ELEMENTS OF DATA SCIENCE AND STATISTICAL LEARNING

SPRING 2017

Week 9

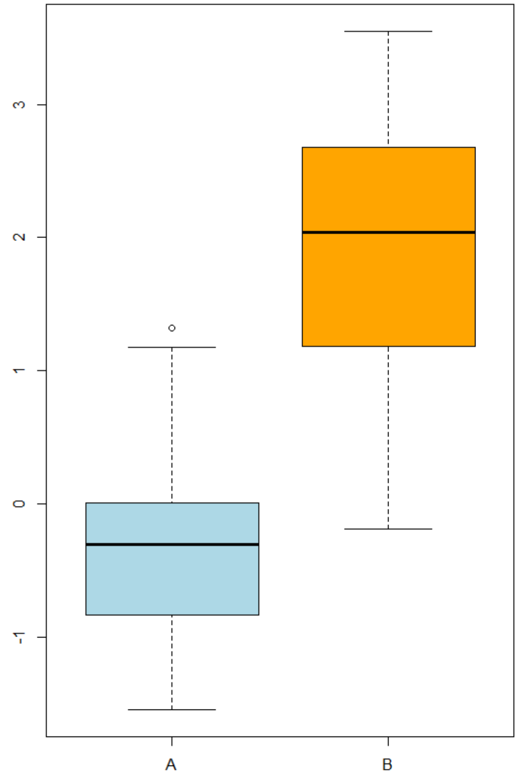
CLASSIFICATION PROBLEM

- Classification problem is the problem of learning a discrete label, in other words, the output variable is categorical
  - Recognizing specific features in the images: cat vs no cat, hand-written digits (multilevel categorical variable!)
  - Disease diagnostics (can be based on imaging data, but also on a number of clinical parameters of course): disease/healthy
  - Text classification: spam/not spam
  - Classification in the population: likely Democrat voter, likely to buy a product, …
  - In natural sciences: a chemical is/is not a ligand for a receptor (or any other property for that matter), a mutation is deleterious vs neutral, a plot in the satellite image is a field/industrial area/forest
  - …
- Arguably there might be more problems out there that require classification than those where regression is needed
  - A simple (collection of) reason(s) is that there are many cases where the outcome variable is technically continuous, but (a) we don't have enough data to perform regression, or (b) we don't have enough variables, and/or we don't have "right" variables and the regression gives mediocre results anyway, (c) we do not really care and all we want to know is whether the "continuous" variable Y is above certain threshold or not. Or all of the above.

SIMULATED DATASET

- In order to gain some intuition, let us simulate a simple dataset first:

```
set.seed(11)
x=c(rnorm(30),rnorm(30,mean=2))
y=rep(c("A","B"),each=30)
boxplot(x~y,col=c("lightblue","orange"))
```



- In the past we learnt how to deal with such data and how to model the dependence of x on y (t-test, anova, linear model x ~ y with categorical independent variable)

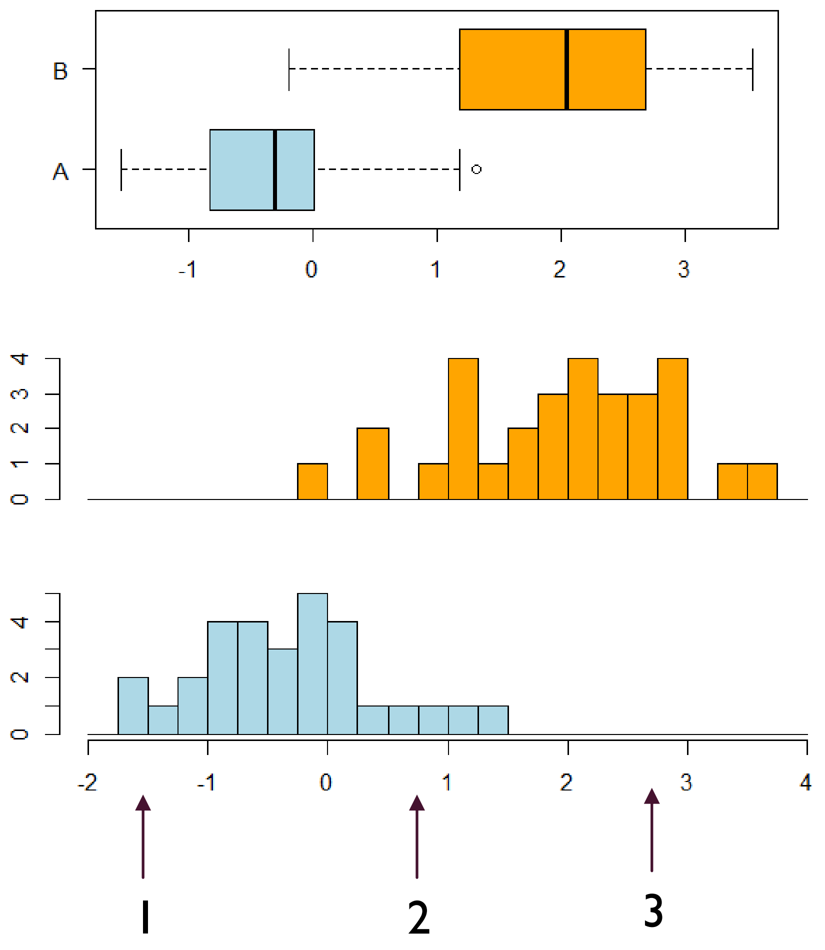
```
> summary(lm(x~y))
...
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.3286     0.1512  -2.173   0.0339 *
yB            2.2764     0.2139  10.644 2.92e-15 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
...
```

PREDICTION OF CATEGORICAL OUTCOME

- The model x~y predicts the (mean value of) X given the (categorical) value of Y
  - Note that regardless of which of the variable(s) is (in)dependent, we can always run t-test/anova/linear model to test for association (association != causation !!), testing "independent" variable vs "outcome" is fine!
- Classification is the opposite situation: variable that we can easily measure is (continuous) X, and what we want to predict is (categorical) Y. We can generate plots that (arguably) better reflect the problem we are trying to solve, as shown on the left:

```
boxplot(x~y,col=c("lightblue","orange"),horizontal=T,las=1)
oldpar=par(mfrow=c(2,1),mar=c(2,2,1,1))
breaks=seq(-2,4,by=0.25)
hist(x[y=="B"],breaks=breaks,col='orange',main="", xaxt='n')
hist(x[y=="A"],breaks=breaks,col='lightblue',main="")
par(oldpar)
```

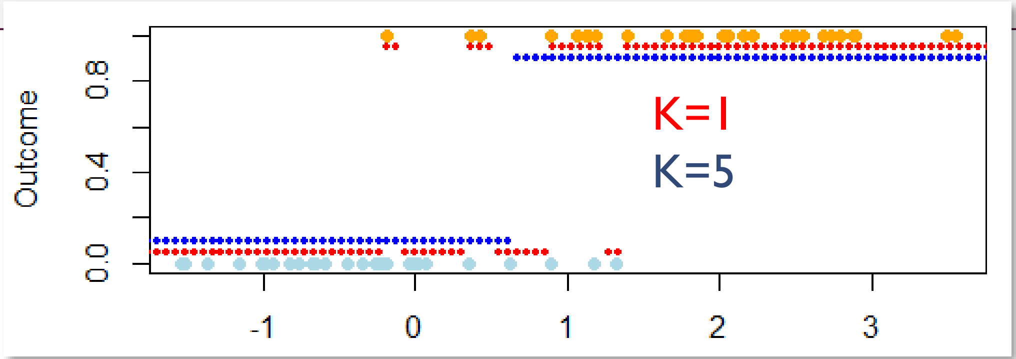
- What can we expect about the label (is it A? or B?) depending on the value of X? What would your prediction be in regions 1, 2, or 3?

