EC999: K-Nearest Neighbours (KNN)

Thiemo Fetzer

University of Chicago & University of Warwick

May 4, 2017

K Nearest Neighbours

- The first classification algorithm we present is called **K nearest neighbors** or KNN.
- KNN assigns a label to a new observation y_i based on "what is the most common class around the vector x_i"?
- The idea is that, if you have similar $X's$ values, then also the label y should be similar.
- One dimensional space [draw example]

K Nearest Neighbours

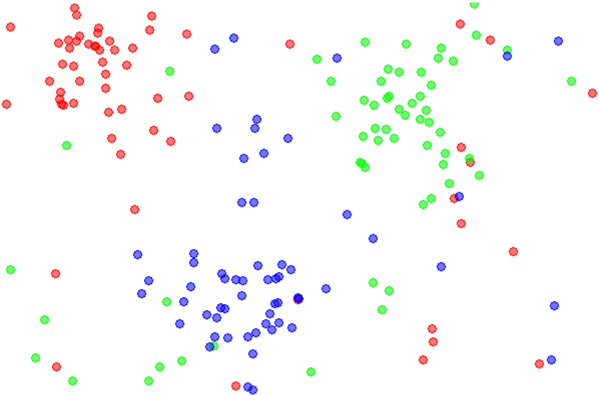


Figure: A three class nearest neighbours visual example with two features X_1, X_2

K Nearest Neighbours

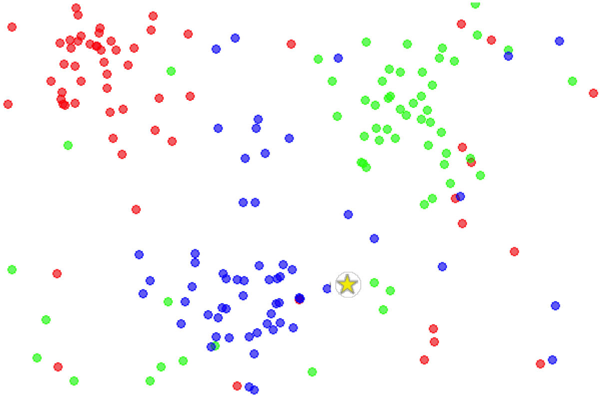


Figure: A three class nearest neighbours visual example with two features X_1, X_2

K Nearest Neighbours

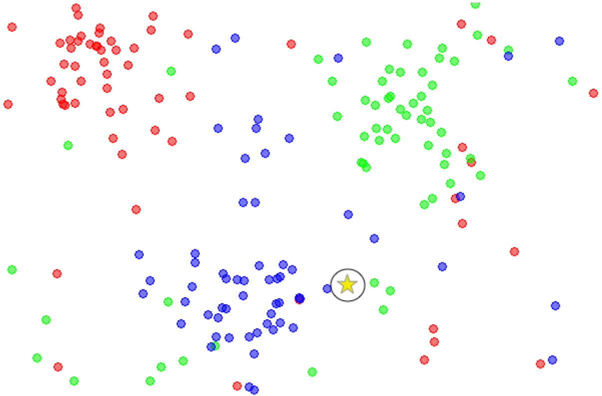


Figure: A three class nearest neighbours visual example with two features X_1, X_2

K Nearest Neighbours

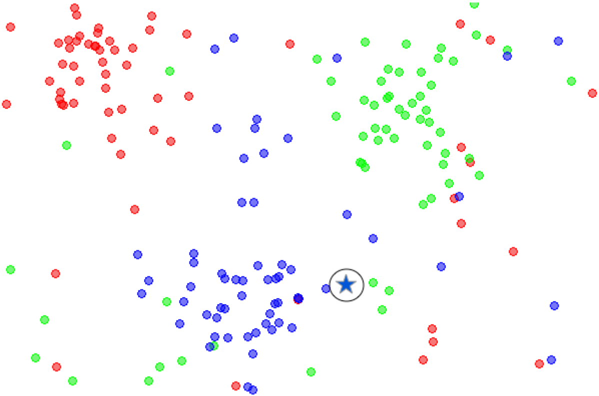


Figure: A three class nearest neighbours visual example with two features X_1, X_2

