Supervised Learning HW3

Team Members:

Adiesha Liyanage, Kaveen Liyanage, Siddat Nesar

A. HOW TO RUN THE SYSTEM

Decision tree: The decision tree algorithm instance can be run using the following command. It will run the decision tree algorithm on the shuttle training data set and will predict the labels for the shuttle test data set.

Command: python decisiontree.py

KNN: The KNN algorithm can be run using the following command.

Command: python knn.py

This script will generate the test results for the three different datasets (iris, satellite, and shuttle). For the iris dataset, there is no separate test-set. So, the script takes the total iris dataset and splits it into 80% as the training set and the remaining 20% as the testing set. This is done randomly. For the satellite and shuttle dataset, there are separate testing sets, which have been used to validate the algorithm. Several iterations have been performed to find the best value of k (number of nearest neighbors) for different datasets. This k value has been incorporated in the script.

K-Fold: Here we have implemented the K-fold algorithm to get the average performances with different sections of the training data to evaluate. By doing so, model overfitting problems can be mitigated by testing different validation sets. The K-fold algorithm will first randomize the training dataset to get rid of any ordered arrangement, Then the data will be divided into approximately equal *K* portions (folds). The *K* value is the user given and usually in the range of *[5,10].* In the i^{th} iteration, the i^{th} portion will be reserved as the validation set and the model will be trained on the remaining datasets. For each iteration, the performance will be measured and finally, the average and the variance will be calculated of all the performances in iterations. The *KFold.py* contains the script to the K-fold cross-validation algorithm.

[$mean(\theta),\ var(\theta)$ **] = crossValidation(data.copy(), K, Model)**

The inputs of the system are the *data* which is in the form of a pandas data frame object with the last column being the label information. *K,* is the number of folds used for cross-validation. *Model,* is an object that belongs to the class *ModelMethod* which contains the classifier model name and the required parameters. In this implementation, only *KNN* and *Decision Tree* methods are implemented but can be expanded to other methods to be included also. The *crossValidation()* prints the average precisions, recall, and F-measure values. Also, it returns the average and the variance of the F-measure values. The *main()* method contains an example usage of the functions.

Performances: We have implemented several basic contingency tables based on supervised performances to evaluate classifier models. Accuracy/Precision, Coverage/Recall, and F-measure. These are implemented in the script *Performances.py*.

Accuracy/Precision: First, the class-specific accuracy is calculated, which represents the fraction of correct predictions over, all points predicted to be in the specified class. Then the precision is calculated by taking the average of class-specific accuracies.

prec_i = precision(result, gTruthCol, predCol, listofclusters)

Coverage/Recall: The class-specific recall is calculated, which represents the fraction of correct predictions over, all points in a specific class.

rec_i = recall(result, gTruthCol, predCol, listofclusters)

F-measure: Tradeoff between precision and recall. First, the class-specific F measure is calculated and then the average is taken for the overall F-measure.

F_i = F_measure(result, gTruthCol, predCol, listofclusters)

The inputs of the system are, the *results* which is in the form of pandas data frame object which contains at least two columns containing ground truth labels and the predications. *gTruthCol* is the column index of the ground truth column and the *predCol* is the column index of the predictions column. *Listofclusters* is the list of unique labels of the dataset. The *main()* method contains an example usage of the functions.

B. EXAMPLE INPUT AND OUTPUT

An example output is shown below after running the *knn.py* algorithm over the shuttle dataset.

```
KNN with Shuttle Dataset
Example Test with 4 nearest neighbors :
True Label : 1
Predicted Label : 1
Accuracy :  0.9972413793103448
Task Complete
```

