Week 6: k Nearest Neighbour (kNN) Classification

Data Science Certificate Program

Ryerson University

Bora Caglayan

4 Feb, 2015

Outline

- Exam Review
- Binary Classification Problem
- Performance Measures in Classification
- k Nearest Neighbour Algorithm (kNN)
- Weighted kNN

Exam Review

- Expected Scores are lower than actual exam scores.
- Problems with <25% correct answers:
  - Training-dataset comparison problem in kNN.
  - Pred(25).
  - Best subset regression question.

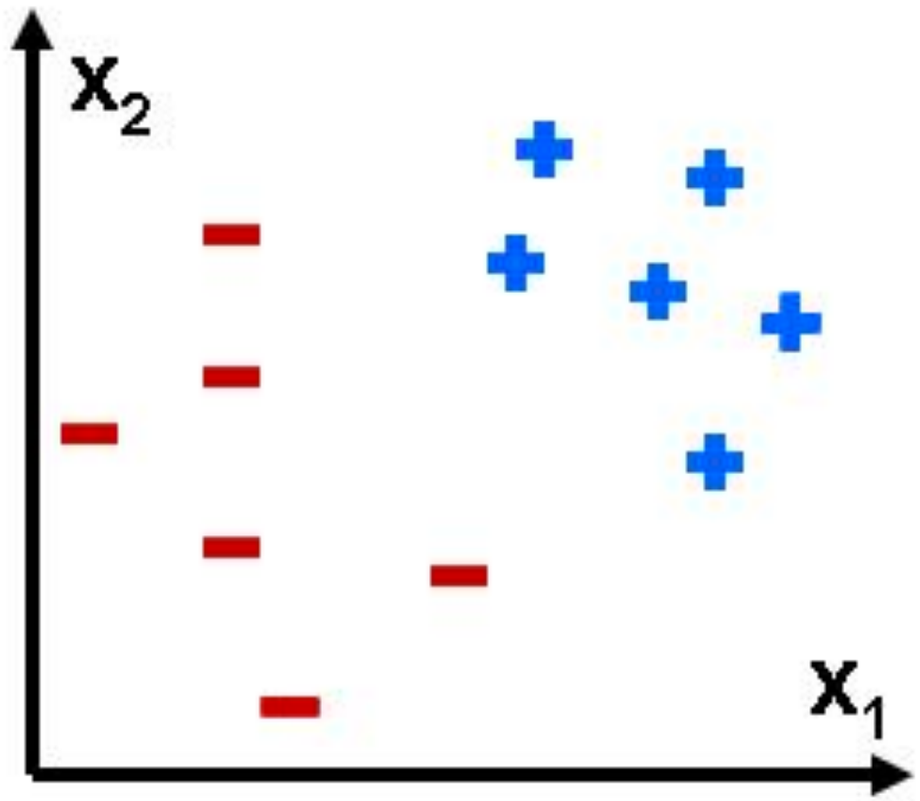
Binary Classification Problem

- **Two classes** $Y = \{0,1\}$
- Our goal is to learn to classify correctly two types of examples
  - Class 0 – labeled as 0,
  - Class 1 – labeled as 1
- We would like to learn $f : X \rightarrow \{0,1\}$
- **Zero-one error (loss) function**

$$Error_1(\mathbf{x}_i, y_i) = \begin{cases} 1 & f(\mathbf{x}_i, \mathbf{w}) \neq y_i \\ 0 & f(\mathbf{x}_i, \mathbf{w}) = y_i \end{cases}$$