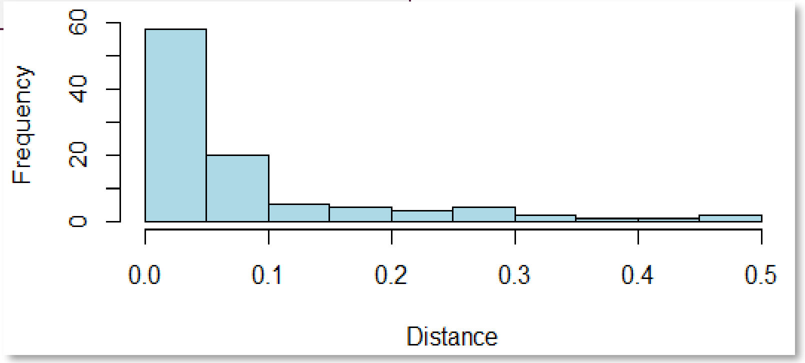


WARM-UP: K NEAREST NEIGHBORS

- We have already discussed KNN when we studied regression problem (Week 4); can be used for classification too!
  - Extremely simple idea: for any point $\mathbf{x} = (x_1, x_2, \ldots, x_p)$ we want to make a prediction for, look for its K nearest neighbors in the training data, $\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(K)}$ and define prediction f($\mathbf{x}$) as the average (regression) or majority vote (classification) of f($\mathbf{x}^{(i)}$)
  - Simple and straightforward; ideally, needs lots of observations and also needs the observations to be reasonably dense across the whole domain; does not "explain"/summarize data; choice of K provides some flexibility/tradeoff between variance and bias

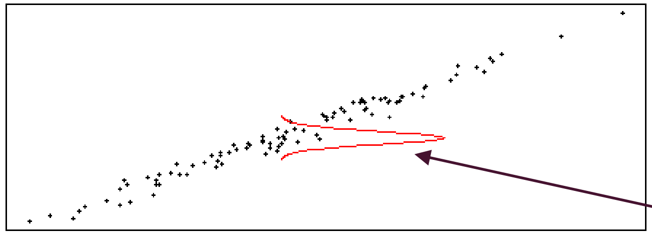
```
library(FNN)
x.new=seq(-2,4,length=100)
knn.pred.1 = knn(train=data.frame(X=x),test=data.frame(X=x.new),cl=y)
knn.pred.5 = knn(data.frame(X=x),data.frame(X=x.new),y,k=5)
plot(x,ifelse(y=="A",0,1),col=ifelse(y=="A","lightblue","orange"),pch=19,ylab="Outcome")
points(x.new,ifelse(knn.pred.1=="A",0.05,0.95),col="red",pch=19,cex=0.6)
points(x.new,ifelse(knn.pred.5=="A",0.1,0.9),col="blue",pch=19,cex=0.6)
hist(attr(knn.pred,"nn.dist"),breaks=15,col='lightblue',xlab="Distance")
```





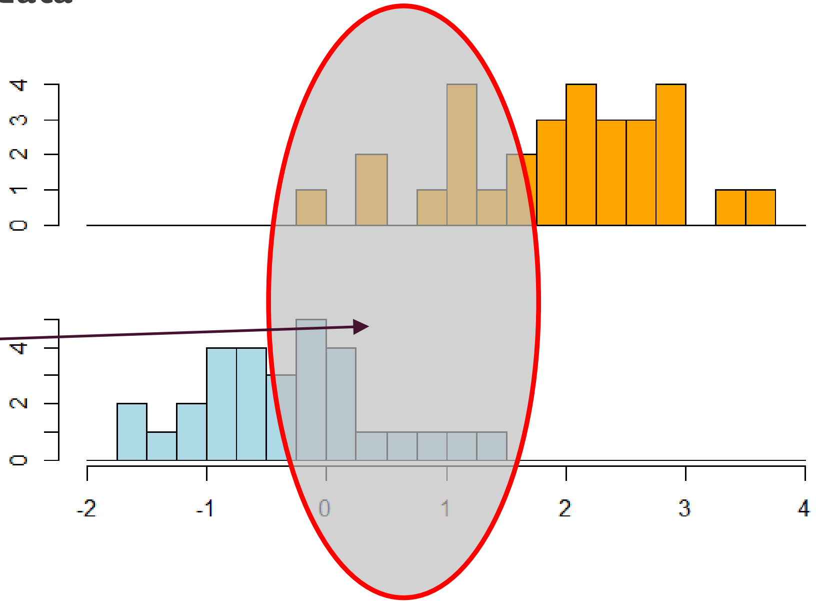
PROBABILITY OF THE OUTCOME

- What we are actually trying to learn are probabilities:
- Joint probability P(X,Y) is everything there is to know.
  - Probability density (in the case of continuous Y) or probability distribution (for categorical Y), P(Y|X) is the remaining uncertainty that *cannot be explained* within the current set of variables
  - Notation: for discrete variables we often write $P(Y=k|X)=p_k(X)$
- In real life we have only measurements and (maybe) some theory/assumptions about the underlying distributions and processes, so we are trying to come up with a model and fit the data
- We can try fitting P(Y|X) directly
- Or in fact we can try fitting P(X|Y)
  - Use Bayes theorem to invert (next slide)



Gray area. Given X, we can never be certain if it's A or B

Remember case of continuous Y? It's the same – we have uncertainty in Y even with $X=x_0$ given!



USE OF BAYES THEOREM

- General relationship between P(Y|X) and P(X|Y) is provided by Bayes Theorem:
- $P(X,Y) = P(X|Y)P(Y) = P(Y|X)P(X) \;\Rightarrow\; P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$
- We can further expand P(X) (here we assume that Y is categorical):

$P(Y|X) = \frac{P(X|Y)P(Y)}{\sum_{y_i} P(X|Y=y_i)P(Y=y_i)}$ [“Bayes classifier” ; we cannot do better!]

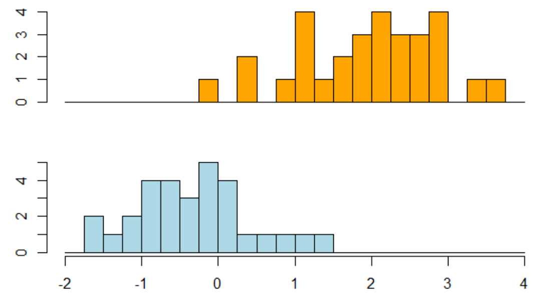
- Note that the denominator is only a normalization factor: it does depend on X but not on Y
  - When comparing probabilities of outcomes $P(Y=k|X=x_0)$ for different k *at the same* x_0, we can safely ignore the denominator!
- So is it better to model P(Y|X) or P(X|Y)?
  - Depends on the data and the assumptions we are willing to make!
  - In our case P(Y|X) is a discrete probability distribution, while (for continuous variable X) P(X|Y) is the probability density function
  - We are often looking for more general and more direct description
  - If inference is important, we would like to directly quantify the effect of X on Y, especially with many variables $X_1, \ldots, X_p$

PROBABILITY OF THE OUTCOME: USING BAYES T.

- For example, in our synthetic data (slide 3), at any value X, we can define the probability of each label, P(Y|X)

$$P(Y=B|X=x) = \exp\left(-\frac{(x-2)^2}{2}\right) / \left[\exp\left(-\frac{x^2}{2}\right) + \exp\left(-\frac{(x-2)^2}{2}\right)\right] \quad \left(\text{ cf. } P(Y|X) = \frac{P(X|Y)P(Y)}{\sum_{y_i} P(X|Y=y_i)P(Y=y_i)}\right)$$

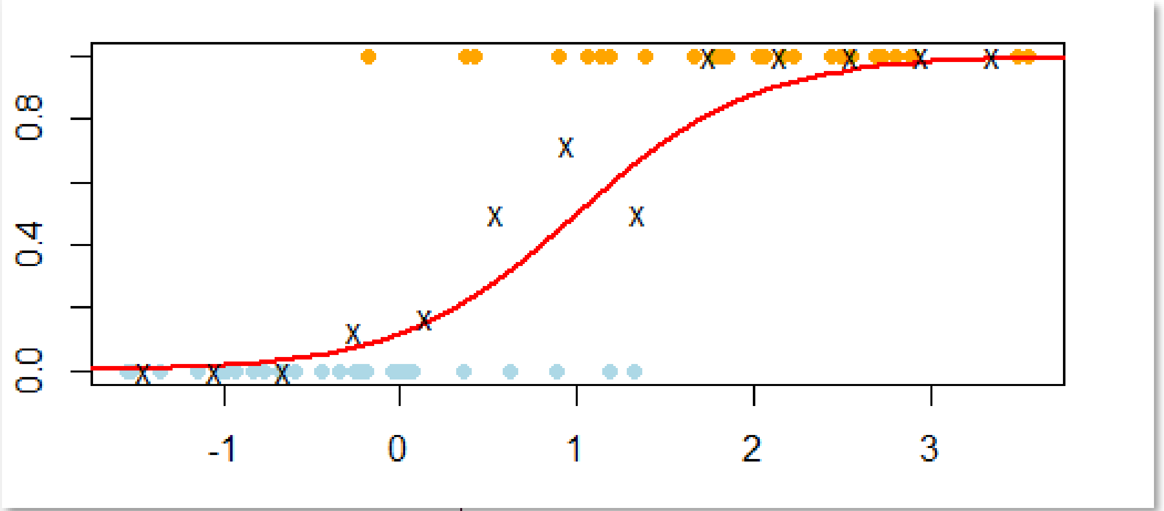
- This is by very definition: that's how we built the dataset. At each point x we sampled ~Norm(x) cases of A and ~Norm(x-2) cases of B (where we denote centered, standardized normal distribution density with Norm)
- By thinking about it: we could obtain empirical estimate of the probability "directly":
  - Split the domain of X into bins
  - In each bin calculate the proportion of each label out of all observations, i.e. $P(Y=B|X=x_0)$ = N(B) / (N(A)+N(B)), associating this proportion (probability) with the value x_0 located e.g. at the bin center
  - Rarely a good method (even KNN is a bit more flexible), we use it here only as an illustration: there is rarely enough points to confidently estimate P(X|Y) in each bin (esp. if we want them reasonably small); think also what happens if we have 100 variables – how many 100 dimensional bins?
  - If we can assume some continuous/regular dependence/distribution, we can pool observations. Compare with linear model: we may have only 1 measurement at each of different, - and sparsely distributed!! – values of X, but by assuming that they all come from the same underlying line we can pool the data together and find the slope, potentially with high confidence



