Lecture 1

IDS575: Statistical Models and Methods
Theja Tulabandhula

We are drowning in information and starving for knowledge. - Rutherford D. Roger

Notes derived from the book titled "Elements of Statistical Learning [2nd edition] (Chapters 1 and 2.1-2.3)

1 Motivation

The impact of data science on a variety of fields, businesses and areas of works needs no introduction. So, here is a list of problems that involve some sort of statistical modeling or other. I am sure you can all relate statistical modeling to your past professional experiences.

- Sports analytics
- Online retail and pricing
- Advertising and personalization
- Social networks
- Education and MOOCs
- Transportation systems
- Automatic speech recognition
- Robotics and computer vision, image processing
- Fraud detection and finance
- ICU monitoring and clinical decision support
- ...

What are some common components among these applications?

1.1 Application in Sports Analytics

Here is an example of how a decision support tool was built using statistical modeling. The context is that a team makes pit stop calls during a race. When to make a call and what to do in it constitutes a pit strategy. for example, replacing four tires (Figure 1) takes more time but the car is much better with respect to lap times.

Figure 1: Car racing: a crew member inspecting a tire.

- The problem to be addressed is the following: use live race measurements to forecast future and decide/update a pit strategy in real time.
- Is there even predictability in this setting?
- Data: text strings are beamed by the organizers over the radio, so there could be several noisy values.

Initial exploration to build features from 17 races ($\sim$ 100000 laps) resulted in the following plots (see Figure 2).

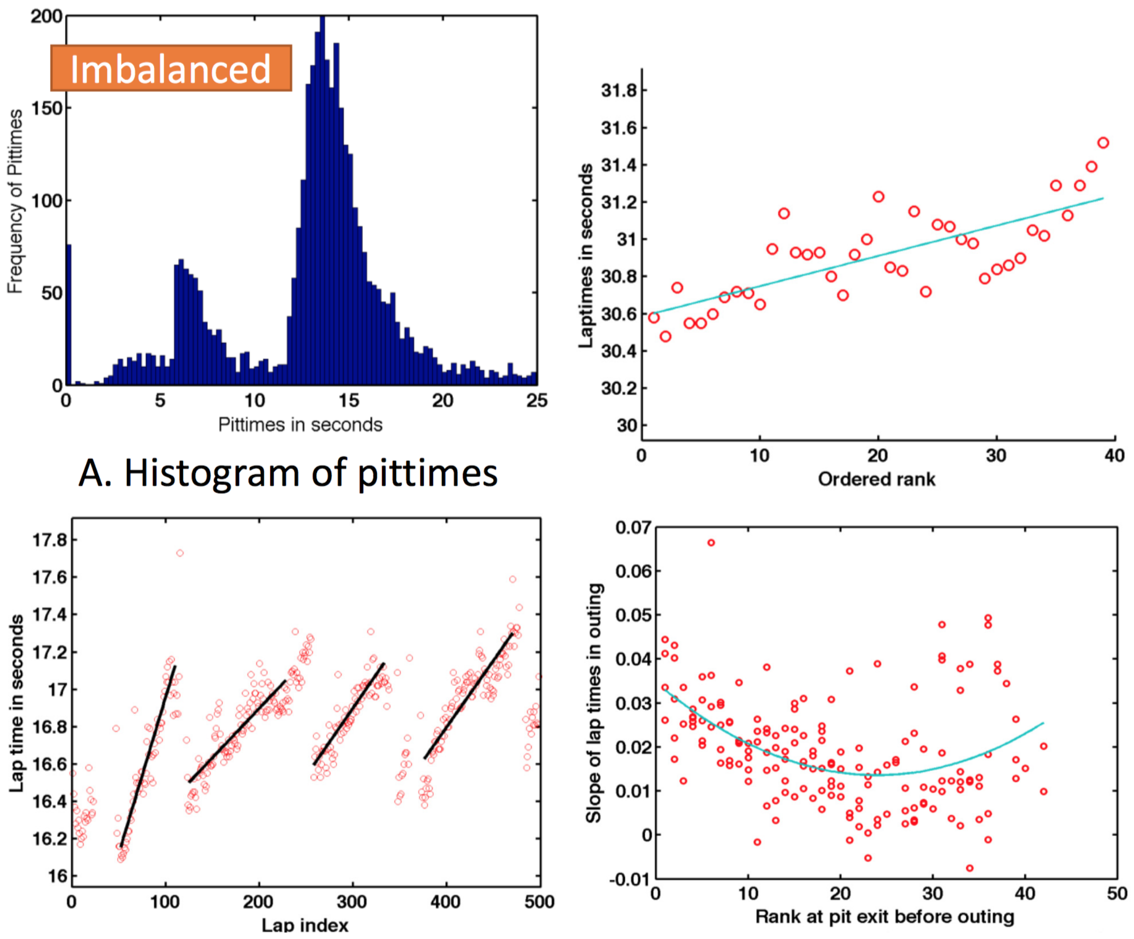


Figure 2: Some histogram and correlation plots to design features.

So after that, one can jump into using off the shelf predictive models to predict a quantity of interest. In this particular application, it was the change in rank position in the laps following a pit stop given measurements of what happened in the pits and what happened before in the race. The performance of various models is shown in Figure 3.

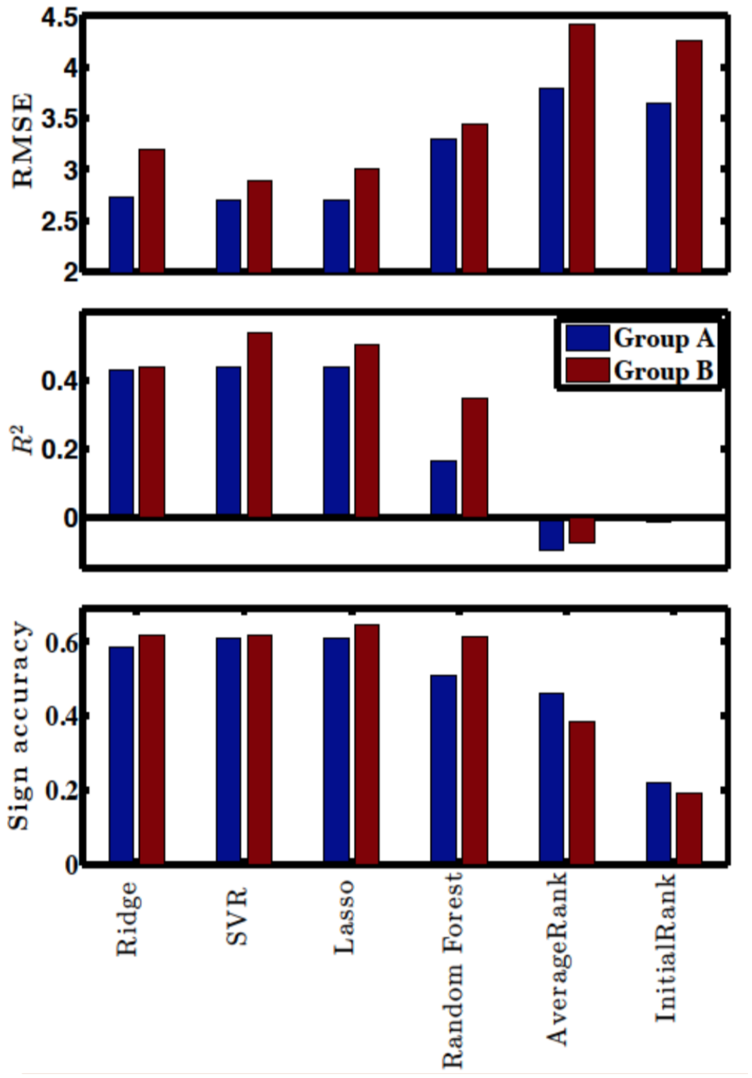


Figure 3: Different predictive models.

All models look similar in performance. What to do next? Can this be improved? Is this accuracy enough to actually build a strategy? By the end of this course, you will be in a better position to answer these questions!

2 Building a Language for Statistical Modeling

In the applications described above, we use some keywords. These are:

- Inputs: these *variables* are measured in some way: e.g., can be physical measurements, can be answers to a survey, etc.
- Outputs: we hypothesize that inputs influence the values these *variables* take.

Informally, we can define supervised learning as the problem of finding a relation between inputs and outputs. You can think of it as a function or a mapping. The method that does the search/finding is typically called a *learner*.

Below is a table of equivalent keywords that you may have come across:

Inputs	Predictors	Independent variables	Features
Outputs	Responses	Dependent variables	Labels

Lets focus on outputs for a bit. These can be quantitative, i.e., taking numerical values. Or they can be qualitative, with no order relationship between the values the outputs take.

When the learner learns a mapping for the quantitative setting, we will call the mapping a regression function and the process *regression*. On the other hand, when the learner learns a mapping for the qualitative setting, we will call the mapping a classifier, and the process *classification*.

2.0.1 Examples

For instance, in the setting where the learner learns a cat classifier, we can specify 1-dimensional representation for the output that takes a value 0 when there is no cat and 1 otherwise. Of course, you could have chosen value 0 to indicate the presence of a cat instead. This is what we mean by no order relationship. These values are sometimes called *targets*.

Another example is that of data drive decisions. Think of a stylized example where an autonomous car's controller is designed using statistical modeling. Then all the features that are measured (location on the road, location of other people and cars, traffic signals) can be mapped to descriptive actions or decisions such as *accelerate, brake, turn left, steer right by 10 degrees* etc., using supervised learning. Outputs such as these are also called *discrete* or *categorical*.