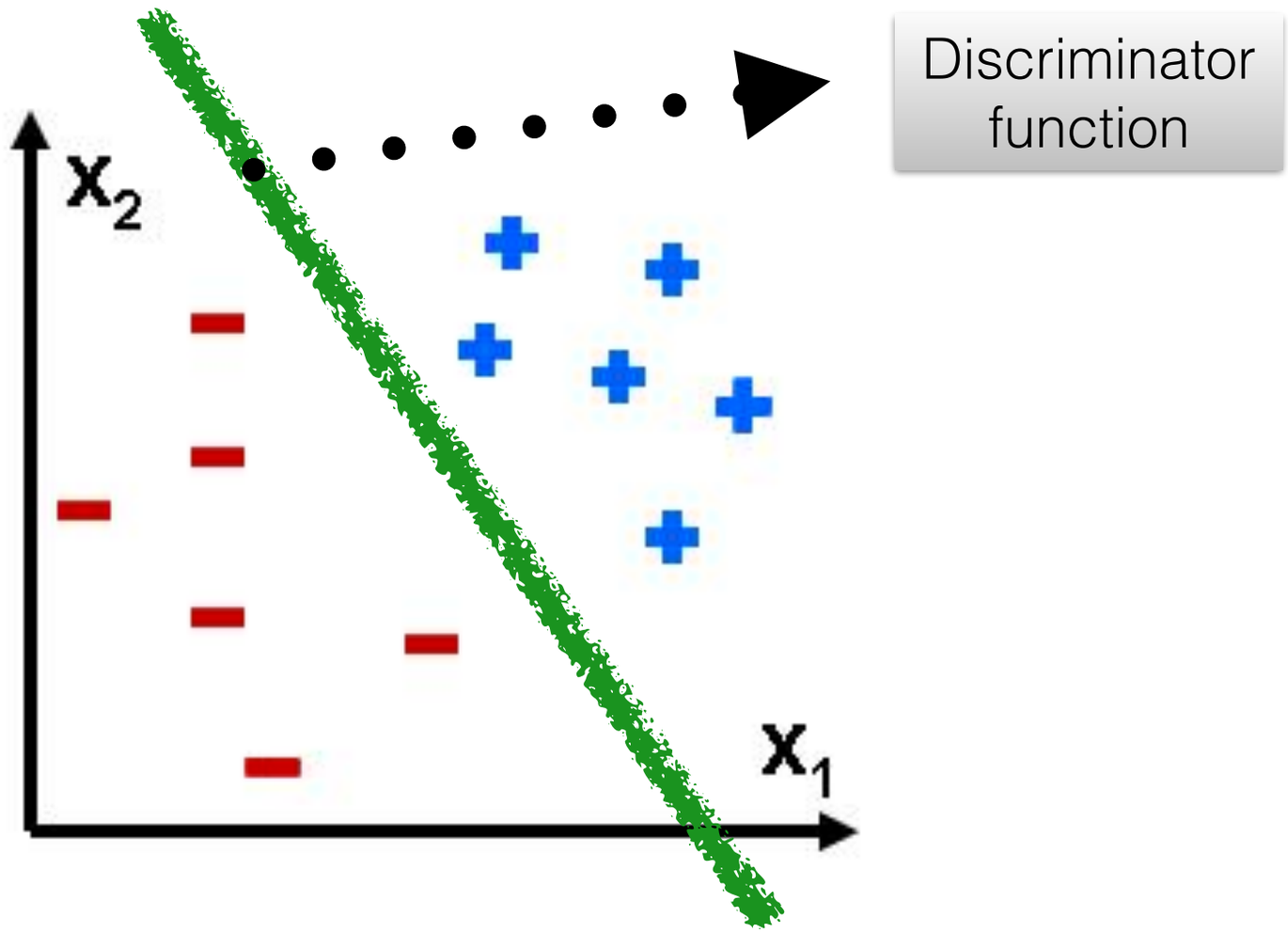
- Error we would like to minimize: $E_{(x,y)}(Error_1(\mathbf{x}, y))$
- **First step:** we need to devise a model of the function

Example



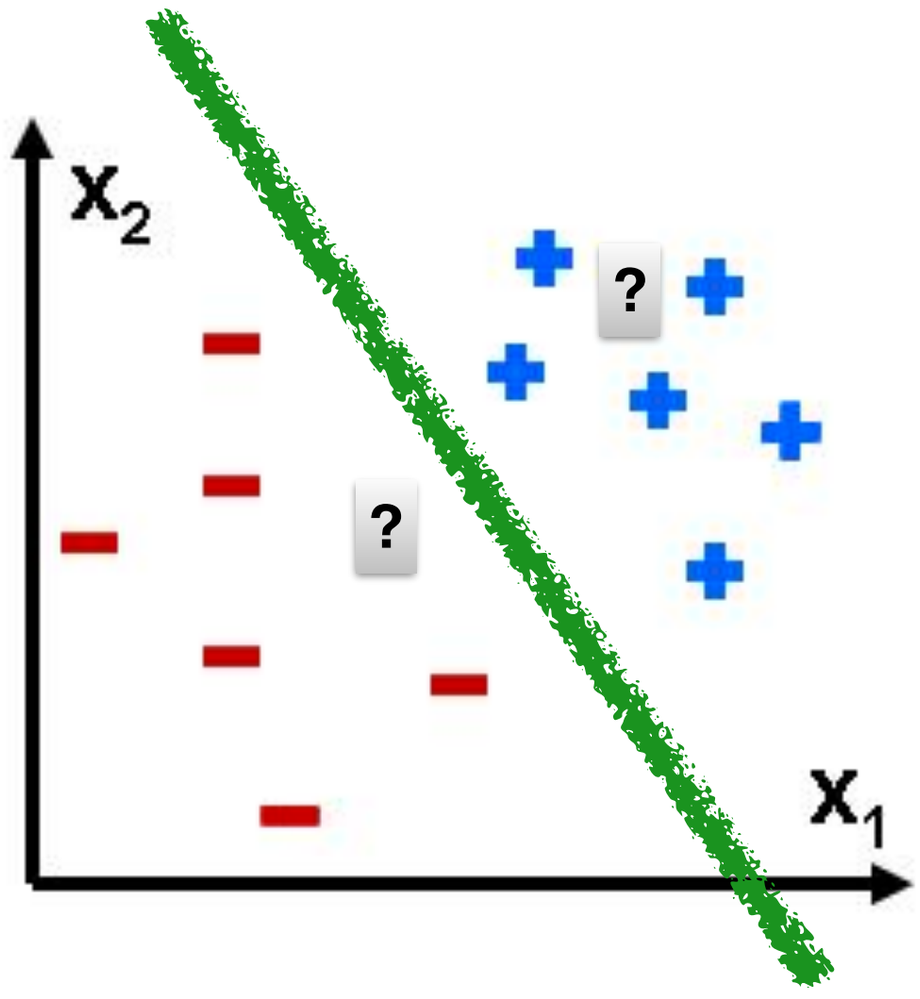
Given a set of instances with
Two features: x_1, x_2
Two labels: -,+