PROBABILITY OF THE OUTCOME: CODE

- Let us illustrate the naïve binning model from previous slide with some code (and of course a figure!):

```
plot(x,ifelse(y=="A",0,1),col=ifelse(y=="A","lightblue","orange"),pch=19)
# theoretical (the formula at the top of prev. slide):
xx=seq(-2,4,by=0.1)
points(xx,exp(-(xx-2)^2/2)/
    (exp(-(xx-2)^2/2)+exp(-xx^2/2)),type='l',col='red',lwd=2)
# and now empirical:
# try smaller bins, the result is not pretty: too few points to get a
# good estimate in each bin when bins are small!
breaks=seq(-2,4,by=0.4)
p=numeric()
for ( i in 2:length(breaks) ) {
  sel= x >= breaks[i-1] & x < breaks[i]
  N=sum(sel)
  if ( N > 0 ) {
     p=c(p, sum(y[sel]=="B")/N )
  } else {
     p=c(p,NA)
  }
}
points(breaks[-length(breaks)]+0.2,p,pch='x',cex=1)
```



PROBABILITY OF THE OUTCOME CORRECTED

- In our first take at computing the theoretical probability P(Y|X) we missed something
- Or rather we implicitly used the fact that sizes of samples Y=A and Y=B are the same
- Take another look at Bayes Theorem:

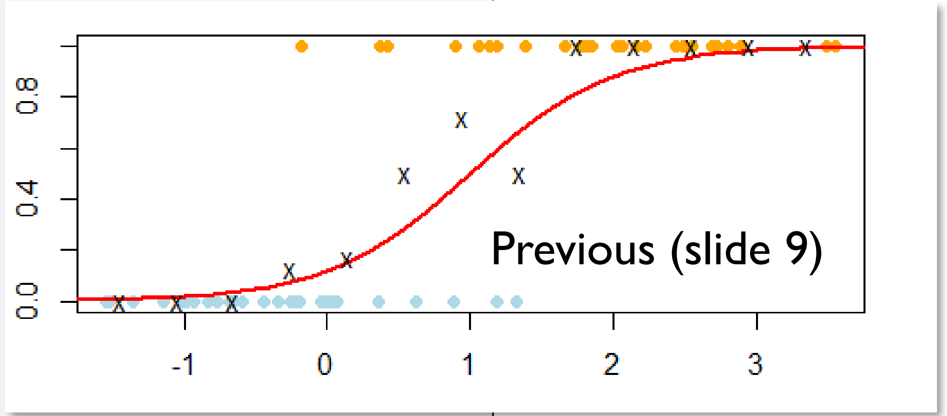
$$P(Y|X) = \frac{P(X|Y)P(Y)}{\sum_{y_i} P(X|Y = y_i)P(Y = y_i)}$$

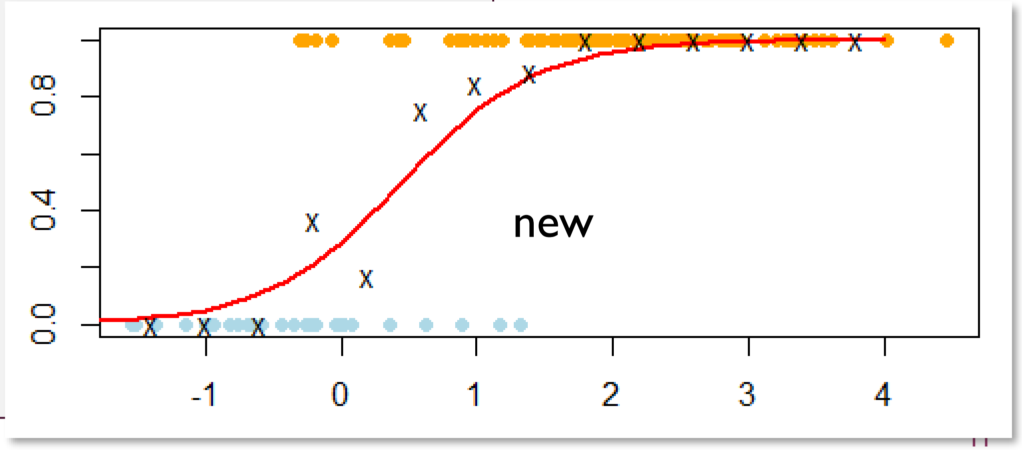
- What about P(Y)? What if we have same 30 observations of Y=A, but 90 observations of Y=B (other things being equal i.e. X coming from the same distributions)
  - It is overall more likely to observe B than A (3 times as likely in this new example to be precise)!
  - If we have even relatively small evidence pointing towards possibility of B, we want to inflate the probability that the outcome is indeed B, since B is just more frequent!
  - Our own brain is wired in the same, truly probabilistic way. Imagine that you are starving in the forest and find a tuber. You know that there are two sorts of the plant, one gives smaller tubers, 2in on average, and very nutritious; the second type gives larger tubers, 4 in on average and poisonous. The widths of the size distributions are the same, for simplicity. The plants are otherwise indistinguishable. The size of the tuber you found is 2.9in. What do you do?

    Consider the following scenarios: (1) you also know that poisonous variety is extremely rare; (2) alternatively, you know that edible variety is extremely rare; (3) in either of cases (1) and (2), imagine that you have been starving for half a day vs 3 days (i.e. what is your *decision boundary*?)

- Let us build an example to illustrate previous slide. This time we will add another 60 cases of B to previously simulated data and take into account P(Y) (now P(Y="A")=0.25 and P(Y="B")=0.75)

```
x1=c(x,rnorm(60,mean=2))  # add 60 more points with label B
y1=c(y,rep("B",60))
plot(x1,ifelse(y1=="A",0,1),
  col=ifelse(y1=="A","lightblue","orange"),pch=19)
# theoretical:
points(xx,exp(-(xx-2)^2/2)*0.75/
      (exp(-(xx-2)^2/2)*0.75+exp(-xx^2/2)*0.25),
      type='l',col='red',lwd=2)
# and now empirical:
breaks=seq(-2,4,by=0.4)
p=numeric()
for ( i in 2:length(breaks) ) {
  sel= x1 >= breaks[i-1] & x1 < breaks[i]
  N=sum(sel)
  if ( N > 0 ) {
     p=c(p, sum(y1[sel]=="B")/N )
  } else {
     p=c(p,NA)
  }
}
points(breaks[-length(breaks)]+0.2,p,pch='x',cex=1)
```

