

Machine Learning and Data Mining II

LABWORK 3: Classification I

Groupwork:

Nguyễn Thị Quỳnh Anh
Vũ Yến Linh

BI11-029
BI11-152

I, K-nearest neighbor classification

The objective is to study the use of the method k-nn and

1. Implement k-nn and test on the Iris dataset. Calculate the classification error? (by comparing the class labels obtained with the prediction and the original labels of test data)

Code:

```
clc
clear

T = readtable('iris.csv');
X = table2array(T(:,1:4));

[m n] = size(X);

% Ground Truth
G = [];
G(1:50) = 0;
G(51:100) = 1;
G(101:150) = 2;
G = G';

% Randomly choose 33 samples (of each Iris class) for training and let 17
remaining samples (of that Iris class) for testing

P = [];
for i=1:3
    p = (i-1)*50 + randperm(50);
    P = [P; p'];
end

trainingData = [X(P(1:33),:); X(P(51:83),:); X(P(101:133),:)];
trainingClass = [G(P(1:33)); G(P(51:83)); G(P(101:133))];
```

```

testingData = [X(P(34:50),:); X(P(84:100),:); X(P(134:150),:)];

D = pdist2(testingData, trainingData);

[D,idx] = sort(D, 2, 'ascend');

% K nearest neighbors

K = 1; % changeable K here

D = D(:,1:K);

idx = idx(:,1:K);

% Majority vote

predict = mode(trainingClass(idx),2);

groundtruth = [G(P(34:50)); G(P(84:100)); G(P(134:150))];

error = nnz(predict - groundtruth)/51*100

accuracy = 100 - error

```

Result:

```

error =

    1.9608

accuracy =

    98.0392

```

Figure 1a: Classification error of K-nn (with k = 1) and test on the Iris dataset

Sometime, K-nn (with k = 1) reaches 100% accuracy on this dataset...

```

error =

    0

accuracy =

    100

```

Figure 1b: Classification error of K-nn (with k = 1) and test on the Iris dataset

2. Vary the value of k, comment on the results.

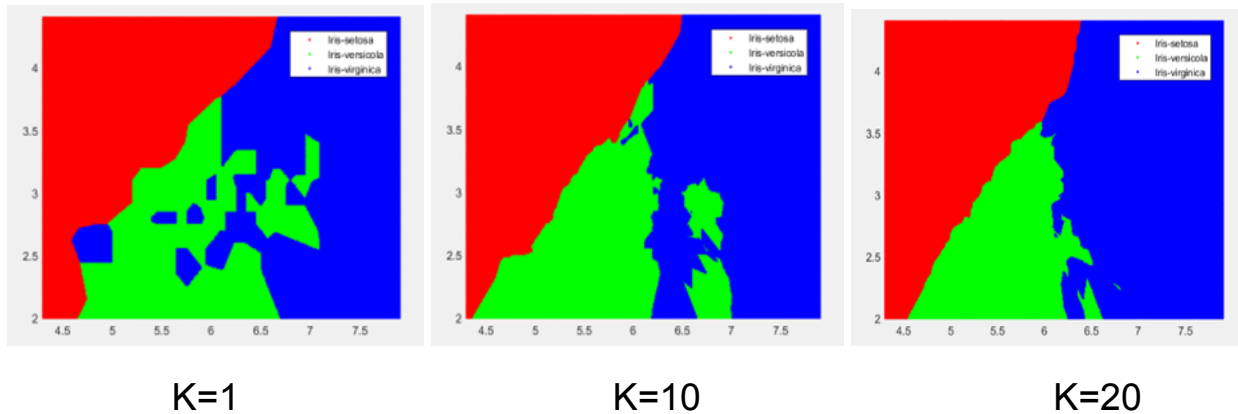


Figure 2: Decision boundary of K-nn classifier (with different k) on the Iris dataset

Comment:

- The accuracy obtained is still very high.
- Increased value of k would result in a smoother decision boundary (in other words: higher k reduces vulnerability to noise in the training data).

Pros

- As K-nn is a straightforward method, there is no need for an explicit training phase. As a result, it is simple to comprehend and apply.
- Obtains high accuracy.
- Useful for nonlinear data.

Cons

- Due to the fact that the algorithms keep all of the training data, K-nn is computationally expensive (to calculate the distance between every testing data and every training data, find majority vote for prediction).
- High memory requirement. Therefore, it is not suitable for large data (~100,000 elements)

3. Try to normalize the input dataset, is the performance better?

Code:

```
% Normalize dataset  
  
X = (X - mean(X)) ./ std(X);
```

Result:

```
error =  
  
5.8824  
  
accuracy =  
  
94.1176
```

Figure 3: Apply K-nn classifier to the normalized Iris dataset

Comment:

- - Even after normalizing the dataset, the result maintains a high level of accuracy and little changes. This is a result of the Iris dataset's tiny feature values. The features' weights are likewise more evenly distributed after normalization, which has an equal impact on the k-nn outcome.