Example output is shown below after running *decisiontree.py* over the Iris data set

```
(base) C:\Users\KLenovo\PycharmProjects\SupervisedLearning>python decisiontree.py
decision tree
       0    1    2    3              4
0    5.1  3.5  1.4  0.2     Iris-setosa
1    4.9  3.0  1.4  0.2     Iris-setosa
2    4.7  3.2  1.3  0.2     Iris-setosa
3    4.6  3.1  1.5  0.2     Iris-setosa
4    5.0  3.6  1.4  0.2     Iris-setosa
..   ...  ...  ...  ...             ...
137  6.4  3.1  5.5  1.8  Iris-virginica
138  6.0  3.0  4.8  1.8  Iris-virginica
139  6.9  3.1  5.4  2.1  Iris-virginica
140  6.7  3.1  5.6  2.4  Iris-virginica
141  6.9  3.1  5.1  2.3  Iris-virginica

[142 rows x 5 columns]
Best v:  5.55  best score:  0.5483208280445977  attribute:  0
Best v:  3.3499999999999996  best score:  0.2797646108168461  attribute:  1
Best v:  2.45  best score:  0.935940714955517  attribute:  2
Best v:  0.8  best score:  0.935940714955517  attribute:  3
Best v:  6.25  best score:  0.1735805289582517  attribute:  0
Best v:  2.45  best score:  0.0522565519120457  attribute:  1
Best v:  4.75  best score:  0.6378575191502841  attribute:  2
Best v:  1.75  best score:  0.660788972265985  attribute:  3
--- 0.9255270957946777 seconds ---
<node.Node object at 0x0000019A91666100>
Iris-virginica
      0    1    2    3              4          result
0   5.8  2.7  5.1  1.9  Iris-virginica  Iris-virginica
1   6.8  3.2  5.9  2.3  Iris-virginica  Iris-virginica
2   6.7  3.3  5.7  2.5  Iris-virginica  Iris-virginica
3   6.7  3.0  5.2  2.3  Iris-virginica  Iris-virginica
4   6.3  2.5  5.0  1.9  Iris-virginica  Iris-virginica
5   6.5  3.0  5.2  2.0  Iris-virginica  Iris-virginica
6   6.2  3.4  5.4  2.3  Iris-virginica  Iris-virginica
7   5.9  3.0  5.1  1.8  Iris-virginica  Iris-virginica
8   5.0  3.5  1.3  0.3     Iris-setosa     Iris-setosa
9   4.5  2.3  1.3  0.3     Iris-setosa     Iris-setosa
10  4.4  3.2  1.3  0.2     Iris-setosa     Iris-setosa
11  6.0  3.4  4.5  1.6  Iris-versicolor  Iris-versicolor
```

An example output is shown after running the *kFold.py* on the Iris dataset

```
K-Fold cross validation
Data = iris
Classification method = KNN
<__main__.ModelMethod object at 0x000001AF1A85A4C0>
3-fold cross validation on 142 data points
From 0.0, to 46.0 as test set
Class-specific F measure
[1.         0.90909091 0.9375    ]
Starting Precision calc
Class-specific precision
[1.         1.         0.88235294]
Overall precision
0.9565217391304348
Starting Recall calc
Class-specific recall
[1.         0.83333333 1.        ]
From 47.0, to 93.0 as test set
```

: : : :

```
**********-FINAL-*********
Class labels
['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
Performance measure in each fold
[0.94886364 0.95520422 0.95008913]
Mean performance measure = 0.9513856596657108
Variance performance measure = 0.9513856596657108
Precision =
[[1.         1.         0.88235294]
 [1.         0.95652174 0.90909091]
 [1.         1.         0.88888889]]
Mean precision = 0.9596504975874116
Variance Precision = 0.002409843370648958
Recall =
[[1.         0.83333333 1.        ]
 [1.         0.95652174 0.90909091]
 [1.         0.83333333 1.        ]]
Mean Recall = 0.9480310349875568
Variance Recall = 0.004600379734837099
```

C. SSH URI OF THE GIT REPO

Ssh of our project: **git@github.com:adiesha/SupervisedLearning.git**

D. DISCUSSION ABOUT THE PROJECT

In this project, we have implemented a decision tree algorithm and K-nearest neighbor algorithm as the supervised learning algorithms. And we used several data sets to test those algorithms.