K Nearest Neighbours

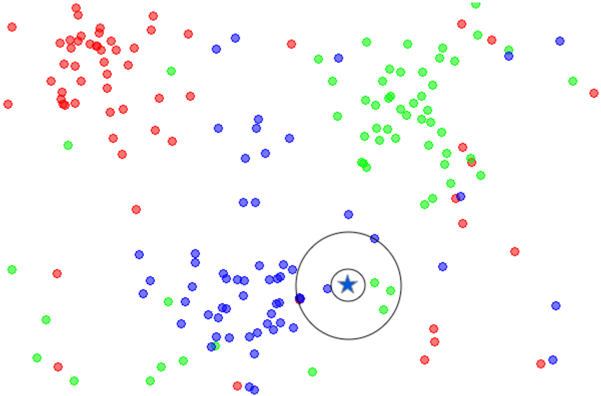


Figure: A three class nearest neighbours visual example with two features X_1, X_2

K Nearest Neighbours

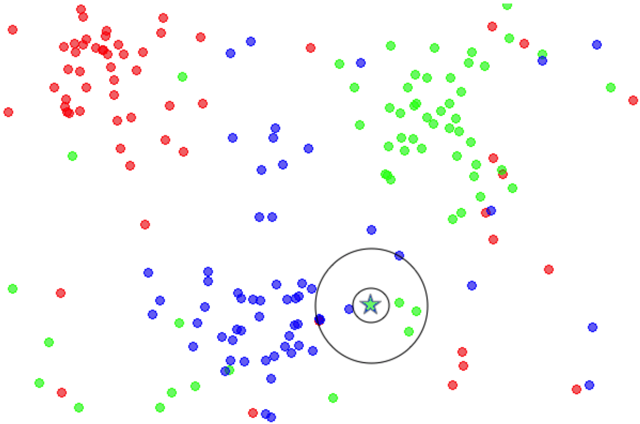


Figure: A three class nearest neighbours visual example with two features X_1, X_2

K Nearest Neighbours

Algorithm (*K Nearest Neighbours*)

1. *Store all the training data* $(y_i, x_{i1}, ..., x_{ip})$
2. *For each new point* $x_q = (x_{q1}, ...x_{qp})$, *find the* K **nearest** *neighbours in the training set, indexed by* $\mathcal{N}_0$.
3. *For each possible value* $j \in \mathcal{C}$, *compute*

$$\hat{P}(Y = j|X = x_q) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} I(y_j = j)$$

4. *Assign the class* j *for which* $\hat{P}(Y = j|X = x_q)$ *is maximal.*

K Nearest Neighbours Estimates Complicated Functions

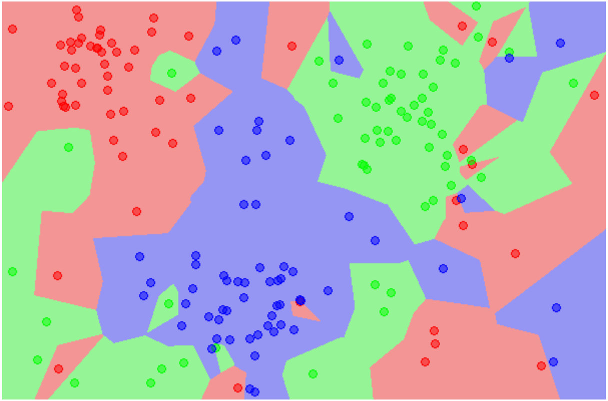


Figure: A three class nearest neighbours visual example with two features X_1, X_2

Some thoughts on KNN

- KNN is among the simplest of all machine learning algorithms.
- Its explicitly non-parametric (you dont estimate any coefficients)
- KNN is referred to as a lazy learning algorithm: the function is only approximated locally and all computation is deferred until classification.
- So there is actually no explicit training step.
- Results from KNN are sensitive to the local structure of the data (example highly skewed, “majority voting” distorted, weights).
- Its (potentially) too slow, as it may be impossible to hold all the training data in memory.

