Statistical Machine Learning
Hilary Term 2020

Pier Francesco Palamara
Department of Statistics
University of Oxford

Slide credits and other course material can be found at:
http://www.stats.ox.ac.uk/~palamara/SML20.html

February 25, 2020

Naïve Bayes

Naïve Bayes: overview

- Naïve Bayes: another **plug-in** classifier with a simple **generative** model - it assumes all measured variables/features are independent given the label.
- Easy to mix and match different types of features, handle missing data.
- Often used with categorical data, e.g. text document classification.
    - A basic standard model for text classification consists of considering a pre-specified dictionary of p words and summarizing each document i by a binary vector x_i (“bag-of-words”):

$$x_i^{(j)} = \begin{cases} 1 & \text{if word } j \text{ is present in document} \\ 0 & \text{otherwise.} \end{cases}$$

where the presence of the word j is the j-th feature/dimension.

Toy Example

Predict voter preference in US elections

Voted in 2012?	Annual Income	State	Candidate Choice
Y	50K	OK	Clinton
N	173K	CA	Clinton
Y	80K	NJ	Trump
Y	150K	WA	Clinton
N	25K	WV	Johnson
Y	85K	IL	Clinton
⋮	⋮	⋮	⋮
Y	1050K	NY	Trump
N	35K	CA	Trump
N	**100K**	**NY**	**?**

Naïve Bayes Classifier (NBC)

- In order to fit a generative model, we'll express the joint distribution as
$$p(\boldsymbol{x}, y \mid \boldsymbol{\theta}, \boldsymbol{\pi}) = p(y \mid \boldsymbol{\pi}) \cdot p(\boldsymbol{x} \mid y, \boldsymbol{\theta})$$
- To model $p(y \mid \boldsymbol{\pi})$, we'll use parameters π_c such that $\sum_c \pi_c = 1$
$$p(y = c \mid \boldsymbol{\pi}) = \pi_c$$
- For class-conditional densities, for class $c = 1, \ldots, C$, we will have a model:
$$p(\boldsymbol{x} \mid y = c, \boldsymbol{\theta}_c)$$
- We assume that the features are conditionally independent given the class label
$$p(\boldsymbol{x} \mid y = c, \boldsymbol{\theta}_c) = \prod_{j=1}^{D} p(x_j \mid y = c, \boldsymbol{\theta}_{jc})$$
- Clearly, the independence assumption is "naïve" and never satisfied. But model fitting becomes very very easy.
- Although the generative model is clearly inadequate, it actually works quite well. Goal is predicting class, not modelling the data!

Naïve Bayes Classifier (NBC)

In our example,

$$p(y = \text{clinton} \mid \boldsymbol{\pi}) = \pi_{\text{clinton}}$$
$$p(y = \text{trump} \mid \boldsymbol{\pi}) = \pi_{\text{trump}}$$
$$p(y = \text{johnson} \mid \boldsymbol{\pi}) = \pi_{\text{johnson}}$$

Given that a voter supports Trump

$$p(\boldsymbol{x} \mid y = \text{trump}, \boldsymbol{\theta}_{\text{trump}})$$

models the distribution over $\boldsymbol{x}$ given $y = \text{trump}$ and $\boldsymbol{\theta}_{\text{trump}}$

Similarly, we have $p(\boldsymbol{x} \mid y = \text{clinton}, \boldsymbol{\theta}_{\text{clinton}})$ and $p(\boldsymbol{x} \mid y = \text{johnson}, \boldsymbol{\theta}_{\text{johnson}})$

We need to pick “model” for $p(\boldsymbol{x} \mid y = c, \boldsymbol{\theta}_c)$

Estimate the parameters π_c, $\boldsymbol{\theta}_c$ for $c = 1, \ldots, C$

Naïve Bayes Classifier (NBC)

Real-Valued Features

- x_j is real-valued annual income
- Example: Use a Gaussian model, so $\boldsymbol{\theta}_{jc} = (\mu_{jc}, \sigma^2_{jc})$
- Can use other distributions, age is probably not Gaussian!

Categorical Features

- x_j is categorical with values in $\{1, \ldots, K\}$
- Use the **multinoulli** distribution, i.e. $x_j = i$ with probability $\mu_{jc,i}$

$$\sum_{i=1}^{K} \mu_{jc,i} = 1$$

- The special case when $x_j \in \{0, 1\}$, use a single parameter $\theta_{jc} \in [0, 1]$

Naïve Bayes Classifier (NBC)

- Assume that all the features are binary, i.e. every $x_j \in \{0, 1\}$
- (In this case, the log-discriminant function of each class assumes the form $a_c + b_c^\top x$ for class c. Verify this.)
- If we have C classes, overall we have only $O(CD)$ parameters, θ_{jc} for each $j = 1, \ldots, D$ and $c = 1, \ldots, C$

- Without the conditional independence assumption
  - We have to assign a probability for each of the 2^D combination
  - Thus, we have $O(C \cdot 2^D)$ parameters!
  - The 'naïve' assumption breaks the *curse of dimensionality* and avoids overfitting!