Example



Learn the pattern of the - and + classes.

Example



Classes of new items are predicted.

Performance Measures in Classification

Classes		Actual	
		yes	no
Prd	Yes	TP:A	FP:B
	no	FN:C	TN:D

$$pd = \frac{A}{A+C}$$

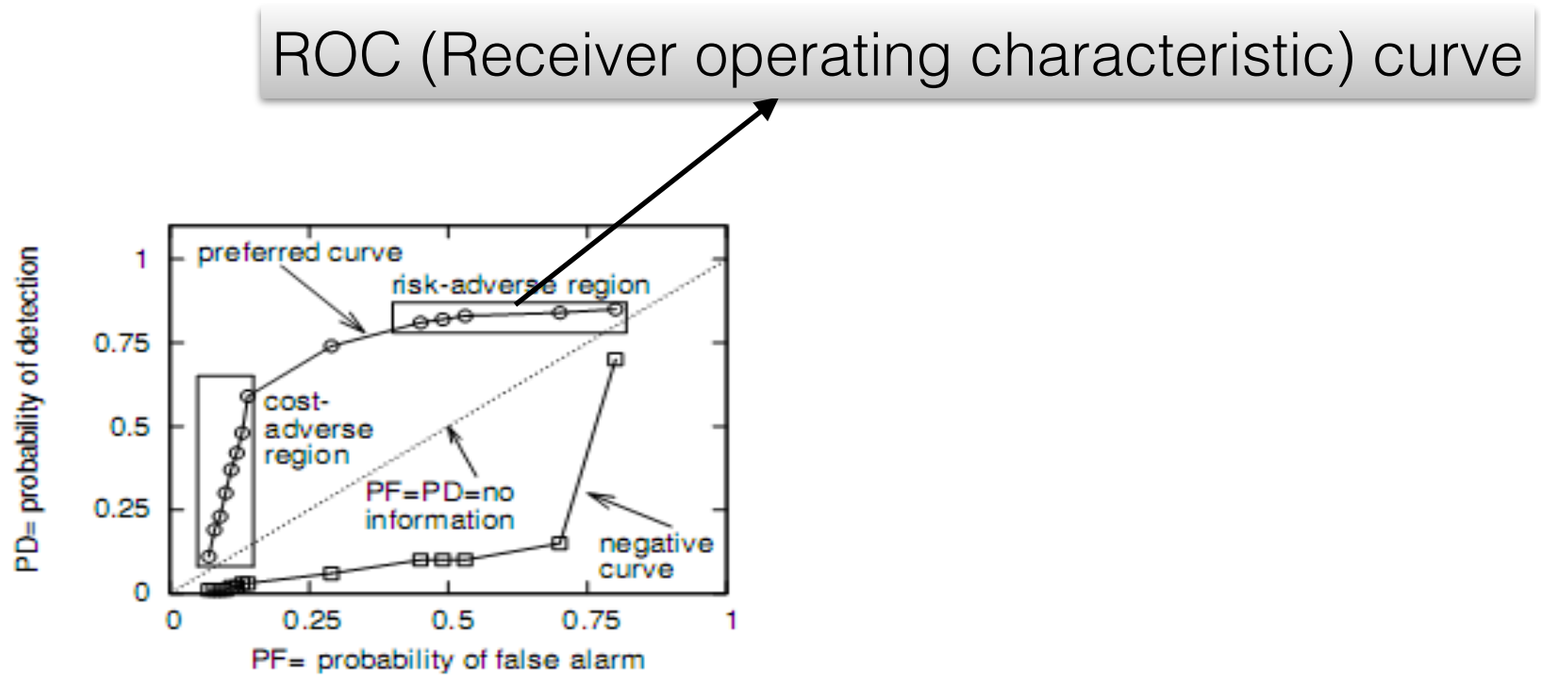
$$pf = \frac{B}{B+D}$$

$$Precision = \frac{A}{A+C}$$

$$Recall = pd = \frac{A}{A+B}$$