2.1 Describing Statistical Modeling

We will use statistical modeling as a way to describe data mining and machine learning solutions where statistics and probabilistic models play a key role. I will enumerate the different types of (application independent) problems that comprise this area:

- Pattern mining: processing databases to find *rules* (e.g., bread $\rightarrow$ oatmeal).
- Clustering: group objects that *belong* together.
- Classification: map inputs to outputs, where the values that outputs take have little/no relation with each other.
- Regression: same as above, but with values of outputs having some relationship. Typically these values are numerical.

- Ranking: order new data using historical data. This is related to classification and regression.
- Density estimation: finding the probability distribution that describes how data realized.
- ...

Both regression and classification can be viewed as a function approximation problem: given inputs and outputs, fit a *nice* function.

A language to describe these various problems is as follows. We will use X to represent an input variable or a vector of input variables (in the latter case, the jth variable is X_j). Quantitative outputs are denoted Y and qualitative ones G.

Lets observe N realizations of input X. The ith observed input is denoted using x_i. Say this is p-dimensional. Then if you stack them together as rows, you get a matrix $\mathbf{X}$ which is $N \times p$ dimensional.

Why did we introduce this notation? We did it to describe the learning task more formally: given X, find a function that predicts $\widehat{\mathrm{Y}}$ that is *close to* Y. For the qualitative setting, replace Y with G.

3 Two Learners

We will look at linear regression and k-nearest neighbor, especially because they are very different from each other in various ways and will give you an idea about the solution space (you could come up with your own learner if you understand these two!)

We will use the word *model* to describe the function or mapping output by a learner.

3.1 Linear Model

A linear model is given by: $\widehat{\mathrm{Y}} = X^T \widehat{\beta}$ (note: vectors are always column vectors unless specified explicitly). $\widehat{\beta}$ is the parameter of the model. That is, $\widehat{\beta}_1$ and $\widehat{\beta}_2$ specify two different models. If $\widehat{\mathrm{Y}}$ is K dimensional, then $\widehat{\beta}$ is $K \times p$ dimensional.

If we want to find a function approximator that maps Xs to Ys and we are allowed to only search in the space of linear models, we can do the following: we can pick an objective that measures how well we are doing.

For example, let the objective be $\sum_{i=1}^{N}(y_i - x_i^T\beta)^2$. This is called the least squared objective[1]. If we minimize this, we get the best approximator $\widehat{\beta}$, where best is defined precisely in terms of the objective function.

[1] sometimes also called the residual sum of squares

If we have enough observations such that $\mathbf{X}^T\mathbf{X}$ is not singular, then there is a single $\widehat{\beta}$ that minimizes the least squares objective, and it is given by $\widehat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$.

Think of it this way: we reduced the $N \times p$ matrix and the N observed outputs (vectorized as $\mathbf{y}$) to a p-dimensional vector. This vector allows us to make predictions. In some sense, we have done *compression*, We do not need the original data (observations of inputs and outputs) anymore.

3.1.1 Example

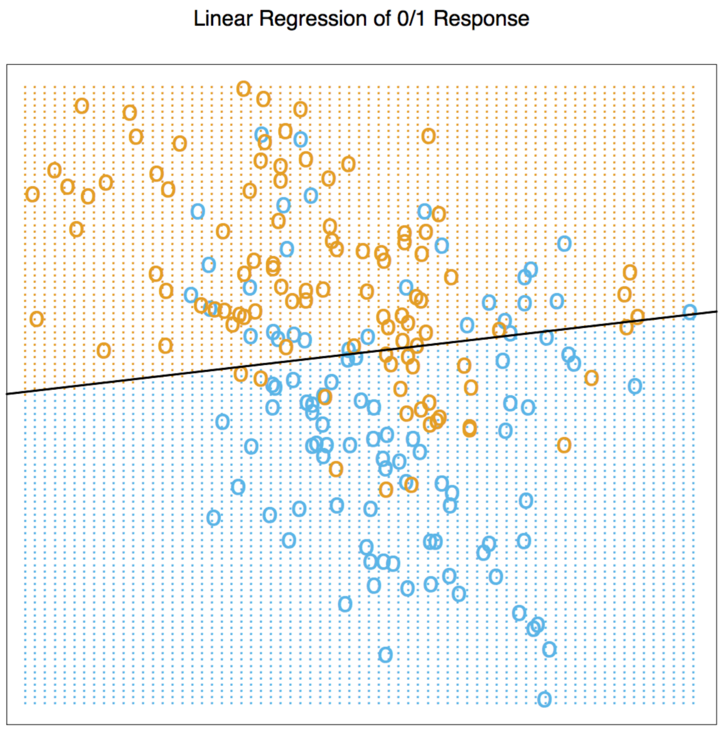


Figure 4: Linear model with 2-dimensional data.

Figure 4 shows a scatterplot of the observations of inputs which are 2-dimensional. The colors of each point represent the value that output variable G takes. We will use a linear model to fit this data (say blue is 1 and orange is 0) and convert $\widehat{\mathrm{Y}}$ to $\widehat{\mathrm{G}}$ via a non-linear transformation:

$$\widehat{\mathrm{G}} = \mathbf{1}[\widehat{\mathrm{Y}} > 0.5].$$

The black line corresponds to the decision boundary. Here $x^T\widehat{\beta} = 0.5$. The decision boundary is linear.

As you can see, the boundary does not separate blue points from red points well. We can eyeball and see this for 2-dimensional data. What if we have 100 dimensions? We will necessarily have to rely on projections or other summary diagnostics to assess model fit. We will discuss this much more detail in a subsequent lecture.

Lets take a tangent and hypothesize how the 2-dimensional data was generated. What if it is just a mixture of two 2-dimensional Gaussians? One can reason that if these two Gaussians overlap too much, a linear decision boundary will not be adequate. Would any other boundary be better?

What if it was a mixture of mixtures of Gaussians, whose means were sampled from Gaussians? Sounds pretty complex, isn't it?

Note that a mixture of Gaussians can be described *generatively*. Think of a Bernoulli random variable. If it takes value 1, you sample a point from one Gaussian, and if it takes value 0, you sample from a different Gaussian. This is a generative description of a mixture of two Gaussians. We will see later that a linear boundary is the best in the least squares sense when data is realized this way.

If it is a much more complex model (mixture of mixture of ...) then a linear boundary may be very bad.

3.2 Nearest-neighbor Methods

These are *lazy* methods. Given a input, you can predict its output $\widehat{\mathrm{Y}}$ as:

$$\widehat{\mathrm{Y}} = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i.$$

Here $N_k(x)$ is the *neighborhood* of value x. It is a set of k points closest to x in terms of some *metric*.

What is a metric?

3.2.1 Example

Figure 5 shows the same data as Figure 4 and uses $k = 15$. There is no fitting here (except for choosing k).

A point is orange if at least 8 of the 15 neighbors of that point are orange, and vice-versa.

As can be inferred from the two figures, we get a non-linear decision boundary with this method. Also, arguably a better fit[2]

What if we choose $k = 1$? We get a Voronoi tesselation. There are many decision boundaries! See Figure 6.

What would change if we were doing regression instead?

[2]We have only been looking at *in-sample* fit!

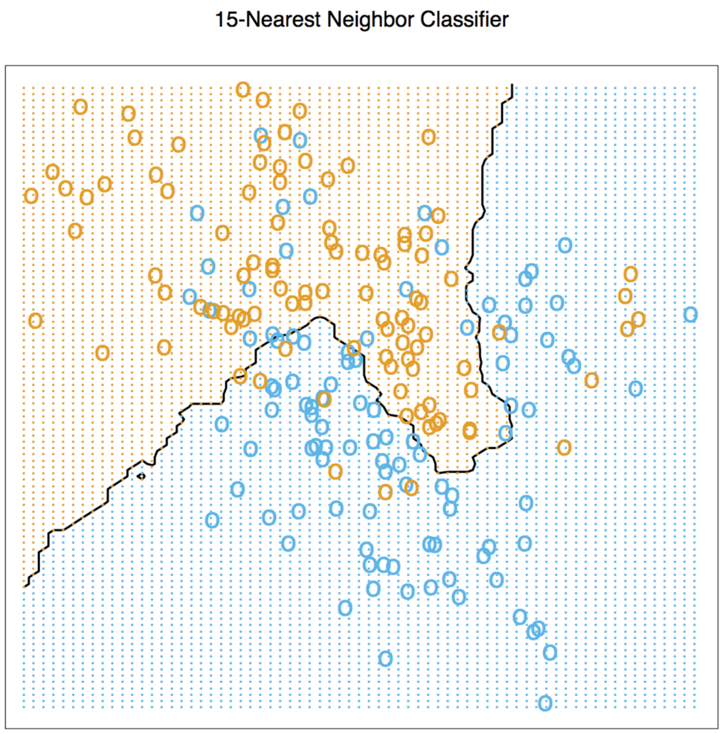


Figure 5: 15-Nearest neighbor method with 2-dimensional data.

3.3 Comparing Nearest-neighbor with Linear Modeling

If you thought nearest-neighbor methods are superior, think again. In Figure 6, all the points are classified correctly. Hence, we can reason that as we increase k, our errors will increase, starting from value 0 when $k = 1$. These errors are somewhat meaningless though. What we should care about is performance on an independent *test set*.

How do we pick k? Does its choice matter? If you used the least squares objective[3] to pick k, what would you get?

If the data in Figures 5 and 6 were from two overlapping Gaussians, k-nearest-neighbor methods would probably capture 'noisy patterns'.

3.3.1 Comparing Parameters

We know that the linear model had p parameters. How many parameters does the k-nearest-neighbor method have? 1. But there is more to this. It turns out that the *effective* number of parameters is N/k, which grows with N and shrinks with k.

The intuition is as follows: lets say the neighborhoods were non-overlapping. Then there would be N/k neighborhoods and we would fit one parameter (the mean) in each neighborhood.

[3]That is, for each k, we find the residual sum of squares.

