University of Tehran
College of Engineering
School of Electrical and Computer Engineering

Intelligent Systems

Dr.Hosseini

Homework 2

Soroush Mesforush Mashhad

SN:810198472

Azar 01

Contents

Abstract

In this assignment, we shall expand our knowledge about intelligent systems in three different sections.

In the first section we shall perform analysis on a decision tree and obtain the decision tree for a given dataset using our knowledge obtained in the course, then we shall continue to construct another tree using an ad-hoc method.

In the second section we shall implement a decision tree using the **ID3 Algorithm** from scratch in python, we shall then proceed to implement the **Random Forest Algorithm** to increase its accuracy.

In the final section we shall become familiar with metric learning methods such as **KNN**, **LMNN**, **LFDA** and **GMML** methods and their implementation.

1 Question 1 : Decision Tree

1.1 Part A : Classifier Design

Here we shall design a **Decision Tree Classifier** to confirm what breed an animal is, we shall do thsi based on **Information Gain** and with the **ID3** algorithm.

شماره	رنگ	تعداد پا	قد	محل زندگی	جاندار
۱	قهوه‌ای	2	بلند	خشکی	A
۲	قهوه‌ای	3	کوتاه	خشکی	B
۳	سبز	2	بلند	آب	B
۴	سبز	3	بلند	آب	B
۵	قهوه‌ای	2	کوتاه	آب	A
۶	قهوه‌ای	2	بلند	آب	A
۷	قهوه‌ای	2	کوتاه	خشکی	B
۸	سبز	2	کوتاه	آب	A
۹	سبز	3	بلند	آب	B
۱۰	قهوه‌ای	2	بلند	خشکی	A

Figure 1: Information needed for classification

We denote the following formulas which shall be used very frequently.

$$Entropy(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$
$$I(A) = \sum_{V \in Values(A)} \frac{|S_v|}{|S|} Entropy(S)$$

Where S is the sample of training samples, p_+ is the proportion of positive examples and p_- is the proportion of negative examples, now we shall write another formatting of the depicted formualae for brevity.

$$Entropy(S) = -\frac{p}{p+n} \log_2 \left(\frac{p}{p+n}\right) - \frac{n}{n+p} \log_2 \left(\frac{n}{p+n}\right)$$
$$I(A) = \sum_{V \in Values(A)} \frac{p_i + n_i}{p+n} Entropy(A)$$

Now we shall define a concept known as **Information Gain** which is the difference in entropy before and after splitting the dataset on attribute A.

$$Gain(S, A) = Entropy(S) - I(A)$$

Step 1

Here we shall calculate the entropy for the entire dataset.

$$p = 5, \quad n = 5, \quad Tot = n + p = 10$$
$$Entropy(S) = -\frac{p}{p+n}\log_2\left(\frac{p}{p+n}\right) - \frac{n}{n+p}\log_2\left(\frac{n}{p+n}\right)$$
$$= -\frac{5}{10}\log_2\left(\frac{5}{10}\right) - \frac{5}{10}\log_2\left(\frac{5}{10}\right) = 1 \longrightarrow Entropy(S) = 1$$

Step 2

Here we shall calculate the entropy for each and every attribute.

Color

We calculate the entropy and information for each of the colors.

Brown

$$p_b = 4, \quad n_b = 2, \quad Entropy(Brown) = -\frac{4}{6}\log_2\left(\frac{4}{6}\right) - \frac{2}{6}\log_2\left(\frac{2}{6}\right) = 0.9183$$

Green

$$p_g = 1, \quad n_g = 3, \quad Entropy(Brown) = -\frac{1}{4}\log_2\left(\frac{1}{4}\right) - \frac{3}{4}\log_2\left(\frac{3}{4}\right) = 0.8113$$

Now we go for the information.

$$I(Color) = \frac{p_b + n_b}{p+n} Entropy\left(Color = Brown\right) + \frac{p_g + n_g}{p+n} Entropy\left(Color = Green\right)$$
$$I(Color) = \frac{6}{10} \times 0.9183 + \frac{4}{10} \times 0.8133 = 0.8763$$

Now the information gain is as follows.

$$Gain(Color) = Entropy(S) - I(Color) = 1 - 0.8763 = 0.1237$$

Number of legs

We calculate the entropy and information for each leg number variation.

Two

$$p_{two} = 5, \quad n_{two} = 2, \quad Entropy(Brown) = -\frac{5}{7}\log_2\left(\frac{5}{7}\right) - \frac{2}{7}\log_2\left(\frac{2}{7}\right) = 0.8631$$

Three

$$p_{three} = 0, \quad n_{three} = 3, \quad Entropy(Brown) = -\frac{0}{3}\log_2\left(\frac{0}{3}\right) - \frac{3}{3}\log_2\left(\frac{3}{3}\right) = 0$$

Now we go for the information.

$$I(NLegs) = \frac{p_{two} + n_{two}}{p+n} Entropy\left(NLegs = Two\right) + \frac{p_{three} + n_{three}}{p+n} Entropy\left(NLegs = Three\right)$$
$$I(NLegs) = \frac{7}{10} \times 0.8631 + 0 = 0.6042$$

Now the information gain is as follows.

$$Gain(Nlegs) = Entropy(S) - I(Nlegs) = 1 - 0.6042 = 0.3958$$

Height

We calculate the entropy and information for each height variation.

Tall

$$p_t = 3, \quad n_t = 3, \quad Entropy(Brown) = -\frac{3}{6}\log_2\left(\frac{3}{6}\right) - \frac{3}{6}\log_2\left(\frac{3}{6}\right) = 1$$

Short

$$p_s = 2, \quad n_s = 2, \quad Entropy(Brown) = -\frac{2}{4}\log_2\left(\frac{2}{4}\right) - \frac{2}{4}\log_2\left(\frac{2}{4}\right) = 1$$

Now we go for the information.

$$I(Height) = \frac{p_t + n_t}{p+n} Entropy\left(Height = Tall\right) + \frac{p_s + n_s}{p+n} Entropy\left(Height = Short\right)$$
$$I(Height) = \frac{6}{10} + \frac{4}{10} = 1$$

Now the information gain is as follows.

$$Gain(Height) = Entropy(S) - I(Nlegs) = 1 - 1 = 0$$

Habitat

We calculate the entropy and information for each habitat variation.

Land

$$p_l = 2, \quad n_l = 2, \quad Entropy(Land) = -\frac{2}{4}\log_2\left(\frac{2}{4}\right) - \frac{2}{4}\log_2\left(\frac{2}{4}\right) = 1$$

Water

$$p_w = 2, \quad n_w = 2, \quad Entropy(Water) = -\frac{3}{6}\log_2\left(\frac{3}{6}\right) - \frac{3}{6}\log_2\left(\frac{3}{6}\right) = 1$$

Now we go for the information.

$$I(Habitat) = \frac{p_l + n_l}{p+n} Entropy\left(Habitat = Land\right) + \frac{p_w + n_w}{p+n} Entropy\left(Habitat = Water\right)$$
$$I(Habitat) = \frac{6}{10} + \frac{4}{10} = 1$$

Now the information gain is as follows.

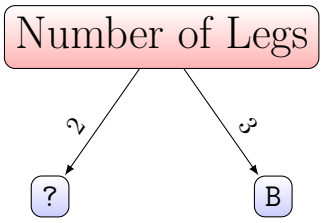
$$Gain(Habitat) = Entropy(S) - I(Habitat) = 1 - 1 = 0$$

Step 3

Now we go on to choose the root node by comparing the gains.

$$Gain(Color) = 0.1237, \quad Gain(Nlegs) = 0.3958,$$
$$Gain(Height) = Gain(Habitat) = 0$$

So we choose the **Number of Legs** attribute as the root node of the decision tree.