I. Iris data set

Ii. Shuttle data set

Satellite data set contains shuttle log data and it contains 43500 data points with 9 different numerical attributes.

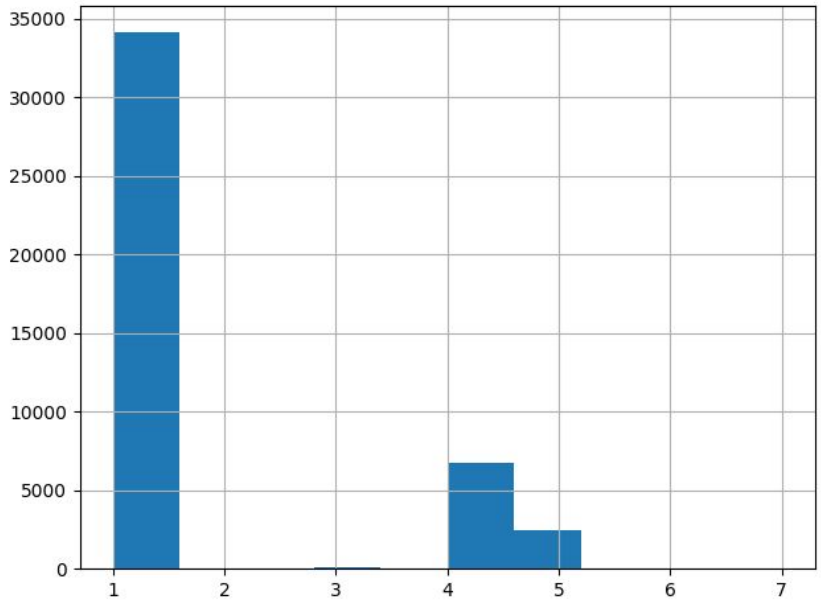
Iii. satellite data set

The database consists of the multi-spectral values of pixels in 3x3 neighborhoods in a satellite image and the classification associated with the central pixel in each neighborhood. This data set contains 36 data attributes.

Then we used k-fold validation measures to test the efficiency of our methods on different data sets. And the k-fold method uses f-measure as the underlying performance measure metric.

Decision tree algorithm results

Shuttle histogram



As you can see before running the supervised learning algorithms we created the histogram of the frequency of classes. As you can see classes 2, 3, 6, 7 have the least frequency. We can consider data points with these classes as the outliers. Table 1 has the performance measures when we ran K-Fold validation (K value is 5) and we ran 5-Fold validations on 3 different neta values while keeping the phi value constant (0.85). F-measure is the average F-Measure of each class. We did not weigh the class frequency when calculating the F-measure, for that we used the accuracy measure Table 2. As you can see classes 1, 4, 5 are predicted really well

using the decision tree algorithm (Accuracy is close to 1) but classes 2,3,6,7 are not predicted well (Accuracy is closer to 0). That is why we get F-measures around 0.42 for the shuttle data set. But we get really good accuracy values for the decision tree algorithm. What we can conclude from this is that the decision tree is not a good algorithm to predict noise points, because in a data set we have less number of outliers/noise points generally. Thus in order to predict these classes correctly, we have to input small neta values when we train because we want to isolate noise data points into leaf nodes. Thus we want to input pphi values closer to 1 and neta values closer to the frequency of noise points. But when we input smaller neta values it takes a longer time to build the decision tree which is not feasible when we have a large data set because it would consume large memory chunks and time. In order to predict the normal classes with good accuracy, we can input neta values to be around the least frequent noise class values. For example for the shuttle data set if the neta value is around 1800 which is around the frequency of least frequent **normal class** (class 5). It will give a fairly good accuracy for the normal classes.

neta	400	500	600
F_measure	0.4263141340744763	0.5120273132240385	0.5405924072021409
Precision	0.4241047002321803	0.42410408293888435	0.42409172417947655
Recall	0.42857142857142855	0.42857142857142855	0.42857142857142855