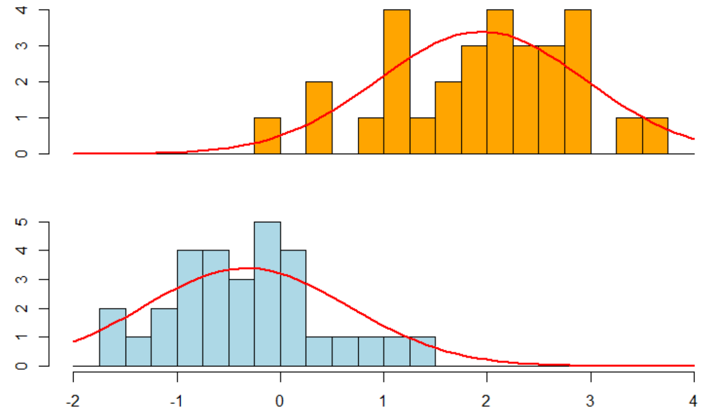




LINEAR DISCRIMINANT ANALYSIS

- Does not hurt to reiterate: when we know P(X|Y), we can always "invert" it using Bayes theorem. So what about real life, when we do not know P(X|Y)?
- **Linear Discriminant Analysis (LDA)** literally traces back the steps we used to generate our synthetic dataset
  - Take subsamples of all measured X values for each label (outcome) Y=k
  - *Assume* that these samples come from ***normal*** distributions with ***the same*** variance
  - Estimate means and (common) variance of these distributions from the data. As soon as we have P(X|Y), and know $P(Y=k)=\pi_k$ (can estimate from the data too!), we are done!
- Simplification: we do not need to calculate all the probabilities that enter the Bayes theorem. Indeed, we want only to compare the probabilities of different outcomes, at given X=x and make the prediction based on the largest P
  - Note that we compare $p_k(X=x)$ *across* k at *fixed* x, so the denominator is not important (not a function of k) and can be discarded! So now we have:
  - $P_k(X) \sim \exp[\,-(x-\mu_k)/2\sigma^2\,]\ \pi_k$

Split the predictor variable(s) – X in our case – by outcome class, estimate distributions in each class



LDA, CONTINUED

- If we take the log on both sides we obtain (discarding the irrelevant constant term):
- $\log P_k(X = x) = -\frac{x^2}{2\sigma^2} + \frac{x\,\mu_k}{2\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log \pi_k$
- Again, all we need to do is to compare $P_k(x)$ at different k, given x, but the first term (proportional to x squared) is *the same* for all k *for as long as σ is constant* (hold that thought!).
- Thus we can discard that first term as well and all we need to compute (and compare) are the quantities

$$\delta_k(x) = \frac{x\,\mu_k}{2\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log \pi_k$$

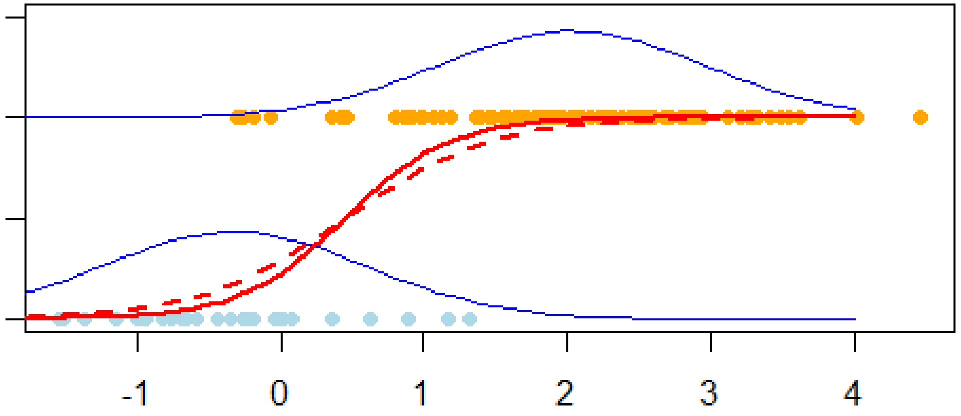
Where $\mu_k = \frac{1}{n_k}\sum_{i:\,y_i=k} x_i$ and $\sigma^2 = \frac{1}{n-K}\sum_{k=1}^{K}\sum_{i:\,y_i=k}(x_i - \mu_k)^2$.

Estimates for marginal probabilities are simply $\pi_k = \frac{n_k}{n}$ (note: if $\delta_i > \delta_j$ then $P_i > P_j$)

- If x is a vector of measurements across multiple variables, nothing changes except that we need to use a "multi-dimensional" normal distributions at each k, with covariance matrix instead of single σ. Everything else holds.
- **NOTE** that the LDA expression shown above is *linear* in x. That means that conditions such as $\delta_k > \delta_q$ define a ***linear boundary*** between cases in the space of variable(s) X

BAYES THEOREM, LDA ILLUSTRATED

- Let us perform another calculation that follows the idea of LDA (but not the simplifications: we will estimate distributions of x and use them directly, just like we did before):



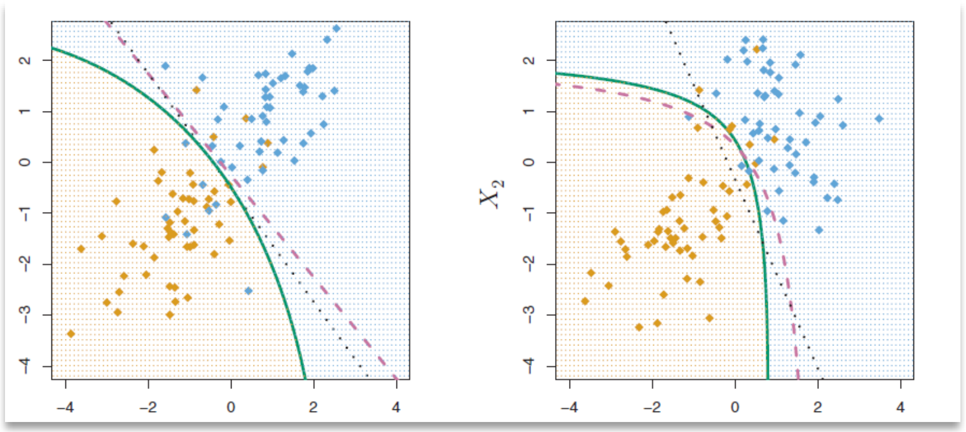
```
x1.a = x1[y1=="A"]; x1.b = x1[y1=="B"]
m.a=mean(x1.a); m.b=mean(x1.b)
# note that we calculate *single* sd across *all* observations:
s=sqrt(1/(length(x1)-2)*(sum((x1.a-m.a)^2)+sum((x1.b-m.b)^2)))
p.a=sum(y1=="A")/length(y1); p.b=1-p.a # priors P(Y=A), P(Y=B)
print(m.a); print(m.b); print(s)
[1] -0.3285999
[1] 1.979425
[1] 0.9196414
plot(x1,ifelse(y1=="A",0,1),col=ifelse(y1=="A","lightblue","orange"),pch=19,ylim=c(0,1.5))
points(xx,exp(-(xx-m.a)^2/(2*s^2))/(sqrt(2*pi)*s),col='blue', type='l') # estimated P(X|Y=A)
points(xx,1+exp(-(xx-m.b)^2/(2*s^2))/(sqrt(2*pi)*s),col='blue', type='l') # estimated P(X|Y=B)

# theoretical probability of B, TRUTH (dashed line):
points(xx,exp(-(xx-2)^2/2)*0.75/
      (exp(-(xx-2)^2/2)*0.75+exp(-xx^2/2)*0.25),type='l',col='red',lwd=2,lty=2)
# from estimated distributions, LDA-style  (STOP here and explain the difference!):
points(xx,exp(-(xx-m.b)^2/(2*s^2))*p.b/
      (exp(-(xx-m.b)^2/(2*s^2))*p.b+exp(-(xx-m.a)^2/(2*s^2))*p.a),type='l',col='red',lwd=2)
```

QUADRATIC DISCRIMINANT ANALYSIS

- Same idea as LDA (assumption of normal distributions of X in each labeled category)
- The ***only*** difference: do not assume equal variance.
  - Estimate variance σ_k^2 in each subsample i: $y_i = k$ separately.
- What happens to δ_k?
  - Now instead of σ we should use σ_k ***and***
  - We ***cannot*** discard quadratic terms anymore (they now have different denominators σ_k^2 for different k!)
- As the result, the boundary between the cases becomes a *quadratic function*.