It is obvious that we need more sub-trees hence we move on to step 4.

Step 4

We take the scenario where the number of legs are 2, so we must repeat what we did to find the root node to find the other node.

$$p = 5, \quad n = 2, \quad Tot = p + n = 7$$

$$Entropy(S') = -\frac{p}{p+n}\log_2\left(\frac{p}{p+n}\right) - \frac{n}{n+p}\log_2\left(\frac{n}{p+n}\right)$$

$$= -\frac{5}{7}\log_2\left(\frac{5}{7}\right) - \frac{2}{7}\log_2\left(\frac{2}{7}\right) = 0.8631 \longrightarrow Entropy(S') = 0.8631$$

Now we must calculate the **Information Gain** for each of the attributes.

Color

We calculate the entropy and information for each of the colors.

Brown

$$p_b = 4, \quad n_b = 1, \quad Entropy(Brown) = -\frac{4}{5}\log_2\left(\frac{4}{5}\right) - \frac{1}{5}\log_2\left(\frac{1}{5}\right) = 0.7219$$

Green

$$p_g = 1, \quad n_g = 1, \quad Entropy(Brown) = -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) = 1$$

Now we go for the information.

$$I(Color) = \frac{p_b + n_b}{p+n} Entropy\,(Color = Brown) + \frac{p_g + n_g}{p+n} Entropy\,(Color = Green)$$

$$I(Color) = \frac{5}{7} \times 0.7219 + \frac{2}{7} \times 1 = 0.8014$$

Now the information gain is as follows.

$$Gain(Color) = Entropy(S) - I(Color) = 0.8631 - 0.8014 = 0.0617$$

Height

We calculate the entropy and information for each height variation.

Tall

$$p_t = 3, \quad n_t = 1, \quad Entropy(Brown) = -\frac{3}{4}\log_2\left(\frac{3}{4}\right) - \frac{1}{4}\log_2\left(\frac{1}{4}\right) = 0.8113$$

Short

$$p_s = 2, \quad n_s = 1, \quad Entropy(Brown) = -\frac{2}{3}\log_2\left(\frac{2}{3}\right) - \frac{1}{3}\log_2\left(\frac{1}{3}\right) = 0.9183$$

Now we go for the information.

$$I(Height) = \frac{p_t + n_t}{p+n} Entropy\left(Height = Tall\right) + \frac{p_s + n_s}{p+n} Entropy\left(Height = Short\right)$$
$$I(Height) = \frac{4}{7} \times 0.8113 + \frac{3}{7} \times 0.9183 = 0.8572$$

Now the information gain is as follows.

$$Gain(Height) = Entropy(S') - I(Nlegs) = 0.8631 - 0.8572 = 0.0059$$

Habitat

We calculate the entropy and information for each habitat variation.

Land

$$p_l = 2, \quad n_l = 1, \quad Entropy(Land) = -\frac{2}{3}\log_2\left(\frac{2}{3}\right) - \frac{1}{3}\log_2\left(\frac{1}{3}\right) = 0.9183$$

Water

$$p_w = 3, \quad n_w = 1, \quad Entropy(Water) = -\frac{3}{4}\log_2\left(\frac{3}{4}\right) - \frac{1}{4}\log_2\left(\frac{1}{4}\right) = 0.8113$$

Now we go for the information.

$$I(Habitat) = \frac{p_l + n_l}{p+n} Entropy\left(Habitat = Land\right) + \frac{p_w + n_w}{p+n} Entropy\left(Habitat = Water\right)$$
$$I(Habitat) = \frac{3}{7} \times 0.9183 + \frac{4}{7} \times 0.8113 = 0.8572$$

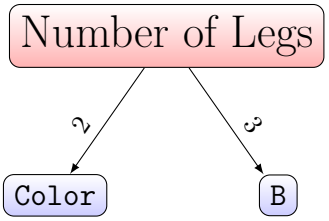
Now the information gain is as follows.

$$Gain(Habitat) = Entropy(S') - I(Habitat) = 0.8631 - 0.8572 = 0.0059$$

Now we compare the gains to choose the next node.

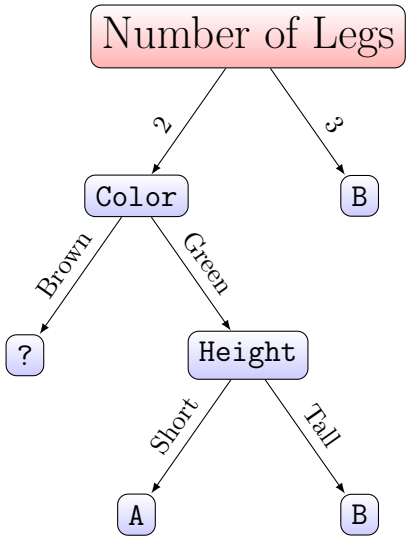
$$Gain(Color) = 0.0617, \quad Gain(Height) = Gain(Habitat) = 0.0059$$

So we choose the color node as the next one.



Step 5

Similar to step 4, we shall choose the next node, if the color is green it is obvious to us that we can classify the animal solely on the height of it, so for this case the next node shall be height as follows:



But for the scenario which the color is brown we have the following:

$$p = 4, \quad n = 1, \quad Tot = p + n = 5$$
$$Entropy(S'') = -\frac{p}{p+n}\log_2\left(\frac{p}{p+n}\right) - \frac{n}{n+p}\log_2\left(\frac{n}{p+n}\right)$$
$$= -\frac{4}{5}\log_2\left(\frac{4}{5}\right) - \frac{1}{5}\log_2\left(\frac{1}{5}\right) = 0.7219 \longrightarrow Entropy(S'') = 0.7219$$

Height

We calculate the entropy and information for each height variation.

Tall

$$p_t = 3, \quad n_t = 0, \quad Entropy(Brown) = -\frac{3}{3}\log_2\left(\frac{3}{3}\right) - \frac{0}{3}\log_2\left(\frac{0}{3}\right) = 0$$

Short

$$p_s = 1, \quad n_s = 1, \quad Entropy(Brown) = -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) = 1$$

Now we go for the information.

$$I(Height) = \frac{p_t + n_t}{p+n} Entropy\left(Height = Tall\right) + \frac{p_s + n_s}{p+n} Entropy\left(Height = Short\right)$$
$$I(Height) = \frac{3}{5} \times 0 + \frac{2}{5} \times 1 = 0.4$$

Now the information gain is as follows.

$$Gain(Height) = Entropy(S'') - I(Nlegs) = 0.7219 - 0.4 = 0.3219$$

Habitat

We calculate the entropy and information for each habitat variation.