Maximum Likelihood for the NBC

- Let us suppose we have data $\langle(\boldsymbol{x}_i, y_i)\rangle_{i=1}^{N}$ i.i.d. from some joint distribution $p(\boldsymbol{x}, y)$
- The probability for a single datapoint is given by:

$$p(\boldsymbol{x}_i, y_i|\boldsymbol{\theta}, \boldsymbol{\pi}) = p(y_i|\boldsymbol{\pi}) \cdot p(\boldsymbol{x}_i|\boldsymbol{\theta}, y_i) = \prod_{c=1}^{C} \pi_c^{\mathbb{I}(y_i=c)} \cdot \prod_{c=1}^{C}\prod_{j=1}^{D} p(x_{ij}|\boldsymbol{\theta}_{jc})^{\mathbb{I}(y_i=c)}$$

- Let N_c be the number of datapoints with $y_i = c$, so that $\sum_{c=1}^{C} N_c = N$
- We write the log-likelihood of the data, assuming points are i.i.d.:

$$\log p(\mathcal{D} \mid \boldsymbol{\theta}, \boldsymbol{\pi}) = \sum_{c=1}^{C} N_c \log \pi_c + \sum_{c=1}^{C}\sum_{j=1}^{D}\sum_{i:y_i=c} \log p(x_{ij} \mid \boldsymbol{\theta}_{jc})$$

- The log-likelihood is easily separated into sums involving different parameters!

Maximum Likelihood for the NBC

- We have the log-likelihood for the NBC

$$\log p(\mathcal{D} \mid \boldsymbol{\theta}, \boldsymbol{\pi}) = \sum_{c=1}^{C} N_c \log \pi_c + \sum_{c=1}^{C} \sum_{j=1}^{D} \sum_{i:y_i=c} \log p(x_{ij} \mid \boldsymbol{\theta}_{jc})$$

- We can use maximum likelihood to estimate the parameters (we have done this before). For instance, let's estimate $\boldsymbol{\pi}$. We have the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & \sum_{c=1}^{C} N_c \log \pi_c \\ \text{subject to :} \quad & \sum_{c=1}^{C} \pi_c = 1 \end{aligned}$$

- This constrained optimization problem can be solved using Lagrange multipliers

$$\Lambda(\boldsymbol{\pi}, \lambda) = \sum_{c=1}^{C} N_c \log \pi_c + \lambda \left(\sum_{c=1}^{C} \pi_c - 1 \right)$$

Maximum Likelihood for the NBC

We can write the Lagrangean form:

$$\Lambda(\boldsymbol{\pi}, \lambda) = \sum_{c=1}^{C} N_c \log \pi_c + \lambda \left(\sum_{c=1}^{C} \pi_c - 1 \right)$$

We can write the partial derivatives and set them to 0:

$$\frac{\partial \Lambda(\boldsymbol{\pi}, \lambda)}{\partial \pi_c} = \frac{N_c}{\pi_c} + \lambda = 0; \qquad \frac{\partial \Lambda(\boldsymbol{\pi}, \lambda)}{\partial \lambda} = \sum_{c=1}^{C} \pi_c - 1 = 0$$

The solution is obtained by setting

$$\frac{N_c}{\pi_c} + \lambda = 0 \quad \rightarrow \quad \pi_c = -\frac{N_c}{\lambda}$$

As well as using the second condition,

$$\sum_{c=1}^{C} \pi_c - 1 = \left(\sum_{c=1}^{C} -\frac{N_c}{\lambda} \right) - 1 = 0 \quad \rightarrow \quad \lambda = -\sum_{c=1}^{C} N_c = -N$$

Thus, we get the estimates,

$$\pi_c = \frac{N_c}{N}$$

Maximum Likelihood for the NBC

- We have the log-likelihood for the NBC

$$\log p(\mathcal{D} \mid \boldsymbol{\theta}, \boldsymbol{\pi}) = \sum_{c=1}^{C} N_c \log \pi_c + \sum_{c=1}^{C} \sum_{j=1}^{D} \sum_{i:y_i=c} \log p(x_{ij} \mid \boldsymbol{\theta}_{jc})$$

- We obtained the estimates, $\pi_c = \frac{N_c}{N}$
- We can estimate $\boldsymbol{\theta}_{jc}$ by taking a similar approach
- To estimate $\boldsymbol{\theta}_{jc}$ we only need to use the j^{th} feature of examples with $y_i = c$
- Estimates depend on the model, e.g. Gaussian, Bernoulli, Multinoulli, etc.
- Fitting NBC is very very fast!