WHEN TO USE LDA (OR QDA)

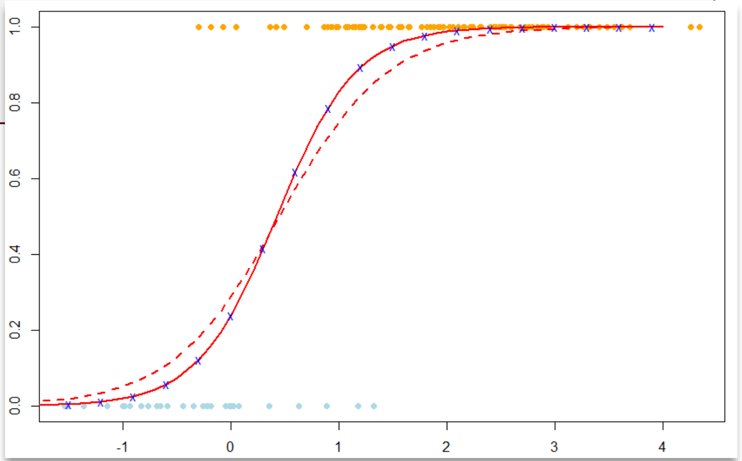
- In general, LDA makes many assumptions and is not very flexible
  - Expect low variance and high bias
  - Works best when the distributions P(X|Y) are indeed close to normal and/or when we have small number of observations
  - Works best when cases are linearly separable
  - Also works well when the classes are well separated (i.e. densities P(X|Y=k) are non- or barely- overlapping)
- QDA uses more parameters (and remember that in a multi-dimensional case, "variance" σ becomes a *p* x *p* covariance matrix, where *p* is the number of variables), so for each k we need to estimate *p(p-1)/2* parameters!
  - Needs more data otherwise serious overfitting may occur
  - Will obviously be a better choice when classes are not linearly separable



LDA FIT AND PREDICTION

- R offers function `lda()` (available from package MASS), as well as `qda()` (same package)
- The syntax and use are very similar to `lm()` (and we can also use `predict()` on fitted objects!)

```
library(MASS)
lda.fit=lda(Y~X,data=data.frame(Y=y1,X=x1))
lda.fit
Prior probabilities of groups:
   A    B
0.25 0.75
Group means:
           X
A -0.3285999
B  1.9794251
```



```
plot(x1,ifelse(y1=="A",0,1),
  col=ifelse(y1=="A","lightblue","orange"),pch=19)
# theoretical probability of B, TRUTH (dashed line):
points(xx,exp(-(xx-2)^2/2)*0.75/
      (exp(-(xx-2)^2/2)*0.75+
       exp(-xx^2/2)*0.25),type='l',col='red',lwd=2,lty=2)
# from estimated distributions, LDA-style (from slide 14):
points(xx,exp(-(xx-m.b)^2/(2*s^2))*p.b/
      (exp(-(xx-m.b)^2/(2*s^2))*p.b+
      exp(-(xx-m.a)^2/(2*s^2))*p.a),
      type='l',col='red',lwd=2)
# Actual LDA predictions:
# we use subset of xx for prediction or too many crosses!
xx.3=xx[(1:length(xx))%%3==0]
points(xx.3,predict(lda.fit,
    newdata=data.frame(X=xx.3))$posterior[,2],
    col='blue',pch='x')
```

NAÏVE BAYES CLASSIFIER

- Naïve Bayes (NB) is a method very closely related to LDA/QDA, which performs surprisingly well
- The formal idea behind NB is again to start from the Bayes rule:

  $$P(Y|X) \propto P(X|Y)P(Y)$$

- We need an expression or estimation for P(X|Y) of course in order to be able to perform the evaluation.
- In the most general form, NB prescription is to (1) use the chain rule (this is an identity that can be obtained by re-applying Bayes theorem many times):

$$P\left(x_1, \dots, x_p \middle| Y\right) = P\left(x_2, \dots, x_p \middle| Y, x_1\right)P(x_1|Y) = P(x_3, \dots, x_n|Y, x_1, x_2)P(x_2|Y, x_1)P(x_1|Y)$$
$$= \cdots = P(x_1|Y)P(x_2|Y, x_1)P(x_3|Y, x_1, x_2) \dots P(x_p|Y, x_1, \dots, x_{p-1})$$

and then (2) making a (very strong!) **assumption** that the predictor variables are *conditionally independent* i.e. that

$P(x_k|Y, x_1, \dots, x_{k-1}) = P(x_k|Y)$ for all *k*.

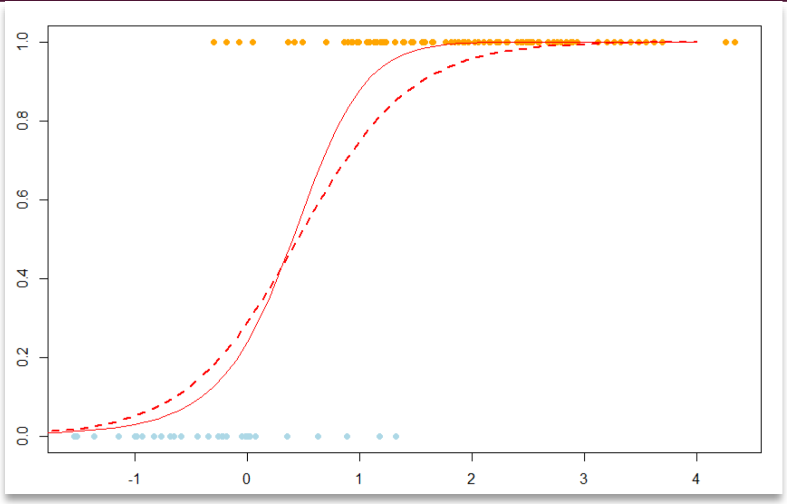
- All that's left is to estimate conditional distributions of each variable x_i *independently*. This can be done in many different ways (including direct brute force binning and counting), but you can also see that if we assume (as it is often done) that $P(x_k|Y)$ are all normal distributions with their own μ_k and σ_k (so these are the only two parameters that need to be estimated for each variable, in each class of the outcome, the NB becomes *equivalent* to QDA with *diagonal* covariance matrix!

FITTING NAÏVE BAYES AND MAKING PREDICTIONS

- Again, interface is very similar to what we have seen before (**need package** 'e1071'):

```
library(e1071)
nb.fit=naiveBayes(Y~X,data=data.frame(Y=y1,X=x1))
nb.fit
...
A-priori probabilities:
Y
   A    B
0.25 0.75
Conditional probabilities:
   X
Y          [,1]      [,2]
  A -0.3285999 0.7261498
  B  1.9794251 0.9744249
...
predict(nb.fit,newdata=data.frame(X=xx))
[1] A A A A A ...
predict(nb.fit,newdata=data.frame(X=xx),type="raw")
 A            B
 [1,] 9.925013e-01 0.007498695
 [2,] 9.916669e-01 0.008333079
 [3,] 9.906629e-01 0.009337121
```

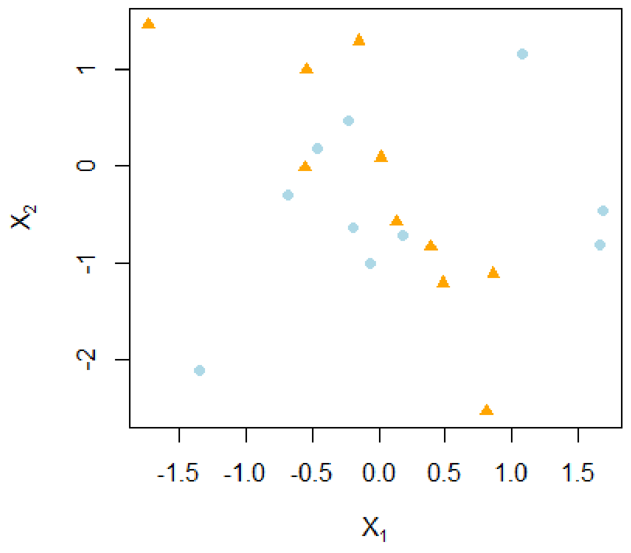
```
plot(x1,ifelse(y1=="A",0,1),
   col=ifelse(y1=="A","lightblue","orange"),
   pch=19)
# theoretical probability of B, TRUTH (dashed):
points(xx,exp(-(xx-2)^2/2)*0.75/
      (exp(-(xx-2)^2/2)*0.75+
       exp(-xx^2/2)*0.25),type='l',
      col='red',lwd=2,lty=2)
points(xx,predict(nb.fit,   # predicted prob.
      newdata=data.frame(X=xx),
      type="raw")[,2],col='red',type='l')
```



DECISION BOUNDARY

- We mentioned "decision boundary" already (i.e. "LDA has linear decision boundary")
- A boundary separating the regions, in which different outcomes are predicted
- A surface in the space $X_1 \ldots X_p$ specified by constant *predicted* probability (e.g. $P(Y=1|X_1 \ldots X_p) = 0.5$)
- Let's illustrate with another simulation:

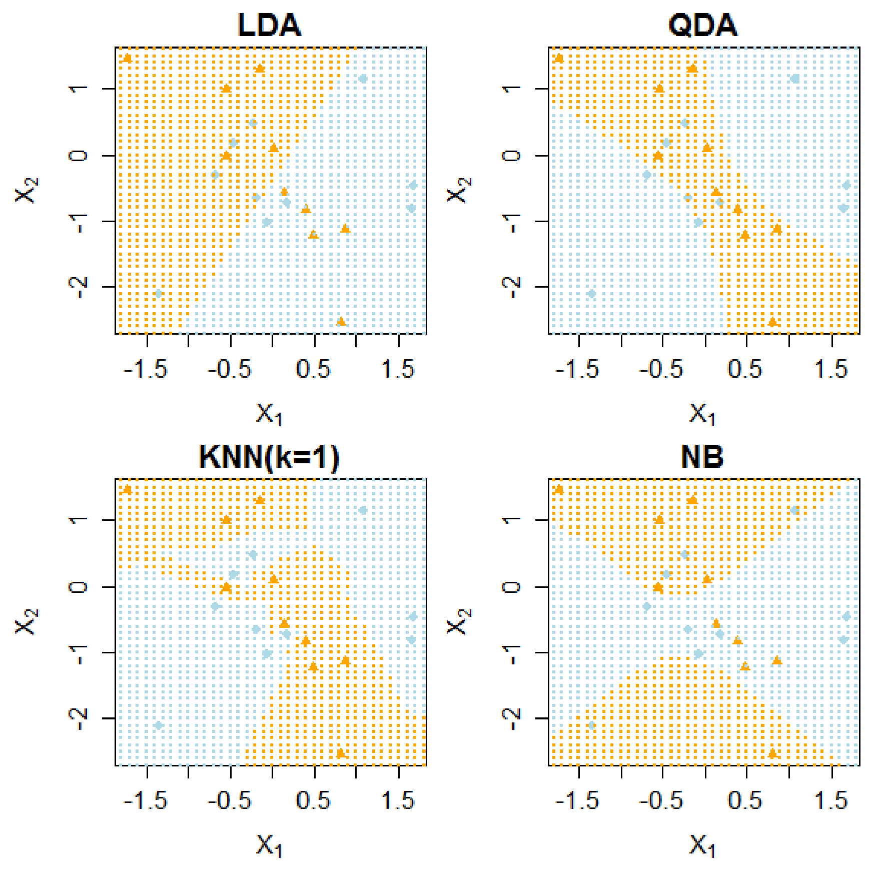
```
library(MASS)
library(class)
library(e1071)
set.seed(4);invisible(rnorm(1000))
xTrain <- cbind(rnorm(20),rnorm(20))
yTrain <- factor(sort(rep(letters[1:2],10)))
xGrid <- seq(-5,5,by=0.1)
grid2D <- cbind(rep(xGrid,length(xGrid)),sort(rep(xGrid,length(xGrid))))
plot(xTrain[,1],xTrain[,2], xlab=expression(X[1]),ylab=expression(X[2]),
   col=c("lightblue","orange")[as.numeric(yTrain)],pch=c(19,17)[as.numeric(yTrain)])
grid2d[1:3,]
     [,1] [,2]
[1,] -5.0   -5
[2,] -4.9   -5
[3,] -4.8   -5
```



DECISION BOUNDARY, CONTINUED

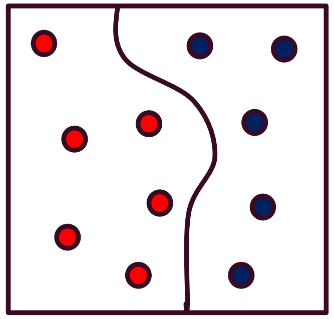
- Let's try to classify with a few methods we just learned

```
old.par <- par(mfrow=c(2,2),ps=16,mar=c(4,4.5,2,0.5))
for ( iMod in 1:4 ) {
  if ( iMod == 1 ) {
    gridPred <- as.numeric(predict(lda(xTrain,yTrain),grid2D)$class)
  } else if ( iMod == 2 ) {
    gridPred <- as.numeric(predict(qda(xTrain,yTrain),grid2D)$class)
  } else if ( iMod == 3 ) {
    gridPred <- as.numeric(knn(xTrain,grid2D,yTrain))
  } else if ( iMod == 4 ) {
    gridPred <- as.numeric(predict(naiveBayes(xTrain,yTrain),
                            newdata=grid2D))
  } else {
    stop("iMod!")
  }
  plot(xTrain[,1],xTrain[,2],
     xlab=expression(X[1]),ylab=expression(X[2]),
     col=c("lightblue","orange")[as.numeric(yTrain)],
     pch=c(19,17)[as.numeric(yTrain)],
     main=c("LDA","QDA","KNN(k=1)","NB")[iMod])
  points(grid2D,col=c("lightblue","orange")[gridPred],pch=20,cex=0.1)
}
par(old.par)
```

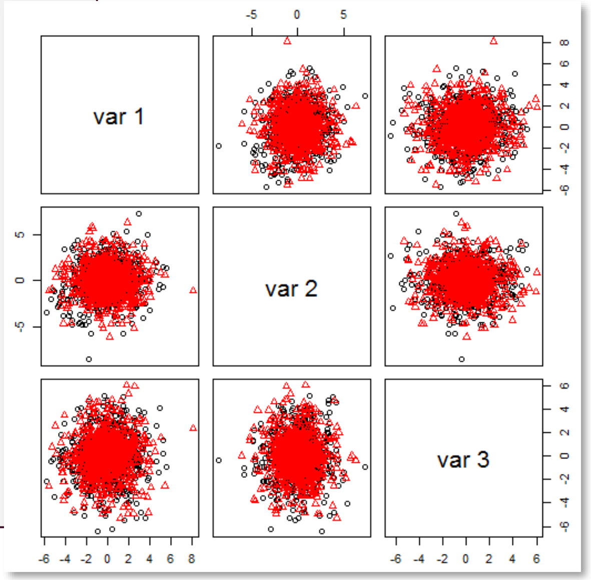


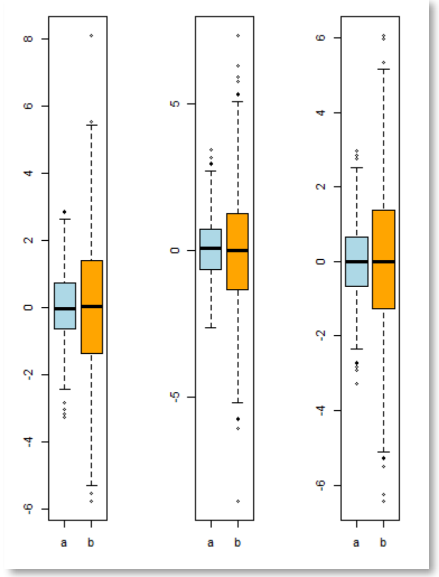
DECISION BOUNDARY FOR CLASSES WITH SAME MEANS

- A simple picture of decision boundary is the one using subsets with different locations/means
- This picture is nice (and correct), but somewhat incomplete. Consider the following example:



```
library(e1071)
library(MASS)
nObs <- 1000
nVars <- 10 # we will have TEN variables
sd2 <- 2
# simulate data: class ‘a’ has all vars normally distributed
# with sd=1, class ‘b’ has all vars normally distributed
# with sd=2. Means for all vars and classes are 0:
xTrain <- rbind(matrix(rnorm(nVars*nObs),ncol=nVars),
                matrix(rnorm(nVars*nObs,sd=sd2),ncol=nVars))
yTrain <- factor(sort(rep(letters[1:2],nObs)))
pairs(xTrain[sample(nrow(xTrain)),1:3],
      col=as.numeric(yTrain),pch=as.numeric(yTrain))
old.par <- par(mfrow=c(1,3))
invisible(lapply(1:3,function(x)boxplot(xTrain[,x]~yTrain,
          col=c("lightblue","orange"))))
par(old.par)
```





CHECKPOINT: do you think we can build a good classifier to predict a vs b?