Land

$$p_l = 2, \quad n_l = 1, \quad Entropy(Land) = -\frac{2}{3}\log_2\left(\frac{2}{3}\right) - \frac{1}{3}\log_2\left(\frac{1}{3}\right) = 0.9183$$

Water

$$p_w = 2, \quad n_w = 0, \quad Entropy(Water) = -\frac{2}{2}\log_2\left(\frac{2}{2}\right) - \frac{0}{2}\log_2\left(\frac{0}{2}\right) = 0$$

Now we go for the information.

$$I(Habitat) = \frac{p_l + n_l}{p+n} Entropy\left(Habitat = Land\right) + \frac{p_w + n_w}{p+n} Entropy\left(Habitat = Water\right)$$
$$I(Habitat) = \frac{3}{5} \times 0.9183 + \frac{2}{5} \times 0 = 0.5510$$

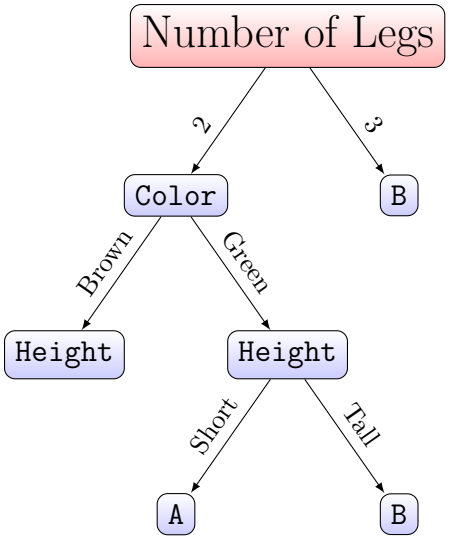
Now the information gain is as follows.

$$Gain(Habitat) = Entropy(S'') - I(Habitat) = 0.7919 - 0.5510 = 0.1709$$

Now we compare the gains to choose the next node.

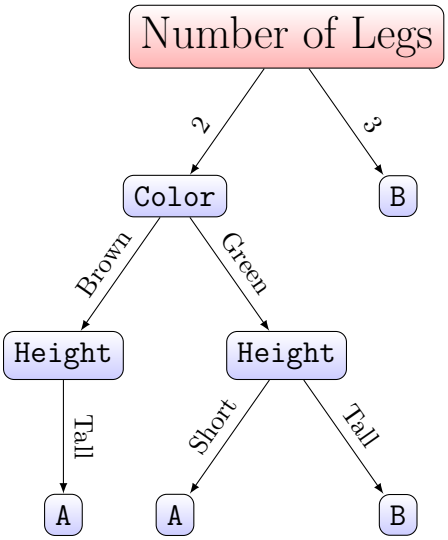
$$Gain(Height) = 0.3219, \quad Gain(Habitat) = 0.1709$$

So the next node shall be height anyway.

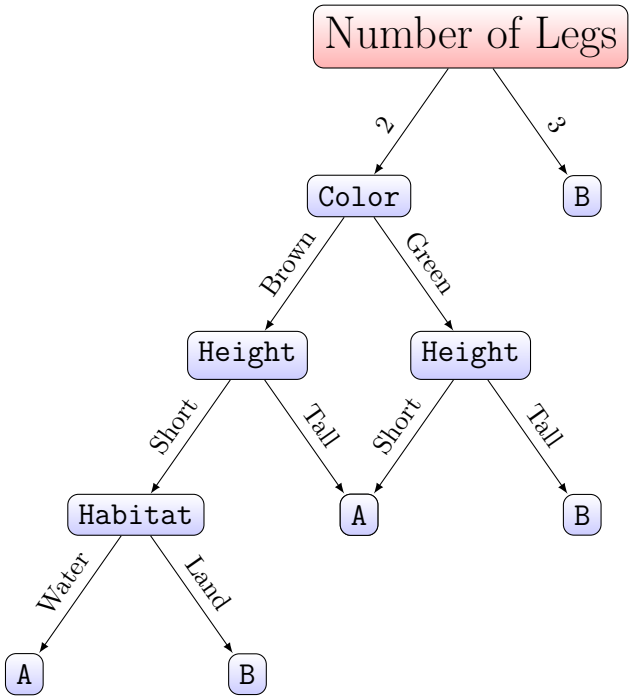


Step 6

It is obvious that if the height is tall then the animal is in the A category.



Step 7 The final node shall be the location and now we complete our decision tree.



1.2 Classifier Testing

Here we shall test our tree for the following table.

شماره	رنگ	تعداد پا	قد	محل زندگی	جاندار
۱	قهوه‌ای	3	بلند	خشکی	B
۲	سبز	2	بلند	خشکی	A
۳	سبز	2	کوتاه	خشکی	A
۴	قهوه‌ای	2	کوتاه	آب	B
۵	قهوه‌ای	2	بلند	خشکی	A

Figure 2: Information needed for classification(Test Data)

1.2.1 Number One

Due to the fact that this case has 3 legs, our tree classifies it as B, which is **correct**.

1.2.2 Number Two

Due to the fact that this case has 2 legs, we check the color, it is green, we go on to check the height which is tall, so our tree classifies it as B, which is **incorrect**.

1.2.3 Number Three

Due to the fact that this case has 2 legs, we check the color, it is green, we go on to check the height which is short, so our tree classifies it as A, which is **correct**.

1.2.4 Number Four

Due to the fact that this case has 2 legs, we check the color, it is brown, we go on to check the height which is short, so we must check the location which

is water, so our tree classifies it as A, which is **incorrect**.

1.2.5 Number Five

Due to the fact that this case has 2 legs, we check the color, it is brown, we go on to check the height which is tall, so our tree classifies it as A, which is **correct**.

Confusion Matrix

We shall design the confusion matrix, to do so we assume that A is positive and B is negative, and $N = 5$ in this example, so we have:

		Truth (+)	**Truth** (−)
Test	(+)	$\mathrm{T_P} = 2$	$\mathrm{F_P} = 1$
	(−)	$\mathrm{F_N} = 1$	$\mathrm{T_N} = 1$

Now we can calculate the accuracy, which essentially depicts how often our tree works, we have:

$$Accuracy = \frac{T_P + T_N}{N} = \frac{3}{5} = 0.6 \longrightarrow Accuracy = 60\%$$

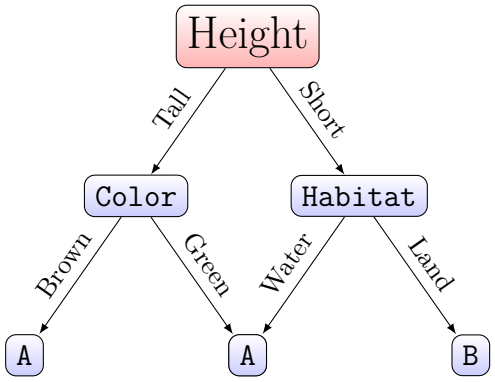
1.3 Greedy implementation of ID3 algorithm

We know that the **ID3 Algorithm** is a greedy algorithm, this algorithm grows the tree from top to down, when it reaches each node, it selects the attribute which best classifies the local training examples, this was demonstrated fully in the first part of this question accordingly, this process is

continued until the tree perfectly classifies the training examples or until it runs out of attributes due to using up all of them.

Due to **Occam's razor** we prefer the simplest hypothesis that fits our data, hence we prefer decision trees with the minimum depth that completely fulfill the training data, in our case with the **Information Gain** metric, the first two nodes chosen are the **Number of legs** and **Color** nodes, these features cannot completely classify the training data.

In continuance of our efforts to complete the decision tree with only two features for each specimen tried to use two attributes to classify A and B accordingly, in this way we could reach a tree with depth 2 which will fulfill our goal, to do so we construct a tree with a different root, we choose **Height** as the root, if it is tall, we choose the next node to be color, and if it is short we chose the next be habitat, this concludes our tree and it works with zero error on the training data.



1.4 Increasing robustness of Classifier

First of all, we must observe why the decision tree is not robust against overfitting.

There are many practical issues in learning decision trees, some of them are determining how deep the tree should grow, handling the continuous attributes, choosing appropriate attributes and overfitting issues.

The **ID3 Algorithm** grows each branch of a tree deeply enough to perfectly classify the training data, this is mostly reasonable but can lead to problems when we are dealing with noisy data or when the number of training examples is too small to sample the true target function accordingly, to prevent overfitting we shall recommend two **Pruning** methods, but first of all, we must know what pruning is.