4. Apply PCA and SVD on the dataset, then what is the performance of k-nn on the new projected data? Justify the answer.

Code:

```
% PCA the dataset  
  
[coeff,score] = pca(X);
```

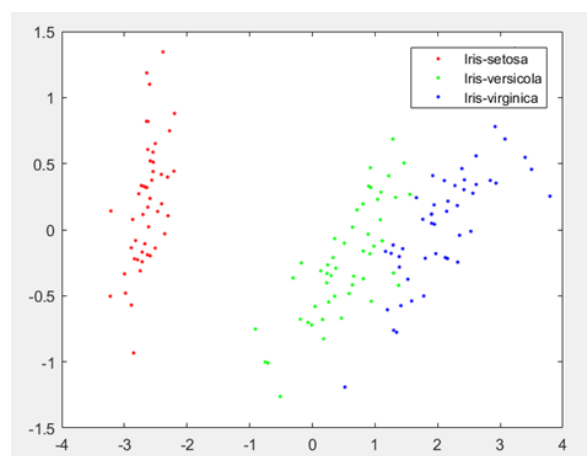


Figure 4: Iris dataset scattered with PCA

Comment:

- When the data dimension was decreased from 4 to 2, the performance seemed to be better. There are now even fewer components (or fewer dimensions) left than before, which allows us to calculate the distance between each testing and training set of data even more quickly.

5. Propose an approach to improve the performance of k-nn with the help of k-cross validation.

Result:

```
error =  
  
2.6667
```

Figure 5a: Using 5-fold for 1-nn on Iris dataset

-The best error with k-fold (k=5) is 2% (which is 98% accuracy) on this dataset...

```
error =  
  
2
```

Figure 5b: Using 5-fold CV for 1-nn on Iris dataset

Comment:

- We first randomize the index of X before storing each one in matrix P. The dataset is then divided into k sections (if the number of observations is not divisible by k, we just remove several dataset until it is satisfied). The remaining k-1 parts serve as the training set after we select each part and treat it as a testing dataset. A total of k times will be tested (as each part should be selected in turn from 1 to k, respectively). We receive an error for that turn every time we test 1 component. The average of all k additional errors is the final error. The k-nn algorithm's overfitting will be diminished with the aid of k-fold cross validation.

6. Apply leave-one-out and calculate the error of classification.

Result:

```
error =  
  
2
```

Figure 6: Using LOOCV for 1-nn on Iris dataset

7. Repeat the same process for two more datasets.

Dataset:

- Banknote Authentication dataset:

<https://archive.ics.uci.edu/ml/datasets/banknote+authentication>

- Wine dataset:

<https://archive.ics.uci.edu/ml/datasets/wine>

Result:

- *Bank dataset:*

```
error =  
  
0.2188  
  
accuracy =  
  
99.7812
```

Figure 7: Classification error of K-nn (with k = 1) and test on the Banknote Authentication dataset