CLASSES WITH SAME MEANS: CONTINUED

- Let's train a few classifiers and see:

```
ldaMod <- lda(xTrain,yTrain)
xTest <- rbind(matrix(rnorm(nVars*10*nObs),ncol=nVars),
               matrix(rnorm(nVars*10*nObs,sd=sd2),
               ncol=nVars))
yTest <- factor(sort(rep(letters[1:2],10*nObs)))
table(predict(ldaMod)$class,yTrain)
   yTrain
      a   b
  a 552 481
  b 448 519
table(predict(ldaMod,newdata=xTest)$class,yTest)
   yTest
       a    b
  a 5283 5182
  b 4717 4818
```

```
nbMod <- naiveBayes(xTrain,yTrain)
table(predict(nbMod,newdata=xTrain),yTrain)
   yTrain
      a   b
  a 965  81
  b  35 919
table(predict(nbMod,newdata=xTest),yTest)
   yTest
       a    b
  a 9546  891
  b  454 9109
```

IDEAS?
Hint: Does decision boundary even exists here? What is it then??
Hint: what would happen when nVars=2? (you can try!)

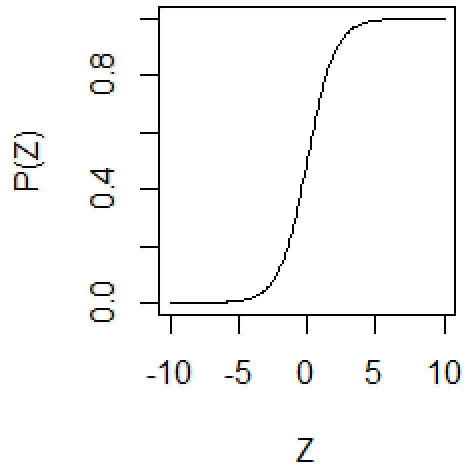
CAN WE USE LINEAR REGRESSION?

- In the past we used a trick to fit continuous vs categorical (would be X~Y for the examples we have considered today so far): we would encode levels of categorical variable with 0, 1 and just proceed with standard calculations. Why cannot we do it now?
  - Machinery **will** work, however this is not a fit we want to work with:
  - Linear regression predicts into continuous (and infinite) range from minus to plus infinity (of course we can trim the predictions and say that whenever the model predicts Y>0.5, we just interpret this as Y=1, or 0 otherwise)
  - In the two-class case, linear regression *will* in fact approximate the probability P(Y="B"), but as we have seen in previous figures, that probability is better approximated by a non-linear function (i.e. it may be asking for a transformation, stay tuned!)
  - What should we do if there are more than 2 classes? We make a separate dummy 0/1 variable *for each class* (just like we had to do with multi-level categorical independent variables in linear regression) and now we have to run regression for each of them separately (which is OK, but see above there are better ways);
    - Encoding the levels with e.g. 0,1,2,… however is *completely* inappropriate for the same reasons it was inappropriate in linear regression: such encoding (a) implies order, and even if the natural order does exist, it also (b) implies that gaps between the levels are quantifiable, comparable and meaningful. Encode as 0,1,5 instead of 0,1,2 and you will obtain a very different model (and different predictions). What is the gap between Bachelor degree and 2 year associate degree compared to associate degree vs high school diploma?

LOGISTIC REGRESSION

- Why don't we fit the probability p(Y=k) directly (assume for now that we have only 2 classes, as in our simulated example, so all we need to fit is P(Y="B")), but in a more principled and adequate way than simple linear regression could provide?
- Let us introduce a *logistic function:* $P(Z) = \frac{e^Z}{1+e^Z}$
- The plot of this function is shown to the left:

```
zz=seq(-10,10,by=0.1)
plot(zz,exp(zz)/(1+exp(zz)),xlab="Z",ylab="P(Z)",type='l')
```



- Apparently it has a nice shape, similar to what we want from P(X)
- Let us fit *Z* as a linear function of X: Z~X or Z=a+bX+e
  - By doing that we *automatically* enforce P in the range [0,1]
  - We approximate the right shape (note that logistic is NOT the only *link function* possible, any similar non-linear function would do in fact and they typically provide very similar results: there is enough flexibility in fitting the linear model Z ~ X
  - We have expressions that can still be evaluated with reasonable efficiency

ODDS RATIO

- We have the model defined. One question remains: what criterion should we use for optimization?
- Note that we can, theoretically, estimate P the way we did (brute force, by binning, or we can be smarter and use sliding window, smoothing filters etc), then use
  - $Z(X) = \log\frac{P(X)}{1-P(X)}$ which is the identity that follows directly from our definition pf P(Z), so now we can fit Z(X) "normally"
  - This is a very good *interpretation* (log p/(1-p) is *logarithm of odds ratio* or *log-odds*), a useful measure of likelihood of an event
  - But this is a bad *calculation:* too involved, the errors are not distributed well (which suggests this is just not the right model) and last but not least if only for clarity and transparency it would be nice to fit the outcome directly without another intermediate calculation

MLE IN CASE OF CATEGORICAL OUTPUT

- Recall the general idea of maximum likelihood estimation: choose parameters of the model M in such a way that obtaining the observed data would be most likely, i.e. P(D|M) is at maximum
  - "I considered parameters a_1, b_1, …, and under those parameters it would be much less likely for the observed data to "happen" compared to the parameters a_2,b_2, …" – which set of parameters is more likely to be correct?
  - If we have predicted probability $\bar{p}_k(x)$ as function of x, what is the likelihood of some particular (observed) realization of data y_1=A, y_2=A, y_3=B, …? The probability to observe y_i=*B* is just $\bar{p}_k(x_i)$, while the probability to observe y_i=A is $1 - \bar{p}_k(x_i)$. Hence,

$$P(D|M) = \prod_{i:y_i=A} \left(1 - \bar{p}_k(x_i)\right) \prod_{j:\, y_j=B} \bar{p}_k(x_j)$$

and correspondingly

$$\log P(D|M) = \sum_{i:\, y_i=A} \log(1 - \bar{p}_k(x_i)) + \sum_{j:\, y_j=B} \log \bar{p}_k(x_j)$$

- Note that this is a generalization of the formula we would use for binomial distribution (*n* trials, some resulted in A=Heads others in B=Tails): there we would use this expression with constant success probability *p* to estimate the total probability of observed numbers of heads and tails – hence "binomial family" (next slide)

RUNNING LOGISTIC REGRESSION

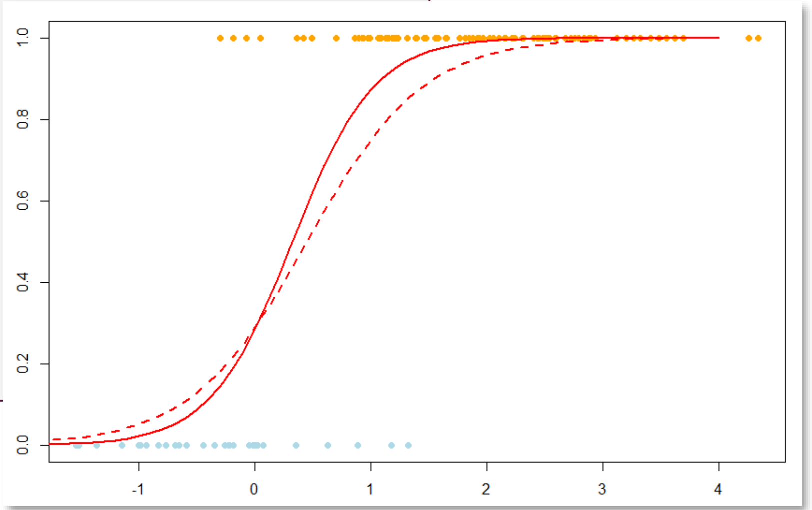
- Easy in R!
- "Generalized linear model" is a general framework for linear regression coupled to outcome via a link function That's what we need to run: generalized linear model (`glm()`) with logistic link function and binomial MLE as described in the previous slide.
  - `glm()` has an argument `family` which should be used for requesting specific error model *and* link function. See help documentation for family: you will discover that "binomial" family has default link set to "logit" (so in principle you can modify it separately) and also uses a binomial error model.
  - Conveniently, we specify the model the same way we are used to: outcome ~ variable1+variable2+…, the function does the rest. **NOTE:** glm() expects a *numeric* response (outcome) vector, so we need to encode as 0/1 or convert into a factor manually (remember, factors have numeric mode!), cant's just use a character vector c("A","A","B",…) as the outcome:

```
d=data.frame(Y=y1,X=x1)
glm.fit=glm(Y~X,data=d,family=binomial)
summary(glm.fit)
...
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -0.9243     0.4521  -2.044   0.0409 *
X             2.8518     0.5637   5.059 4.21e-07 ***
...
```