1.4.1 Pruning

Normally a decision tree is allowed to grow to its full depth, when we perform pruning some parts of the tree are removed and prevented from growing to its full extent, this can be useful in preventing overfitting, we shall explain the two methods in due course.

1.4.2 Method 1 : Reduced Error Pruning

In this pruning method each of the decision nodes in the tree are considered as pruning candidates, pruning a decision node is essentially removing the subtree rooted at the node and transforming it into a leaf node and assigning it the most common classification of the training examples affiliated with the said node.

It is important to take notice that nodes are removed only if the resulting pruned tree performs no worse than the original tree over the validation set.

1.4.3 Method 2 : Rule Post-Pruning

In this pruning method we allow the decision tree to grow until the training data fits as well as possible and we allow overfitting to occur, then we convert the learned tree into an equivalent by creating one rule for each path from the root to the leaf node, then we prune each rule by removing preconditions that result in improving the estimated accuracy, in the end we sort the pruned

rules by their estimated accuracy and consider them in this ordered sequence whilst we classify the subsequent instances.

This method is handy in preventing overfitting, it is good to know that a variant of this method is used by **C4.5**. which is an outgrowth of the **ID3 Algorithm**.

2 Question 2 : Decision Tree Implementation

2.1 Tree Model Implementation

Here we shall implement the tree using the **Titanic** dataset, there are many attributes in this dataset, all of which aren't effective in the tree, so we have dropped them in the preprocessing section, these attributes are the port where the passengers have boarded the titanic(logically this has literally nothing to do with them surviving) the other attribute is the fair which also has nothing to do with them surviving because we already have the **Pclass** attribute which is the level of the ticket.

In our implementation due to the fact the test file given to us had no labels, I was forced to do everything with the train file, what we did was that we moved the labels (AKA **Survived**) to the final column of the train data, we do this to split the data according to the best split obtained in our functions.

For brevity we have broken the implementation into different functions to make the code more understandable, in the end we test our code as depicted below.

```
Test = Train.sample(frac=0.2,random_state=42)
idx = Test.index
Train_ = Train.iloc[list(set(range(len(Train)))-set(idx)), :]
tree = ID3(Train_, MaxDepth=3)
print(tree)
Output:
{'Sex = 1.0': [{'Pclass = 3.0': [{'SibSp <= 2.0': [1.0, 0.0]},
                                 {'Age <= 2.0': [0.0, 1.0]}]}, {
                                'Age <= 6.0': [{'SibSp <= 2.0':
                                [1.0, 0.0]}, 0.0]}]}
```

Figure 3: Testing the tree for depth = 3

```
calculate_accuracy(Test, tree)
Output:
0.797752808988764
```

Figure 4: Accuracy depth = 3

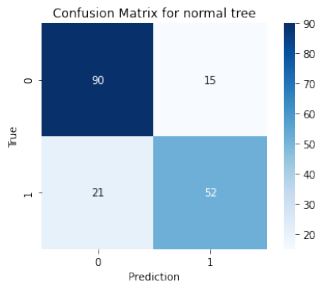


Figure 5: Confusion Matrix for *Depth* = 3

```
tree = ID3(Train_, MaxDepth=5)
print(tree)
Output:
{'Sex = 1.0': [{'Pclass = 3.0': [{'SibSp <= 2.0': [{'Parch
                                 <= 3.0': [1.0, 0.0]}, {'Parch <=
                                  0.0': [1.0, 0.0]}]}, {'Age <= 2
                                 .0': [{'Parch <= 1.0': [1.0, 0.0
                                 ]}, 1.0]}]}, {'Age <= 6.0': [{'
                                 SibSp <= 2.0': [1.0, 0.0]}, {'
                                 Pclass = 1.0': [{'Age <= 70.0':
                                 [0.0, 1.0]}, 0.0]}]}]}
```

Figure 6: Testing the tree for depth = 5

```
calculate_accuracy(Test, tree)
Output:
0.7865168539325843
```

Figure 7: Accuracy depth = 5

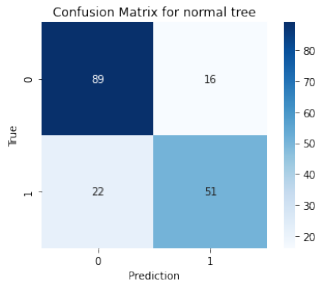


Figure 8: Confusion Matrix for *Depth* = 5

2.2 Improving the decision tree algorithm

As discussed in class, some of disadvantages and drawbacks of the **ID3 Algorithm** is as follows.

- Many algorithms (like ID3 and C4.5) require that the target attribute to have only discrete values and don't work well with continuous values.
- The greedy characteristic of decision trees lead to different problems, such as over sensitivity to the training data and irrelevant attributes and noise which make the tree unstable.
- A small change in the data can result in a major change in the structure of the decision tree.
- If small samples are tested data may be overfitted.
- Only one attribute at a time is tested for making a decision.

To solve these problems we resort to the **Bagging** and **Random Forest** methods.

2.2.1 Bagging(Bootstrap Aggregation)

In a simple interpretation, bagging means building different models using sample subset and then aggregating the predictions of the different models to reduce variance, due to the fact that bagging reduces the mean of the observations by factor n the variance shall be reduced by this factor, so by increasing the data points, the variance of the mean is reduced and this is the concept behind bagging decision trees.

Now we shall explain the two steps of bagging, the first step is bootstrapping helps to create multiple subsets from the training data. Then we can build multiple trees on the bootstrapped samples.

Next we have aggregation, Model predictions undergo aggregation to combine them for the final prediction to consider all the possible outcomes. The

aggregation can be done based on the total number of outcomes or the probability of predictions derived from the bootstrapping of every model in the procedure.

All in all, bagging significantly raises the stability of models in improving accuracy and reducing variance, which eliminates the challenge of overfitting. Bagging in ensemble machine learning takes several weak models, aggregating the predictions to select the best prediction.

2.2.2 Random Forest

The concept behind the random forest is quite simple, it is merely a collection of decision trees whose results are aggregated into one final result. This model is quite powerful, this is due to their ability to limit overfitting without substantially increasing error due to bias.

Random forest avoids the overfitting problem of decision trees by instead scaling by adding more trees instead of building one big tree. Averaging the outputs of the trees in the forest means that it does not matter as much if the individual trees are overfitting.

In conclusion we point out some of the advantages of the random forest.

- Random forests can handle categorical and continuous variables equally well.
- They can be used to solve both classification and regression problems.
- Random forest is comparatively less affected by noise.
- Random forest can handle missing values automatically.
- The algorithm is very stable. If a new data point is introduced in the dataset, the overall algorithm is not impacted much.

2.3 Implementation of Random Forest

Here we have implemented the random forest, the codes and results are included as follows.

```
def random_forest(Train, Test, k=5):
preds = []
trees = []
results = []
for i in range(k):
Test_ = Train.sample(frac=0.2, random_state=random.randint(0
                                , 2000))
idx = Test_.index
Train = Train.iloc[list(set(range(len(Train)))-set(idx)), :]
tree = ID3(Train, MaxDepth=3)
trees.append(tree)
for i, row in Test.iterrows():
for tree in trees:
predictions = PredictExpl(row, tree)
preds.append(predictions)
results.append(max(set(preds), key = preds.count))
preds = []
return np.array(results)
```

Figure 9: Random Forest code

```
predictions = random_forest(Train_, Test, 9)
print("Accuracy =", (np.array(predictions) == Test.label).
                                mean())
Output:
Accuracy = 0.8146067415730337
```

Figure 10: Random Forest Answers

The confusion Matrix shall be depicted below.