$$f\_measure = 2 * \frac{Precision * Recall}{(Precision + Recall)}$$

Performance Measures in Classification

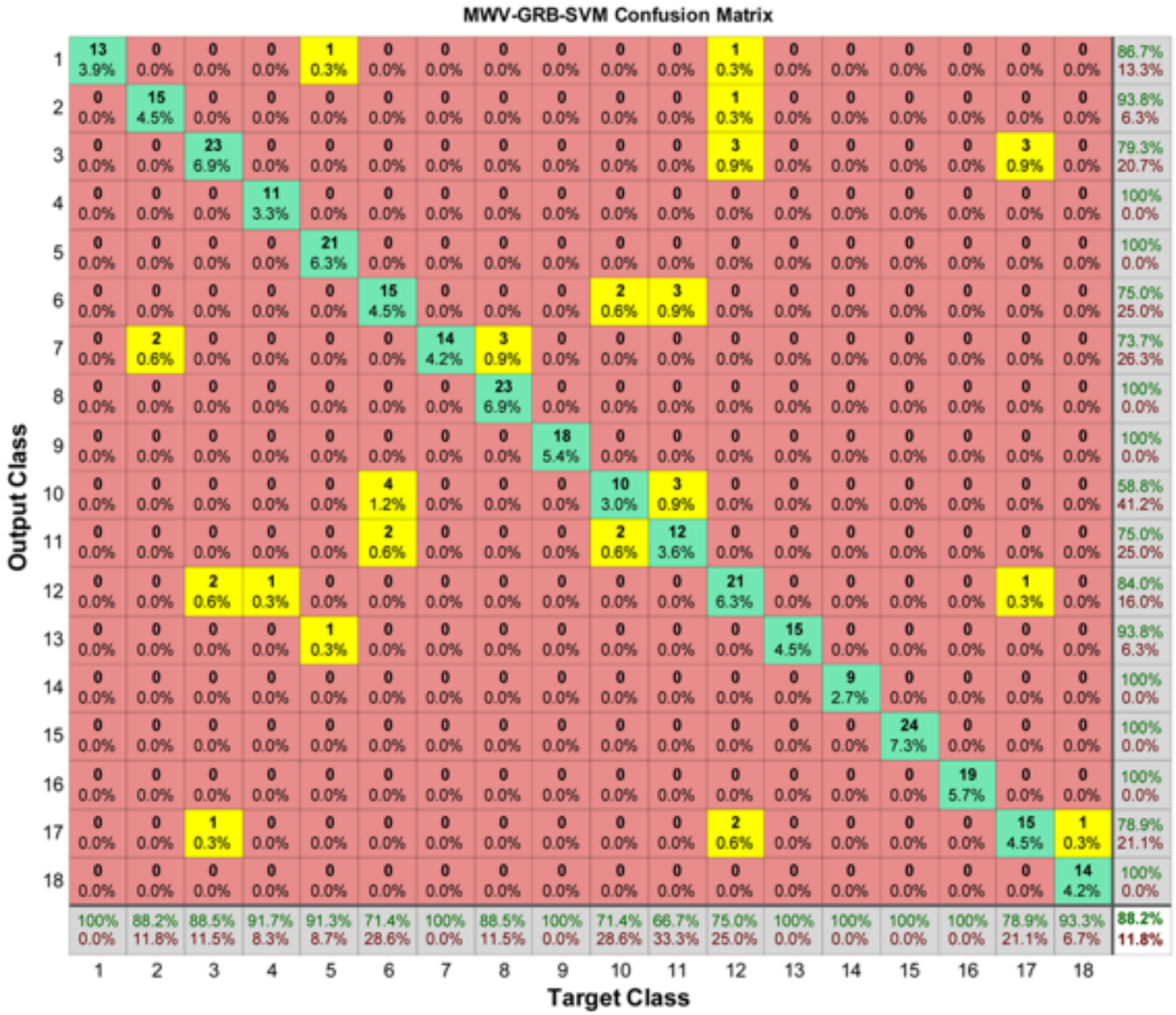


	precision	pd	pf	effort[#]
Early in a contractor/client relationship	hi			
Risk adverse (e.g. airport bomb detection, morning sickness)		hi	hi	
Cost adverse (e.g. budget conscious)		med	lo	
Arisholm and Briand [2006]				$< pd$