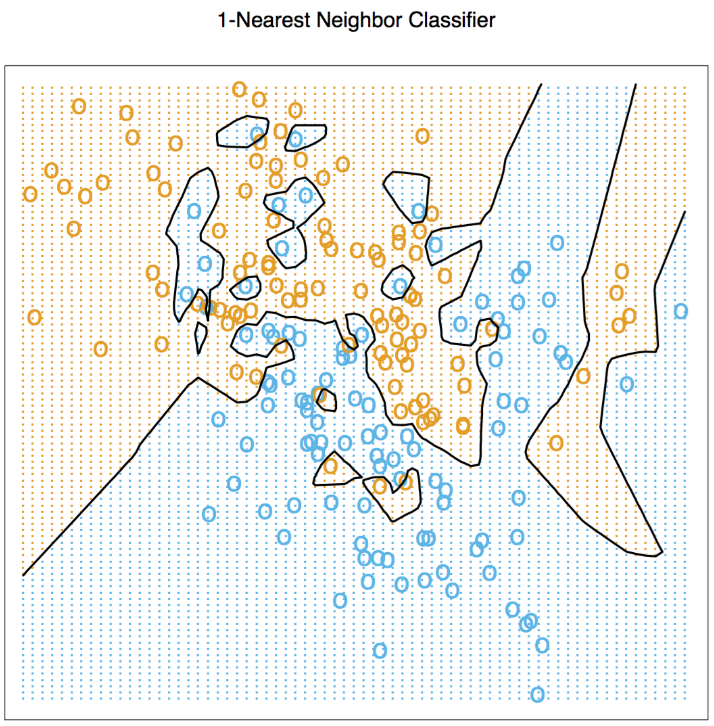


Figure 6: 1-Nearest neighbor method with 2-dimensional data. More-irregular than the 15-Nearest neighbor method.

3.3.2 Comparing Decision Boundaries

The decision boundary from a linear model is *stable* and *smooth*[4] This is a manifestation of *low variance* and *high bias.*

For the k-nearest-neighbor, since any classification decision only depends on a few nearby points, it is unstable and wiggly. But it can adapt to complex data. This is a manifestation of *high variance* and *low bias.*

As you will see throughout the course, there is no single best method for making the data talk. Any method can shine if its properties are aligned with the unknown true characteristics of the data. This is true of deep neural nets as well.

3.3.3 Comparing Testing Performance

In the examples, we have 200 *training* observations. The performance of nearest neighbor methods is shown in Figure 7 as a function of N/k over an independent test data (10000 in number, so fairly accurate representation of how these methods do in reality). The

[4]We are using both these terms loosely here.

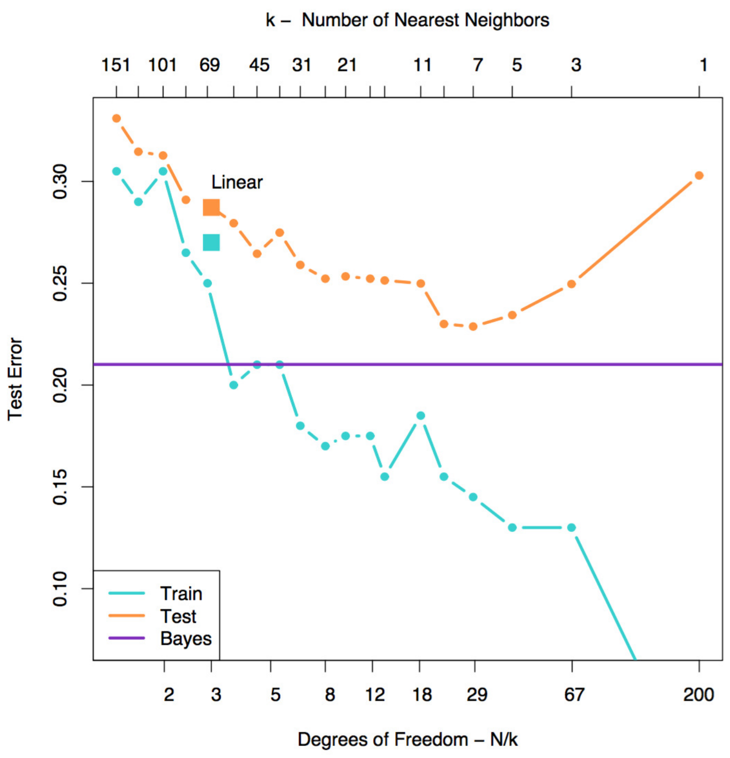


Figure 7: Error for different models.

performance is just the number of points on the wrong side of the decision boundary. The figure also includes the performance of a linear model with $p = 3$.

Many methods discussed in the course can be thought about in similar terms, and will work intuitively the same way as either of the two methods we just saw.

4 Summary

We learned the following two things:

- Introduced vocabulary to describe statistical modeling.
- Looked at two intuitive prediction models: the linear model and the k-nearest-neighbor method.

In the next lecture, we will discuss the bias-variance tradeoff and jump into the nuances of regression modeling.

Lecture 2

IDS575: Statistical Models and Methods
Theja Tulabandhula

Notes derived from the book titled "Elements of Statistical Learning [2nd edition] (Sections 2.4-2.9, 3.1-3.2)

1 Statistical Decision Theory

We will now formally make the variables into random variables! Allowing us to bring in statistics and probability into the mix.

Let $Pr(X,Y)$ be a joint distribution. We want a function $f(X)$ for predicting Y, and its quality is measured using a *loss function* $L(Y, f(X))$. The loss function allows us to define what we mean by a 'best' model.

Example 1. Loss function: $(Y - f(X))^2$. □

So from the informal least squares objective, we can generalize to the following objective, called the *expected (squared) prediction error* (**EPE**):

$$\begin{aligned} EPE(f) &= E(Y - f(X))^2 \\ &= E_X E_{Y|X}([Y - f(X)]^2 | X). \end{aligned}$$

We want to minimize this to get an f. One way to thin about it is to observe that the function value $f(x)$ at each point x is unrelated to other points. So you have one optimization problem per x which looks like:

$$\min_z E_{Y|X}([Y - z]^2 | X = x).$$

This is minimized at $z = E(Y|X = x)$. Thus, the function defined by $f(x) = E(Y|X = x)$ is the optimal solution. It is called *the regression function*.

Lets revisit k-nearest-neighbor and linear modeling: they are in fact trying to estimate the regression function $E(Y|X = x)$!

1.1 Reinterpreting the Nearest-neighbor Method and the Linear Model

Nearest-neighbor: If we want to estimate $E(Y|X = x)$ at a point x, a natural thing to do is: average all y_i such that $x_i = x$.

Further relax this by taking those y_i into account for which $x_i \sim x$ (i.e., in the neighborhood).

For large number of points, we can expect that the neighbors are close to x so this approximation make sense.

The intuition here is that: (a) large N will get you many neighbors, and large k will get you stable estimation.

In fact, theoreticians have shown that for many different $Pr(X, Y)$, as long as N and k go to ∞, and k/N goes to 0, $\widehat{\mathrm{f}}(x) \to E(Y|X = x)$.

Is this the end of the story? Obviously no.

- The catch is, this is an *asymptotic* result.
- There is also another issue which will make Nearest-neighbor methods ineffective, which is called the ***curse of dimensionality***.

Linear model: What if we assume that the regression function $E(Y|X = x)$ is of the form $x^T\beta$? We are assuming something about the data then. If we make this assumption, it turns out that:

$$\beta = [E(XX^T)]^{-1}E(XY).$$

We have a single parameter β that does not depend on a specific value of x, it depends on the entire distribution of X (through the expectation).

If you compare this to the least squares solution, you will notice that the least squares solution is the empirical average version of the above.

Lets summarize Nearest-neighbor method and the linear model via least squares, now that we know about the expected prediction error EPE and $E(Y|X = x)$.

- Both methods do averaging
- Nearest neighbor assumes $E(Y|X)$ is locally constant.
- Least squares assumes a (globally) linear function for $E(Y|X)$.

Note 1. Instead of assuming $f(X) := E[Y|X] = X^T\beta$, one could assume $f(X) = \sum_{i=1}^{p} f_j(X_j)$. Can you see why this is more general than the linear model?

Note 2. What if we change the objective to $E|Y - f(X)|$? It turns out that the best fit is $f(x) = \text{median}(Y|X = x)$. It is not clear why one would choose one objective over the other a priori[1].

1.2 EPE for Classification

If we have categorical variable G (with K values) instead of Y then we can define an 'EPE' objective using a $K \times K$ matrix.

Example 2. This is the reasoning behind the zero-one loss function: a loss of 0 when correct prediction happens, and a loss of value 1 when incorrect prediction happens. □

The expected prediction error (EPE) in this case is:

$$EPE = E_X \sum_{k=1}^{K} L(G = k, \widehat{\mathrm{G}}(X)) Pr(G = k|X),$$

where $\widehat{\mathrm{G}}(x)$ is the classifier and the output variable takes K values between $1, ..., K$.

If we repeat our calculations as before and *use the zero-one loss*, we get the classifier:

$$\widehat{\mathrm{G}}(x) = \max_{k \in \{1,...,K\}} Pr(G = k|X = x).$$

Note 3. This is a very intuitive classifier and is known as the *Bayes classifier*. See Figure 1 for how it looks like.

Note 4. A nearest-neighbor classifier approximates this as well because it is essentially taking a 'majority vote'.

Note 5. As mentioned before, as a heuristic, one can convert this classification setting to a regression setting ($G \to Y$) and then do appropriate thresholding (say 0.5 for binary).

2 Curse of Dimensionality

Because it looks like no assumptions are needed for the nearest-neighbor methods, one should always try them first. Especially when there is large amount of data.