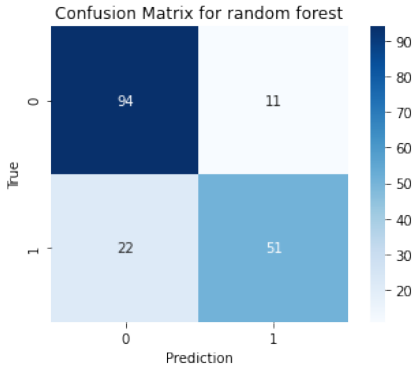


Figure 11: Confusion Matrix for random forest

As we predicted, the **Random Forest Algorithm** works beautifully, it successfully increases the accuracy of the decision tree, so we can be of the same opinion and in accord that our implementation works properly.

In the last section, we shall study the concepts of metric learning.

3 Question 3: Metric Learning

3.1 K Nearest Neighbor Classifier

3.1.1 Classifier Design

Here we give a brief explanation about are flow in implementing this algorithm.

Firstly we choose a distance function, here I have used the euclidean distance, then we must go on to find the nearest neighbors(the number of them are determined by k) after which we shall go on to predict the classification and find the accuracy and confusion matrix accordingly.

```
def KNearestNeighbors(K,train_data,test_data):
#we need to find the closest neighbors hence we need an
                                array
distance = []
distance = Distance(train_data,test_data)
#Now we need the indices of the minimum distances to choose
                                the neighbors
Minindices = distance.argsort(axis=0)
neighbors = np.zeros(len(distance))
neighbors = train_data[Minindices]
#We must choose the first k ones as the final neighbors
neighbors = neighbors[0:K,:]
Minindices = Minindices[0:K,:]
return neighbors,Minindices
```

Figure 12: KNN code

```
def Predict(K,train_data, test_data,train_labels):
Classification=list()
Neighbors,NeighborIndices = KNearestNeighbors(K,train_data,
                                test_data)
counter1 = 0
NeighborLabels = train_labels[NeighborIndices]
while(counter1<NeighborIndices.shape[1]):
counter2 = 0
Temp = list()
while(counter2<NeighborIndices.shape[0]):
Temp.append(train_labels[NeighborIndices[counter2][counter1]])
counter2 +=1
Classification.append(max(Temp,key = Temp.count))
counter1 +=1
Classification = np.array(Classification)
return Classification,NeighborLabels
```

Figure 13: KNN Prediction code

```
  def Accuracy (Real,Pred):
  counter = 0
  for i in range(len(Real)):
  if Real[i]==Pred[i]:
  counter+=1
  return counter/len(Real)
```

Figure 14: Accuracy code

```
def CMatrix(Pred,Real):
#First we must find the classes, i.e: 0 1 2
Cls = np.unique(Real)
Len  = len(Cls)
#We know that the confusion matrix is always a square matrix
                                hence
Cm = np.zeros((Len,Len))
for i in range (Len):
for j in range(Len):
Cm[i][j] = np.sum((Real == Cls[i]) & (Pred == Cls[j]))
return Cm
```

Figure 15: Confusion Matrix code

The accuracies for the different KNNs are as follows.

```
Acc1: 0.7777777777777778
Acc5: 0.8333333333333334
Acc10: 0.75
Acc20: 0.7777777777777778
```

Figure 16: Accuracies for $K = 1, 5, 10, 20$

Now we take a look at the different confusion matrices.

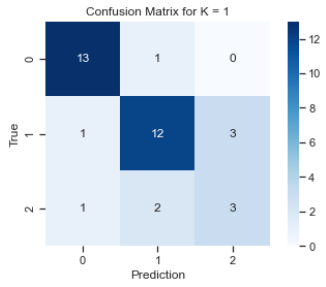


Figure 17: Confusion Matrix for $K = 1$

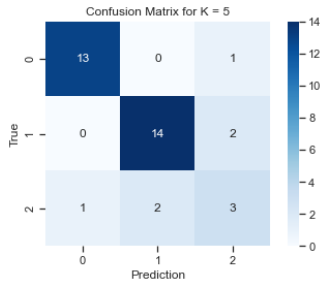


Figure 18: Confusion Matrix for $K = 5$

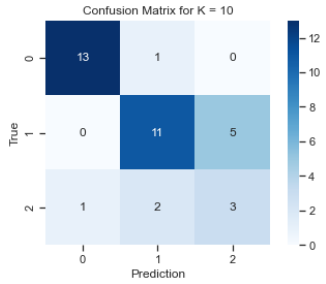


Figure 19: Confusion Matrix for $K = 10$

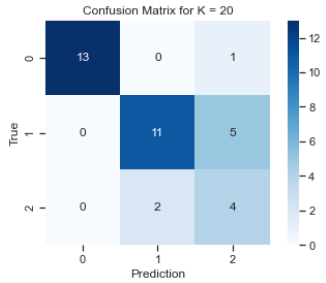


Figure 20: Confusion Matrix for $K = 20$

3.1.2 Probability of being in a class

The plots are as follows.