NBC: Handling Missing Data

Let's recall our example about trying to predict voter preferences

Voted in 2012?	Annual Income	State	Candidate Choice
Y	50K	OK	Clinton
N	173K	CA	Clinton
Y	80K	NJ	Trump
Y	150K	WA	Clinton
N	25K	WV	Johnson
Y	85K	IL	Clinton
⋮	⋮	⋮	⋮
Y	1050K	NY	Trump
N	35K	CA	Trump
?	**100K**	**NY**	**?**

Suppose a voter does not reveal whether or not they voted in 2012

For now, let's assume we had no missing entries during training

NBC: Handling Missing Data

The prediction rule in a generative model is

$$p(y = c \mid \boldsymbol{x}_{\text{new}}, \boldsymbol{\theta}) = \frac{p(y = c \mid \boldsymbol{\theta}) \cdot p(\boldsymbol{x}_{\text{new}} \mid y = c, \boldsymbol{\theta})}{\sum_{c'=1}^{C} p(y = c' | \boldsymbol{\theta}) p(\boldsymbol{x}_{\text{new}} \mid y = c', \boldsymbol{\theta})}$$

Let us suppose our datapoint is $\boldsymbol{x}_{\text{new}} = (?, x_2, \ldots, x_D)$, e.g. $(?, \text{100K}, \text{NY})$

$$p(y = c \mid \boldsymbol{x}_{\text{new}}, \boldsymbol{\theta}) = \frac{\pi_c \cdot \prod_{j=1}^{D} p(x_j \mid y = c, \boldsymbol{\theta}_{cj})}{\sum_{c'=1}^{C} p(y = c' | \boldsymbol{\theta}) \prod_{j=1}^{D} p(x_j \mid y = c', \boldsymbol{\theta}_{jc})}$$

Since x_1 is missing, we can marginalize it out,

$$p(y = c \mid \boldsymbol{x}_{\text{new}}, \boldsymbol{\theta}) = \frac{\pi_c \cdot \prod_{j=2}^{D} p(x_j \mid y = c, \boldsymbol{\theta}_{cj})}{\sum_{c'=1}^{C} p(y = c' | \boldsymbol{\theta}) \prod_{j=2}^{D} p(x_j \mid y = c', \boldsymbol{\theta}_{jc})}$$

This can be done for other generative models, but marginalization requires summation/integration

NBC: Handling Missing Data

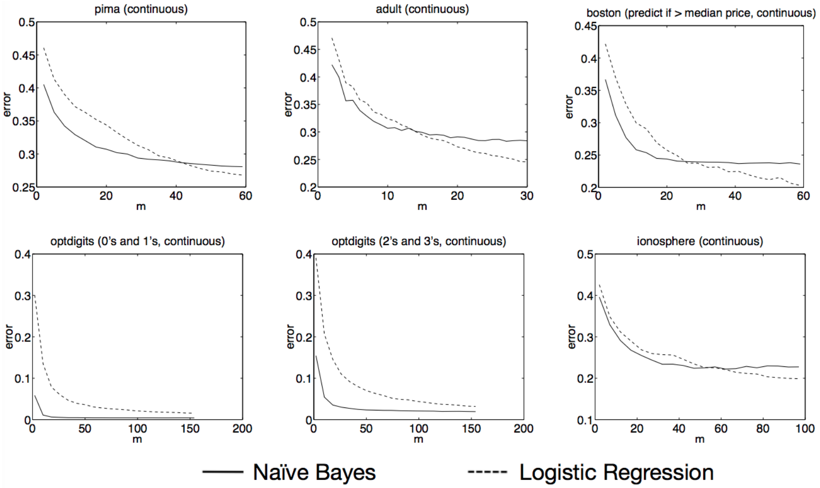
For Naïve Bayes Classifiers, training with missing entries is quite easy

Voted in 2012?	Annual Income	State	Candidate Choice
?	50K	OK	Clinton
N	173K	CA	Clinton
?	80K	NJ	Trump
Y	150K	WA	Clinton
N	25K	WV	Johnson
Y	85K	?	Clinton
⋮	⋮	⋮	⋮
Y	1050K	NY	Trump
N	35K	CA	Trump
?	**100K**	**NY**	**?**

Let's say for Clinton voters, 103 had voted in 2012, 54 had not, and 25, didn't answer

You can simply set $\theta = \frac{103}{157}$ as the probability that a voter had voted in 2012, conditioned on being a Clinton supporter

Naïve Bayes vs Logistic regression



“On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes” by A. Ng and M. Jordan, NIPS 2001. m represents training dataset size.