Not so correct intuition: With large data, one could approximate the conditional expectation very well because you can find a lot of neighbors for a point x.

The reason it breaks down is termed as the *curse of dimensionality*. How does it impact statistical modeling? Say you have three inputs and $N = 10000$, then the nearest-neighbor method may work well. But if you have 3000 inputs (because you collected more measurements), nearest-neighbor will most likely not work from a statistical modeling point of view!

[1]Perhaps, computational considerations may influence this choice.

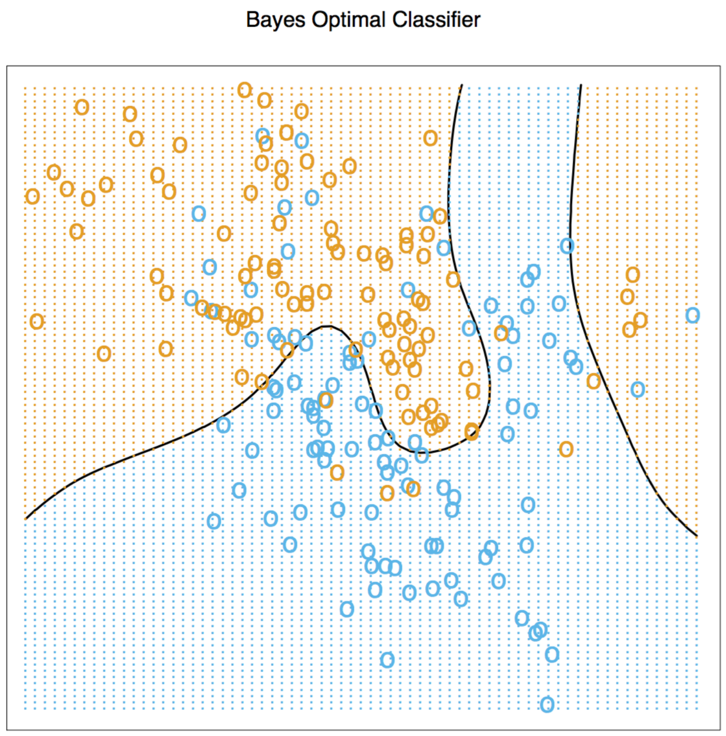


Figure 1: Bayes decision boundary.

Note 6. You may be worried about computing nearest neighbors in a decent amount of time as well. There are some solutions to the computational problem (such as approximately finding the nearest neighbors). Spotify has an open source implementation called Annoy that tries to address the computational issue (i.e., avoid linear time). In their own words: ... *"We use it at Spotify for music recommendations. After running matrix factorization algorithms, every user/item can be represented as a vector in f-dimensional space. This library helps us search for similar users/items. We have many millions of tracks in a high-dimensional space, so memory usage is a prime concern."*

Example 3. If you want to find a neighborhood-cube of side length l that covers r fraction (say 10%) of the volume of a unit hypercube in a p dimensional space, then the side length of such a cube is $(r)^{1/p}$ (say $0.1^{1/p}$). Figure 2 shows how long this is for even $p = 10$. Thus to capture a fixed percentage of data (if it is uniformly spread), you will need almost the full range of each variable coordinate! This is no longer local.

The consequence of all this is that doing statistical modeling for high dimensions is very different from low dimensions and we somehow need exponentially more observations in the training set.

Thus, nearest neighbor methods will fail because:

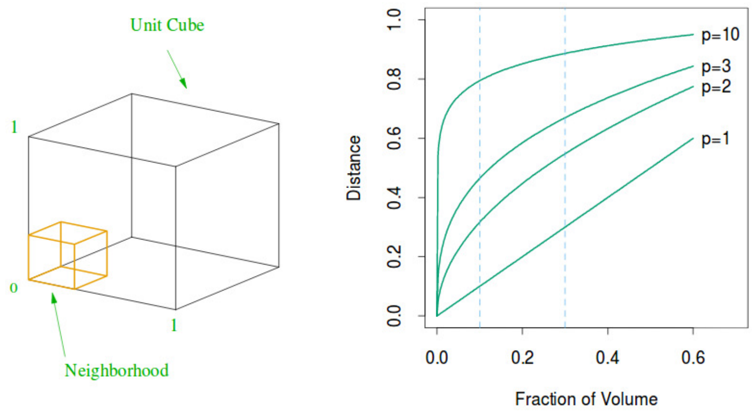


Figure 2: Large neighborhoods for the same fraction of volume.

- neighbors will not be close to the target point for which we are making a prediction, and
- if there is special structure in the data that you are aware of, significantly better prediction error can be obtained.

This is one of the key reasons why we want to develop different statistical modeling techniques!

Take for instance the linear model: if the strong assumption (conditional expectation is linear) holds, then the prediction error does not scale too badly compared to the nearest-neighbor methods. Figure 3 illustrate the relative errors between 1-nearest neighbor and the linear model for two specific datasets, showing that the latter is better. Note that such a trend can be easily reversed if we change the datasets.

3 Supervised Learning

3.1 Beyond Expected (squared) Prediction Error

We have only looked at k-nearest neighbor, linear models and squared loss function, But we can think of any function, parameterized by a parameter (say θ), and minimize the residual sum-of-squares (RSS, which is the empirical version of EPE):

$$RSS(\theta) = \sum_{i=1}^{N} (y_i - f_\theta(x_i))^2.$$

For $f_\theta(X) = X^T\theta$, we get a closed form expression. Otherwise, we may have to numerically optimize.

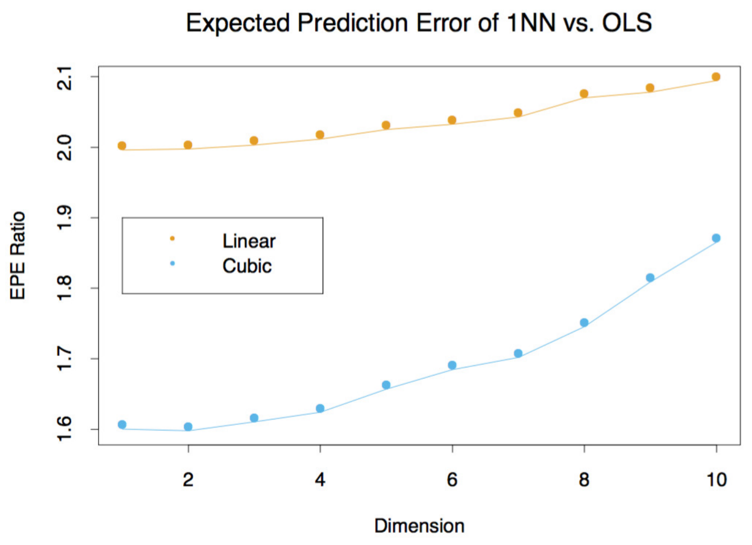


Figure 3: Linear model ($Y =$) doing better than 1-nearest-neighbor for two specific datasets (orange: $Y = X + \epsilon$, blue: $Y = \frac{1}{2}(X+1)^3$, X is 1-dimensional and $\epsilon \sim N(0,1)$). The y-axis is the EPE ratio of 1-neaerst-neighbor and linear modeling.

While least squares objective is great, there is a more general principle of estimation called *Maximum Likelihood Estimation* (**MLE**). Say random variable $Z \sim Pr_\theta(Z)$, where the density is indexed by parameter θ, then the log-probability of observing $z_1, ..., z_N$ is:

$$L(\theta) = \sum_{i=1}^{N} \log Pr_\theta(z_i).$$

The principle of MLE says that the most reasonable value for θ is that one, for which the probability of the observed sample is the largest.

Example 4. If $Y = f_\theta(X) + \epsilon$, $\epsilon \sim N(0, \sigma^2)$, then least squares is *equivalent* to MLE using $P(Y|X,\theta) = N(f_\theta(X), \sigma^2)$. The conditional log-likelihood of data is

$$L(\theta) \propto \sum_{i=1}^{N} (y_i - f_\theta(x_i))^2,$$

which is the same as $\text{RSS}(\theta)$.

Example 5. Consider qualitative output G. Let the model be $Pr(G = k|X = x) = p_{k,\theta}(x)$ for $1 \leq k \leq K$. Then the log-likelihood (also known as cross-entropy[2]) is $L(\theta) = \sum_{i=1}^{N} \log p_{g_i,\theta}(x_i)$.

[2]You may come across this term when you look at deep learning and neural network classifiers.

3.2 Different Model Families

For a fixed N, if we optimize $RSS(f)$ over all functions, we will have infinitely many solutions!

So we should restrict the search space. Such restrictions are based on additional knowledge and can be imposed using:

1. constraints on θ, or
2. implicitly though the learning method.

Example 6. For example, local linear fits in large neighborhoods is almost a globally linear model and is quite restrictive.

Note 7. Generally, any method that overcomes curse of dimensionality has an associated metric for defining the size of neighborhoods that are not small.

Lets list some methods here:

- Regularlized models: Here, the search space is controlled by defining Penalized RSS (PRSS): $PRSS(f, \lambda) = RSS(f) + \lambda J(f)$. These are also called penalty methods. And are equivalent to having a log-prior from a Bayesian point of view.
- Kernel methods: These explicitly specify the local neighborhood using a kernel function $K_\lambda(x_0, x)$.
  - $K_\lambda(x_0, x)$ assigns a weight to point x in a region around x_0.
  - Example: $K_\lambda(x_0, x) = \frac{1}{\lambda} \exp(-\frac{\|x-x_0\|_2^2}{2\lambda})$ assigns weights that die exponentially with squared Euclidean distance.
  - Given a $K_\lambda(x_0, x)$, an example model is:

    $$\widehat{f}(x_0) = \frac{1}{\sum_{i=1}^N K_\lambda(x_0, x_i)} \sum_{i=1}^N K_\lambda(x_0, x_i) y_i,$$

    which resembles a weighted average!
  - $K_k(x_0, x) = 1[\|x - x_0\|_2 \leq \|x_{(k)} - x_0\|_2]$ gives us the k-nearest-neighbor method. Here, $x_{(k)}$ is the k^{th} nearest point to x_0.
- Dictionary methods: These are linear in the coefficients, but arbitrary in the inputs. A function in this family would be $f_\theta(x) = \sum_{m=1}^M \theta_m h_m(x)$. For instance, $h_m(x) = \exp(-\frac{1}{2\lambda_m}\|x - \mu_m\|_2^2)$. The are called dictionary methods, because we can choose from a set of candidate functions to define each $h_m()$.

