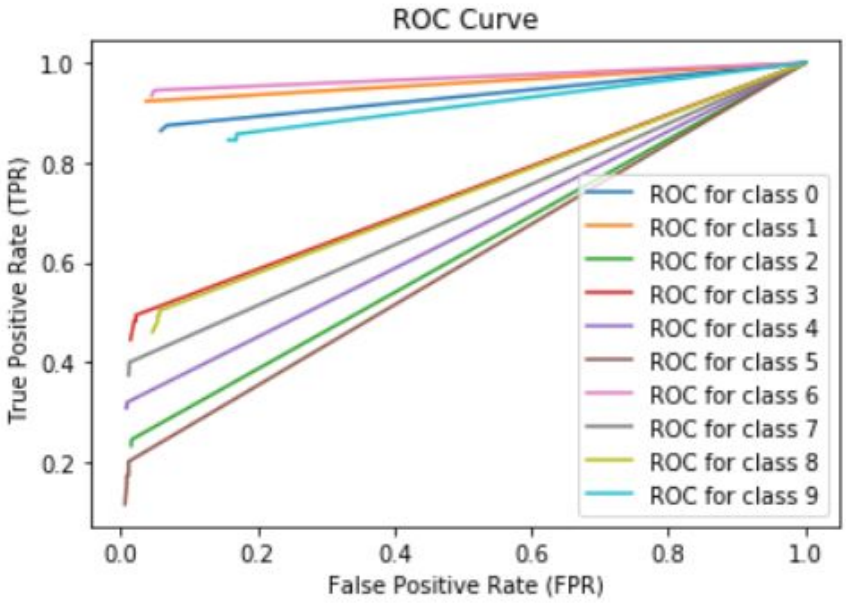
```
l = [[accuracy_score,pres_score_micro,pres_score_macro,recall_score_micro,recall_score_macro,f1_score_micro,f1_score_macro]]
table = tabulate(l, headers=["Accuracy","Micro Precision ", "Macro precision ", "Micro Recall ", "Macro Recall ", "Micro f1 ","M
print("\n")
print(table)
```

```
|   Accuracy |   Micro Precision  |   Macro precision  |   Micro Recall  |   Macro Recall  |   Micro f1  |   Macro f1  |
|------------+--------------------+--------------------+-----------------+-----------------+-------------+-------------|
|   0.586905 |           0.586905 |           0.633009 |        0.586905 |        0.572167 |    0.586905 |    0.601052 |
```

3. **ROC curve**

   Since this is a multiclass dataset, the roc curves are plotted for all the classes for this dataset.

```
prob = clf.predict_proba(x_test)
metrics.plot_roc_curve(y_test, prob)
```



Dataset B

The dataset was split into **60-20-20 train-test-val** splits.

```
X,Y = load_dataset(1)
train_ratio = 0.6
validation_ratio = 0.2
test_ratio = 0.2
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=1 - train_ratio)
x_val, x_test, y_val, y_test = train_test_split(x_test, y_test, test_size=test_ratio/(test_ratio + validation_ratio))

print(" x_train shape : ", x_train.shape, "\n", "x_val shape : ", x_val.shape,"\n","x_test shape :", x_test.shape)
```

```
(4200, 2048) (4200, 2)
The class frequencies are :
The frequency of class 0 is 2097 / 4200
The frequency of class 1 is 2103 / 4200
 x_train shape :  (2520, 2048)
 x_val shape :  (840, 2048)
 x_test shape : (840, 2048)
```

(a) To find the optimal depth in case of **Decision Tree (DT)** using **MyGridSearch** implemented in Q3.py file

```
X,Y = load_dataset(1)
std_slc = StandardScaler()

X = std_slc.fit_transform(X)

dec_tree = tree.DecisionTreeClassifier()

max_depth = list([i for i in range(1,30,1)])

parameters = dict( max_depth=max_depth)

clf_GS = MyGridSearchCV(dec_tree, parameters)
clf_GS.fit(X, Y)
```

The parameters were :

Depth : 1 to 30
Cv = 5 (default)

The optimal depth achieved was **6.**

```
Best  max_depth  :  6
DecisionTreeClassifier ( {'class_weight': None, 'criterion': 'gini', 'max_depth': 29, 'max_features': None, 'max_leaf_nodes': N
one, 'min_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fract
ion_leaf': 0.0, 'presort': False, 'random_state': None, 'splitter': 'best'} )
```

The accuracy achieved on **depth = 6** was **Best validation score achieved : 0.6047619047619047** and **Average validation score achieved : 0.5895238095238096**

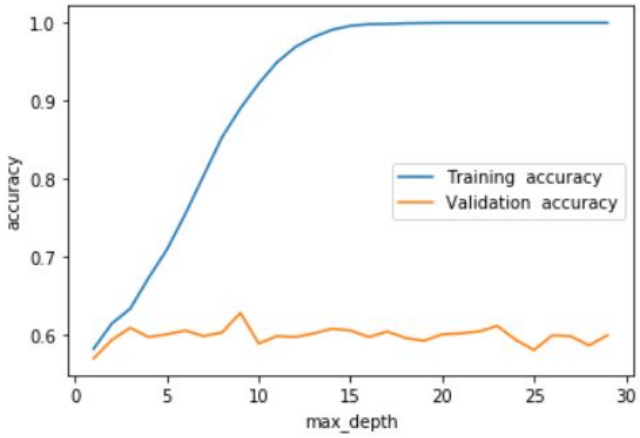
```
Current parameters :  {'max_depth': 6}
Best validation score achieved :  0.6047619047619047  Average validation score achieved :  0.5895238095238096
```

Running **Gaussian Naive Bayes** Classification with 5 fold cross validation, the results obtained were :

Best Model	Average Validation Score	Best Validation Score	Best training Score
GaussianNB(priors=None, var_smoothing=1e-09)	0.571905	0.586905	0.57619

Analysis:
The Gaussian Naive Bayes model performed relatively poorly as compared to the Decision Tree model. This can be due to the fact that Dataset B has a lot of **0 columns** which makes the gaussian estimate to fluctuate a lot and potentially reach an underflow.

(b) Training vs Validation accuracy with respect to tree depth plot.



Analysis:
The curve clearly shows us that the model overfits at higher depth. The optimal depth chosen from our grid search was 6, which is very correct as after 6 depth the model begins to overfit.