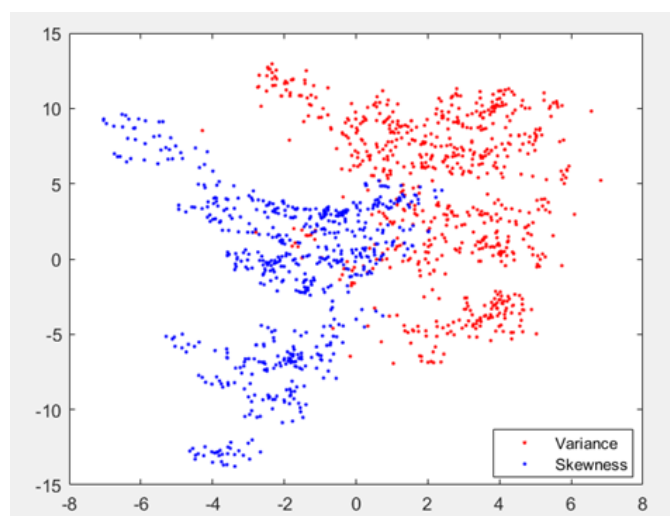


Figure 8: Banknote Authentication dataset distribution

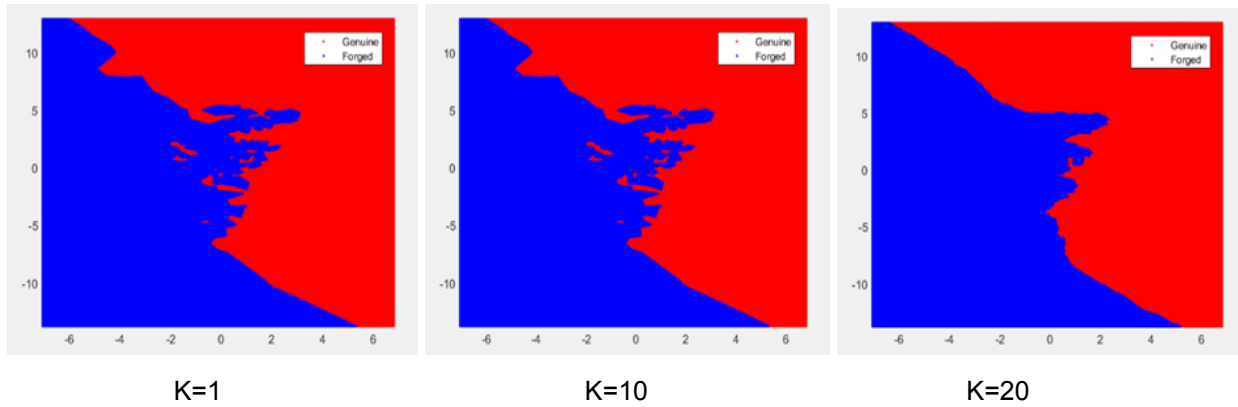


Figure 9: Decision boundary of K-nn classifier (with different k) on the Banknote Authentication dataset

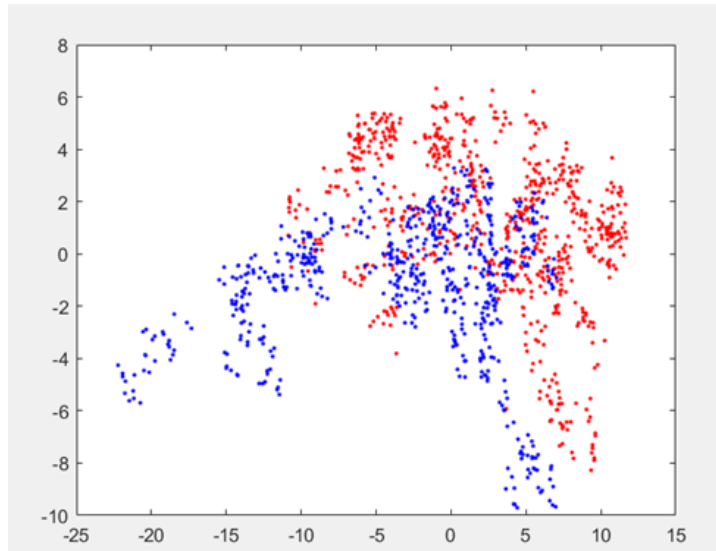


Figure 10: Banknote Authentication dataset scattered with PCA

```
error =  
  
0.2188
```

Figure 11: Using 5-fold CV for 1-nn on Banknote Authentication dataset

```
error =  
  
0.2188
```

Figure 12: Using LOOCV for 1-nn on Banknote Authentication dataset

- *Wine dataset:*

```
error =  
  
1.7544  
  
accuracy =  
  
98.2456
```

Figure 13: Classification error of K-nn (with $k = 1$) and test on the Wine dataset