Naïve Bayes vs Logistic regression

- For infinite data
  - If generative model is correct (independence assumption holds)

$$Error_{LR,\infty} \sim Error_{NB,\infty}$$

  - If generative model is inaccurate (independence assumption does not hold)

$$Error_{LR,\infty} < Error_{NB,\infty}$$

- For finite data (e.g. n points, d features), NB will require less training to converge to its (possibly asymptotically higher) error

$$Error_{LR,n} \leq Error_{LR,\infty} + O\left(\sqrt{\frac{d}{n}}\right)$$
$$Error_{NB,n} \leq Error_{NB,\infty} + O\left(\sqrt{\frac{\log d}{n}}\right)$$

Preventing numerical underflow (not examinable)

- Generative classifiers often require multiplying a large number of small quantities, leading to **numerical underflow**.

$$\begin{aligned}\log p(y=c|\boldsymbol{x}) &= \log\left[\frac{p(y=c)p(\boldsymbol{x}|y=c)}{\sum_{c'} p(y=c')p(\boldsymbol{x}|y=c')}\right]\\ &= b_c - \log\left[\sum_{c'=1}^{C} e^{b_{c'}}\right]\\ b_c &\triangleq \log p(\boldsymbol{x}|y=c) + \log p(y=c)\end{aligned}$$

- The terms $e^{b_{c'}}$ are extremely small (e.g. in Naive Bayes), but we cannot sum in the log domain to evaluate $\log\sum_{c'} e^{b_{c'}}$.
- Idea: factor out the largest term[1]. For example:

$$\log(e^{-120}+e^{-121}) = \log(e^{-120}(e^{0}+e^{-1})) = \log(e^{0}+e^{-1}) - 120.$$

- In general, having defined $B = \max_c b_c$:

$$\log\sum_c e^{b_c} = \log\left[\left(\sum_c e^{b_c-B}\right)e^{B}\right] = \left[\log\left(\sum_c e^{b_c-B}\right)\right] + B$$

[1] Also see Murphy 3.5.3.

Naïve Bayes code example: Titanic data I

Predicting Titatinc survival from passenger data using Naïve Bayes[2]:

```
#Install the package
install.packages("e1071")
#Loading the library
library(e1071)
?naiveBayes #The documentation also uses Titanic data
#Next load the Titanic dataset
data("Titanic")
#Save into a data frame and view it
Titanic_df=as.data.frame(Titanic)
#Creating data from table
#This will repeat each combination equal to the frequency
repeating_sequence=rep.int(seq_len(nrow(Titanic_df)),
    Titanic_df$Freq)

#Create the dataset by row repetition created
```

Naïve Bayes code example: Titanic data II

```
Titanic_dataset=Titanic_df[repeating_sequence,]
#We no longer need the frequency, drop the feature
Titanic_dataset$Freq=NULL

#Fitting the Naive Bayes model
Naive_Bayes_Model=naiveBayes(Survived ~.,
    data=Titanic_dataset)
#What does the model say? Print the model summary
Naive_Bayes_Model

#Prediction on the dataset
NB_Predictions=predict(Naive_Bayes_Model,Titanic_dataset)
#Confusion matrix to check accuracy
table(NB_Predictions,Titanic_dataset$Survived)
```

[2] code from https://r-posts.com/understanding-naive-bayes-classifier-using-r/

K-Nearest Neighbors

Nearest neighbor (NN) classification/regression

Training

- Store the entire training set.

Classification/regression rule

$$y = f(\boldsymbol{x}) = y_{\mathsf{nn}(\boldsymbol{x})}$$

where $\mathsf{nn}(\boldsymbol{x}) \in [\mathsf{N}] = \{1, 2, \cdots, \mathsf{N}\}$, i.e., the index to one of the training instances

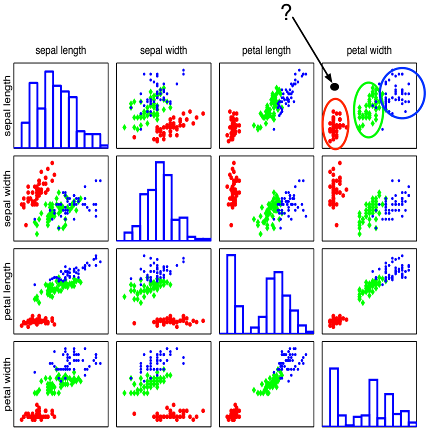
$$\mathsf{nn}(\boldsymbol{x}) = \operatorname*{argmin}_{n \in [\mathsf{N}]} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2 = \operatorname*{argmin}_{n \in [\mathsf{N}]} \sum_{d=1}^{\mathsf{D}} (x_d - x_{nd})^2$$

Inductive bias

- Label of point is similar to the label of nearby points.