(c) Saving the **best model** to disk.

```
#best model save
dec_tree = tree.DecisionTreeClassifier(max_depth = 6)
dec_tree.fit(x_train, y_train)
filename = 'models/DT_datasetB.sav'
pickle.dump(dec_tree, open(filename, 'wb'))
```

Loading the best model from disk and **predicting** on the test set.

```
loaded_model = pickle.load(open('models/DT_datasetB.sav', 'rb'))
score = loaded_model.score(x_test, y_test)

print(score)
```

```
0.5904761904761905
```

The accuracy received on the test set is **0.5904761904761905.**

(d)

For Decision Tree Classifier

The model was evaluated using **MyEvaluationMetric** implemented in Q3.py.

```
metrics = MyEvaluationMetric()
confusion_matrix = metrics.confusion_matrix(y_test,y_pred)
accuracy_score = metrics.accuracy_score(y_test,y_pred)
pres_score= metrics.precision_score(y_test,y_pred)
recall_score= metrics.recall_score(y_test,y_pred)
f1_score = metrics.f1_score(y_test,y_pred)
print("Confusion matrix \n", confusion_matrix)
```

The results obtained are :

1. **Confusion Matrix**

```
Confusion matrix
 [[335  93]
 [251 161]]
```

2. **Other metrics**

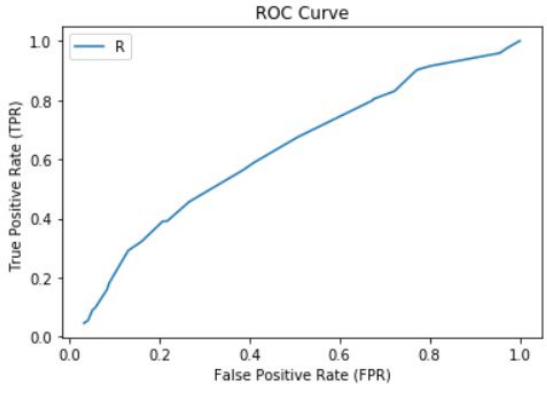
   This being a binary classification, there is no need for micro or macro averages. The results obtained on the dataset were :

```
l = [[accuracy_score,pres_score,recall_score,f1_score]]
table = tabulate(l, headers=["Accuracy Score"," Precision Score", "Recall Score", "f1 Score"], tablefmt='orgtbl')
print("\n")
print(table)
```

```
|   Accuracy Score |    Precision Score |   Recall Score |   f1 Score |
|------------------+--------------------+----------------+------------|
|         0.590476 |           0.571672 |        0.78271 |    0.66075 |
```

3. **ROC curve**

```
prob = loaded_model.predict_proba(x_test)
metrics.plot_roc_curve(y_test, prob)
```



For Gaussian Naive Bayes Classifier

The model was evaluated using **MyEvaluationMetric** implemented in Q3.py.

```
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
```

```
metrics = MyEvaluationMetric()
confusion_matrix = metrics.confusion_matrix(y_test,y_pred)
accuracy_score = metrics.accuracy_score(y_test,y_pred)
pres_score= metrics.precision_score(y_test,y_pred)
recall_score= metrics.recall_score(y_test,y_pred)
f1_score = metrics.f1_score(y_test,y_pred)
print("Confusion matrix \n", confusion_matrix)
```

The results obtained are :

1. **Confusion Matrix**

```
Confusion matrix
 [[288 140]
 [222 190]]
```

2. **Other metrics**

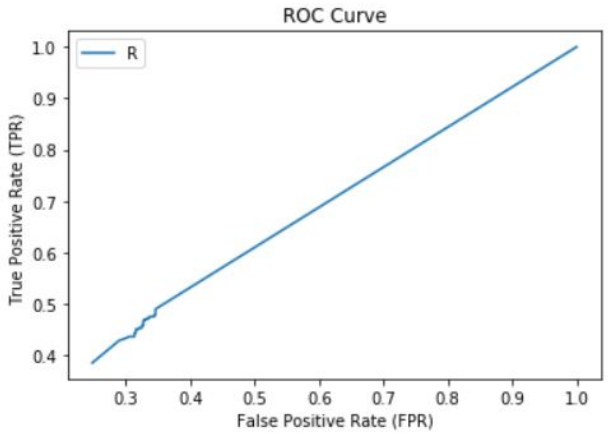
   This being a binary classification, there is no need for micro or macro averages.
   The results obtained on the dataset were :

```
l = [[accuracy_score,pres_score,recall_score,f1_score]]
table = tabulate(l, headers=["Accuracy Score"," Precision Score", "Recall Score", "f1 Score"], tablefmt='orgtbl')
print("\n")
print(table)
```

```
|   Accuracy Score |    Precision Score |   Recall Score |   f1 Score |
|------------------+--------------------+----------------+------------|
|         0.569048 |           0.564706 |       0.672897 |   0.614072 |
```

3. **ROC curve**

```
prob = clf.predict_proba(x_test)
metrics.plot_roc_curve(y_test, prob)
```



Question 4

The source code for **MyGaussianNaiveBayes** is in Q4.py.

Dataset A

The results obtained from running the **MyGaussianNaiveBayes** is :

```
nb = MyGaussianNaiveBayes()
nb.fit(X_train,y_train)
```

```
y_pred = nb.predict(X_test)
```

```
accuracy = met.accuracy_score(y_test,y_pred)
precision = met.precision_score(y_test,y_pred, average = "macro")
recall = met.recall_score(y_test,y_pred,average = "macro")
f1 = met.f1_score(y_test,y_pred, average = "macro")
l = [[ accuracy, precision, recall, f1]]
table = tabulate(l, headers=["Accuracy", "Precision ", " Recall", "f1"], tablefmt='orgtbl')
print("\n")
print(table)
```

```
|   Accuracy |   Precision  |    Recall |       f1 |
|------------+--------------+-----------+----------|
|   0.644048 |     0.699825 |  0.629772 | 0.616185 |
```

The results obtained from **sklearn Naive Bayes** is :

```
from sklearn.naive_bayes import GaussianNB
```

```
gnb = GaussianNB()
y_pred = gnb.fit(X_train, y_train).predict(X_test)
```

```
accuracy = met.accuracy_score(y_test,y_pred)
precision = met.precision_score(y_test,y_pred, average = "macro")
recall = met.recall_score(y_test,y_pred,average = "macro")
f1 = met.f1_score(y_test,y_pred, average = "macro")
l = [[ accuracy, precision, recall, f1]]
table = tabulate(l, headers=["Accuracy", "Precision ", " Recall", "f1"], tablefmt='orgtbl')
print("\n")
print(table)
```

```
|   Accuracy |   Precision  |    Recall |       f1 |
|------------+--------------+-----------+----------|
|   0.563095 |     0.615859 |  0.545691 | 0.515025 |
```

Analysis :
The custom naive Bayes obtained **higher** accuracy than sklearn. This can be due to the fact that the dataset had a lot of columns which has a variance less than $\mathbf{10^{-7}}$. These types of columns were dropped in our custom implementation as they were subject to underflow in finite precision.