Applications in Research: Fetzer and Marden, 2014

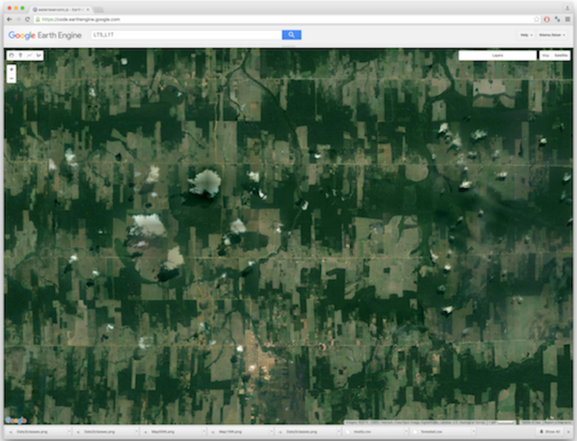


Figure: Classifiying historical landsat imagery into Forest or Non Forest Status.

[See forested.R]

Applications in Research: Fetzer and Marden, 2014

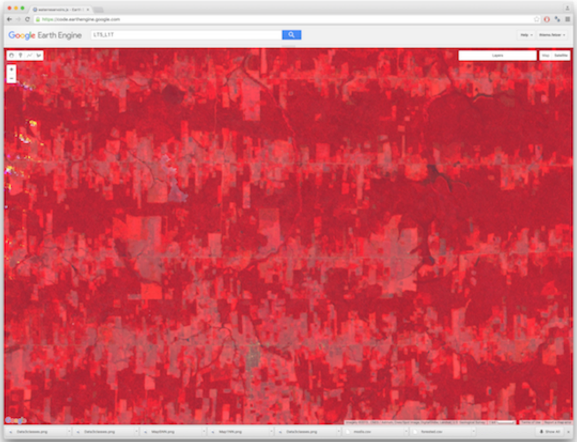


Figure: Classifiying historical landsat imagery into Forest or Non Forest Status.

[See forested.R]

Applications in Research: Fetzer and Marden, 2014

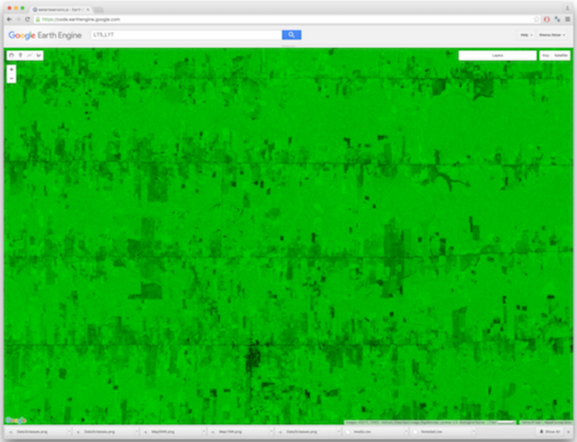


Figure: Classifiying historical landsat imagery into Forest or Non Forest Status.

[See forested.R]

Applications in Research: Fetzer and Marden, 2014

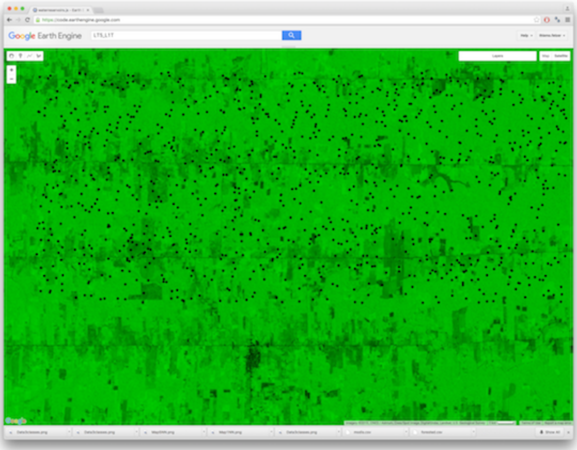
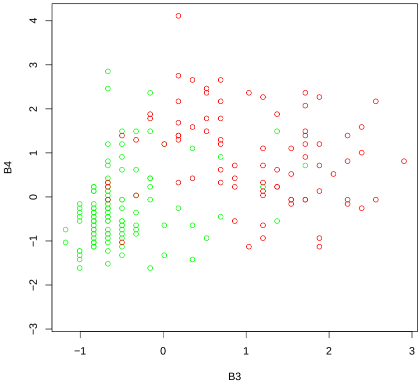