Labeling an unknown flower type

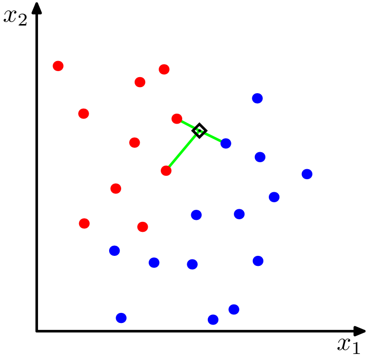
Ex: Iris data (click here for all data)
Using two features: petal width and sepal length



Closer to red cluster: so labeling it as setosa

Visual example

In this 2-dimensional example, the nearest point to $\boldsymbol{x}$ is a red training instance, thus, $\boldsymbol{x}$ will be labeled as red.



(a)

How to measure nearness with other distances?

Previously, we use the Euclidean distance

$$\mathsf{nn}(\boldsymbol{x}) = \operatorname*{argmin}_{n \in [\mathsf{N}]} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2$$

We can also use alternative distances

$$\|\boldsymbol{x} - \boldsymbol{x}_n\|_p = \left(\sum_d |x_d - x_{nd}|^p \right)^{1/p}$$

for $p \geq 1$.
E.g., the L_1 distance for $p = 1$ (i.e., city block distance, or Manhattan distance)

$$\begin{aligned} \mathsf{nn}(\boldsymbol{x}) &= \operatorname*{argmin}_{n \in [\mathsf{N}]} \|\boldsymbol{x} - \boldsymbol{x}_n\|_1 \\ &= \operatorname*{argmin}_{n \in [\mathsf{N}]} \sum_{d=1}^{\mathsf{D}} |x_d - x_{nd}| \end{aligned}$$

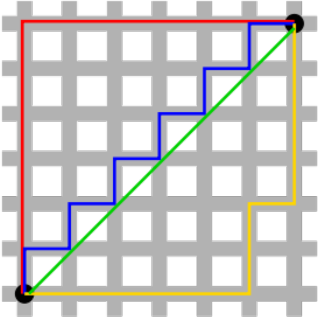
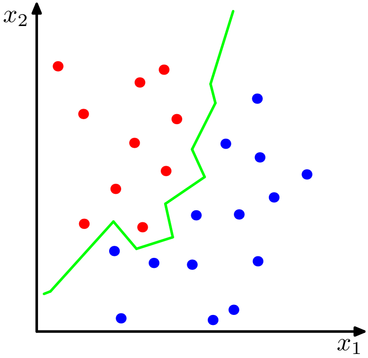


Figure: Green line is Euclidean distance. Red, Blue, and Yellow lines are L_1 distance

Decision boundary

For every point in the space, we can determine its label using the NN classification rule. This gives rise to a **decision boundary** that partitions the space into different regions.



(b)

Regression/classification with K neighbors?

Classification rule

- Every neighbor votes: suppose y_n (the true label) for $\boldsymbol{x}_n$ is c, then
  - vote for c is 1
  - vote for $c' \neq c$ is 0

  We use the indicator function $\mathbb{I}(y_n == c)$ to represent.
- Aggregate everyone's vote

$$v_c = \sum_{n \in \mathsf{knn}(\boldsymbol{x})} \mathbb{I}(y_n == c), \quad \forall \quad c \in [\mathsf{C}]$$

- Label with the majority

$$y = f(\boldsymbol{x}) = \operatorname*{argmax}_{c \in [\mathsf{C}]} v_c$$

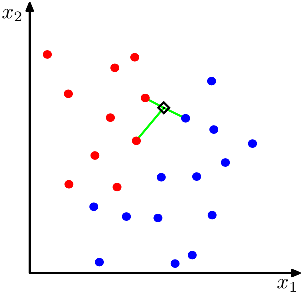
Regression rule

- Average across nearest neighbors:

$$y = f(\boldsymbol{x}) = \frac{\sum_{n \in \mathsf{knn}(\boldsymbol{x})} y_n}{K}$$

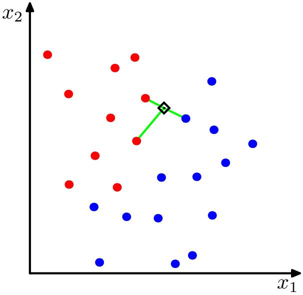
Example

K=1, Label: red



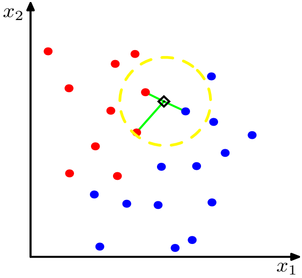
(a)