Dataset B

The results obtained from running the **MyGaussianNaiveBayes** is :

```
nb = MyGaussianNaiveBayes()
nb.fit(X_train,y_train)
```

```
y_pred = nb.predict(X_test)
```

```
accuracy = met.accuracy_score(y_test,y_pred)
precision = met.precision_score(y_test,y_pred, average = "macro")
recall = met.recall_score(y_test,y_pred,average = "macro")
f1 = met.f1_score(y_test,y_pred, average = "macro")
l = [[ accuracy, precision, recall, f1]]
table = tabulate(l, headers=["Accuracy", "Precision ", " Recall", "f1"], tablefmt='orgtbl')
print("\n")
print(table)
```

```
|   Accuracy |   Precision  |    Recall |       f1 |
|------------+--------------+-----------+----------|
|   0.557143 |     0.557224 |  0.555869 | 0.553865 |
```

The results obtained from **sklearn Naive Bayes** is :

```
gnb = GaussianNB()
y_pred = gnb.fit(X_train, y_train).predict(X_test)
```

```
accuracy = met.accuracy_score(y_test,y_pred)
precision = met.precision_score(y_test,y_pred, average = "macro")
recall = met.recall_score(y_test,y_pred,average = "macro")
f1 = met.f1_score(y_test,y_pred, average = "macro")
l = [[ accuracy, precision, recall, f1]]
table = tabulate(l, headers=["Accuracy", "Precision ", " Recall", "f1"], tablefmt='orgtbl')
print("\n")
print(table)
```

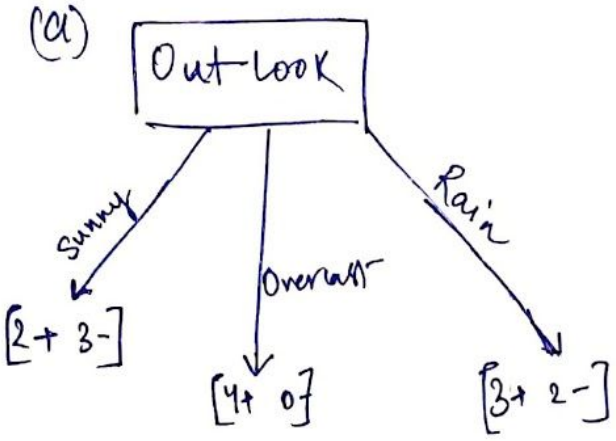
```
|   Accuracy |   Precision  |    Recall |       f1 |
|------------+--------------+-----------+----------|
|   0.557143 |     0.557224 |  0.555869 | 0.553865 |
```

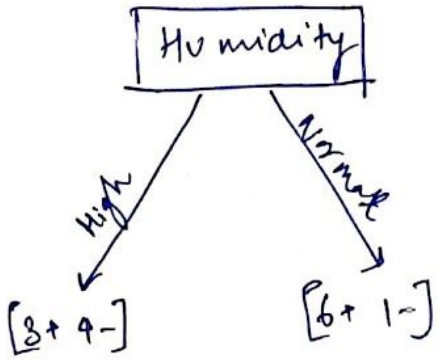
Analysis :
The custom naive Bayes obtained **similar** accuracy to sklearn.

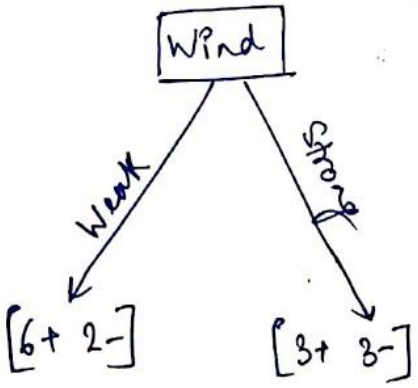
Question 5

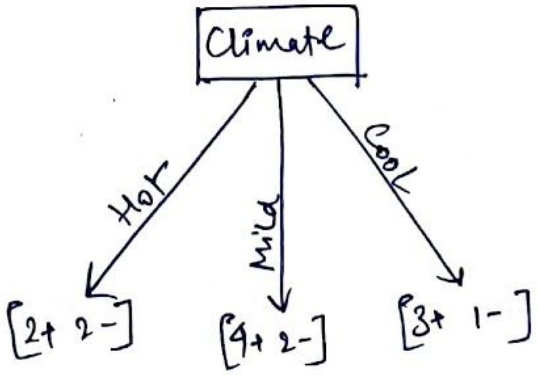
Day	Outlook	Climate	Humidity	Wind	PlayMatch
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

(a)