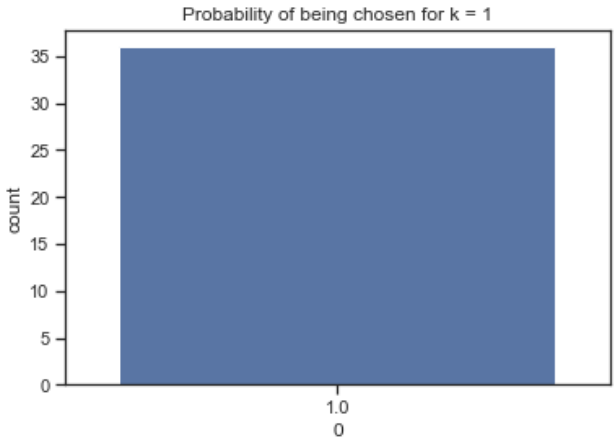


Figure 21: Probability of being chosen for $K = 1$

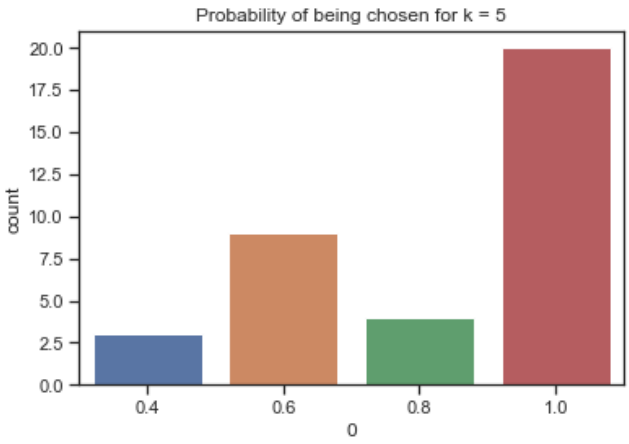


Figure 22: Probability of being chosen for $K = 5$

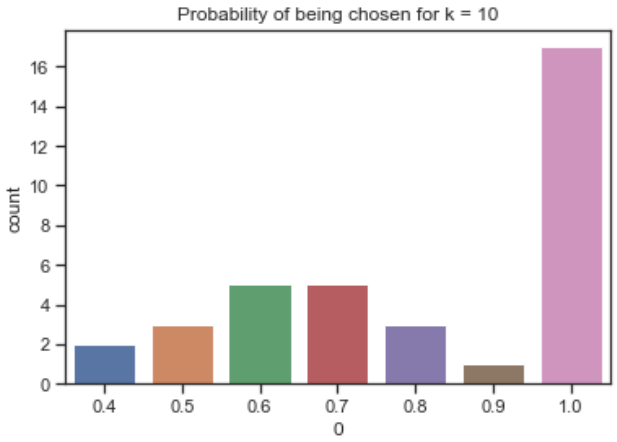


Figure 23: Probability of being chosen for $K = 10$

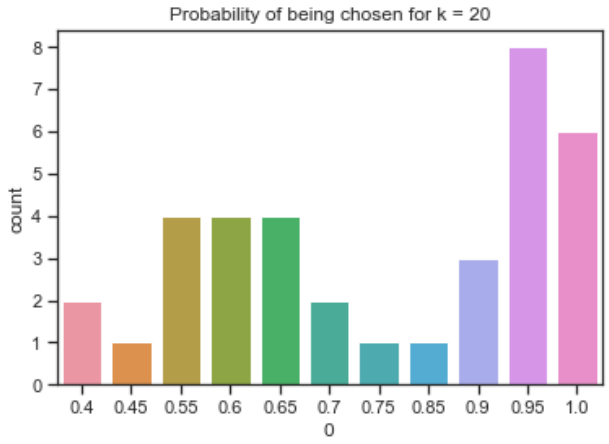


Figure 24: Probability of being chosen for $K = 20$

3.2 Metric Learning with LMNN and LFDA

3.2.1 Overall observation of learning method

Here we shall take a look at the **LMNN** and **LFDA** methods.

LMNN

The Large margin nearest neighbor classification is a statistical machine learning algorithm for metric learning, it is somewhat similar to **KNN** and it i a sub-class of convex optimization.

In this method we have target neighbors, these neighbors are selected before training, each instance has an amount of target neighbors within a range, these neighbors are the ones that should become nearest neighbors under the learned metric.

Then we have the impostors, consider a datapoint such as x_i, if another datapoint is in the nearest neighbors of x_i with a different class label this datapoint is an impostor.

All in all the constraints in this algorithm are used to fulfill the following goal, we want the target neighbors to be close and the impostors to be far.

LFDA

The Local Fisher Discriminant Analysis method is a very well performing supervised dimensionality-reducing metric learning, what it does is that it automatically learn a matrix to capture the characteristics of the original data.

In short this method attempts to use supervised learning to obtain a transformation matrix to transform the original dataset according to the learned metrics.

3.2.2 Dataset plotting in new space

Part 1

In the **LMNN** method k is the number of neighbors to consider excluding the self-edges but in the **KNN** method k is the number of nearest neighbors

without any constraints compared to **LMNN**.

In the **LFDA** method k is the number of nearest neighbors used in local scaling method, but this is not the case in **KNN**.

Part 2

Here we shall plot the dataset after changing the space of it, we shall plot it for different values of k.

To reduce the dimentions from 13 to 2, we use **PCA**.

Then we perform KNN on both of the train and test sets just to be sure, then we get the accuracies and the corresponding scatter plots.(The KNN here was performed via scikit learn as approved by the TA of this question.)

```
from sklearn.model_selection import train_test_split
train_data,test_data, train_labels, test_labels =
                                train_test_split(wine.data, wine
                                .target, test_size=0.2,
                                random_state=0)
```

Figure 25: Train-Test Split

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
PCAData = pca.fit_transform(train_data)
PCATest = pca.transform(test_data)
print(np.shape(PCAData))
print(np.shape(PCATest))
```

Figure 26: PCA code

Next we shall go on to perform KNN, to do so we use the appropriate functions and commands as instructed below.

```
#Import knearest neighbors Classifier model
from sklearn.neighbors import KNeighborsClassifier

knn1 = KNeighborsClassifier(n_neighbors=1)
knn1.fit(PCAData, train_labels)
Pred1 = knn1.predict(PCAData)

knn5 = KNeighborsClassifier(n_neighbors=5)
knn5.fit(PCAData, train_labels)
Pred5 = knn5.predict(PCAData)

knn15 = KNeighborsClassifier(n_neighbors=15)
knn15.fit(PCAData, train_labels)
Pred15 = knn15.predict(PCAData)


#Now for the test data.

knn1t = KNeighborsClassifier(n_neighbors=1)
knn1t.fit(PCAData, train_labels)
Pred1t = knn1t.predict(PCATest)

knn5t = KNeighborsClassifier(n_neighbors=5)
knn5t.fit(PCAData, train_labels)
Pred5t = knn5t.predict(PCATest)

knn15t = KNeighborsClassifier(n_neighbors=15)
knn15t.fit(PCAData, train_labels)
Pred15t = knn15t.predict(PCATest)
```

Figure 27: KNN

Now we shall go on to obtain the accuracies.

```
Accuracy1train: 1.0
Accuracy5train: 0.7816901408450704
Accuracy15train: 0.7535211267605634
Accuracy1test: 0.75
Accuracy5test: 0.75
Accuracy15test: 0.7222222222222222
```

Figure 28: Accuracies for different Ks

Now we obtain the needed scatter plots.

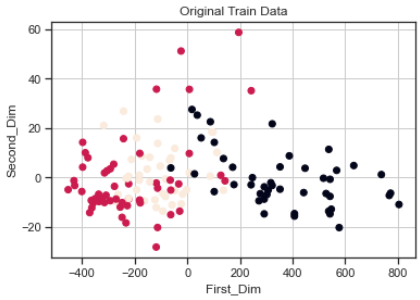


Figure 29: Original Train Data

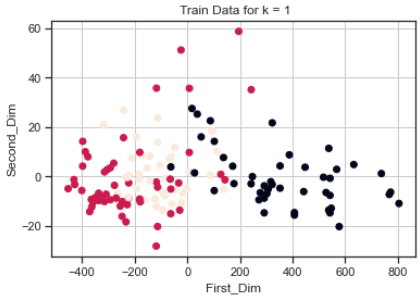


Figure 30: Train Data for $K = 1$

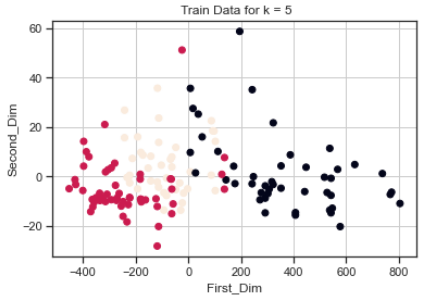


Figure 31: Train Data for $K = 5$

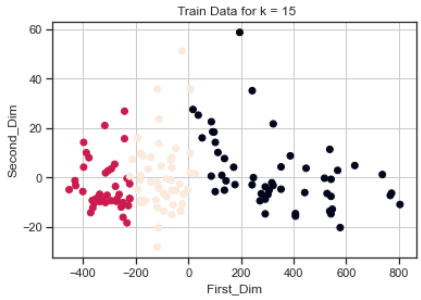


Figure 32: Train Data for $K = 15$

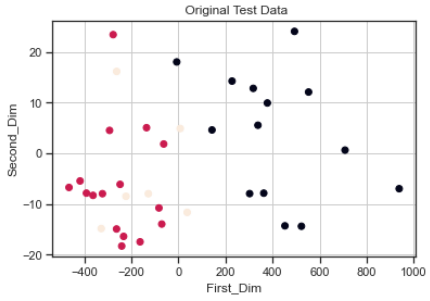


Figure 33: Original Test Data

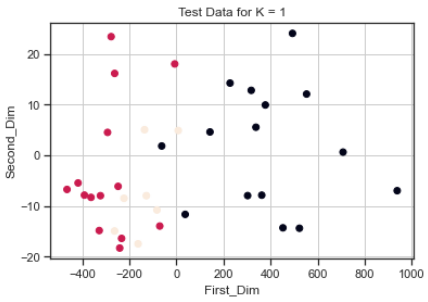


Figure 34: Test Data for $K = 1$

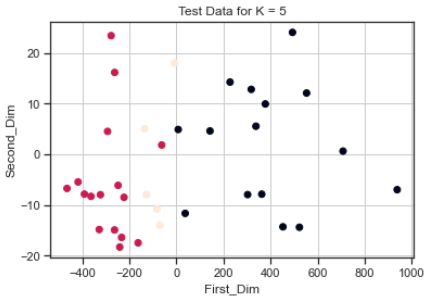


Figure 35: Test Data for $K = 5$

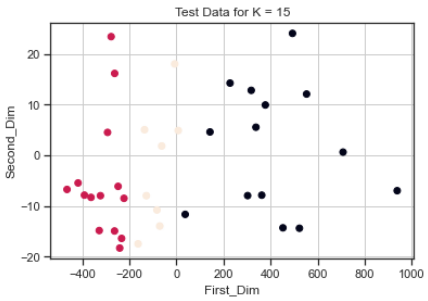


Figure 36: Test Data for $K = 15$

It is understandable that for $K = 15$ the space is easier to classify in a heuristic manner, but this does not mean that the accuracy for $K = 15$ is the best option.

3.2.3 Classifier Comparison

Due to the results we obtained on the test data, we shall go out on a limb and claim that $K = 5$ has a better accuracy and $K = 15$ is better for visual classification, here we have included the confusion matrix for both of these versions.

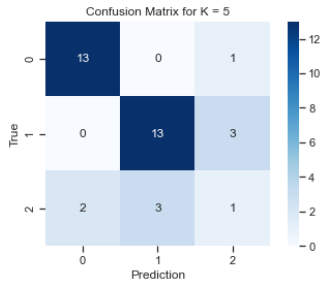


Figure 37: Confusion Matrix for $K = 5$

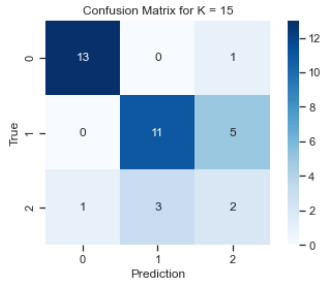


Figure 38: Confusion Matrix for $K = 15$

We can see that the results are quite similar to the previous part of this assignment.

Now we shall do all the above steps for LMNN and LFDA.

LFDA

We shall do the following:

```
from metric_learn import LFDA
ndim = 2
lfda = LFDA(ndim)
lfda.fit(wine.data, wine.target)
LFDAtraindata = lfda.transform(train_data)
LFDAtestdata = lfda.transform(test_data)
print(LFDAtraindata.shape)
print(LFDAtestdata.shape)
```

Figure 39: LFDA Dimension reduction

```
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy1trainLFDA:",metrics.accuracy_score(
                                train_labels, Pred1))
print("Accuracy5trainLFDA:",metrics.accuracy_score(
                                train_labels, Pred5))
print("Accuracy15trainLFDA:",metrics.accuracy_score(
                                train_labels, Pred15))
print("Accuracy1testLFDA:",metrics.accuracy_score(test_labels,
                                 Pred1t))
print("Accuracy5testLFDA:",metrics.accuracy_score(test_labels,
                                 Pred5t))
print("Accuracy15testLFDA:",metrics.accuracy_score(test_labels
                                , Pred15t))
```

Figure 40: Accuracy

```
Accuracy1trainLFDA: 1.0
Accuracy5trainLFDA: 0.971830985915493
Accuracy15trainLFDA: 0.9788732394366197
Accuracy1testLFDA: 0.9722222222222222
Accuracy5testLFDA: 1.0
Accuracy15testLFDA: 0.9444444444444444
```

Figure 41: Accuracies for different Ks

Now we obtain the needed scatter plots.

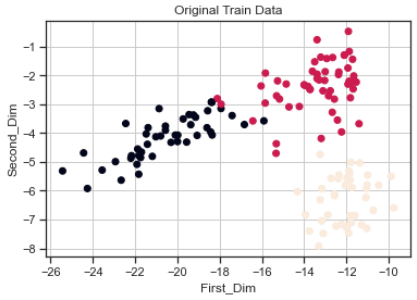


Figure 42: Original Train Data

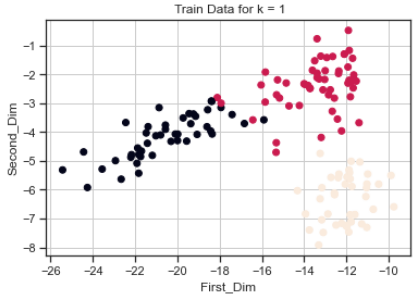


Figure 43: Train Data for $K = 1$

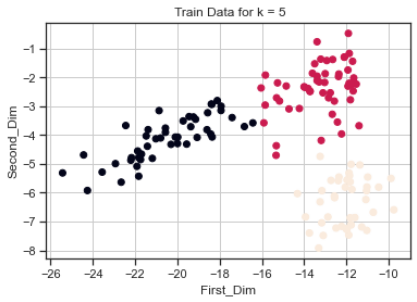


Figure 44: Train Data for $K = 5$

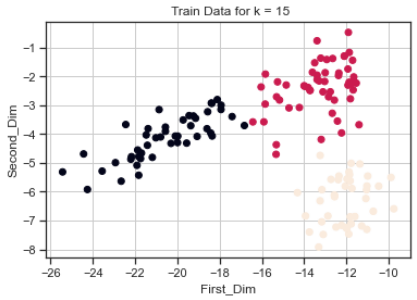


Figure 45: Train Data for $K = 15$

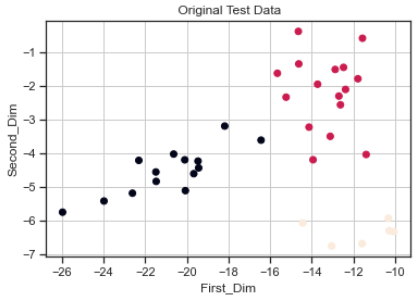


Figure 46: Original Test Data

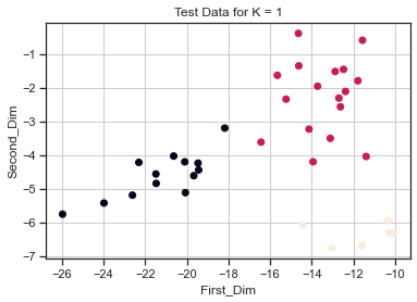


Figure 47: Test Data for $K = 1$

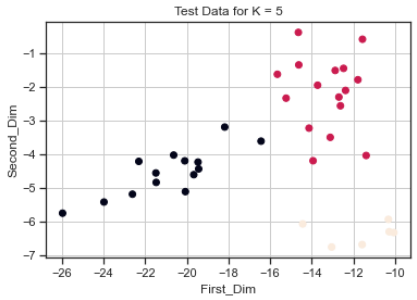


Figure 48: Test Data for $K = 5$

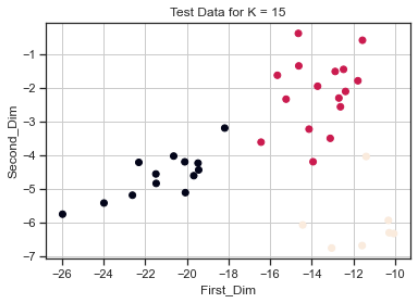


Figure 49: Test Data for $K = 15$

As we can see the accuracy is much higher than the normal PCA and the scatter plots are more classifiable.

LMNN

Here we perform the reduction with **PCA** then proceed to implement LMNN.

The sample code is shown below.