K=3, Label: red



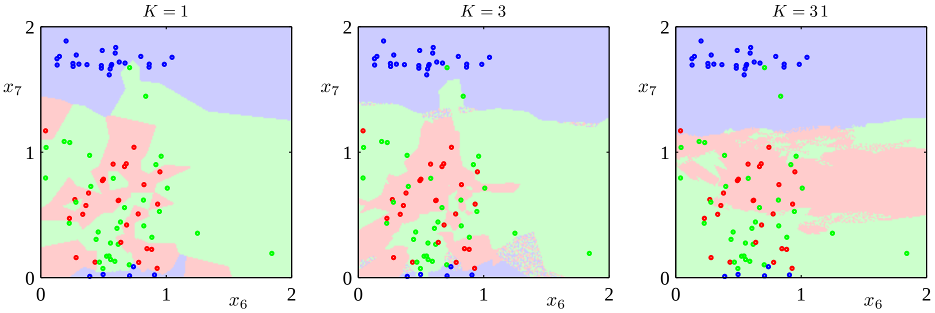
(a)

K=5, Label: blue



(a)

How to choose an optimal K?



When K increases, the decision boundary becomes smooth.

Hypeparameters in K-NN

Two practical issues about K-NN

- Choosing key parameter K, the number of nearest neighbors
  - May be chosen using cross-valitation.
  - How does K affect bias/variance?
- Choosing the right distance measure (e.g. Euclidean distance).

Those are not specified by the algorithm itself — resolving them requires empirical studies and are task/dataset-specific.

Preprocess data

Assumes all features are equally important!

- Distances depend on units of the features.

Normalize data to have zero mean and unit standard deviation in each dimension

- Compute the means and standard deviations in each feature

$$\bar{x}_d = \frac{1}{\mathrm{N}} \sum_n x_{nd}, \qquad s_d^2 = \frac{1}{N-1} \sum_n (x_{nd} - \bar{x}_d)^2$$

- Scale the feature accordingly

$$x_{nd} \leftarrow \frac{x_{nd} - \bar{x}_d}{s_d}$$

Many other ways of normalizing data — you would need/want to try different ones and pick them using (cross)validation

Summary

Advantages of K-NN

- Simple and easy to implement – just computing distance
- Theoretically, has strong guarantees of “doing the right thing”

How well does K-NN do?

Theorem (Cover-Hart Inequality)

For the 1-NN rule f^{1NN} for binary classification, we have (see problem sheet),

$$R(f^*) \leq R(f^{\text{1NN}}) \leq 2R(f^*)(1 - R(f^*)) \leq 2R(f^*)$$

The expected risk is at worst twice that of the Bayes optimal classifier.

Disadvantages of K-NN

- Computationally intensive: $O(\text{N}D)$ for labeling a data point
- Need **a lot of data** for large D. May want to reduce dimensions first.
- Not useful for understanding relationships between attributes.
- We need to “carry” the training data around (**nonparametric** approach).
- Choosing the right distance measure and K can be involved.

k-Nearest Neighbour Demo – R Code I

```
library(MASS)
## load crabs data
data(crabs)
ct <- as.numeric(crabs[,1])-1+2*(as.numeric(crabs[,2])-1)
## project to first two LD
cb.lda <- lda(log(crabs[,4:8]),ct)
cb.ldp <- predict(cb.lda)
x <- as.matrix(cb.ldp$x[,1:2])
y <- as.numeric(crabs[,2])-1
x <- x + rnorm(dim(x)[1]*dim(x)[2])*1.5
eqscplot(x,pch=2*y+1,col=1)
n       <- length(y)

#get training indices
i <- sample(rep(c(TRUE,FALSE),each=n/2),n,replace=FALSE)

kNN <- function(k,x,y,i,gridsize=100) {

  p      <- dim(x)[2]

  train  <- (1:n)[i]
  test   <- (1:n)[!i]
  trainx <- x[train,]
  trainy <- y[train]
  testx  <- x[test,]
  testy  <- y[test]

  trainn <- dim(trainx)[1]
  testn  <- dim(testx)[1]

  gridx1 <- seq(min(x[,1]),max(x[,2]),length=gridsize)
  gridx2 <- seq(min(x[,2]),max(x[,2]),length=gridsize)
  gridx  <- as.matrix(expand.grid(gridx1,gridx2))
```