[#]effort = LOC in the modules predicted to be faulty

Performance Measures in Classification

Sample Confusion Matrix for 2+ Class Prediction Task



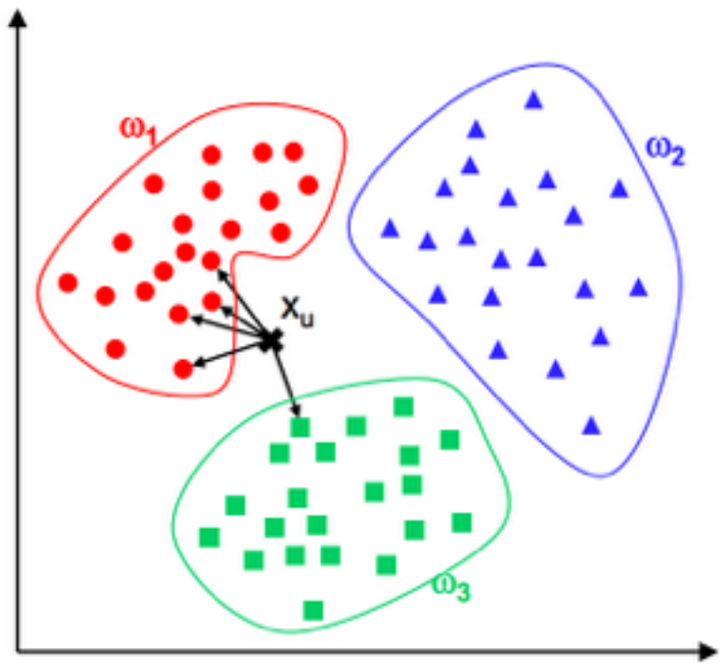
Fraction of correctly classified instances (fraccorrect)

$$fraccorrect = \frac{\sum_{i=1}^{N}(C_{ii})}{\sum_{j=1}^{N}(\sum_{i=1}^{N}(C_{ij}))}$$

k Nearest Neighbour (kNN) Algorithm

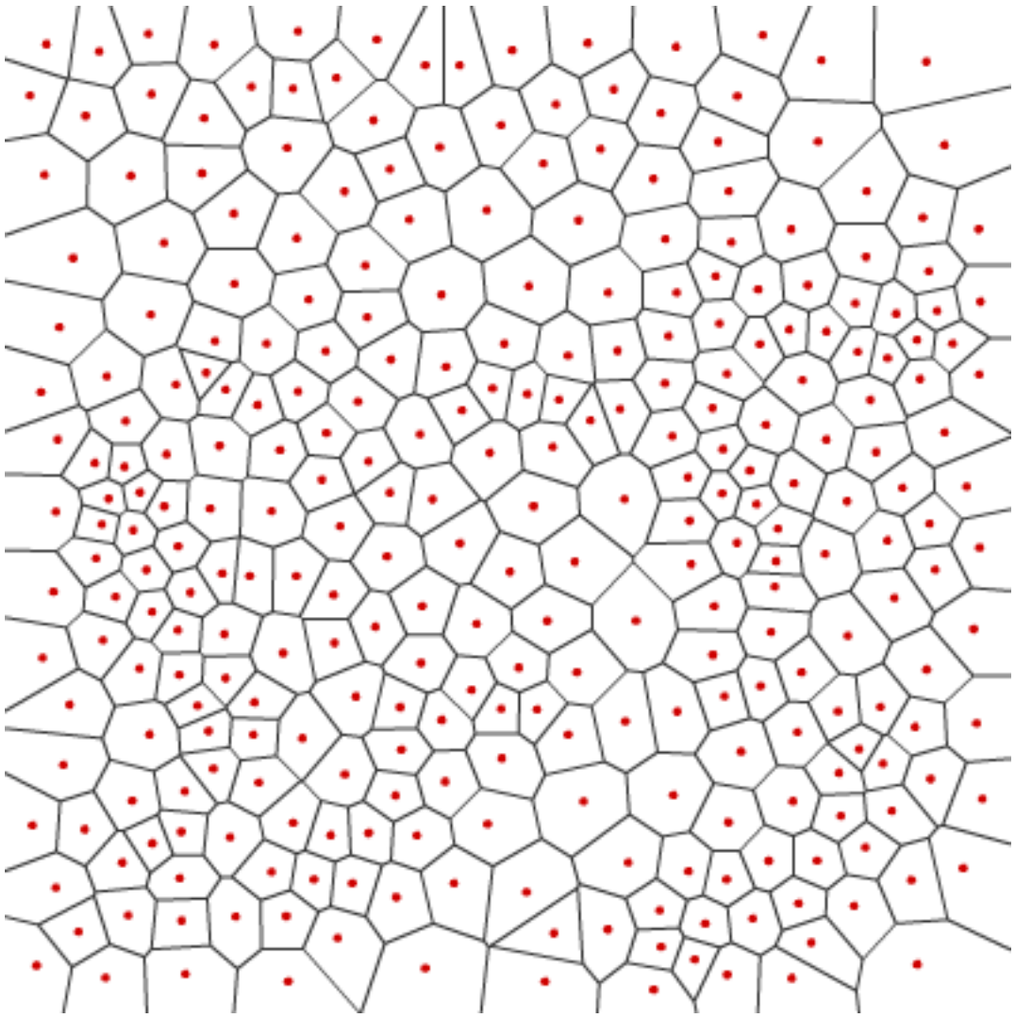
Definition

- The kNN rule is a very intuitive method that classifies unlabeled examples based on their similarity to examples in the training set
- For a given unlabeled example $x_u \in \Re^D$, find the k "closest" labeled examples in the training data set and assign x_u to the class that appears most frequently within the k-subset
- The kNN only requires
  - An integer k
  - A set of labeled examples (training data)
  - A metric to measure "closeness"
- Example
  - In the example here we have three classes and the goal is to find a class label for the unknown example x_u
  - In this case we use the Euclidean distance and a value of $k = 5$ neighbors
  - Of the 5 closest neighbors, 4 belong to ω_1 and 1 belongs to ω_3, so x_u is assigned to ω_1, the predominant class



k Nearest Neighbour

Every instance have some neighbourhood.
(Voronoi Tessaract)