```
from metric_learn import LMNN
X = PCAData
X2 = PCATest
Y = train_labels
Y2 = test_labels
lmnntr1 = metric_learn.LMNN(k=1, learn_rate=1e-6)
lmnnte1 = metric_learn.LMNN(k=1, learn_rate=1e-6)
# fit the data!
lmnntr1.fit(X, Y)
knntr1 = KNeighborsClassifier(n_neighbors = 1)
knntr1.fit(lmnntr1.transform(X),Y)
Predtr1 = knntr1.predict(lmnntr1.transform(X))
print("Accuracy1trainLMNN:",metrics.accuracy_score(
                            train_labels, Predtr1))
lmnnte1.fit(X2, Y2)
knnte1 = KNeighborsClassifier(n_neighbors = 1)
knnte1.fit(lmnnte1.transform(X2),Y2)
Predte1 = knnte1.predict(lmnnte1.transform(X2))
print("Accuracy1testLMNN:",metrics.accuracy_score(
                            test_labels, Predte1))
```

Figure 50: LMNN Code

The scatter plots are shown below.

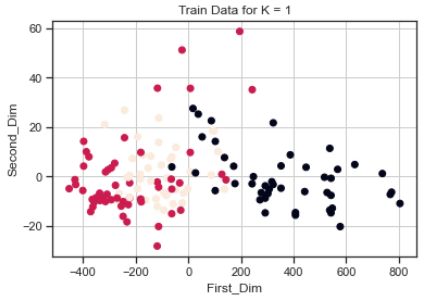


Figure 51: Train Data for $K = 1$

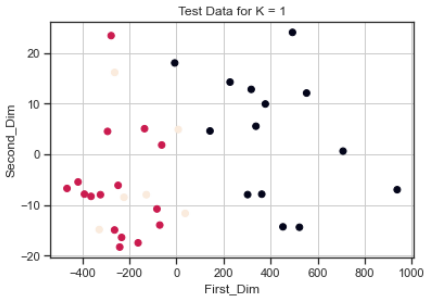


Figure 52: Test Data for $K = 1$

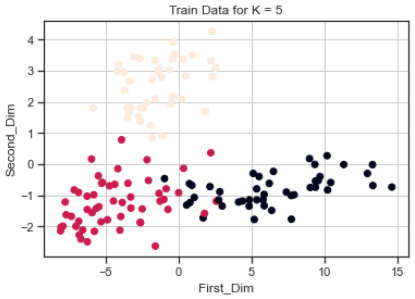


Figure 53: Train Data for $K = 5$

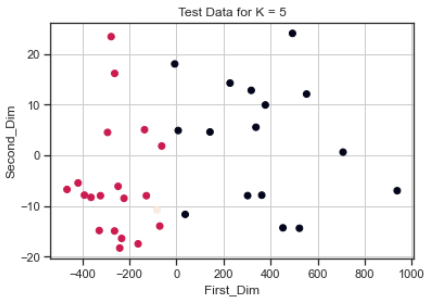


Figure 54: Test Data for $K = 5$

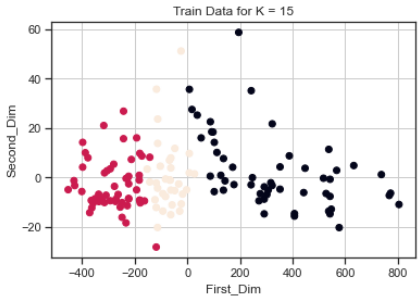


Figure 55: Train Data for $K = 15$

3.2.4 Correlation Coefficient

The correlation matrix can be seen as below.

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins
alcohol	1.000000	0.094397	0.211545	-0.310235	0.270798	0.289101	0.236815	-0.155929	0.136698
malic_acid	0.094397	1.000000	0.164045	0.288500	-0.054575	-0.335167	-0.411007	0.292977	-0.220746
ash	0.211545	0.164045	1.000000	0.443367	0.286587	0.128980	0.115077	0.186230	0.009652
alcalinity_of_ash	-0.310235	0.288500	0.443367	1.000000	-0.083333	-0.321113	-0.351370	0.361922	-0.197327
magnesium	0.270798	-0.054575	0.286587	-0.083333	1.000000	0.214401	0.195784	-0.256294	0.236441
total_phenols	0.289101	-0.335167	0.128980	-0.321113	0.214401	1.000000	0.864564	-0.449935	0.612413
flavanoids	0.236815	-0.411007	0.115077	-0.351370	0.195784	0.864564	1.000000	-0.537900	0.652692
nonflavanoid_phenols	-0.155929	0.292977	0.186230	0.361922	-0.256294	-0.449935	-0.537900	1.000000	-0.365845
proanthocyanins	0.136698	-0.220746	0.009652	-0.197327	0.236441	0.612413	0.652692	-0.365845	1.000000
color_intensity	0.546364	0.248985	0.258887	0.018732	0.199950	-0.055136	-0.172379	0.139057	-0.025250
hue	-0.071747	-0.561296	-0.074667	-0.273955	0.055398	0.433681	0.543479	-0.262640	0.295544
od280/od315_of_diluted_wines	0.072343	-0.368710	0.003911	-0.276769	0.066004	0.699949	0.787194	-0.503270	0.519067
proline	0.643720	-0.192011	0.223626	-0.440597	0.393351	0.498115	0.494193	-0.311385	0.330417

Figure 56: Corrlation Matrix for Wine Data

	0	1
0	1.000000	0.000000
1	0.000000	1.000000

Figure 57: Corrlation Matrix for Wine Data

	0	1
0	1.000000	-0.026416
1	-0.026416	1.000000

Figure 58: Corrlation Matrix for Wine Data

	0	1
0	1.000000	0.730804
1	0.730804	1.000000

Figure 59: Corrlation Matrix for Wine Data

3.2.5 GMML

We know that many machine learning algorithms need to compute the distance between data points, therefore it is important to select the **distance measure**.

Now we shall attempt to explain the **GMML** method formulation and solution, the main idea discussed is how to choose to include the impact of the dissimilar points. It has been proposed to find a matrix, that decreases the sum of distances over all the similar points, but unlike other methods, instead of treating dissimilar points asymmetrically, it is proposed to measure their interpoint distances using the inverse matrix and to add their contribution to the objective.

Some experiments have been conducted in this model, one important difference between this model and the **LMNN** model is the difference between the number of constraints, the **GMML** method has more than the **LMNN** method. The experiments also denote that this new proposed method works better than **LMNN** method.

As for another difference between these two methods we can point out that the **LMNN** method doesn't do anything with dissimilar points but the **GMML** method uses these points accordingly.

References

[1] Reshad Hosseini, *Intelligent Systems Lecture Notes, Fall 01*

[2] Pouya Habib Zadeh, Reshad Hosseini,Suvrit Sra *Geometric Mean Metric Learning*

[3] Tom Mitchell, *Machine Learning , McGraw-Hill, 1997*

[4] Decision Tree ID3 Algorithm

[5] A simple guide to confusion matrix terminology

[6] Using Bagging and Boosting to Improve Classification Tree Accuracy

[7] How To Label Data For Machine Learning In Python

[8] Bagging (Bootstrap Aggregation)

[9] Random Forest Algorithm

[10] Why is random forest an improvement of decision tree?

[11] Data Mining Decision Trees