We define our ~~the~~ training set and test set ~~as~~ in the following manner:-

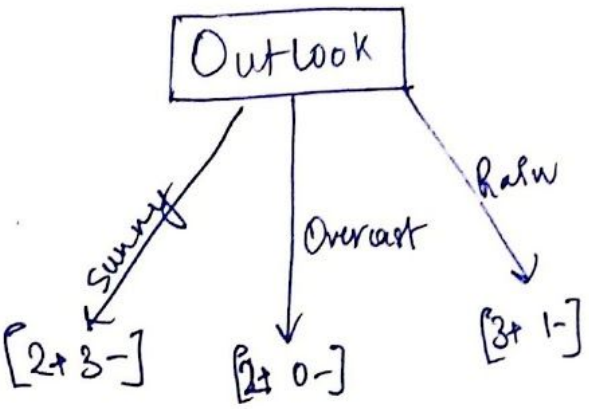
Training Set :- D1 – D11

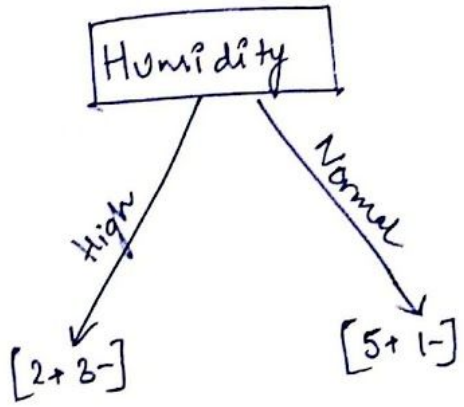
Test Set :- D12 – D14

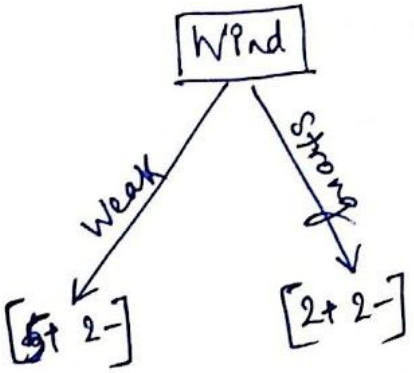
Now,

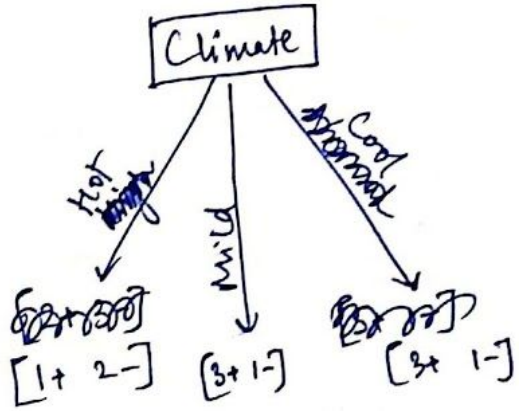
Building the Decision Tree from our Training set.

⑭ Training the Model.









→ Finding the model parameter.

① Finding the root node:

Calculating gini impurities of each feature:

a) Outlook

$$\text{Sunny} = 1 - \left(\frac{2}{5}\right)^2 - \left(\frac{3}{5}\right)^2 = 0.48$$

$$\text{Overcast} = 1 - \left(\frac{2}{2}\right)^2 - \left(\frac{0}{2}\right)^2 = 0.$$

$$\text{Rain} = 1 - \left(\frac{3}{4}\right)^2 - \left(\frac{1}{4}\right)^2 = 0.375$$

$$\text{gini} = \frac{5}{11} \times 0.48 + \frac{2}{11} \times 0 + \frac{4}{11} \times 0.375$$

$$= 0.354$$

b) Wind

$$\text{Weak} = 1 - \left(\frac{5}{7}\right)^2 - \left(\frac{2}{7}\right)^2 = 0.4081$$

$$\text{Strong} = 1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2 = 0.5$$

$$\text{gini} = \frac{7}{11} \times 0.4081 + \frac{4}{11} \times 0.5$$

$$= 0.4415$$

c) Humidity

$$\text{High} = 1 - \left(\frac{2}{5}\right)^2 - \left(\frac{3}{5}\right)^2 = 0.48$$

$$\text{Normal} = 1 - \left(\frac{5}{6}\right)^2 - \left(\frac{1}{6}\right)^2 = 0.277$$

$$\text{gini} = \frac{6}{11} \times 0.277 + \frac{5}{11} \times 0.48$$

$$= 0.369$$

d) Climate

$$\text{Hot} = 1 - \left(\frac{1}{3}\right)^2 - \left(\frac{2}{3}\right)^2 = 0.44$$

$$\text{Mild} = 1 - \left(\frac{3}{4}\right)^2 - \left(\frac{1}{4}\right)^2 = 0.375$$

$$\text{Cool} = 1 - \left(\frac{3}{4}\right)^2 - \left(\frac{1}{4}\right)^2 = 0.375$$

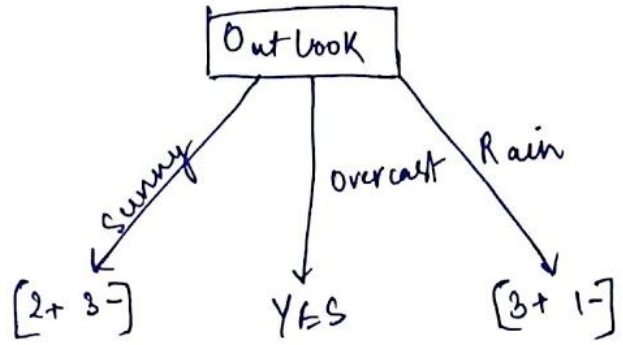
$$\text{gini} = \frac{3}{4} \times 0.44 + \frac{4}{11} \times 0.375 + \frac{4}{11} \times 0.375$$

$$= 0.393$$

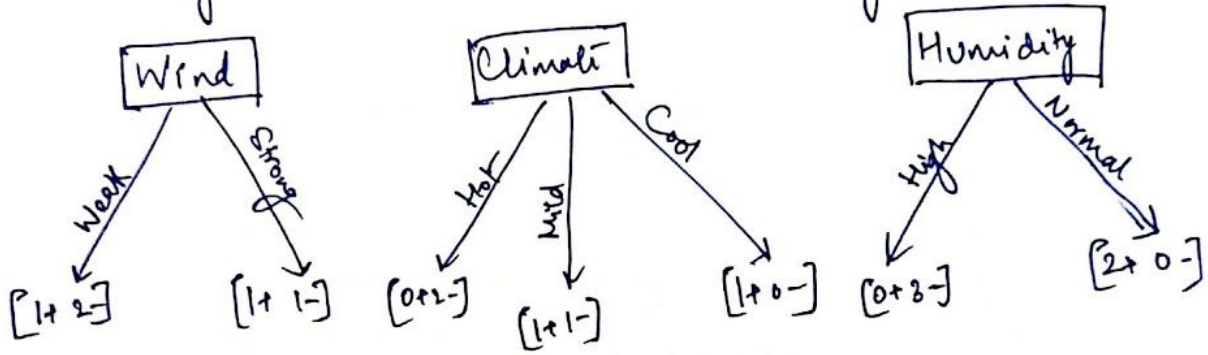
Since, Outlook has the lowest gini impurity value.

We pick Outlook as the root Node

Current Tree.



② Finding the next node in the Sunny Branch.