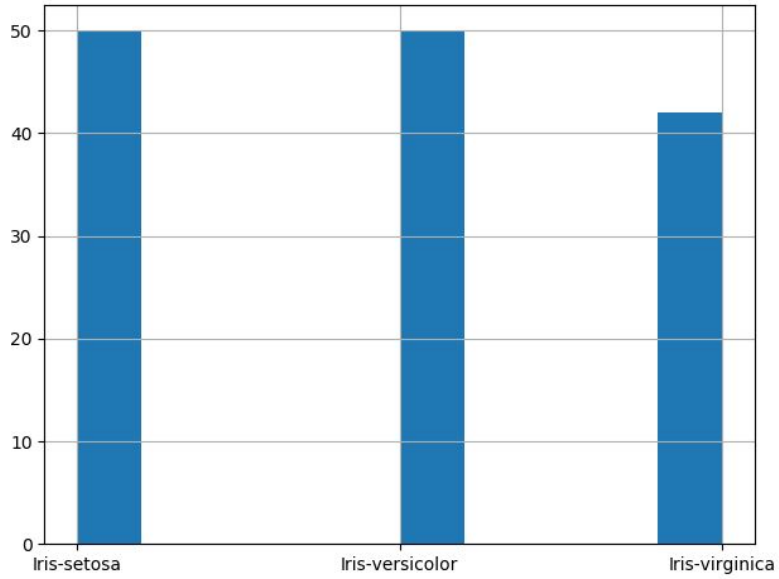
Table 1

Class-specific average accuracies with Shuttle data, 5-fold cross-validation DTree

Class	400	500	600
1	1	1	1
2	0	0	0
3	0	0	0
4	0.9756	0.9756	0.9757
5	0.9931	0.9931	0.9947
6	0	0	0
7	0	0	0

Table 2

Iris data set results



If you look at the histogram of the iris data set we can see it has an even frequency of classes. And as the 5-Fold result, we get really good F-measure and accuracy values. This is because unlike the shuttle data set frequency of classes are similar and it helps the algorithm to predict better. Table 4 contains class-specific precision values. Iris-setosa classes can be predicted with 100% precision while versicolor and virginica classes can be predicted with precision around 90% precision.

5- fold cross-validation results on Iris data

	Dtree	KNN
F_measure	0.916873159047072	0.9667349970291147
Precision	0.9311457061457061	0.9661616161616161
Recall	0.9201322751322751	0.96986531986532

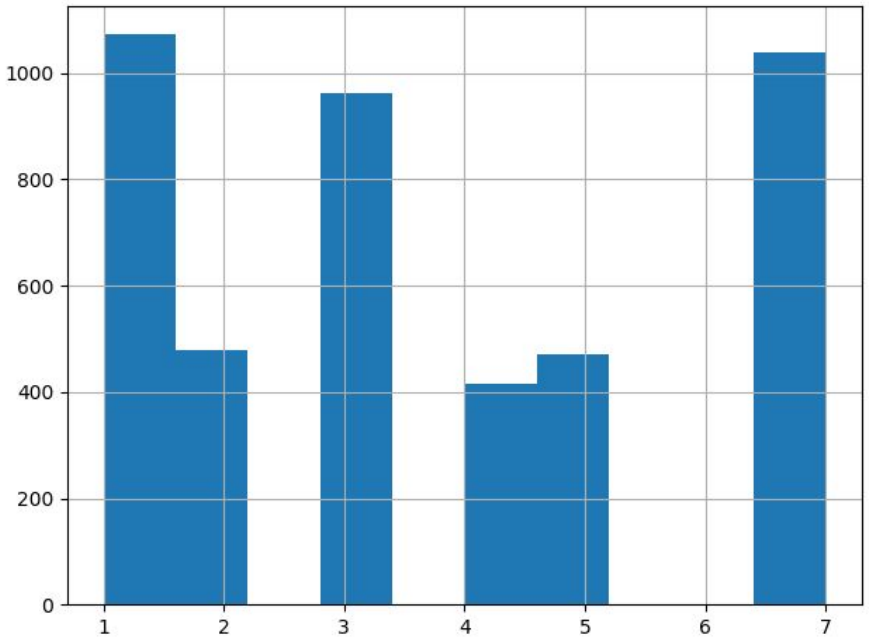
Table 3

Class-specific accuracy Decision Tree algorithm on Iris dataset

Dtree	Iris-setosa	Iris-versicolor	Iris-virginica
Precision	1	0.9073	0.8861

Table 4

Satellite data set results



Histogram of the satellite data set

This data set contains 36 attributes. And only 1 noise point class. This data set takes a long time to run compared to other data sets because it contains a lot of data attributes.