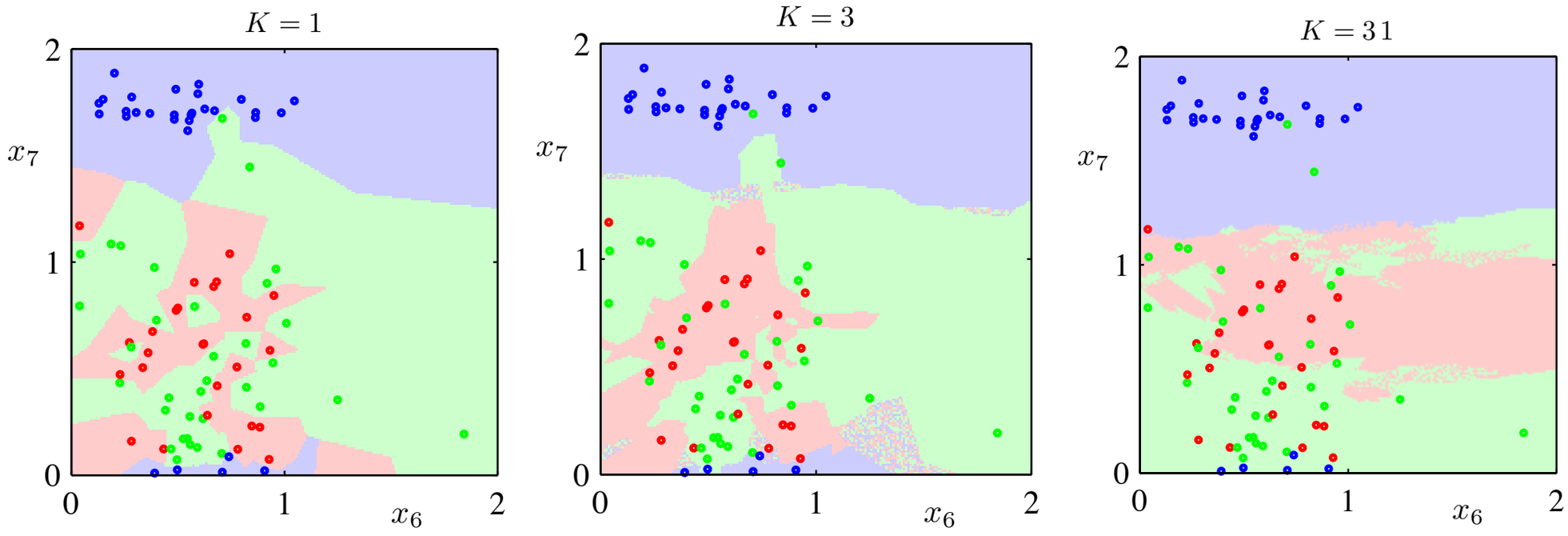


k Nearest Neighbour (kNN) Algorithm

Every instance have some neighbourhood.
(Voronoi Tessaract)