Figure: Classifiying historical landsat imagery into Forest or Non Forest Status.

[See forested.R]

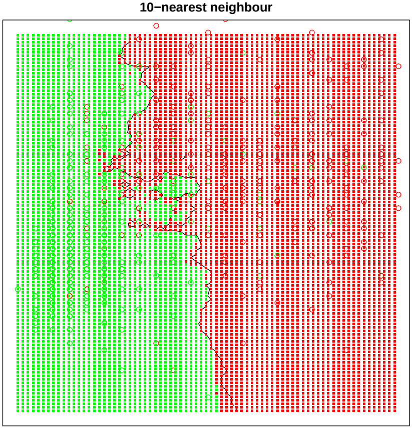
Plotting Out Band Values for Red, Near Infrared

```
options(stringsAsFactors=FALSE)
library(data.table)
library(plyr)
library(class)
FORESTED <- data.table(read.csv("R/forested.csv"))
COMPOSITE <- data.table(read.csv("R/composite.csv"))
MODIS <- data.table(read.csv("R/modis.csv"))
setnames(MODIS, "mean", "landcover")
setnames(FORESTED, "mean", "forestcover")
DF<- join(join(FORESTED[, c("system.index", "forestcover"), with=F],
               MODIS[, c("system.index", "landcover"), with=F]), COMPOSITE)
DF[, Forested := forestcover>.8]
DF[, MODISforested := (landcover>0 & landcover<=5)]
DF[, B3:=scale(B3)]
DF[, B4:=scale(B4)]
df.xlim <- range(DF$B3)
df.ylim<- range(DF$B4)
plot(DF[Forested==TRUE][1:120, c("B3", "B4"), with=F], col="green", xlim=df.xlim, ylim=df.ylim)
points(DF[Forested==FALSE][1:80, c("B3", "B4"), with=F], col="red", xlim=df.xlim, ylim=df.ylim)
```



Results for KNN with K=10

```
set.seed(1151)
train<-sample(1:1000, 800)
# get the contour map
px1 <- range(DF[-train]$B3)
px1<-seq(px1[1], px1[2], 0.05)
px2 <- range(DF[-train]$B4)
px2<-seq(px2[1], px2[2],0.05)
xnew <- expand.grid(x1 = px1, x2 = px2)
knn10 <- knn(DF[train, c("B3","B4"), with=F], test = xnew, cl = DF[train]$Forested, k = 10, prob = TRUE)
prob <- attr(knn10, "prob")
prob <- ifelse(knn10==TRUE, prob, 1-prob)
prob10 <- matrix(prob, nrow = length(px1), ncol = length(px2))
par(mar = rep(2,4))
contour(px1, px2, prob10, levels=.5, labels="", xlab="", ylab="", main= "10-nearest neighbour", axes=FALSE)
points(DF[train,c("B3","B4"), with=F], col=ifelse(DF[train]$Forested==TRUE, "green", "red"))
points(xnew, pch=".", cex=3.5, col=ifelse(prob10>.5, "green", "red"))
box()
```



How do we perform in terms of classification?

```
knn10 <- knn(DF[train, c("B3","B4"), with=F], test = DF[-train, c("B3","B4"), with=F],
             cl = DF[train]$Forested, k = 10, prob = TRUE)
prob <- attr(knn10, "prob")
prob <- ifelse(knn10==TRUE, prob, 1-prob)
table(prob>0.5,DF[-train]$Forested)


##
##          FALSE TRUE
##   FALSE     52   10
##   TRUE      15  123
```

Overall error rate is quite low: $25/200 = 12.5\%$.