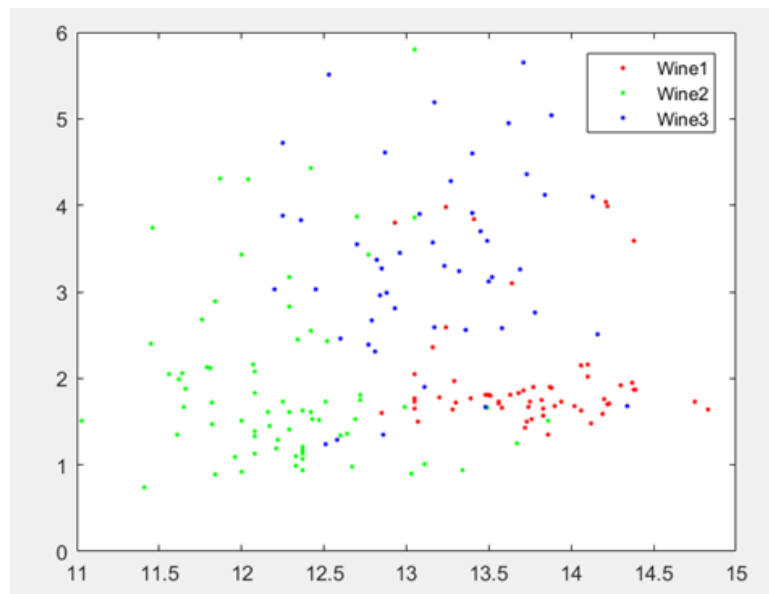


Figure 14: Wine dataset distribution

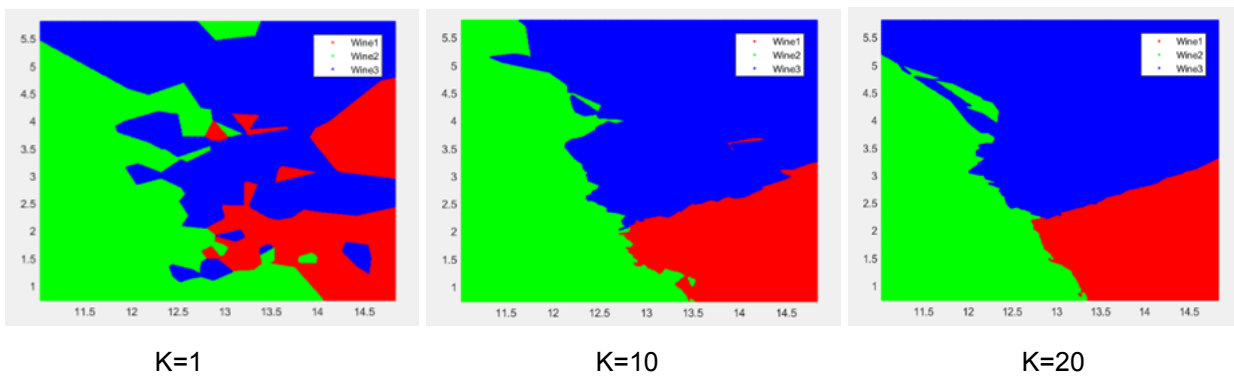


Figure 15: Decision boundary of K-nn classifier (with different k) on the Wine dataset

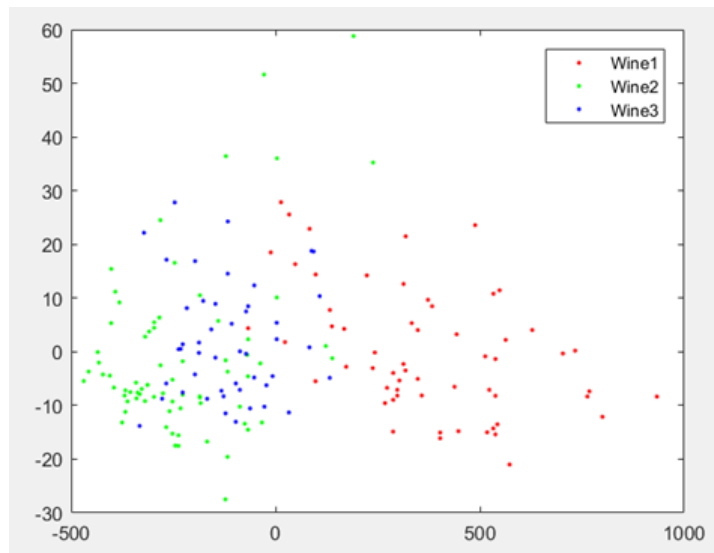


Figure 16: Wine dataset scattered with PCA

```
error =
```

```
10.5263
```

Figure 17: Using 5-fold CV for 1-nn on Wine dataset

```
error =
```

```
5.2632
```