Satellite result

We tested the satellite data set on two parameter settings. Neta 200 and phi 0.8 and Neta 150 and phi 0.9

Neta: 200 phi: 0.8
Class-specific F measure
[0.57381258 0.93735499 0.85649718 0.23826715 0.69189189 0.0125261]
Class-specific precision
[0.40747493 0.97584541 0.77663934 0.5 0.96240602 0.33333333]
Overall precision
0.596

Neta 250 , phi: 0.9
Error Rate = 0.4095
Accuracy = 0.5905

Class-specific precision
[0.40747493 0.97584541 0.77663934 0.55 0.96610169 0.2]

Overall precision
0.5905

Class-specific recall
[0.96963124 0.90178571 0.95465995 0.1563981 0.48101266 0.01276596]
Class-specific F measure
[0.57381258 0.93735499 0.85649718 0.24354244 0.64225352 0.024   ]

As you can see for high frequency normal classes like 1,2,3 it gives good recall values and fairly okay F-measure values but for low frequency classes it doesn't give good recall and F-measure values. It is harder to tweak the parameters because it takes a long time to run this data set since it has 36 attributes. My guess is when a data set contains a large number of attributes the performance of the decision tree algorithm decreases. It becomes harder to find good splitting attributes. Maybe it is not optimal to split the data set using the highest gain attribute in high dimensional data, maybe the data set has a different structure that does not follow the highest gain dimension is the best split value attribute assumption. Although we have fairly good results for this data set. Since it takes a long time to run this we did not use the k-Fold validation method. What we did was we ran the training data set to create the decision tree and then used the test data set to predict the label and calculate the performance measures.

KNN algorithm result

The accuracy with different k values for the different datasets are as follows:

Data | K	K = 2	K =3	K = 4	K = 5	K = 6
Iris Data	Acc = 0.9655 F = 0.9602	Acc = 0.9655 F = 0.9602	Acc = 0.9655 F = 0.9602	Acc = 0.931 F = 0.9212	Acc = 0.8965 F = 0.8824
Satellite Data	Acc = 0.8855 F = 0.8727	Acc = 0.872 F = 0.8576	Acc = 0.854 F = 0.8389	Acc = 0.8415 F = 0.8264	Acc = 0.828 F = 0.8113
Shuttle Data	-	-	Acc = 0.9972 F = 0.8775	-	-

Table 5

The number of points for the nearest neighbors plays a vital role in the classification accuracy of knn algorithms. For the iris dataset, it has been observed that the accuracy remains the maximum for k value of 2 to 4, but it drops after that. Taking 5 or more nearest neighbors results to get points from other clusters, which drops the overall accuracy. For the satellite dataset also, the same characteristics have been observed. K values of 2 give the maximum accuracy of 0.8855 and drops with the increase of k value. It can be noted that this dataset contains 36 features, and the points are nearby, which is a good reason to drop the overall classification accuracy using this algorithm. The shuttle dataset is too big in size for this kind of lazy learning algorithm. But, with a k-value of 4, it gives an overall accuracy of 0.9972. Due to time constraints, this dataset was not tested with other k values. The F-measures also exhibit a

similar property. To run the F-measure for all of this dataset with different k values, the following command can be used.

Command: python test.py

Class-specific accuracy KNN on Iris dataset with 5-fold cross-validation

KNN	Iris-setosa	Iris-versicolor	Iris-virginica
Precision	1	0.9750	0.9235