4 Linear Regression

Linear regression: just means that $E(Y|X)$ is linear in the inputs $X_1, ..., X_p$.

Note 8. Linear methods can be applied to transformed inputs, considerably expanding their scope. This generalization leads to what are called *basis-function* methods.

Model:

$$Y = \sum_{j=1}^{p} X_j \beta_j$$

Here, X_js can be: (a) quantitative, (b) functions of raw data (e.g., log(), $\surd$ etc.), (c) or even encode ***interactions between*** variables (e.g., $X_3 = X_1 \cdot X_2$).

Note 9. X_j can also be categorical. Say it has K levels, then we can create a 1-hot encoding to get a group of input variables X_k, $k = 1, ..., K$. Here $X_k = 1[X_j = k]$.

Irrespective of how X_js came about, the model is linear in the parameter β.

4.1 Least Squares

Recall that $RSS(\beta) = \sum_{i=1}^{N}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2$. The geometry of least squares is illustrated in Figure 4.

Lets derive the closed form expression for $\widehat{\beta}$ that we saw before.

$$RSS(\beta) = (\mathbf{y} - \mathbf{X}\,\beta)^T(\mathbf{y} - \mathbf{X}\,\beta)$$

Here, $\mathbf{X}$ is $N \times p$. This is a quadratic function of β. If we differentiate with respect to β, we get:

$$\frac{\partial RSS}{\partial \beta} = -2\,\mathbf{X}^T(\mathbf{y} - \mathbf{X}\,\beta),$$

and

$$\frac{\partial^2 RSS}{\partial \beta \partial \beta^T} = 2\,\mathbf{X}^T\,\mathbf{X}\,.$$

If $\mathbf{X}$ has full column rank, then $\mathbf{X}^T\,\mathbf{X}$ is positive definite. Setting the first derivative equal to zero, i.e.,

$$\mathbf{X}^T(\mathbf{y} - \mathbf{X}\,\beta) = 0, \text{we get}$$
$$\Rightarrow \widehat{\beta} = (\mathbf{X}^T\,\mathbf{X})^{-1}\,\mathbf{X}^T\,\mathbf{y}\,.$$

Note 10. Does it happen that the rank of $\mathbf{X} = [\mathbf{x}_1 \ldots \mathbf{x}_p]$ is not p? The answer is yes. For instance, if $\mathbf{x}_2 = 4\,\mathbf{x}_1$. Another example: when we transform categorical variable X_j into 1-hot encoding.

Note 11. When $\mathbf{X}$ is not full rank, then $\widehat{\beta}$ is not unique! One fix is to drop redundant columns.

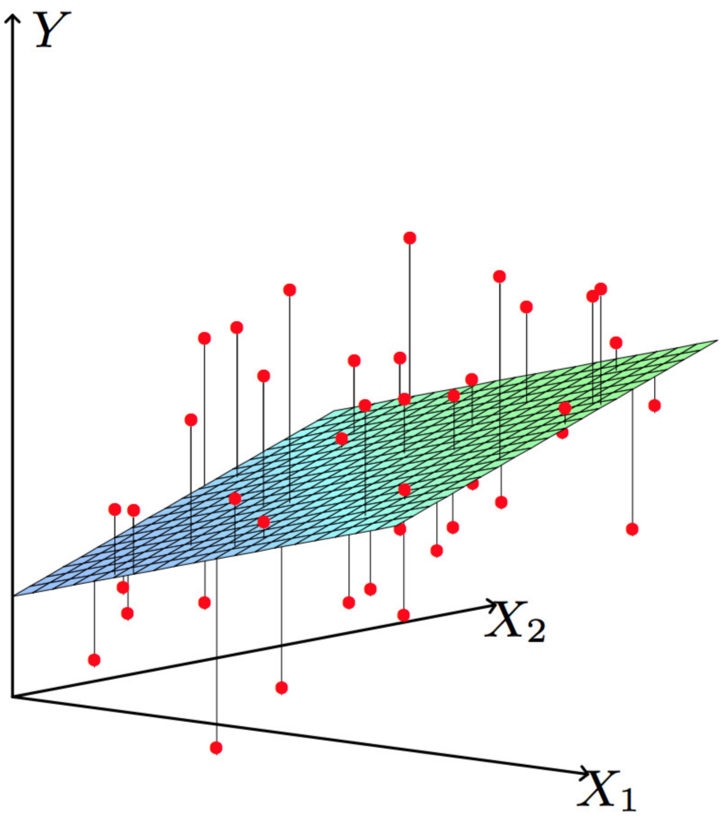


Figure 4: We seek a linear function of X that minimizes $RSS(\beta)$.

5 Summary

We learned the following things:

- Introduced probability while defining supervised learning problems.
- Curse of dimensionality motivated us to look for other statistical models (we will see them in future lectures!).
- Linear Regression: interaction terms, categorical variables.

In the next lecture, we will discuss subset selection and LASSO methods in regression. And move to the nuances of classification.

A Sample Exam Questions

- What is the regression function in the context of minimizing EPE?
- In what ways does k-nearest neighbor get affected by the curse of dimensionality?

- What is the equivalent maximum likelihood setup that justifies minimizing RSS?

Lecture 3

IDS575: Statistical Models and Methods
Theja Tulabandhula

Notes derived from the book titled "Elements of Statistical Learning [2nd edition] (Sections 3.2-3.4)

1 Bias Variance Tradeoff

Let us now understand the expressiveness or complexity of models. As seen in the previous section, such complexity is controlled by:

- regularizer coefficient
- kernel parameters, or
- number and type of basis functions.

We cannot use RSS(f), which is defined on training data, to determine these parameters. This is because we will always pick those that give the least residuals. Such models will fail spectacularly on test data.

Lets look at what the impact of the parameter is by first writing down the EPE at a test point x_0, and then specializing it for the k-nearest-neighbor method.

Let $Y = f(X) + \epsilon$ with $E[\epsilon] = 0$ and $Var(\epsilon) = \sigma^2$. Let x_i be non-random.

$$\begin{aligned} EPE(x_0) &= E_{\epsilon,\tau}[(Y - \widehat{f}(x_0))|X = x_0] \\ &= \sigma^2 + \text{Bias}^2(\widehat{f}(x_0)) + \text{Var}_\tau(\widehat{f}(x_0)) \end{aligned}$$

There are three key terms above:

- First term: Irreducible error σ^2. This is the variance of the new test output variable, and cannot be removed/reduced even if you know $f(x_0)$.
- Second term: Is called the bias term. it is the difference between the true mean $f(x_0)$ and the expected value of the estimate.

- Third term: is the variance of the estimate.

Example 1. The k-nearest-neighbor method: Bias is $f(x_0) - \frac{1}{k}\sum_{l=1}^{k} f(x_{(l)})$ and Variance is $\frac{\sigma^2}{k}$. Here $x_{(l)}$ is the l^{th} nearest neighbor. Bias may increase with increasing k because neighbors are further away. The variance is just the variance of the average, so it decreases as k increases. Thus, there is a bias-variance tradeoff with respect to k.

When model complexity increases, there is more variance and less bias. When model complexity decreases, there is more bias and less variance.

We want to choose model complexity to trade bias with variance so as to minimize test error (e.g, EPE). Training error (e.g., RSS) is not a good estimate of test error. Figure 1 shows an illustration of the behavior of test and training errors as model complexity of some model family is varied.

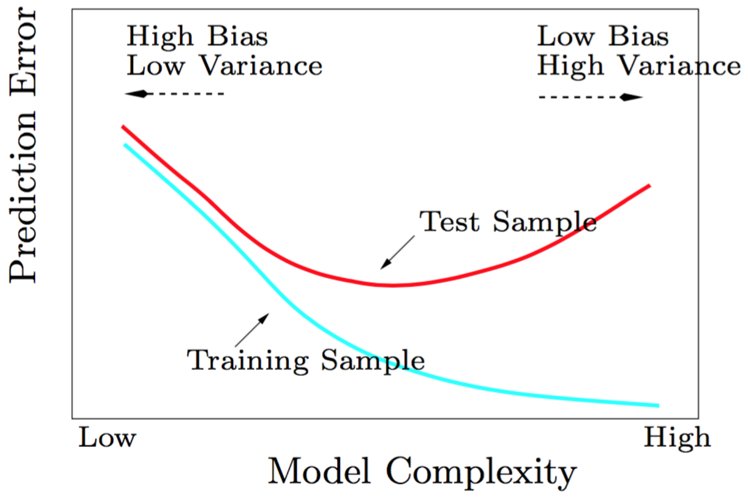


Figure 1: Bias Variance tradeoff for some model family. Example: model complexity for the k-nearest-neighbor family of methods is the parameter k.

With larger model complexity, the model adapts itself too much to the training data, leading to overfitting. On the other hand, if the model is not complex enough, it will lead to underfitting.

2 Linear Regression (Continued)

Recall that

$$\widehat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}\,\mathbf{y}$$

minimizes RSS(β).

2.1 Sampling Properties of $\widehat{\beta}$

Now we bring in the joint distribution and use it to describe properties of $\widehat{\beta}$. Lets assume:

- y_i are uncorrelated and have variance σ^2.
- x_i are non-random.