Calculating gini impurities

① Wind

$$\text{weak} = 1 - \left(\frac{1}{3}\right)^2 - \left(\frac{2}{3}\right)^2 = 0.44$$

$$\text{Strong} = 1 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2 = 0.5$$

$$\text{gini} = \frac{3}{5} \times 0.44 + \frac{2}{5} \times 0.5$$

$$= 0.464$$

② Climate.

$$\text{Hot} = 1 - \left(\frac{0}{2}\right)^2 - \left(\frac{2}{2}\right)^2 = 0$$

$$\text{Mild} = 1 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2 = 0.5$$

$$\text{Cool} = 1 - \left(\frac{1}{1}\right)^2 - \left(\frac{0}{1}\right)^2 = 0.$$

$$gini = \frac{2}{5} \times 0 + \frac{2}{5} \times 0.5 + \frac{1}{5} \times 0$$

$$= 0.2$$

(3) Humidity

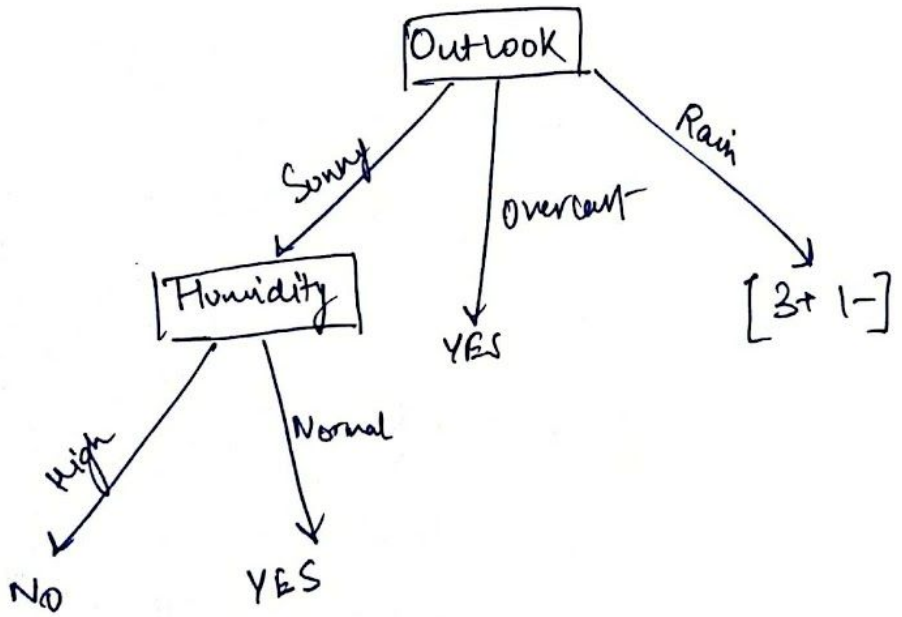
$$\text{Normal} = 1 - \left(\frac{2}{2}\right)^2 - \left(\frac{0}{2}\right)^2 = 0$$

$$\text{High} = 1 - \left(\frac{0}{3}\right)^2 - \left(\frac{3}{3}\right)^2 = 0$$

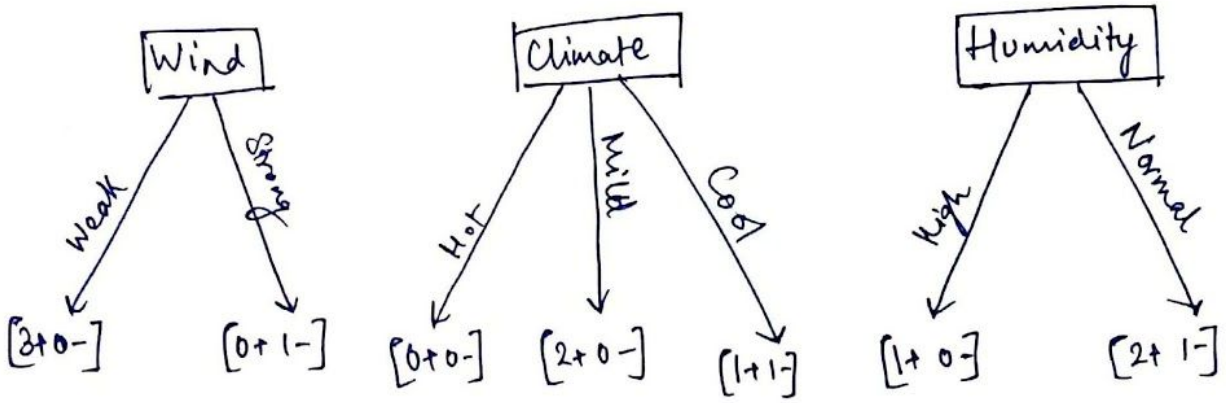
$$gini = 0.$$

Since, Humidity has the lowest gini impurity we pick Humidity as our next node

Current Tree:



(3) Finding the next node in the Rain Branch.



Calculating the gini impurities

① Wind:

$$\text{Weak} = 1 - \left(\frac{3}{3}\right)^2 - \left(\frac{0}{3}\right)^2 = 0$$

$$\text{Strong} = 1 - \left(\frac{0}{1}\right)^2 - \left(\frac{1}{1}\right)^2 = 0$$

$$\text{gini} = 0$$

② Climate:

$$\text{Hot} = 1 - \left(\frac{0}{0}\right)^2 - \left(\frac{0}{0}\right)^2 = \text{N. A.}$$

$$\text{Mild} = 1 - \left(\frac{2}{2}\right)^2 - \left(\frac{0}{2}\right)^2 = 0$$

$$\text{Cool} = 1 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2 = 0.5$$

$$\text{gini} = \frac{2}{4} \times 0 + \frac{2}{4} \times 0.5$$

$$= 0.25$$

③ Humidity:

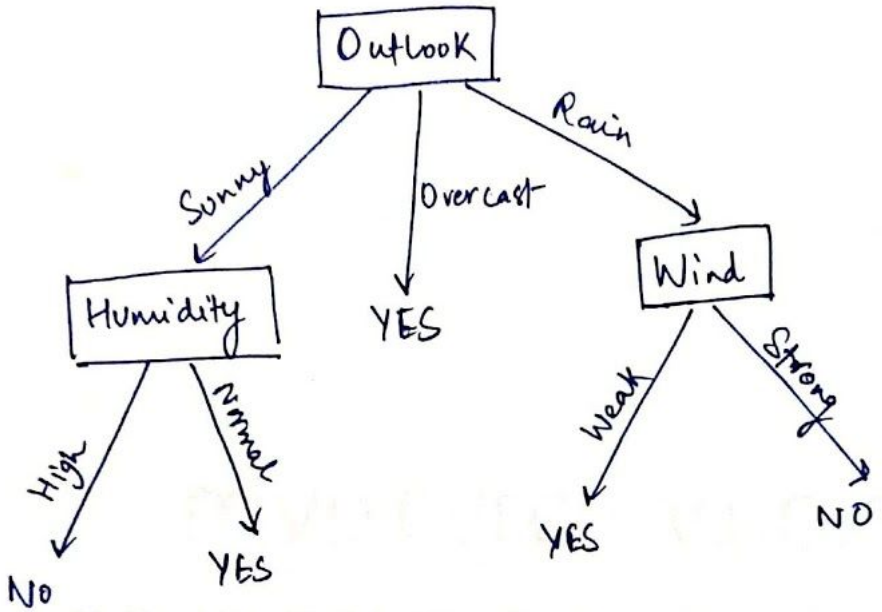
$$\text{High} = 1 - \left(\frac{1}{1}\right)^2 - \left(\frac{0}{1}\right)^2 = 0$$

$$\text{Normal} = 1 - \left(\frac{2}{3}\right)^2 - \left(\frac{1}{3}\right)^2 = 0.44$$

$$\text{gini} = \frac{3}{4} \times 0.44 + \frac{1}{4} \times 0 = 0.33$$

Since, wind has the lowest gini impurity we pick wind as our next node.

