STAT 621 Lecture Notes
Statistical Learning Part 1

Next we begin discussing the very broad field of *machine learning*, or since we'll approach this from a statistics point of view, *statistical learning*. This field combines techniques from statistics and computer science with the objective of learning about processes by analyzing data in a computationally efficient way. Often data sets are very large and complex, and there is little basis for making distributional or functional assumptions. Thus many of these procedures are inherently nonparametric.

A number of the procedures we've already encountered fall into the category of *supervised learning*. Here the focus is on building a statistical model to predict or estimate an output (Y) based on one or more inputs (Xs). Some examples:

The focus in these investigations includes

By contrast, in *unsupervised learning*, all variables are inputs. That is, there is no specific output or response. The objective is not prediction but just understanding the general structure and relationships among the variables. Some examples:

First we'll look at a few additional supervised learning techniques where the focus is on classifying a response. Here the objective is to find a set of rules to classify subjects into groups based on the values of measured inputs. First describe the data and give some examples.

1. Logistic Regression for Classification

Recall that logistic regression is commonly-used to model the probability of a binary outcome as a function of one or more covariates. Commonly the response Y is binary but models can be defined for multinomial responses as well.

First let Y be a binary variable that takes the value 0 and 1, and $\boldsymbol{X} = (X_1, \ldots, X_p)^T$ be predictors. Let $p(\boldsymbol{X}) = P(Y = 1|\boldsymbol{X})$. The logistic model is,

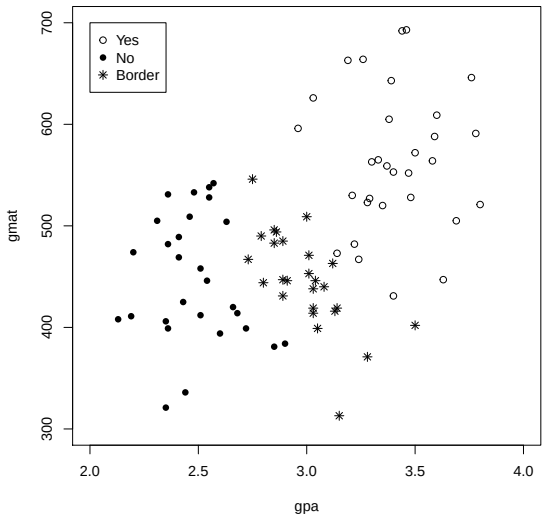
Parameters are usually estimated with maximum likelihood. Given a fitted model, a classification rule can be defined to classify a response as 1 or 0 for any given values of $\boldsymbol{X}$. For example,

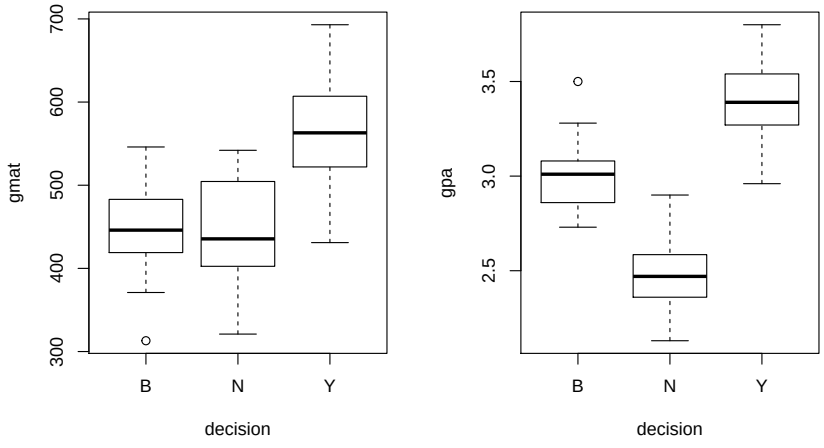
Example: We consider data from the admissions office of a business school. The data set includes each applicant's GPA, score on the GMAT, and the decision of the admissions committee: admit, no admit or borderline. There were a total of 85 applicants.

```
  decision  gpa gmat
1        Y 2.96  596
2        Y 3.14  473
...
84        B 3.03  414
85        B 3.04  446
```

First look at some plots of the variables. Comment.

```
attach(admit)
plot(gpa[decision=="Y"],gmat[decision=="Y"],xlab="gpa",ylab="gmat", xlim=c(2,4),ylim=c(300,700))
points(gpa[decision=="N"],gmat[decision=="N"],pch=16)
points(gpa[decision=="B"],gmat[decision=="B"],pch=8)
plot(gmat~decision)
plot(gpa~decision)
```





Now for the logistic regression. The basic logistic regression model is for a binary response, but our `decision` response has three possible outcomes: yes, no and borderline. To make the response binary, will just model the probability of `yes` vs. `no` or `borderline`.

```
> admit.yes=as.numeric((admit$admit==1))
> new.admit=cbind(admit,admit.yes)
```

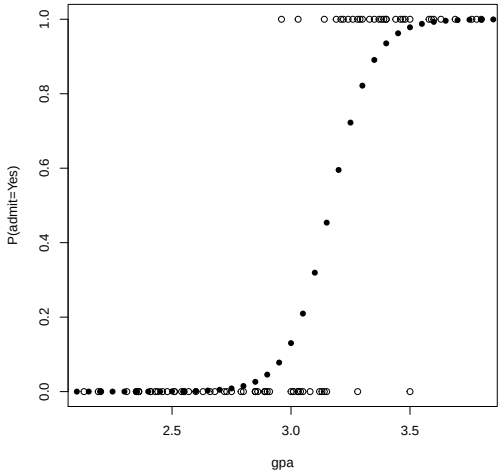
Recall the function to fit the logistic regression, as well as other generalized linear models, is called `glm`. It turns out that for these data, there is something called *complete separation*, which is why an error message occurs when we try the model with both `gpa` and `gmat`:

```
model=glm(admit.yes~gpa+gmat,data=new.admit,family=binomial)

Warning message:
glm.fit: fitted probabilities numerically 0 or 1 occurred
```

Anyone see the problem?