- Pixels are wrongly classified as forested (type 1 error) 10/200
- Pixels are wrongly classified as non forested 15/200
- Suppose you are sending an enforcement agent to fine people who have illegally deforested land (predicted value = FALSE), i.e. for 62 pixels.
- It costs a lot of money to send police out there, but in 16% (10/(52+10)) of the cases you find the forest is actually standing...

Changing $\bar{c}$...

Make it less likely, that we have pixels wrongly classified as being forested...reduce the threshold $\bar{c}$.

```
table(prob>0.4,DF[-train]$Forested)
```

```
##
##         FALSE TRUE
##   FALSE    51    5
##   TRUE     16  128
```

- Now, we will send enforcement teams out 56 times, but only in 8% of the cases, will the forest still be standing...
- While the threshold $\bar{c} = 0.5$, theoretically minimizes overall prediction error, you may still prefer different $\bar{c}$, in case there are different costs associated with type 1 or type 2 errors.

Exercise: Plot an ROC Curve...

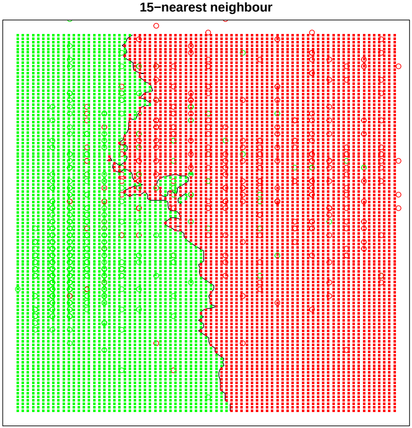
Can you try to compute ROC curves and plot them out?

How to Choose K?

- As you will have guessed, it turns out that K is a tuning parameter that we need to select.
- In binary (two class) classification problems, it is helpful to choose K to be an odd number – why?
- Cross validation is our preferred method, i.e. for each different value of K, we perform cross validation and plot out the overall average error rates.
- Why does the choice of K matter?

Results for KNN with K=15

```
set.seed(1151)
train<-sample(1:1000, 800)
# get the contour map
px1 <- range(DF[-train]$B3)
px1<-seq(px1[1], px1[2], 0.05)
px2 <- range(DF[-train]$B4)
px2<-seq(px2[1], px2[2],0.05)
xnew <- expand.grid(x1 = px1, x2 = px2)
knn15 <- knn(DF[train, c("B3","B4"), with=F], test = xnew, cl = DF[train]$Forested, k = 15, prob = TRUE)
prob <- attr(knn15, "prob")
prob <- ifelse(knn15==TRUE, prob, 1-prob)
prob15 <- matrix(prob, nrow = length(px1), ncol = length(px2))
par(mar = rep(2,4))
contour(px1, px2, prob15, levels=.5, labels="", xlab="", ylab="", main= "15-nearest neighbour", axes=FALSE)
points(DF[train,c("B3","B4"), with=F], col=ifelse(DF[train]$Forested==TRUE, "green", "red"))
points(xnew, pch=".", cex=3.5, col=ifelse(prob15>.5, "green", "red"))
box()
```