Under the above assumptions, $Var(\widehat{\beta}) = (\mathbf{X}^T\mathbf{X})^{-1}\sigma^2$

To make further inference, lets assume $E(Y|X) = X^T\beta$ and $Y = X^T\beta + \epsilon$, with $\epsilon \sim N(0, \sigma^2)$. Then,

$$\widehat{\beta} \sim N(\beta, (\mathbf{X}^T\mathbf{X})^{-1}\sigma^2)$$
$$\widehat{\sigma} \sim \frac{\sigma^2}{N-p}\chi^2_{N-p}$$

Note 1. In the ESLII book, the assume $\mathbf{X}$ is $p+1$ dimensional, so there is a slight change in the above formula.

This allows us to define some hypothesis tests and confidence intervals for β and β_j.

1. Let *Z-score* be $\frac{\widehat{\beta}}{\widehat{\sigma}\sqrt{v_j}}$, where v_j is the j^{th} diagonal entry of $(\mathbf{X}^T\mathbf{X})^{-1}$. Under the null that β_j is 0, z_j is distributed as t_{N-p-1} (t distribution). If we know σ, z_j will be a standard normal.

2. To test for groups of variables (e.g., if the original X_j is categorical, then several variables X_k will be related): We define the F statistic as:

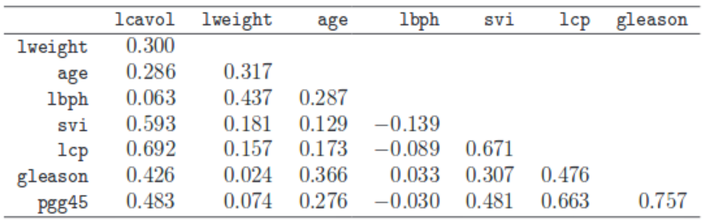
$$F = \frac{(RSS_0 - RSS_1)/(p_1 - p_0)}{RSS_1/(N - p_1)},$$

   where RSS_1 is for the bigger model. This statistic will be $F_{p_1-p_0, N-p_1}$ distributed under the null that the smaller model is correct.

3. Confidence intervals: Isolating β_j, we get a $1 - 2\alpha$ confidence interval as

$$\{\widehat{\beta} - z^{1-\alpha}\sqrt{v_j}\,\widehat{\sigma}, \widehat{\beta} + z^{1-\alpha}\sqrt{v_j}\,\widehat{\sigma}\}.$$

   For instance, $z^{1-0.05} = 1.645$ for standard normal.



	lcavol	lweight	age	lbph	svi	lcp	gleason
lweight	0.300						
age	0.286	0.317					
lbph	0.063	0.437	0.287				
svi	0.593	0.181	0.129	−0.139			
lcp	0.692	0.157	0.173	−0.089	0.671		
gleason	0.426	0.024	0.366	0.033	0.307	0.476	
pgg45	0.483	0.074	0.276	−0.030	0.481	0.663	0.757

Figure 2: Prostate cancer dataset: predictor correlations.

2.2 Prostate Cancer Example

Figure 2 and 3 shows how the input variables are correlated. These are: log cancer volume (lcavol), log prostate weight (lweight), age, log of benign prostate hyperplastia (lbph), seminal vesicle invasion (svi), log of capsular penetration (lcp), Gleason score (gleason) and percent of Gleason scores 4 or 5 (pgg45). The response variable is the level of prostate-specific antigen (lpsa).

We can fit a linear regression model to this data after standardizing the inputs. Training has 67 observations and test has 30. The output of regression is shown in Figure 4. With 9 parameters, the 0.025 quantile of t_{67-9} is ± 2. Notice that lcp and lcavol are strongly correlated but only the latter is significant.

The prediction error is 0.521. Predicting using the mean value of lpsa would have gotten 1.057 (the *base error rate*). The linear model reduces this error by about 50%.

Most models are distortion of truth. A good model is obtained when a suitable trade-off is made between bias and variance.

Note 2. Mean Squared Error (MSE) is related to Expected (Squared) Prediction Error (EPE) at a point x_0. For instance, for estimate $x_0^T\widehat{\beta}$, $E(Y_0 - x_0^T\widehat{\beta}) = \sigma^2 + E(x_0^T\widehat{\beta} - x_0^T\beta)^2$.

Example 2. If X is 1-dimensional and there is no intercept, then $\widehat{\beta} = \frac{\sum_{i=1}^N x_i y_i}{\sum_{i=1}^N x_i^2}$.(Check that this is true yourself!)

Example 3. Say the columns of input data matrix $\mathbf{X}$, $\mathbf{x}_j$ are orthogonal. Then, each coefficient $\widehat{\beta}_j = \frac{\mathbf{x}_j^T\mathbf{y}}{\mathbf{x}_j^T\mathbf{x}}$. You can get this by expanding out the closed form formula and using the fact that $\mathbf{x}_i^T\mathbf{x}_j = 0$ for $i \neq j$.

Note 3. What happens if Y is K-dimensional? Well, clearly β is not a vector anymore. It is a $p \times K$ dimensional matrix and the model is $Y = X\beta + E$ (E is the error matrix). It turns out that the least squares estimated matrix $\widehat{\beta}$ is still the same as before: i.e., $\widehat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$. What this means is that the coefficients for the k^{th} outcome/output variable are just the least squares estimates when the k^{th} coordinate is regressed on X. In other words, it is as if you are solving K different single-dimensional-output linear regression problems.

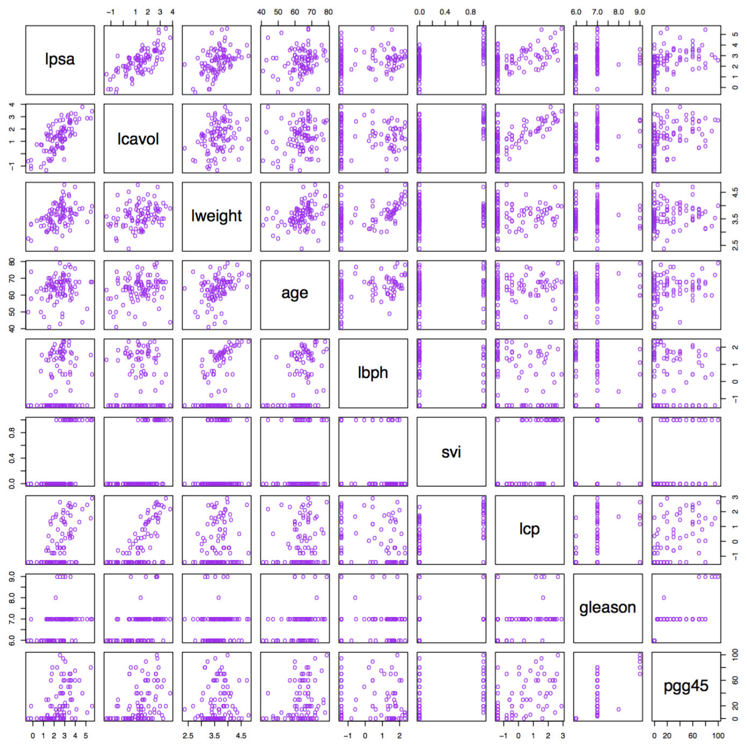


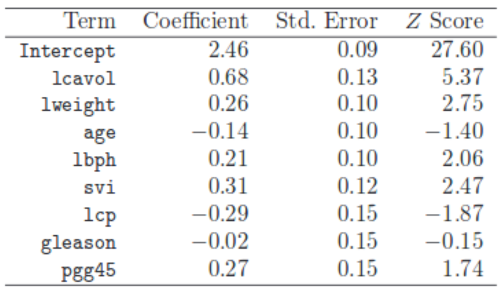
Figure 3: Scatterplot of predictors.

Least squares is great, especially due to its rich history and use in practice. But what happens when: (a) the prediction accuracy is not good enough? And (b) there are lots of feature coordinates? We will see that introducing more bias can sometimes improve prediction accuracy as well as tell us about feature importance. This is what we will do next!

Example 4. When there are many correlated input variables, it is empirically observed that linear regression coefficients become poorly determined and exhibit high variance. A large positive coefficient on one coordinate can be cancelled by a similarly large negative coefficient on another related coordinate.

3 Biasing Linear Regression via Subset Selection, Ridge and LASSO

We will introduce bias to linear regression via three methods: subset selection, ridge regression and LASSO. The bias will be such that some input variables will have higher coefficients, allowing us to infer that these are the important ones. This is related to *model selection*, which is a topic that will be discussed later.



Term	Coefficient	Std. Error	Z Score
`Intercept`	2.46	0.09	27.60
`lcavol`	0.68	0.13	5.37
`lweight`	0.26	0.10	2.75
`age`	−0.14	0.10	−1.40
`lbph`	0.21	0.10	2.06
`svi`	0.31	0.12	2.47
`lcp`	−0.29	0.15	−1.87
`gleason`	−0.02	0.15	−0.15
`pgg45`	0.27	0.15	1.74

Figure 4: Prostate cancer dataset: Linear regression output.

3.1 Subset Selection

What do we want to do here? We want to retain only a subset of input variables and discard the rest of the input variables.

How do we do this?

- Strategy 1: Say the number of features is p. Then for every $1 \le k \le p$, we find the RSS with linear regression with just k features. Note that for $\binom{p}{k}$ such subsets of input variables to consider. See Figure 5 for a plot of RSS for the prostate cancer example.

The best-subset curve (red line in Figure 5) is decreasing, so it cannot be used to pick k. The choice of k depends on bias and variance. One way to choose is: we pick k that minimizes an estimate of error. Such an estimate can be obtained by *cross validation* (see below).