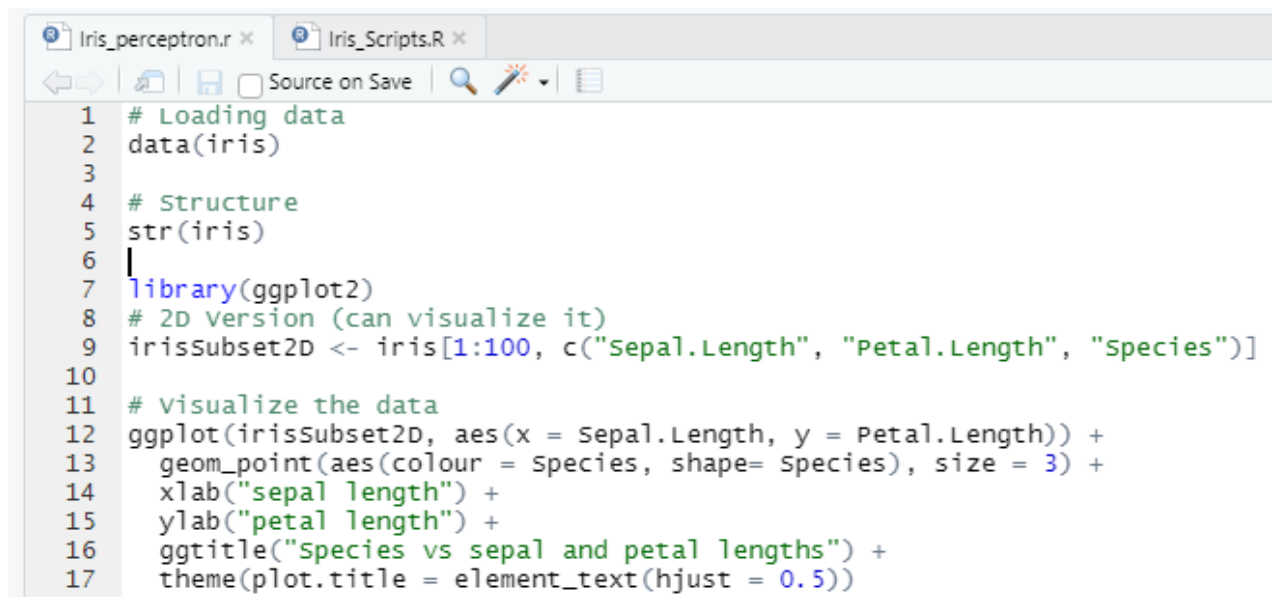
Figure 18: Using LOOCV for 1-nn on Wine dataset

II, Perceptron classifier

In this problem, we aim to implement Perceptron classifier.

- Initialize values for the weight vector w and fraction α :

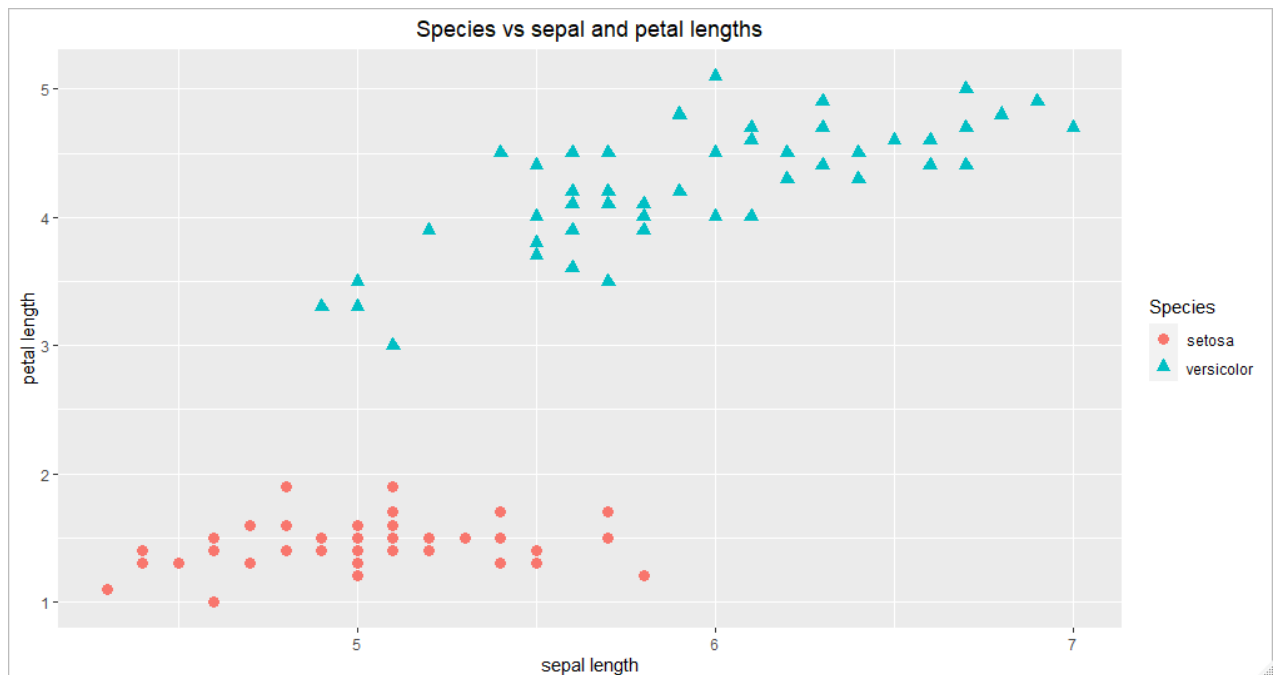
We will load the two flower classes Setosa and Versicolor from the Iris data set to evaluate our perceptron implementation. For visualization purposes, we will just take into account the two features sepal length and petal length.



```
1 # Loading data
2 data(iris)
3
4 # Structure
5 str(iris)
6
7 library(ggplot2)
8 # 2D Version (can visualize it)
9 irisSubset2D <- iris[1:100, c("Sepal.Length", "Petal.Length", "Species")]
10
11 # visualize the data
12 ggplot(irisSubset2D, aes(x = Sepal.Length, y = Petal.Length)) +
13   geom_point(aes(colour = Species, shape = Species), size = 3) +
14   xlab("sepal length") +
15   ylab("petal length") +
16   ggtitle("Species vs sepal and petal lengths") +
17   theme(plot.title = element_text(hjust = 0.5))
```

```
> iris <- read.csv("C:/Users/admin/Downloads/iris.data", header=FALSE)
> view(iris)
> # Loading data
> data(iris)
>
> # Structure
> str(iris)
'data.frame': 150 obs. of 5 variables:
 $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
> library(ggplot2)
> irisSubset2D <- iris[1:100, c("Sepal.Length", "Petal.Length", "Species")]
> # visualize the data
> ggplot(irisSubset2D, aes(x = Sepal.Length, y = Petal.Length)) +
+   geom_point(aes(colour = Species, shape = Species), size = 3) +
+   xlab("sepal length") +
+   ylab("petal length") +
+   ggtitle("Species vs sepal and petal lengths") +
+   theme(plot.title = element_text(hjust = 0.5))
> |
```