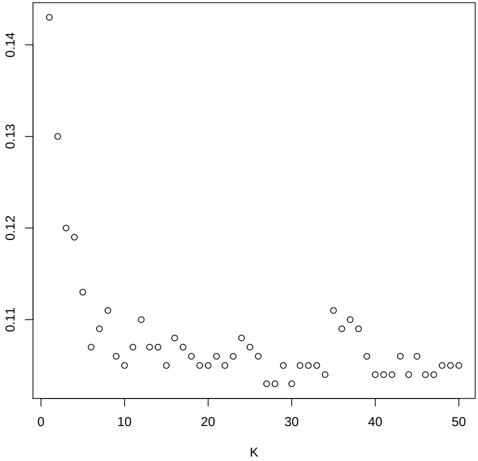


Results for KNN with K=1

```
set.seed(1151)
train<-sample(1:1000, 800)
# get the contour map
px1 <- range(DF[-train]$B3)
px1<-seq(px1[1], px1[2], 0.05)
px2 <- range(DF[-train]$B4)
px2<-seq(px2[1], px2[2],0.05)
xnew <- expand.grid(x1 = px1, x2 = px2)
knn1 <- knn(DF[train, c("B3","B4"), with=F], test = xnew, cl = DF[train]$Forested, k = 1, prob = TRUE)
prob <- attr(knn1, "prob")
prob <- ifelse(knn1==TRUE, prob, 1-prob)
prob1 <- matrix(prob, nrow = length(px1), ncol = length(px2))
par(mar = rep(2,4))
contour(px1, px2, prob1, levels=.5, labels="", xlab="", ylab="", main= "1-nearest neighbour", axes=FALSE)
points(DF[train,c("B3","B4"), with=F], col=ifelse(DF[train]$Forested==TRUE, "green", "red"))
points(xnew, pch=".", cex=3.5, col=ifelse(prob1>.5, "green", "red"))
box()
```

Cross Validation and choice of K

```
RES<-NULL
for(K in 1:50) {
knncv <- knn.cv(DF[, c("B3","B4"), with=F],
                cl = DF$Forested, k = K, prob = TRUE)
RES<-rbind(RES,cbind(K,1-(table(knncv,DF$Forested)[1,1]+table(knncv,DF$Forested)[2,2])/1000))
}
plot(RES)
```



Comparing Logistic Regression Versus kNN Classification

- Cross Validation suggested that we should use $k \approx 10$ for kNN classification.
- Lets compare the performance of logistic regression versus 10NN for this example.

```
set.seed(1151)
train<-sample(1:1000, 800)

glm.fit<-glm(Forested ~  B3 + B4,              data=DF[train], family=binomial(link=logit))

glm.predict <- predict.glm(glm.fit, DF[-train], type="response")

knn10 <- knn(DF[train, c("B3","B4"), with=F], test = DF[-train, c("B3","B4"), with=F],
             cl = DF[train]$Forested, k = 10, prob = TRUE)
prob <- attr(knn10, "prob")
prob <- ifelse(knn10==TRUE, prob, 1-prob)
table(prob>0.5,DF[-train]$Forested)

##
##         FALSE TRUE
##   FALSE    52   10
##   TRUE     15  123

table(glm.predict>0.5,DF[-train]$Forested)

##
##         FALSE TRUE
##   FALSE    48    6
##   TRUE     19  127
```

Comparing Logistic Regression Versus kNN Classification

- The result is not surprising. The data suggests a near linear decision boundary, both kNN as well as logistic regression estimate that linear boundary reasonably well.
- You may still decide that you prefer logistic regression, since it is much more interpretable and less of a black box.
- With logistic regression, you get interpretable coefficients; here we do not care much for inference, so it doesnt really matter.
- But in general, a (reasonable) rule of thumb is that you should prefer a simple parametric interpretable method over a non parametric method in case they yield similar results.

When does kNN outperform?

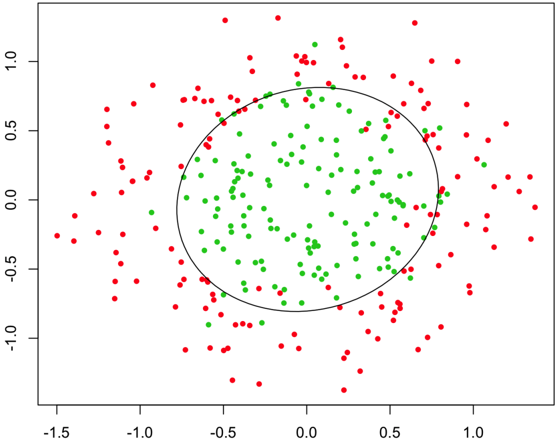


Figure: Example of Non-Linear Classification Problems

When does kNN outperform?