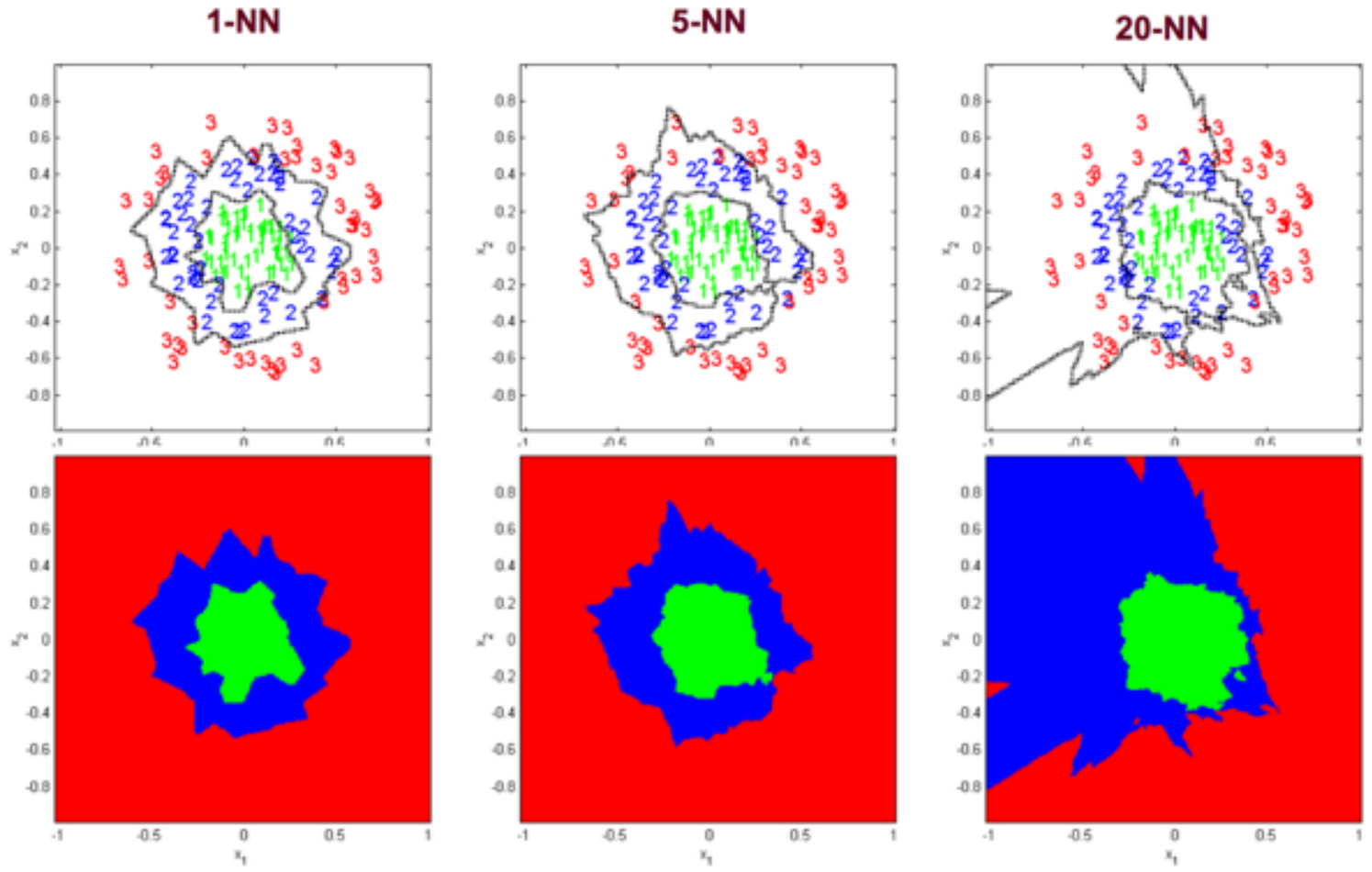
k Nearest Neighbour (kNN) Algorithm

Larger values of k reduce the effect of noise, but make boundaries between classes less distinct.



k Nearest Neighbour (kNN) Algorithm

Larger values of k reduce the effect of noise, but make boundaries between classes less distinct.



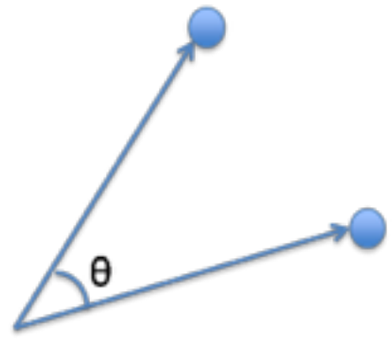
Distance Measures

- Eucledian Distance

$$\|\mathbf{x}-\mathbf{y}\| = \sqrt{(x_1-y_1)^2+\ldots+(x_p-y_p)^2}$$

- Cosine Similarity

$$sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$



Scaling Attributes

- Attributes with very different scales might be an important problem for the models. There are several methods to re-scale the attributes.

- **Normalization:** Rescale attribute so that its minimum is 0 (or –1) and its maximum is 1.

$$x_{new} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

- **Standardization:** Rescale attribute so that its mean is 0 and its standard deviation is 1.

$$z = \frac{x - \mu}{\sigma}$$

μ = Mean
σ = Standard Deviation

k Nearest Neighbour

Strengths of kNN

- Easy training
- Easy implementation
- Performs well for problems with many classes.

Weaknesses of kNN

- Slow testing, may not be suitable for real-time applications.
- Dealing with many dimensions is hard.
- Large storage requirements
- Curse of dimensionality

Curse of Dimensionality

- Distance usually relates to all the attributes and assumes all of them have the same effects on distance

- The similarity metrics do not consider the relation of attributes which result in inaccurate distance and then impact on classification precision. Wrong classification due to presence presence of many irrelevant irrelevant attributes attributes is often termed as the curse of dimensionality.

- For example: Each instance is described by 20 attributes out of which only 2 are relevant in determining the classification of the target function. In this case, instances that have identical values for the 2 relevant attributes may nevertheless be distant from one another in the 20 dimensional instance space.

Weighted kNN

- Approach 1
  - Associate weights with the attributes
  - Assign weights according to the relevance of attributes
    - Assign random weights
    - Calculate the classification error
    - Adjust the weights according to the error
    - Repeat till acceptable level of accuracy is reached
- Approach 2
  - Backward Elimination
  - Starts with the full set of features and greedily removes the one that most improves performance, or degrades performance slightly

Attribute Weighting

Idea: All attributes might not be equally important

1. Assign random weight to each attribute.
2. Divide the number of training examples into N sets.
3. Train the weights by cross validation.
4. For every test instance change the weights of the attributes until some condition is met:

$$W_i = W_i + \alpha * ERROR * value$$

$Error$: enum(actual class) - enum(predicted class)
$value$: value of the attribute
α : Learning rate

Backward Elimination

1. Remove an attribute.
2. Check if performance decreases:
    A. If performance decreases, add the attribute again.
    B. Otherwise, remove attribute permanently.
3. Repeat until convergence.