Final Decision Tree Obtained



Calculate Accuracy

#	Y_pred	Y_test
D12 - 1	YES	YES
D13 - 2	YES	YES
D-14-3	No	NO

Accuracy obtained = $\frac{2+1}{3}$ = 100 %

To include Climate in the decision Tree, we have to choose a training set that will have the gini impurity 0.

The following is the training set.

#	Outlook	Climate	Humidity	Wind	Play
D1	Sunny	Hot	High	Weak	NO
D2	Sunny	Hot	High	Strong	NO
D4	Rain	Mild	High	Weak	YES
D5	Rain	Cool	Normal	Weak	YES
D10	Rain	Mild	Normal	Weak	YES
D11	Sunny	Mild	Normal	Strong	YES
D12	Overcast	Mild	High	Strong	YES

Gini impurity of Climate :

$$\text{Hot} = 1 - \left(\frac{0}{2}\right)^2 - \left(\frac{2}{2}\right)^2 = 0$$

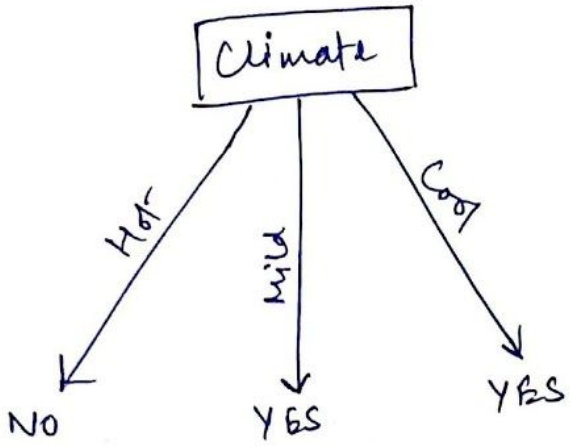
$$\text{Mild} = 1 - \left(\frac{4}{4}\right)^2 - \left(\frac{0}{4}\right)^2 = 0$$

$$\text{Cool} = 1 - \left(\frac{1}{1}\right)^2 - \left(\frac{0}{1}\right)^2 = 0$$

$\therefore$ gini = 0.

Thus gini would be 0 $\Rightarrow$ Climate would be selected as the root node.

Thus the final Decision Tree is:

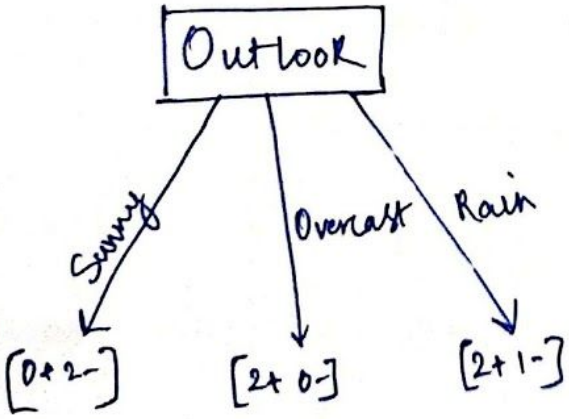


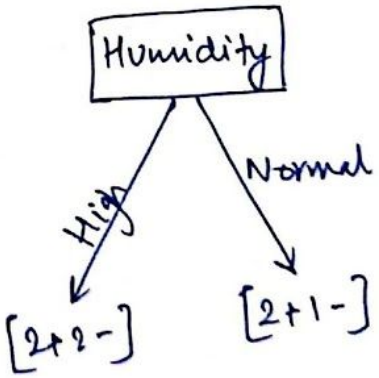
(c) The train test split is :-

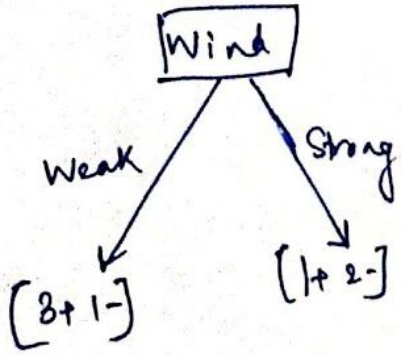
Training :- D1 – D7

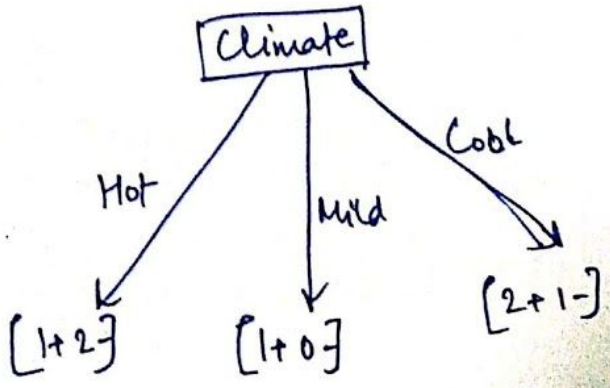
Test Set :- D8 – D14

Training Model.









(1) Finding the root Node

Calculating gini impurities.