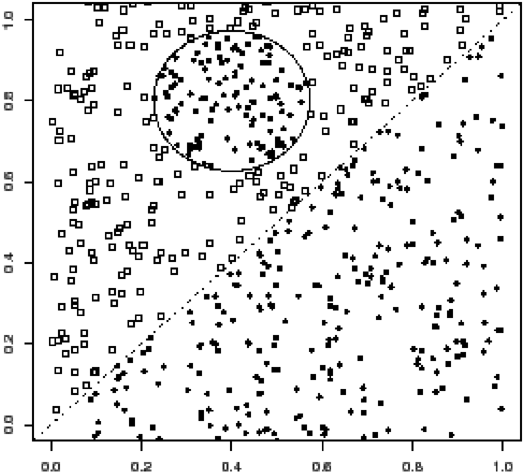
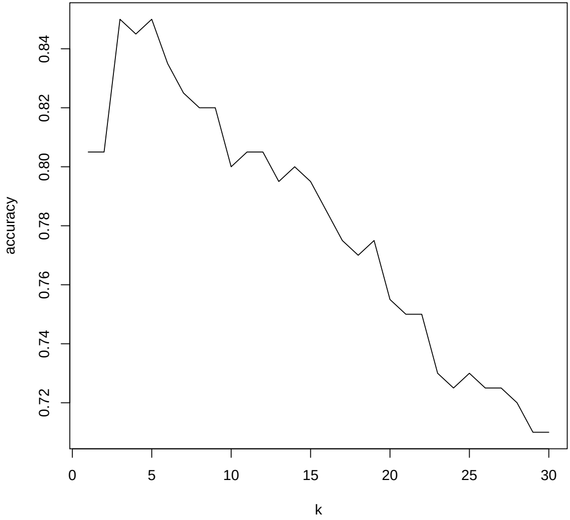


Figure: Example of Non-Linear Classification Problems

KNN for text classification

- As you can see from the specification, KNN uses Euclidian distances, which may be problematic if you want to classify documents of different lengths
- In text classsification, KNN is also called
- Its explicitly non-parametric (you dont estimate any coefficients)
- KNN is referred to as a lazy learning algorithm: the function is only approximated locally and all computation is deferred until classification.
- So there is actually no explicit training step.
- Results from KNN are sensitive to the local structure of the data (example highly skewed, “majority voting” distorted, weights).
- Its (potentially) too slow, as it may be impossible to hold all the training data in memory.