We can visualize via a two-dimensional scatter plot:



Perceptron learning algorithm in R

```

19 # Map Setosa to -1 and Versicolor to 1
20 irisSubset2D$class <- lapply(irisSubset2D$species, function(x) {
21   if(x == 'setosa')
22     irisSubset2D$class <- -1
23   else if(x == 'versicolor')
24     irisSubset2D$class <- 1
25   else
26     irisSubset2D$class <- NULL
27 })
28
29 # Break down into input (x) and output(y)
30 x <- irisSubset2D[, c("Sepal.Length", "Petal.Length")] # Input Matrix
31 y <- irisSubset2D$class # Output Vector
32
33 # Perceptron Learning Algorithm
34 perceptron <- function(X, y, numEpochs) {
35   results <- list()
36   w <- runif(ncol(X), -10, 10) #Initialize weights
37
38   # For loop - number of generations(epochs) - number of times dataset is ran through
39   for(j in 1:numEpochs) {
40     predictedResult <- numeric(length=100) # Initialize predictedResult vector
41     numIncorrect = 0 # Keeps track of # of misclassified points
42
43     # For loop - loop through dataset
44     for(i in 1:length(y)) {
45       xi = as.numeric(unlist(X[i,])) # Convert dataframe to vector
46       predictedResult[i] = sign(w %*% xi) # Predict the point
47
48       # If predicted point is incorrect - change weight
49       if(predictedResult[i] != y[i]) {
50         numIncorrect = numIncorrect + 1 # Add one to # of misclassified points
51         w <- w + as.numeric(y[i]) * xi # Update the weight w <- w + wixi
52       }
53     }
54     # Print results of this generation(epoch)
55     cat("\nEpoch #: ", j)
56     cat("\nNumber Incorrect: ", numIncorrect)
57     cat("\nFinal weight: ", w)
58   }
59 }

```

Data	
iris	150 obs. of 5 variables
irissubset2D	100 obs. of 4 variables
x	100 obs. of 2 variables
y	List of 100
Values	
results	NULL
Functions	
perceptron	function (X, y, numEpochs)