(1) Outlook

Sunny = $1 - (\frac{2}{2})^2 - (\frac{1}{2})^2 = 0$

Overcast = $1 - (\frac{2}{2})^2 - (\frac{0}{2})^2 = 0$

Rain = $1 - (\frac{2}{3})^2 - (\frac{1}{3})^2 = 0.45$

gini = $\frac{2}{7} \times 0 + \frac{1}{7} \times 0 + \frac{3}{7} \times 0.45$

$= 0.192$

(2) Wind

Weak = $1 - (\frac{3}{4})^2 - (\frac{1}{4})^2 = 0.375$

Strong = $1 - (\frac{1}{3})^2 - (\frac{2}{3})^2 = 0.45$

gini = $\frac{4}{7} \times 0.375 + \frac{3}{7} \times 0.45$

$= 0.407$

(3) Climate

Hot = $1 - (\frac{1}{3})^2 - (\frac{2}{3})^2 = 0.45$

Mild = $1 - (\frac{1}{1})^2 - (\frac{0}{1})^2 = 0$

Cool = $1 - (\frac{1}{3})^2 - (\frac{1}{3})^2 = 0.45$

gini = $\frac{3}{7} \times 0.45 + \frac{1}{7} \times 0 + \frac{3}{7} \times 0.45$

$= 0.385$

④ Humidity

$$\text{Normal} = 1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2 = 0.5$$

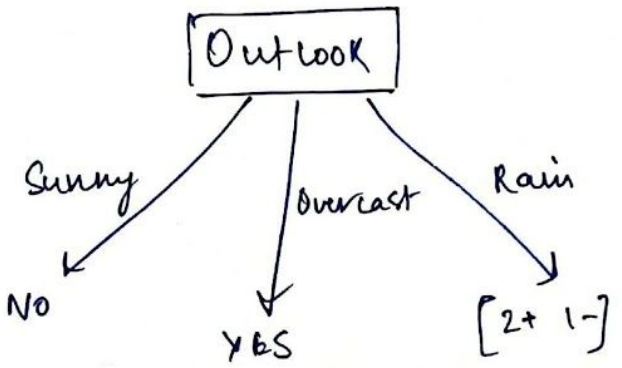
$$\text{High} = 1 - \left(\frac{2}{3}\right)^2 - \left(\frac{1}{3}\right)^2 = 0.45$$

$$\text{gini} = \frac{4}{7} \times 0.5 + \frac{3}{7} \times 0.45$$

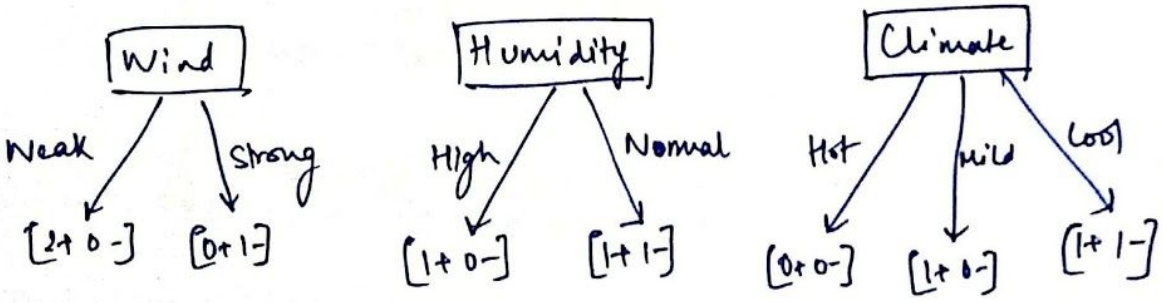
$$= 0.4785$$

Since, Outlook has the lowest gini impurity, we pick Outlook as the root node.

Current Tree:



② Next node in Rain branch.



Calculating gini Impurities

① Wind

$$\text{Weak} = 1 - \left(\frac{2}{2}\right)^2 - \left(\frac{0}{2}\right)^2 = 0$$

$$\text{Strong} = 1 - \left(\frac{1}{1}\right)^2 - \left(\frac{1}{1}\right)^2 = 0$$

$$\text{gini} = \frac{2}{3} \times 0 + \frac{1}{3} \times 0 = 0$$

② Climate

$$\text{Hot} = 1 - \left(\frac{0}{0}\right)^2 - \left(\frac{0}{0}\right)^2 = \text{N.A.}$$

$$\text{Mild} = 1 - \left(\frac{1}{1}\right)^2 - \left(\frac{0}{1}\right)^2 = 0$$

$$\text{cool} = 1 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2 = 0.5$$

$$\text{gini} = \frac{1}{3} \times 0 + \frac{2}{3} \times 0.5 = 0.33$$

③ Humidity

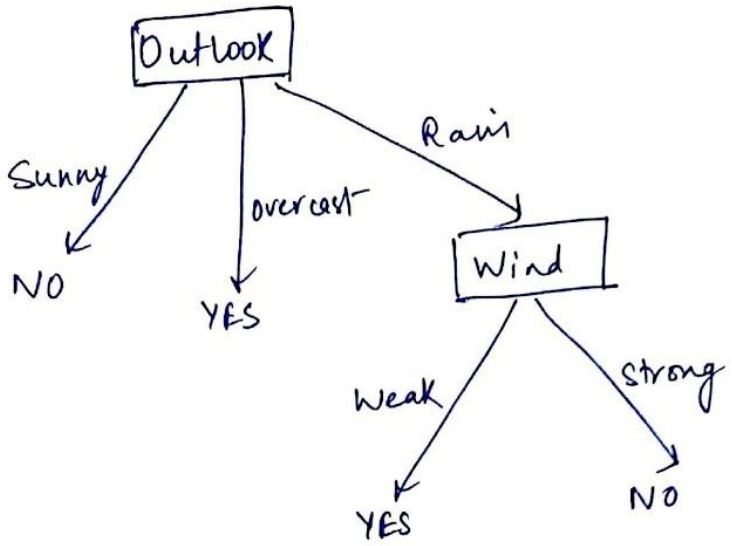
$$\text{Normal} = 1 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2 = 0.5$$

$$\text{High} = 1 - \left(\frac{1}{1}\right)^2 - \left(\frac{0}{1}\right)^2 = 0$$

$$\text{gini} = \frac{2}{3} \times 0.5 + \frac{1}{3} \times 0 = 0.33$$

Since, wind has the lowest gini impurity we pick wind as our next node.

④ Thus, the final Decision Tree is



(*) Calculating Accuracy on the Test set.

#	Y_pred	Y_test
D8	NO	NO
D9	NO	YES
D10	YES	YES
D11	NO	YES
D12	YES	YES
D13	YES	YES
D14	NO	NO

Accuracy obtained = $\frac{5}{7}$ = 71.42%

(d) To avoid overfitting of our decision Tree, some of the pruning strategies that can be applied directly on the algorithm are :-

a) Use K-Fold Cross Validation for Training. (14)

b) Set a max depth limit on the tree, as more depth means more nodes thus more overfitting.

c) We can set a minimum limit for the number of leaf nodes in the Tree.