KNN for text classification

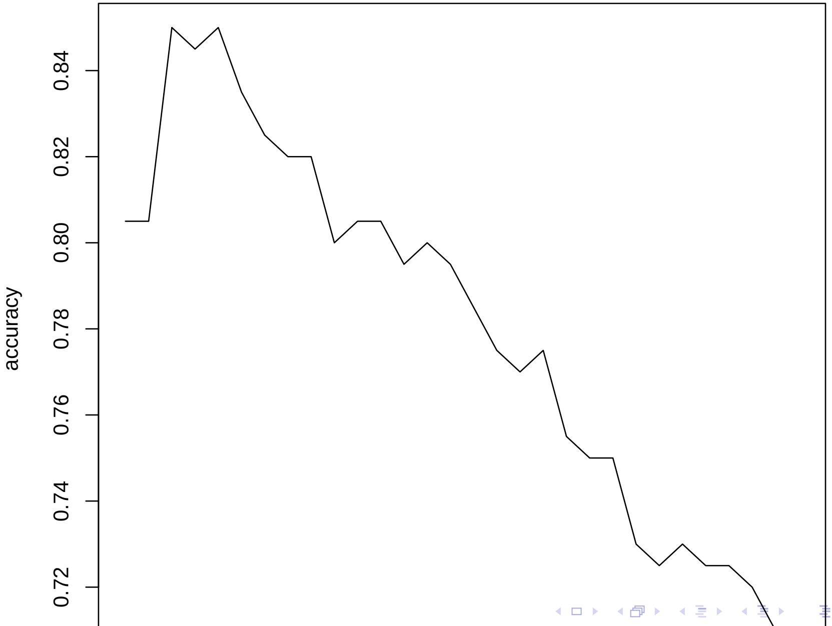


KNN for text classification

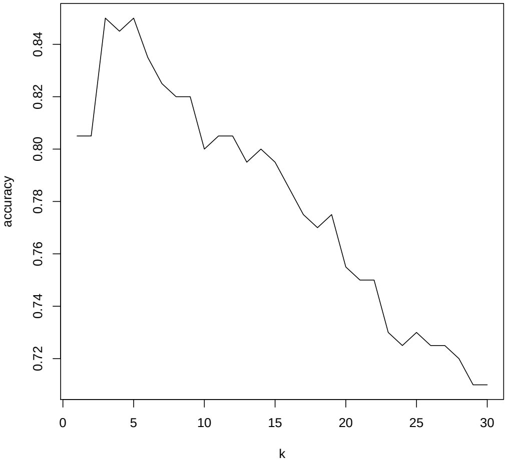
```
library(RTextTools)
##knn function
library(class)
TTIM<-TTIM[order(sid)]
set.seed(06022017)
TTIM<-TTIM[sample(1:nrow(TTIM), nrow(TTIM))]
L1 <- create_matrix(TTIM[,paste(objectcleanpp,sep=" ")],
                    language="english",stemWords=FALSE)
##CREATION OF NON SPARSE MATRIX
DTM<-as.matrix(L1)

dim(DTM)


## [1] 1397 2460


#knn(train, test, labelsfortrainingset, k=1)
res<-knn(DTM[201:1200,], DTM[1:200,], cl=as.factor(TTIM[201:1200]$label1))

table(res, TTIM[1:200]$label1)


##
## res         civilian security terrorist
##   civilian        40        5        17
##   security         7       31         1
##   terrorist        7        2        90
```

KNN for text classification

```
ACC<-NULL
for(k in 1:30) {
#knn(train, test, labelsfortrainingset, k=1)
res<-knn(DTM[201:1200,], DTM[1:200,], cl=as.factor(TTIM[201:1200]$label1), k=k)

ACC<-rbind(ACC, data.frame(k=k, accuracy=sum(diag(3) * table(res, TTIM[1:200]$label1))/200))

}
```


accuracy
0.72
0.74
0.76
0.78
0.80
0.82
0.84

Accuracy for different values of k



KNN is a discriminative classifier [non examinable]

KNN is a discriminative algorithm since it models the conditional probability of a sample belonging to a given class. To see this just consider how one gets to the decision rule of kNNs.

A class label corresponds to a set of points which belong to some region in the feature space R. If you draw sample points from the actual probability distribution, p(x), independently, then the probability of drawing a sample from that class is,

$$P = \int_R p(x)dx$$

What if you have N points? The probability that K points of those N points fall in the region R follows the binomial distribution,

$$Prob(K) = \binom{N}{K} P^K (1 - P)^{N-K}$$

As $N \to \infty$ this distribution is sharply peaked, so that the probability can be approximated by its mean value KN.

KNN is a discriminative classifier[non examinable]

An additional approximation is that the probability distribution over R remains approximately constant, so that one can approximate the integral by,

$$P = \int_R p(x)dx \approx p(x)V$$

where V is the total volume of the region. Under this approximations $p(x) \approx \frac{K}{NV}$

Now, if we had several classes, we could repeat the same analysis for each one, which would give us,

$$p(x|C_k) = \frac{K_k}{N_k V}$$

where K_k is the amount of points from class k which falls within that region and Nk is the total number of points which fall in that region. Notice $\sum_k N_k = N$.

Repeating the analysis with the binomial distribution, it is easy to see that we can estimate the prior $P(C_k) = \frac{N_k}{N}$

Using Bayes rule,

$$P(C_k|x) = \frac{p(x|C_k)p(C_k)}{p(x)} = \frac{K_k}{K}$$

which is the rule for kNNs.