And we run the algorithm

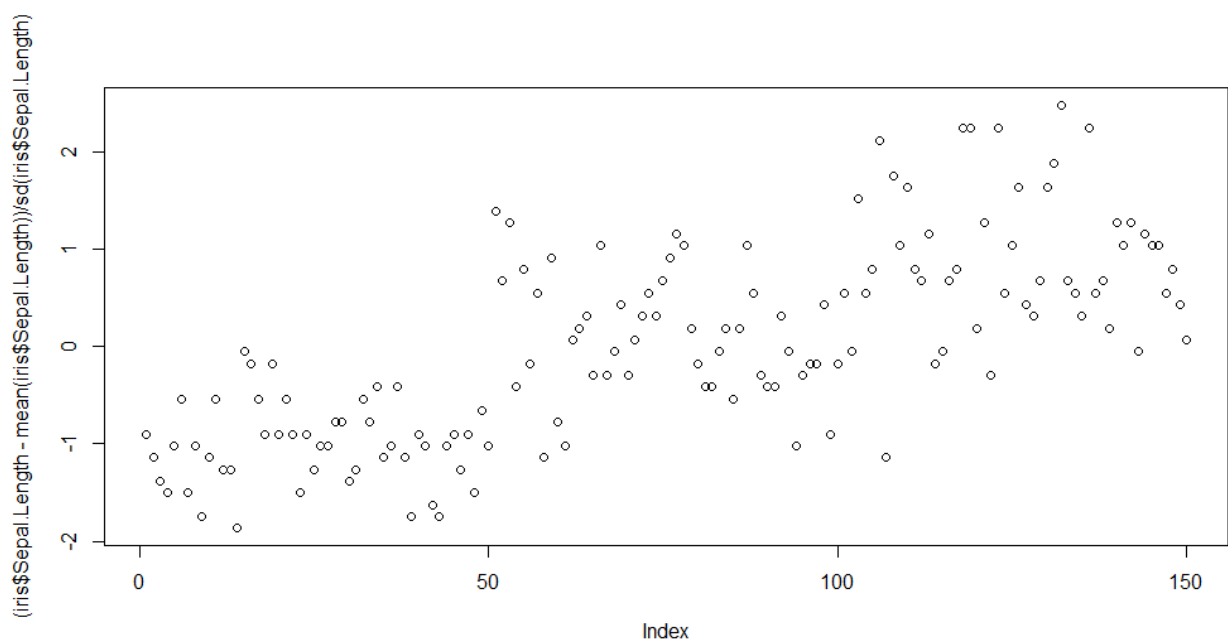
```
> # Run Perceptron algorithm
> results = perceptron(X,y, 8)

Epoch #: 1
Number Incorrect: 2
Final weight: 4.120964 -4.035452
Epoch #: 2
Number Incorrect: 2
Final weight: 6.020964 -0.735452
Epoch #: 3
Number Incorrect: 3
Final weight: 3.020964 1.164548
Epoch #: 4
Number Incorrect: 2
Final weight: 4.920964 4.464548
Epoch #: 5
Number Incorrect: 3
Final weight: 1.920964 6.364548
Epoch #: 6
Number Incorrect: 2
Final weight: 2.520964 8.464548
Epoch #: 7
Number Incorrect: 3
Final weight: -0.3790356 9.864548
Epoch #: 8
Number Incorrect: 2
Final weight: 0.1209644 12.06455
> |
```

- Plot linear classifiers for Iris dataset (using PCA or SVD to reduce the number of dimensions to 2D)

Centre and scale

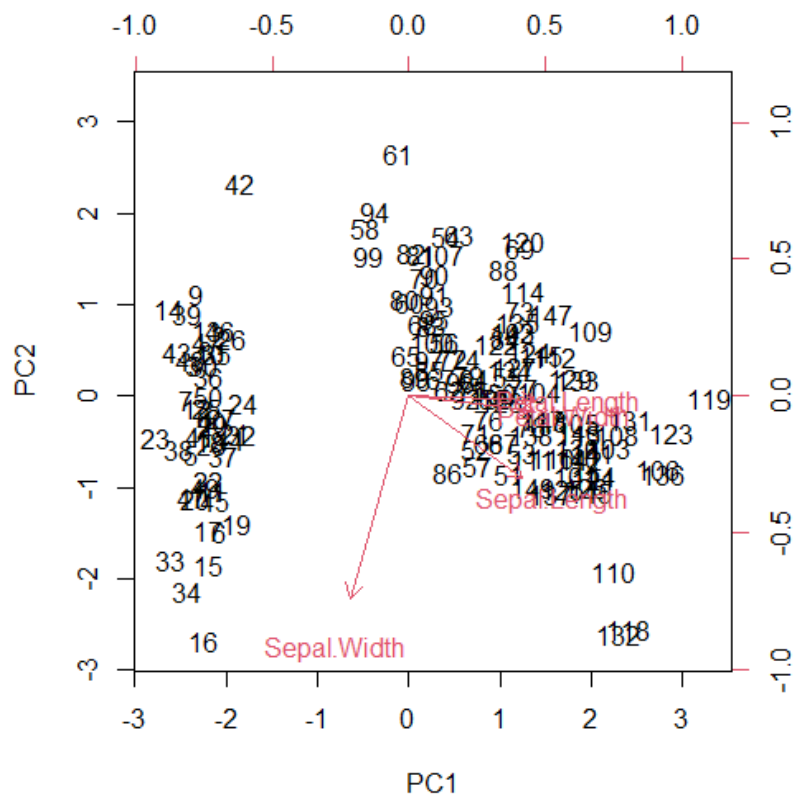
```
> data(iris)
> head(iris)
  Sepal.Length Sepal.width Petal.Length Petal.width Species
1          5.1         3.5         1.4         0.2   setosa
2          4.9         3.0         1.4         0.2   setosa
3          4.7         3.2         1.3         0.2   setosa
4          4.6         3.1         1.5         0.2   setosa
5          5.0         3.6         1.4         0.2   setosa
6          5.4         3.9         1.7         0.4   setosa
> myPr <- prcomp(iris[, -5])
> View(iris)
> myPr <- prcomp(iris[, -5], scale = TRUE)
> plot(iris$Sepal.Length, iris$Sepal.width)
> plot(scale(iris$Sepal.Length), scale(iris$Sepal.width))
> plot((iris$Sepal.Length - mean(iris$Sepal.Length)) / sd(iris$Sepal.Length))
```