LOGISTIC REGRESSION: PREDICTING OUTCOMES

- Just like we could use linear model's fitted object to calculate predictions on either the original or new data, we can perform predictions using glm fitted object (using the overloaded function `predict()` !).
- Note that by default predictions are returned "on the scale of linear predictors" (see the docs!), in other words the returned values are Z (the log-odds). If you set optional argument type="response", the returned values will be outcome probabilities (you can choose your own decision boundary to make the final outcome class assignment)

```
d=data.frame(Y=y1,X=x1)
glm.fit=glm(Y~X,data=d,family=binomial)
summary(glm.fit)
# let us predict log odds (use type="response" to predict P directly!)
Z=predict(glm.fit,newdata=data.frame(X=xx))
# convert manually, for practice (type="response" would do just that):
P=exp(Z)/(1+exp(Z))
plot(x1,ifelse(y1=="A",0,1),
  col=ifelse(y1=="A","lightblue","orange"),pch=19)
# theoretical probability of B, TRUTH (dashed):
points(xx,exp(-(xx-2)^2/2)*0.75/
  exp(-(xx-2)^2/2)*0.75+exp(-xx^2/2)*0.25),
  type='l',col='red',lwd=2,lty=2)
# predicted with fitted glm:
points(xx,P,col='red',lwd=2,type='l')
```



ASSESSING CLASSIFICATION PERFORMANCE

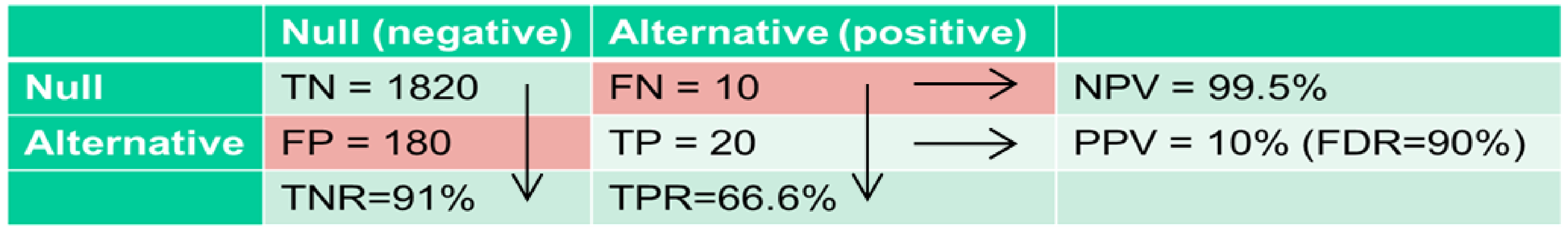
- Assessment of the performance of a classifier requires a multi-dimensional approach.
- Naïve way: the accuracy = % correct predictions. But this is far from the whole story
  - False positive and false negatives can have very different costs
  - The positive/negative cases may have very different (marginal) probabilities.
- Example 1: only 2 cases out of a 100 are "true" or `1` (can be anything: default vs non-default, disease vs healthy, bought a product vs skipped on the offer). If we "predict" "false" or `0` in all cases we achieve 98% accuracy (and if the frequency of the event `1` is indeed correctly represented in the training dataset, we will get similar "accuracy" on the test set as well!)
- Example 2: we diagnose a serious disease *mostly* correctly but miss a few cases (false negatives). With a different model, we diagnose correctly all disease cases, but also erroneously diagnose few healthy patients with the disease (false positives). Overall accuracy is good and in fact can be the same in both cases. We might prefer to have a few more false positives but catch all the disease cases (how many false positives is too many? Overdiagnosing healthy people with serious disease too often is not a good thing either!)

PERFORMANCE METRICS

- Negative Predictive Value, NPV =TN/(TN+FN); low NPV means that we are biased towards the null (lots of false negatives) and tend to miss true events, i.e. we are very conservative.
- Positive Predictive Value, PPV = TP/(TP+FP); low PPV means that we are too willing to reject the null even when we should not and thus make too many false positive calls
- False Discovery Rate, FDR=1-PPV=FP/(TP+FP); obviously, quantifies the same effect as PPV but defined as fraction of false positive calls out of all positive ("significant") calls.
- ***Sensitivity*** or True Positive Rate, TPR = TP/P = TP/(TP+FN); low TPR means that we are missing many true events (so the test is not very sensitive)
- ***Specificity,*** or True Negative Rate, TNR = TN/N = TN/(TN+FP)=1-FPR; low TNR means that we are calling too many no-events as significant (i.e. false positives), so the test is not very specific (picks up lots of cases that it should not).

Truth

Test



	Null (negative)	Alternative (positive)	
Null	TN = 1820 ↓	FN = 10 ↓ →	NPV = 99.5%
Alternative	FP = 180	TP = 20 →	PPV = 10% (FDR=90%)
	TNR=91%	TPR=66.6%	

specificity sensitivity

HELPER FUNCTION FOR ASSESSING PREDICTION QUALITY

- Count various statistics on the predictions vs truth (you can modify this function to return the list with slots TPR, TNR, PPV, … instead of just printing those to the screen)

```
assess.prediction=function(truth,predicted) {
   # check for missing values (we are going to
   # compute metrics on non-missing values only)
   predicted = predicted[ ! is.na(truth) ]
   truth = truth[ ! is.na(truth) ]
   truth = truth[ ! is.na(predicted) ]
   predicted = predicted[ ! is.na(predicted) ]
   cat("Total cases that are not NA: ",
         length(truth),"\n",sep="")
   # overall accuracy of the test: how many cases
   # (both positive and
   # negative) we got right:
   cat("Correct predictions (accuracy): ",
     sum(truth==predicted),
     "(",signif(sum(truth==predicted)*100/
     length(truth),3),"%)\n",sep="")
   # how predictions align against known
   # training/testing outcomes:
   # TP/FP= true/false positives,
   # TN/FN=true/false negatives
```

```
   TP = sum(truth==1 & predicted==1)
   TN = sum(truth==0 & predicted==0)
   FP = sum(truth==0 & predicted==1)
   FN = sum(truth==1 & predicted==0)
   P = TP+FN  # total number of
         # positives in the truth data
   N = FP+TN  # total number of
              # negatives

   cat("TPR (sensitivity)=TP/P: ",
       signif(100*TP/P,3),"%\n",sep="")
   cat("TNR (specificity)=TN/N: ",
       signif(100*TN/N,3),"%\n",sep="")
   cat("PPV (precision)=TP/(TP+FP): ",
       signif(100*TP/(TP+FP),3),"%\n",sep="")
   cat("FDR (false discovery)=1-PPV: ",
       signif(100*FP/(TP+FP),3),"%\n",sep="")
   cat("FPR =FP/N=1-TNR: ",
      signif(100*FP/N,3),"%\n",sep="")
}
```

COMPARING MODELS (A PRIMER)

- Here's how we can use the helper function from the previous slide (note that it is quite naïve and expects the classes to be encoded as 0/1 – we may fix that later!)

```
> assess.prediction(ifelse(y1=="B",1,0),ifelse(predict(lda.fit)$posterior[,2]>0.5,1,0))
Total cases that are not NA: 120
Correct predictions (accuracy): 110(91.7%)
TPR (sensitivity)=TP/P: 93.3%
TNR (specificity)=TN/N: 86.7%
PPV (precision)=TP/(TP+FP): 95.5%
FDR (false discovery)=1-PPV: 4.55%
FPR =FP/N=1-TNR: 13.3%
> assess.prediction(ifelse(y1=="B",1,0),ifelse(predict(glm.fit,type = "response")>0.5,1,0))
Total cases that are not NA: 120
Correct predictions (accuracy): 111(92.5%)
TPR (sensitivity)=TP/P: 95.6%
TNR (specificity)=TN/N: 83.3%
PPV (precision)=TP/(TP+FP): 94.5%
FDR (false discovery)=1-PPV: 5.49%
FPR =FP/N=1-TNR: 16.7%
```

CROSS-VALIDATION

- ***Everything*** we have been saying so far about cross-validation (and variable selection) applies 100%. A model, whether it is a regression model or a classifier, might have too much flexibility (too many parameters) and as the result become too eager to fit particular training dataset.
- Cross-validation can and should be used to properly assess a classifier in exactly the same way as we did for regression models:
  - Leave-n-out
  - N-fold cross-validation
  - Bootstrap
  - …
- However, when evaluating the model on the test dataset we have to assess all the performance metrics relevant for classifiers, not just the "MSE"

SUMMARY

- Classification problem
- Categorical outcome, probability of the class label
- Bayes theorem as Bayes classifier
- Classifiers:
  - LDA/QDA
  - Naïve Bayes
  - Logistic regression
- Metrics for assessing the performance of a classifier