Summary of kNN

- **Definition:** Predict the value of a class based on one or more numeric variables from the classes of .
- **Exploratory analysis:** Check correlations, scatter plot all the pairs, check the ranges of attribute.
- **Preprocessing:** Changing scales,
- **Algorithms:** Gradient descent, closed form solution with linear algebra, (optional: changed cost function for *regularized regression, stepwise regression)*.
- **Experiment:** k-fold cross validation, random split, leave-one-out
- **Performance criteria:** recall, accuracy, ROC curve
- **Advantages:** Simplicity, low training computational cost. Good baseline model. Easy to apply on distributed frameworks.
- **Disadvantage:** May not fit the data if. Very high testing computation cost.

Other Related Concepts

- Other methods such as PCA can be used to reduce the number of features.
- Instance weighted regression.
- Different distance measures.
- Faster algorithms for distance estimations.

Questions

Can we use kNN for regression?

What would happen if k is very large?

References

- http://metacademy.org/graphs/concepts/k_nearest_neighbors

- http://www.cs.waikato.ac.nz/ml/weka/

Week 6 Application Part

March 5, 2015

Preparation

```
library("RWeka") # rweka (embedded Weka software)
library("kknn") # knn library
```

Control structure overview

For loops:

```
for(i in seq(1,5)){
    print(i);
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Control structure overview

While loops:

```
i <- 0;
while(i < 5) {
  i <- i+1;
  print(i);
  }
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Initially, we will implement kNN the hard way.

Without additional libraries

Step 1: Training and testing samples:

Let us define training and testing samples.

```
x <- c(1, 7, 3, 1)
y <- c(4, 1, 12, 7)
class <- c("o", "o", "x", "x")
training <- data.frame(x=x, y=y, class=class,
                       stringsAsFactors=F)

test <- c(5, 4)
```

Step 2: Visualize the dataset:

```
plot(training[c(1,2),1], training[c(1,2),2], pch="o",
     xlim=c(0,8), ylim=c(0,13), col="red",xlab="x",
     ylab="y")
points(training[c(3,4),1], training[c(3,4),2], pch="x",
       col="blue")
points(test[1], test[2], pch="%", col="green")
```

Step 3: Setup initial minimum distances:

```
k= 2;
min_distances <- data.frame(d=rep(c(Inf), k),
                            c=rep(c("unknown"), k),
                            stringsAsFactors=F)
```

Step 4: Training and find the min distances:

```
for(i in 1:nrow(training)) {
    row <- training[i,];
    distance <- sqrt((test[1] - row$x)**2 +
                         (test[2] - row$y)**2);
    if(distance< max(min_distances$d)){
      min_distances[which.max(min_distances$d),"d"] <-
        distance;
      min_distances[which.max(min_distances$d),"c"] <-
        row$class;
    }
}
```

Step 5: Find the most common class

```
table(min_distances$c)
```

```
o
2
```

Here is an easier method

Weka is a well known machine learning platform. You may try weka implementation of knn!

http://www.cs.waikato.ac.nz/ml/weka/

```
iris <- read.arff(system.file("arff", "iris.arff",
                              package = "RWeka"))
classifier <- IBk(class ~., data = iris,
          control = Weka_control(K = 20))
evaluate_Weka_classifier(classifier, numFolds = 10)
```

Scaling variables

Standardization:

```
a <- c(1, 100, 10000)
scale(a)
```

```
             [,1]
[1,] -0.5859457
[2,] -0.5687120
[3,]  1.1546577
attr(,"scaled:center")
[1] 3367
attr(,"scaled:scale")
[1] 5744.56
```

Scaling variables

Between 0 to 1:

```
scale(a, center = min(a), scale = max(a) - min(a))
```

```
            [,1]
[1,] 0.00000000
[2,] 0.00990099
[3,] 1.00000000
attr(,"scaled:center")
[1] 1
attr(,"scaled:scale")
[1] 9999
```

Lab Preparation

```
library("RWeka") # rweka (embedded Weka software)
library("kknn") # knn library
diabetes <- read.arff(system.file("arff", "diabetes.arff",
                                  package = "RWeka"))
```

Lab problems:

1. Visualize the classes for the distributions of insu and age.
2. Do knn with weka interface using diabetes data.
3. Scale the numeric columns of diabetes between 0 and 1.
4. Scale the numeric columns of diabetes with mean=0 and stddev=1.
5. Do knn with weka interface and updated column scales. Do results change?
6. (optional) Implement knn without using any third party libraries.