Plotting:

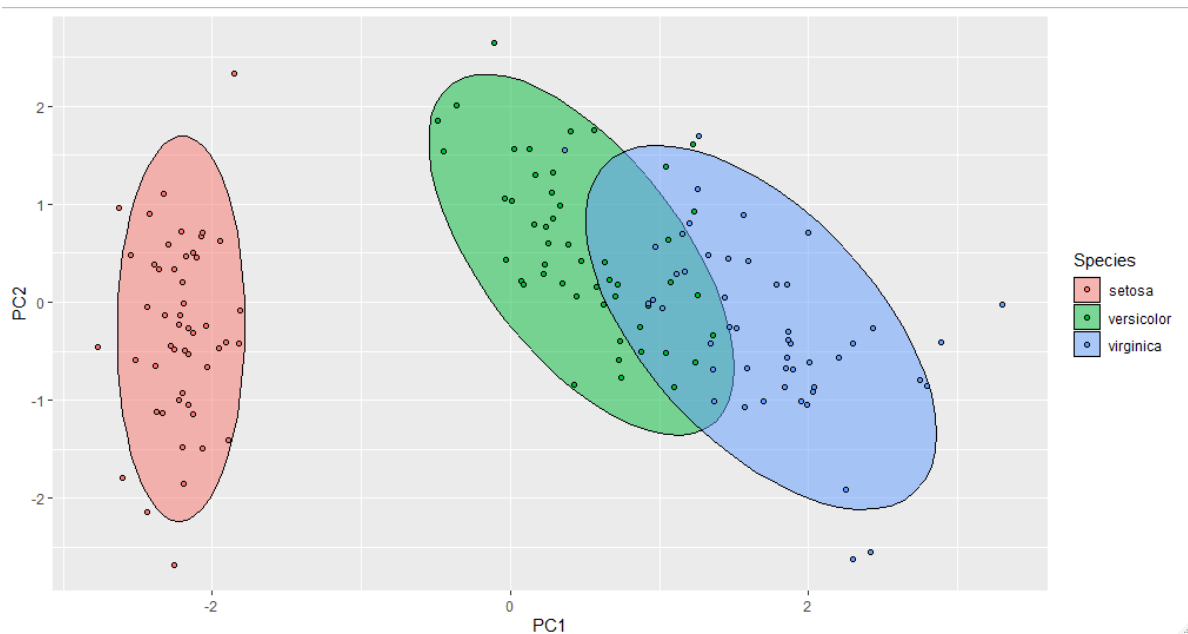
biplot

```
> plot(myPr, type = "l")
> biplot(myPr)
> biplot(myPr, scale = 0)
> |
```



ggplot

```
> iris2 <- cbind(iris, myPr$x)
> library(ggplot2)
>
> ggplot(iris2, aes(PC1, PC2, col = species, fill = species)) +
+   stat_ellipse(geom = "polygon", col = "black", alpha = 0.5) +
+   geom_point(shape = 21, col = "black")
> |
```



- Comment on the convergence rate of Perceptron on Iris? How to make it converging faster?

Convergence is one of the perceptron's main challenges even though it correctly recognizes the two Iris flower classes.

If a linear hyperplane can be used to divide the two classes, the perceptron learning rule converges.

Nevertheless, unless we define a maximum number of epochs, the weights will never stop updating if classes cannot be completely separated by such a linear decision boundary.