Question 6

$P(\text{tough} \mid \text{tough}) = 0.7$ $\quad$ $P(\text{tough} \mid \text{course}) = 0.5$

$P(\text{course} \mid \text{tough}) = 0.3$ $\quad$ $P(\text{course} \mid \text{course}) = 0.5$

We ~~set~~ have to find $P(w3 \mid w1, w2, w4)$

Using Markov Assumption, we can drop w1,

$\therefore$

$$P(w3 \mid w2, w4) = \frac{P(w3, w2, w4)}{P(w2, w4)}$$

$$\Rightarrow \frac{P(w4 \mid w2, w3) * P(w3 \mid w2)}{P(w4 \mid w2)}$$

$$\Rightarrow \frac{P(w4 \mid w3) * P(w3 \mid w2)}{P(w4 \mid w2)}$$

{As we can drop w2 from $P(w4 \mid w2, w3)$}

$$\Rightarrow \frac{P(W4|W3) \times P(W3|W2)}{\sum_{W_3} P(W4|W3) \times P(W3|W2)} \quad \text{---(1)}$$

$W3 =$ "tough" or $W3 =$ "course" $\therefore$ (1) becomes.

$$\Rightarrow \frac{P(\text{course}|W3) \times P(W3|\text{course})}{P(\text{course}|\text{course}) \times P(\text{course}|\text{course}) + P(\text{course}|\text{tough}) \times P(\text{tough}|\text{course})}$$

$$\Rightarrow \frac{P(\text{course}|W3) + P(W3|\text{course})}{0.5 \times 0.5 + 0.3 \times 0.5}$$

$$\Rightarrow \frac{P(\text{course}|W3) + P(W3|\text{course})}{0.4}$$

Therefore, the posterior probability that $W3 =$ "course" or $W3 =$ "tough" is :-

$W3 =$ "tough" :- $\frac{0.3 \times 0.5}{0.4} = \frac{0.15}{0.4} = 0.375$

$W3 =$ "course" :- $\frac{0.5 \times 0.5}{0.4} = \frac{0.25}{0.4} = 0.625$.

is the required Answers

Question 8

$X = \langle X_1, X_2, \ldots, X_n \rangle$, X_i are boolean variables.

and $P(Y=1) = \pi$

To show:- $P(Y=1 \mid X) = \dfrac{1}{\left(1 + \exp\left(w_0 + \sum_{i=1}^{n} w_i X_i\right)\right)}$

Solution:

$$P(Y=1 \mid X) = \frac{P(Y=1, X)}{P(X)} \quad \text{---(1)}$$

$\theta_{i1} = P(X_i=1 \mid Y=1)$ and $\theta_{i0} = P(X_i=1 \mid Y=0)$ ⟶ (2)

Equation (1) can be written as.

$$P(Y=1 \mid X) = \frac{P(Y=1, X)}{P(X, Y=1) + P(X, Y=0)}$$

$$P(Y=1 \mid X) = \frac{P(Y=1, X)}{P(Y=1)\,P(X \mid Y=1) + P(Y=0) \cdot P(X \mid Y=0)}$$

$$P(Y=1 \mid X) = \frac{P(Y=1)\,P(X \mid Y=1)}{P(Y=1)\,P(X \mid Y=1) + P(Y=0) P(X \mid Y=0)}$$

$$P(Y=1|X) = \frac{1}{1 + \frac{P(Y=0)\,P(X|Y=0)}{P(Y=1)\,P(X|Y=1)}}$$

$$= \frac{1}{1 + \left(\frac{1-\pi}{\pi}\right) \cdot \frac{P(X|Y=0)}{P(X|Y=1)}}$$

$$= \frac{1}{1 + \underbrace{\left(\frac{1-\pi}{\pi}\right) \cdot \frac{P(X_1, X_2, \ldots X_n \mid Y=0)}{P(X_1, X_2, \ldots X_n \mid Y=1)}}_{T}}$$

Taking exp and ln of Term T.

$$P(Y=1|X) = \frac{1}{1 + \exp\left(\ln\left(\frac{1-\pi}{\pi}\right) + \ln\left(\frac{P(X_1, X_2, \ldots X_n \mid Y=0)}{P(X_1, X_2, \ldots X_n \mid Y=1)}\right)\right)}$$

$$= \frac{1}{1 + \exp\left(\ln\left(\frac{1-\pi}{\pi}\right) + \sum_i \ln\left(\frac{P(X_i|Y=0)}{P(X_i|Y=1)}\right)\right)} \quad -(3)$$

Substituting (2) in (3), we get.

$$P(Y=1|X) = \left[1 + \exp\left(\ln\left(\frac{1-\pi}{\pi}\right) + \sum_i \ln\left(\frac{\theta_{i0}^{X_i}\,(1-\theta_{i0})^{(1-X_i)}}{\theta_{i1}^{X_i}\,(1-\theta_{i1})^{(1-X_i)}}\right)\right)\right]^{-1}$$

(18)

$$P(Y=1|X) = \left[1 + \exp\left(\ln\left(\frac{1-\pi}{\pi}\right) + \sum_i \left(x_i \ln\left(\frac{\theta_{i0}}{\theta_{i1}}\right) + (1-x_i)\ln\left(\frac{(1-\theta_{i0})}{1-\theta_{i1}}\right)\right)\right)\right]^{-1}$$

$$P(Y=1|X) = 1 + \exp\left(\ln\left(\frac{1-\pi}{\pi}\right) + \sum_i \left(x_i \ln\left(\frac{\theta_{i0}}{\theta_{i1}}\frac{(1-\theta_{i1})}{(1-\theta_{i0})}\right) + \ln\left(\frac{1-\theta_{i0}}{1-\theta_{i1}}\right)\right)\right)^{-1}$$

Thus, this can be written in the form of:

$$P(Y=1|X) = \frac{1}{1+\exp\left(w_0 + \sum_i w_i x_i\right)}.$$

Hence, proved.

Question 7

(a) Advantages of Decision Trees over Logistic Regression are :

(i) Decision Trees are extremely easy to compute and require less computational power as compared to Logistic Regression.

(ii) Decision Tree models don't assume the features to be independent, while Logistic Regression assume the features to be independent.

(iii) Decision tree models are very easy to visualize as compared to Logistic Regression.

(b) Biggest weakness of Decision tree over Logistic Regression are :

(i) Decision tree trees are very likely to overfit the data while logistic regression assigns weights to each feature which are less likely to get overfitted as it uses gradient descent to find the optimal parameters.

(ii) Decision trees are very sensitive to change in training data, a small change in the data and drastically change the tree.

(c) SInce the data can be classified using a regression classifier, the dataset is linearly separable. Thus we can split the dataset on x1 and can assume a threshold on each of x1 and x2. Thus the tree would be of depth log(n).

(d) Now since the data is not linearly separable we cannot assume a cut off for both x1 and x2. Hence we need to split for x1 and x2 separately. Thus the tree would be of depth 2*log(n).