Since `gpa` seemed to be the better discriminator from the plots, we try a model with just that covariate.

```
model=glm(admit.yes~gpa,data=new.admit,family=binomial)
summary(model)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -36.168      8.811  -4.105 4.04e-05 ***
gpa           11.423      2.787   4.099 4.14e-05 ***
---

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 111.533  on 84  degrees of freedom
Residual deviance:  36.719  on 83  degrees of freedom
AIC: 40.719

Number of Fisher Scoring iterations: 7

plot(gpa, admit.yes, xlab="gpa", ylab="P(admit=Yes)")
newdata=data.frame(gpa=seq(2,4,.05))
phat=predict(model, newx, type="response")
points(newdata$gpa,phat, pch=16)
```



Multi-Category Logistic Regression Models: Often the response of interest has more than two categories (multinomial). The basic logistic model can be extended to these responses. We'll describe these models just briefly, and discuss how they may be used for classification.

Consider a response Y that has J categories. Below we'll give a brief description of logistic models for this response, assuming that the J categories are *nominal*, or unordered. Other models take advantage of the structure of responses that are *ordinal*.

2. Kernel Classification and Naive Bayes

Again assume we have a response Y that takes one of J classes. Also assume we have a single continuous predictor variable, X. Our goal is to estimate $P(Y = j|X = x)$, the probability that the response falls into class j as a function of the value of the predictor.

The kernel density estimator that we discussed earlier can be used in the following way to estimate this probability. First the following quantities are estimated.

- $\widehat{f}_j(x)$ for $j = 1, \ldots J$

- $\widehat{\pi}_j$ for $j = 1, \ldots, J$

An expression for $P(Y = j|X = x)$ follows from Bayes' Theorem.

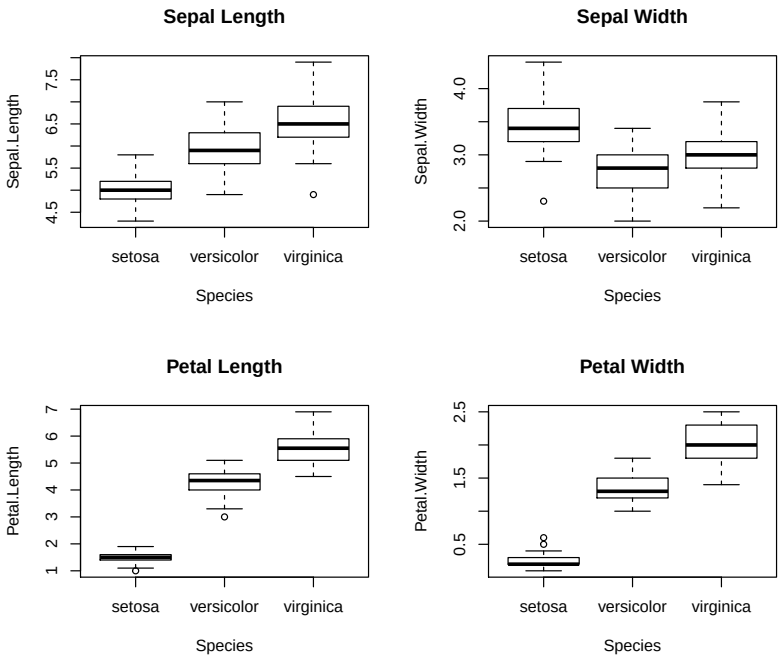
Substituting in our above estimates yields the kernel classifier,

This method generally works well when n is large and we have a single predictor X. However it brakes down in the situation where the objective is to predict the response class from multiple predictors, $X_1, \ldots, X_m$. Then, we try to estimate $P(Y = j|X_1 = x_1, X_2 = x_2, \ldots, X_m = x_m)$. What's the problem?

Naive Bayes Classifier: This is essentially the kernel classifier with the added (naive) assumption that given $X_1, \ldots, X_m$ are conditionally independent given $Y = j$. Under this assumption the kernel classifier becomes:

Example: This example uses the `iris` data set in R to classify flowers according to species based on measurements of four predictors, sepal length, sepal width, petal length and petal width. The three species of iris are setosa, versicolor and virginica. The data set has $n = 150$ observations, 50 of each species.

First some plots. Also I break the data set up into 100 observations to train the classifier, and 50 observations to test the classifier.




```
a=sample(1:150, 50, replace=F)
test.data=iris[a,]
train.data=iris[-a,]
```

The R function `klaR` will compute the naive Bayes classifier using the function `NaiveBayes`. By default this estimates the conditional densities of the predictors by assuming they are Normal. Use the `usekernel=T` option to estimate these with a kernel density estimator.

Below the model is fit and used to predict `Species` on the training set. I make a summary table and some plots.

```
libaray(klaR)
nb.iris=NaiveBayes(Species ~ ., usekernel=T, data=train.data)   # formula Y~dataframe of X's
pred=predict(nb.iris, test.data)

table(pred$class, test.data$Species)
```

```
            setosa versicolor virginica
  setosa         12          0         0
  versicolor      0         23         0
  virginica       0          2        13
```

```
library(ggplot2)         # basically plot the above table
ggplot(test.data, aes(Species, pred, color = Species)) +
  geom_jitter(width = 0.2, height = 0.1, size=2) +
  labs(title="Predicted vs. Observed from Iris dataset",
       y="Predicted", x="Truth")

plot(nb.iris)   # plot estimated cond. densities
```