k-Nearest Neighbour Demo – R Code II

```
  gridn  <- dim(gridx)[1]

  # calculate distances
  trainxx <- t((trainx*trainx) %*% matrix(1,p,1))
  testxx  <- (testx*testx)   %*% matrix(1,p,1)
  gridxx  <- (gridx*gridx)   %*% matrix(1,p,1)
  testtraindist <- matrix(1,testn,1) %*% trainxx +
    testxx %*% matrix(1,1,trainn) -
    2*(testx %*% t(trainx))
  gridtraindist <- matrix(1,gridn,1) %*% trainxx +
    gridxx %*% matrix(1,1,trainn) -
    2*(gridx %*% t(trainx))

  # predict
  testp <- numeric(testn)
  gridp <- numeric(gridn)
  for (j in 1:testn) {
    nearestneighbors <- order(testtraindist[j,])[1:k]
    testp[j] <- mean(trainy[nearestneighbors])
  }
  for (j in 1:gridn) {
    nearestneighbors <- order(gridtraindist[j,])[1:k]
    gridp[j] <- mean(trainy[nearestneighbors])
  }
  predy <- as.numeric(testp>.5)

  plot(trainx[,1],trainx[,2],pch=trainy*3+1,col=4,lwd=.5)
  points(testx[,1],testx[,2],pch=testy*3+1,col=2+(predy==testy),lwd=3)
  contour(gridx1,gridx2,matrix(gridp,gridsize,gridsize),
          levels=seq(.1,.9,.1),lwd=.5,add=TRUE)
  contour(gridx1,gridx2,matrix(gridp,gridsize,gridsize),
          levels=c(.5),lwd=2,add=TRUE)
}
```

Evaluating performance

Example: Spam Dataset

A data set collected at Hewlett-Packard Labs, that classifies 4601 e-mails as spam or non-spam. 57 variables indicate the frequency of certain words and characters.

```
> library(kernlab)
> data(spam)
> dim(spam)
[1] 4601   58
> spam[1:2,]
  make address  all num3d  our over remove internet order mail receive will
1 0.00    0.64 0.64     0 0.32 0.00   0.00     0.00     0 0.00    0.00 0.64
2 0.21    0.28 0.50     0 0.14 0.28   0.21     0.07     0 0.94    0.21 0.79
  people report addresses free business email  you credit your font num000
1   0.00   0.00      0.00 0.32     0.00  1.29 1.93      0 0.96    0   0.00
2   0.65   0.21      0.14 0.14     0.07  0.28 3.47      0 1.59    0   0.43
  money hp hpl george num650 lab labs telnet num857 data num415 num85
1  0.00  0   0      0      0   0    0      0      0    0      0     0
2  0.43  0   0      0      0   0    0      0      0    0      0     0
  technology num1999 parts pm direct cs meeting original project re edu table
1          0    0.00     0  0      0  0       0        0       0  0   0 0
2          0    0.07     0  0      0  0       0        0       0  0   0 0
  conference charSemicolon charRoundbracket charSquarebracket charExclamation
1          0             0            0.000                 0       0.778
2          0             0            0.132                 0       0.372
  charDollar charHash capitalAve capitalLong capitalTotal type
1       0.00    0.000      3.756          61          278 spam
2       0.18    0.048      5.114          101        1028 spam
> str(spam$type)
 Factor w/ 2 levels "nonspam","spam": 2 2 2 2 2 2 2 2 2 2 ...
```

Spam Dataset

Use logistic regression to predict spam/not spam.

```
## let Y=0 be non-spam and Y=1 be spam.
Y <- as.numeric(spam$type)-1
X <- spam[ ,-ncol(spam)]

gl <- glm(Y ~ ., data=X,family=binomial)
```

Spam Dataset

How good is the classification?

```
> table(spam$type)
nonspam    spam
   2788    1813
> proba <- predict(gl,type="response")
> predicted_spam <- as.numeric( proba>0.5)
> table(predicted_spam,Y)
              Y
predicted_spam    0    1
             0 2666  194
             1  122 1619
> predicted_spam <- as.numeric( proba>0.95)
> table(predicted_spam,Y)
              Y
predicted_spam    0    1
             0 2766  810
             1   22 1003
```

Advantage of a probabilistic approach: predictive probabilities give interpretable confidence to predictions. Soft classification rules for other classifiers, e.g., support vector machines can be poorly calibrated if we are to interpret them as probabilities.

Spam Dataset

- We are viewing the prediction error on the training set. Not necessarily representative of the generalization ability.
- Separate in training and test set 50/50.

```
n <- length(Y)
train <- sample( n, round(n/2) )
test<-(1:n)[-train]
```

- Fit only on training set and predict on both training and test set.

```
gl <- glm(Y[train] ~ ., data=X[train,],family=binomial)

proba_train <- predict(gl,newdata=X[train,],type="response")
proba_test  <- predict(gl,newdata=X[test,],type="response")
```

Spam Dataset

Results for training and test set:

```
> predicted_spam_lr_train <- as.numeric(proba_train > 0.95)
> predicted_spam_lr_test  <- as.numeric(proba_test > 0.95)

> table(predicted_spam_lr_train, Y[train])
predicted_spam_lr_train    0    1
                      0 1401  358
                      1    8  533


> table(predicted_spam_lr_test, Y[test])
predicted_spam_lr_test    0    1
                     0 1357  392
                     1   22  530
```

Note: testing performance is worse than training performance.