- Strategy 2: Because considering all subsets is time consuming[1], one could start with the intercept and sequentially add a variable that reduces the RSS the most. This is called a *greedy* procedure, and has a couple of advantages: We only consider $O(p^2)$ subsets instead of $O(2^p)$, implying less computation. Further, lesser subsets to consider also implies more bias and less variance[2].

Example 5. Although we omit the details here, both strategy 1 and strategy 2 above give the same subset.

When we do subset selection and get the input variables that seem to be the best in predicting the dependent variable, we should avoid displaying the standard errors and z-scores for the corresponding model. Why is this the case? This is because data was not just used to get the model $\widehat{\beta}$, but also used to subset features. This makes the regression analysis we saw before, inapplicable!

[1]The number of all non-empty subsets of a set with p elements is $2^p - 1$.

[2]This is a hand-wavy remark for now, we haven't made this precise.

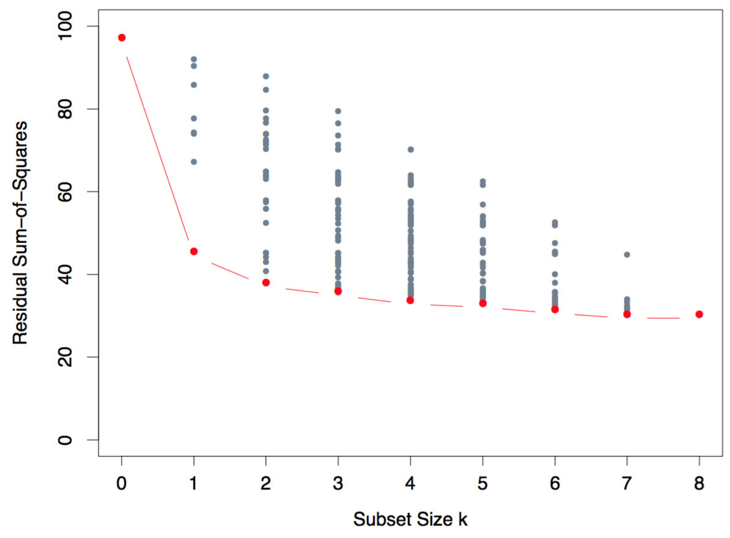


Figure 5: Prostate cancer dataset: Subset selection.

3.1.1 Cross Validation Primer

Term	LS	Best Subset	Ridge	Lasso	PCR	PLS
`Intercept`	2.465	2.477	2.452	2.468	2.497	2.452
`lcavol`	0.680	0.740	0.420	0.533	0.543	0.419
`lweight`	0.263	0.316	0.238	0.169	0.289	0.344
`age`	−0.141		−0.046		−0.152	−0.026
`lbph`	0.210		0.162	0.002	0.214	0.220
`svi`	0.305		0.227	0.094	0.315	0.243
`lcp`	−0.288		0.000		−0.051	0.079
`gleason`	−0.021		0.040		0.232	0.011
`pgg45`	0.267		0.133		−0.056	0.084
Test Error	0.521	0.492	0.492	0.479	0.449	0.528
Std Error	0.179	0.143	0.165	0.164	0.105	0.152

Figure 6: Prostate cancer dataset: Coefficients estimated using extensions of linear regression. The last two lines are post cross-validation and report numbers using the 30 observation test data.

Figure 6 shows the performance of subset selection compared to vanilla linear regression (there are a few other methods there, and we will discuss them soon).

Lets go into the details of cross-validation that we mentioned earlier. It is a useful tool to get an estimate of test error, which itself is useful to pick the right subset size.

Note 4. Why do we need an estimate of test error? We need it to find the right subset size k, for instance. And we should not use the original test data for this. Because that data is to score the final model. The choice of k should only be determined using the data you are working with, i.e., training data!

Cross validation happens as follows. You want to estimate test error for a k that you chose. You do the following:

- Break training data randomly into 10 buckets[3] (sets or *folds*).
- For a given k, do the following for each subset:
  - Compute the linear model using the subset of input features, using 9 of the folds. Evaluate the performance of the corresponding model on the 10th fold.
  - Repeat the above step in-turn for each of the 10 folds.
  - Average the errors. This is your estimate of the prediction error for this subset.
- Find the best subset of size k by doing the above for all subsets of the given size k, and picking the subset with the lowest estimated prediction error.
- Do the above for each k to get the corresponding error estimates, one per value of k. Now pick the best k (and the subset) using these error estimates.

Once the k and the subset is picked, you could use the full training data to build the final regression model. This can then be scored against the original test data (e.g., in the prostate cancer example, we had 67 observations in training and 30 observations as part of the test data).

4 Summary

We learned the following things:

- Bias variance trade-off.
- Linear Regression: sampling properties of $\widehat{\beta}$.
- Subset selection.

In the next lecture, we will discuss ridge regression, LASSO, and the nuances of classification through: (a) Linear Discriminant Analysis (LDA), and (b) logistic regression.

[3]The choice of 10 was arbitrary here.

A Sample Exam Questions

1. Is k-nearest neighbor the best method among all supervised learning methods? Explain why or why not.

2. How would you use a categorical feature (say, taking three values (cat,dog,fox)) in linear regression?

3. In which situations would $\mathbf{X}$ not be full rank? Does this pose any issue in linear regression modeling? If so, how would you mitigate it?

4. How would you perform a hypothesis test to check the null $\beta_j = 0$?

Lecture 4

IDS575: Statistical Models and Methods
Theja Tulabandhula

Notes derived from the book titled "Elements of Statistical Learning [2nd edition] (Sections 3.4 and 4.1-4.4)

We have briefly discussed bias-variance trade-off in Lecture 3. We will again revisit it in Lecture 5 (Model Assessment and Selection).

1 Applications

1.1 Prostate Cancer Prediction

Term	LS	Best Subset	Ridge	Lasso	PCR	PLS
Intercept	2.465	2.477	2.452	2.468	2.497	2.452
lcavol	0.680	0.740	0.420	0.533	0.543	0.419
lweight	0.263	0.316	0.238	0.169	0.289	0.344
age	−0.141		−0.046		−0.152	−0.026
lbph	0.210		0.162	0.002	0.214	0.220
svi	0.305		0.227	0.094	0.315	0.243
lcp	−0.288		0.000		−0.051	0.079
gleason	−0.021		0.040		0.232	0.011
pgg45	0.267		0.133		−0.056	0.084
Test Error	0.521	0.492	0.492	0.479	0.449	0.528
Std Error	0.179	0.143	0.165	0.164	0.105	0.152

Figure 1: Prostate cancer dataset: Coefficients estimated using extensions of linear regression. The last two lines are post cross-validation and report numbers using the 30 observation test data.

Figure 1 shows the performance of certain biased methods (we saw subset selection last time) compared to vanilla linear regression.

1.2 Salary Calculator by Stackoverflow

Here is an example of regression task used to create an information tool[1] (Announced on Sept 19th 2017).

The folks at Stackoverflow, an online question answering website that also provides a job board, released a salary calculator. Its purpose is to help job seekers as well as employers find typical salaries based on input variables such as experience level, location, technologies used and educational qualification.

So they collected data using a survey. And the survey seeks the information related to the input variables defined above. As an example of some preliminary analysis, a plot of salary versus years of experience is shown in Figure 2. The authors mention that developers in San Francisco and Seattle have different salaries compared to the rest of the US. In the survey, they did not ask for the city. To overcome this, they used the IP addresses to geolocate respondents.

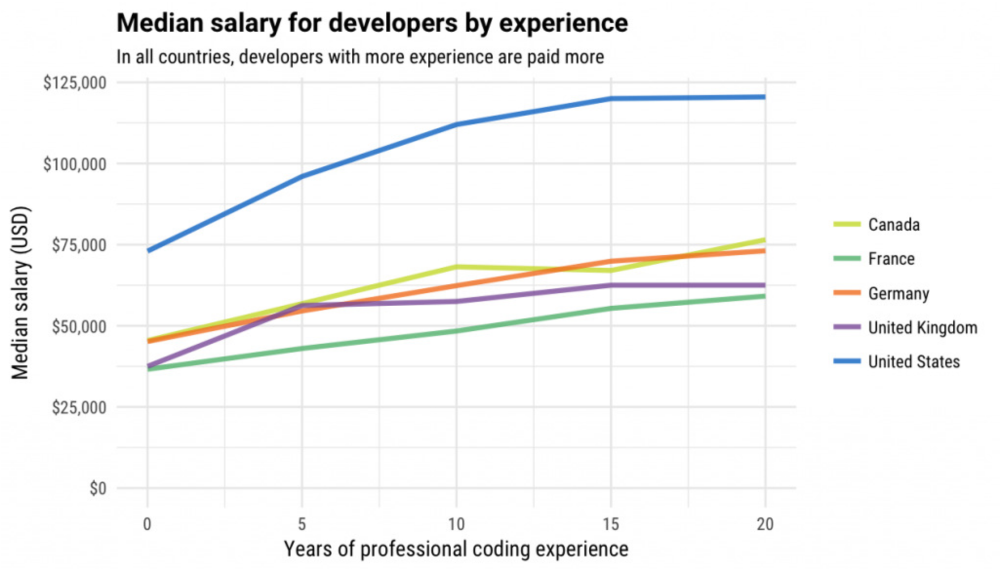


Figure 2: Exploratory plot from the salary survey (Image source).

Here is another exploratory plot (Figure 3) showing the impact of type of work (technologies used) on salaries.

The authors chose to create salary predictions only when there were enough observations for a given location (city or country). And the model they used is the familiar *linear regression*[2]. Specifically, they modeled the salaries in log scale, because they found the salary distribution was log-normal, 'with a long tail of very high salaries'.