Spam Dataset

Compare with LDA.

```
library(MASS)
lda_res <- lda(x=X[train,],grouping=Y[train])

proba_lda_test <- predict(lda_res,newdata=X[test,])$posterior[,2]
predicted_spam_lda_test <- as.numeric(proba_lda_test > 0.95)
```

```
> table(predicted_spam_lr_test, Y[test])
predicted_spam_lr_test    0    1
                     0 1357  392
                     1   22  530

> table(predicted_spam_lda_test, Y[test])
predicted_spam_lda_test    0    1
                      0 1361  533
                      1   18  389
```

- LDA has a larger number of false positives but a smaller number of false negatives.
  - Above results are for a single threshold (0.95) - how to keep track of what happens across multiple thresholds?
  - More generally, how to compare the classifiers fairly when the number of positive and negative examples is very different?

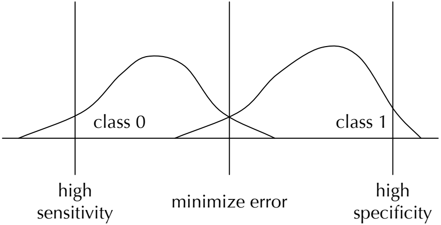
Performance Measures

- **Confusion matrix**:

True	state	0	1
Prediction	0	**# true negative**	**# false negative**
	1	**# false positive**	**# true positive**

  - **Accuracy**: $(TP+TN)/(TP+TN+FP+FN)$.
  - **Error rate**: $(FP+FN)/(TP+TN+FP+FN)$.
  - **Sensitivity (true positive rate)**: $TP/(TP+FN)$.
  - **Specificity (true negative rate)**: $TN/(TN+FP)$.
  - **False positive rate** (1-Specificity): $FP/(TN+FP)$.
  - **Precision**: $TP/(TP+FP)$.
  - **Recall** (same as Sensitivity): $TP/(TP+FN)$.
  - **F1**: harmonic mean of precision and recall.

- As we vary the prediction threshold c from 0 to 1:
  - Specificity varies from 0 to 1.
  - Sensitivity goes from 1 to 0.



ROC Curves

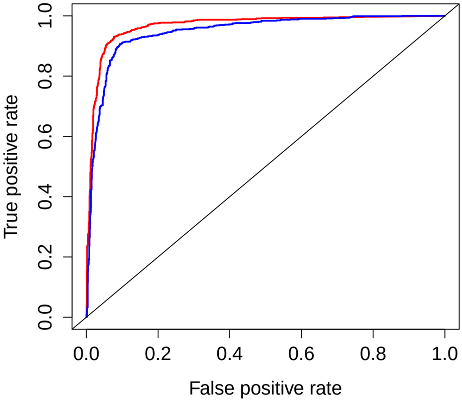
ROC curve plots sensitivity versus specificity as threshold varies.

```
cvec <- seq(0.001,0.999,length=1000)
specif <- numeric(length(cvec))
sensit <- numeric(length(cvec))
speciflr <- numeric(length(cvec))
sensitlr <- numeric(length(cvec))

for (cc in 1:length(cvec)){
  sensit[cc] <- sum( proba_lda_test> cvec[cc] & Y[test]==1)/sum(Y[test]==1)
  specif[cc] <- sum( proba_lda_test<=cvec[cc] & Y[test]==0)/sum(Y[test]==0)
  sensitlr[cc] <- sum( proba_test> cvec[cc] & Y[test]==1)/sum(Y[test]==1)
  speciflr[cc] <- sum( proba_test<=cvec[cc] & Y[test]==0)/sum(Y[test]==0)
}
plot(specif,sensit,xlab="Specificity",ylab="Sensitivity",type="l",lwd=2)
lines(speciflr,sensitlr,col='red',lwd=2)
```

ROC (Receiver Operating Characteristic) Curves

ROC curve: plot TPR (sensitivity) vs FPR (1-specificity). LDA = blue; LR = red.



LR beats LDA on this dataset in terms of the **area under ROC (AUC): probability that the classifier will score a randomly drawn positive example higher than a randomly drawn negative example.** Also called Wilcoxon-Mann-Whitney statistic.

ROC Curves

R library `ROCR` contains various performance measures, including AUC.

```
> library(ROCR)
> pred_lr <- prediction(proba_test,Y[test])
> perf <- performance(pred_lr, measure = "tpr", x.measure = "fpr")
> plot(perf,col='red',lwd=2)
> pred_lda <- prediction(proba_lda_test,Y[test])
> perf <- performance(pred_lda, measure = "tpr", x.measure = "fpr")
> plot(perf,col='blue',add=TRUE,lwd=2)
> abline(a=0,b=1)
> auc_lda <- as.numeric(performance(pred_lda,"auc")@y.values)
> auc_lda
[1] 0.9472542
> auc_lr <- as.numeric(performance(pred_lr,"auc")@y.values)
> auc_lr
[1] 0.9673279
```