Here is a plot (Figure 4) of the residuals, giving a hint to whether linear modeling is a good enough model choice here. The residuals seem mostly flat, except for the US where they are low around the ends. This is an indicator that a linear model is not sufficient here.

[1]See link for more information.

[2]They may have done an ad-hoc subset selection, which is not discussed.

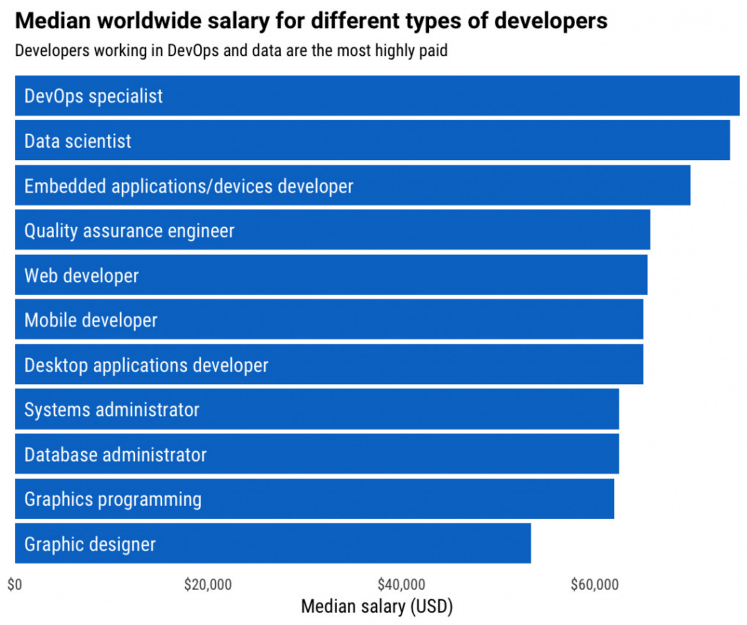


Figure 3: Another exploratory plot from the salary survey (Image source).

Here is a link to fun discussion page, by people who may use such a predictive tool!

2 Biasing Linear Regression via Ridge Regression and LASSO

Other than k-nearest neighbors, we have been focusing on linear methods for regression so far. Today, we will see linear methods for classification. This does not mean that linear methods are the only ones out there. One can use many other non-linear methods for both regression and classification. One of the reasons we have been focusing on linear methods is because they are intuitive and simpler to understand.

Subset selection is great, but is a discrete process: you either retain a coordinate of the input variable or discard it. Intuitively, variance can be further reduced if we don't completely discard these coordinates. One such attempt is via ridge regression and LASSO.

2.1 Ridge Regression

The idea of ridge regression is pretty straightforward. We want to shrink regression coefficients ($\widehat{\beta}_j$s) by imposing a penalty. What do we mean by this? Remember the RSS objective? We will just add another term to that:

$$RSS(\lambda, \beta) = (\mathbf{y} - \mathbf{X}\,\beta)^T(\mathbf{y} - \mathbf{X}\,\beta) + \lambda\beta^T\beta.$$

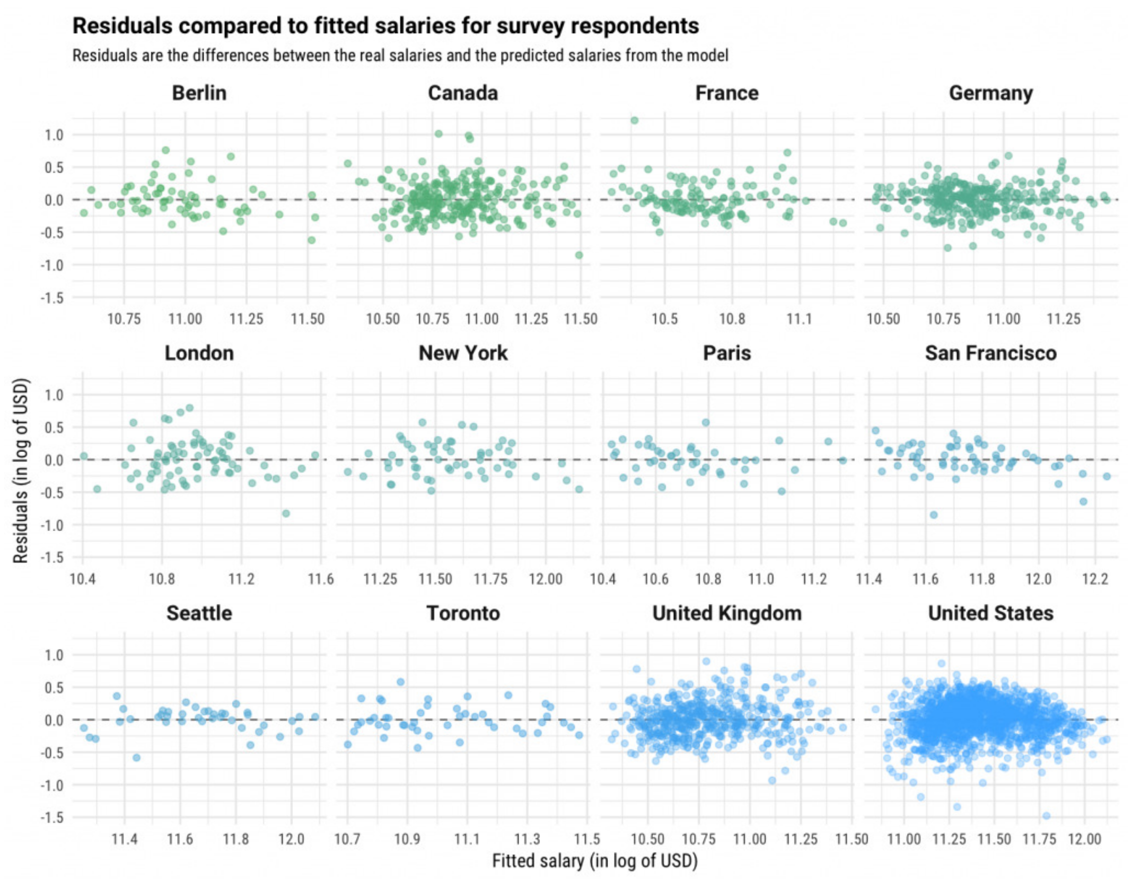


Figure 4: Plot of residuals with linear regression: salary dataset. (Image source).

What does λ do? It is a non-negative parameter. If it is large, it will make the coefficients shrink towards 0.

It can be shown that for a given λ, there exists a t such that the solution to the following problem is the same as that of $\text{RSS}(\lambda, \beta)$:

$$(\mathbf{y} - \mathbf{X}\,\beta)^T(\mathbf{y} - \mathbf{X}\,\beta)$$
$$\text{subject to } \beta^T\beta \leq t.$$

Why is this useful? It is useful because we can think of t as controlling the size of the coefficients directly. Recall that if we do this, then the chances of high positive coefficients on some variables and high negative coefficients on related variables is potentially mitigated.

Note 1. There is no point penalizing the intercept (corresponding to the coefficient of the last column of $\mathbf{X}$, which is all ones). If we *center* our data (i.e., subtract the feature means from training), we can compute the intercept separately. So, in our exposition here, lets assume data is centered and there is no intercept.

(Not so) surprisingly, the $\widehat{\beta}$ for ridge regression looks like the solution for linear regression:

$$\widehat{\beta} = (\mathbf{X}^T\mathbf{X} + \lambda I)^{-1}\mathbf{X}^T\mathbf{y}\,.$$

Figure 5 shows the ridge coefficients as a function of (another function [3] $df()$ of) λ. λ is like the k in subset selection, and will need to be determined using cross-validation.

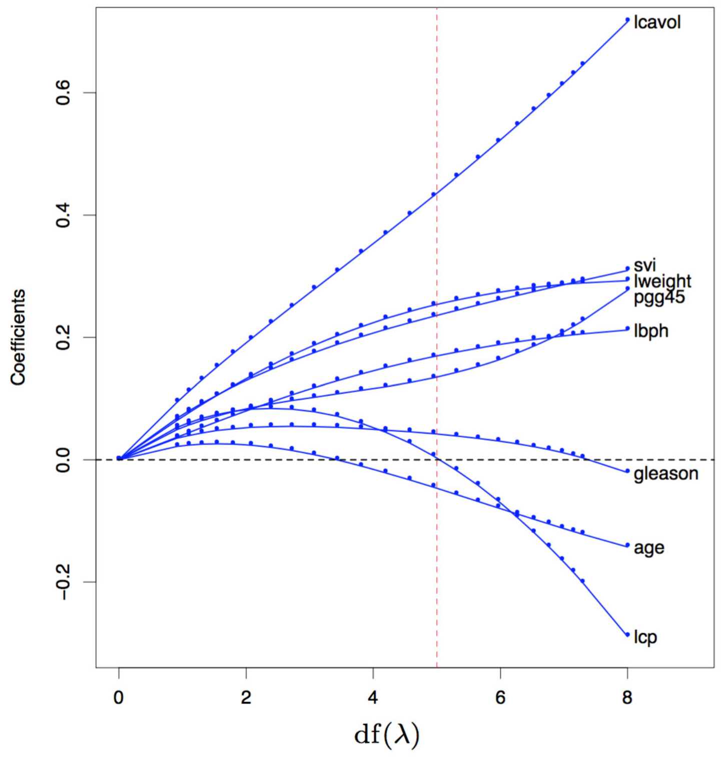


Figure 5: Prostate cancer dataset: Ridge regression coefficients as a function of (some function $df()$ of) λ.

Note 2. Just like linear regression is related to Maximum Likelihood Estimation, ridge regression is related to Maximum A posteriori Estimation (MAP). The latter estimation problem (MAP) is like MLE but with notions of priors and posteriors.

Note 3. Quick aside about the terms *prior* and *posterior*: If you remember Bayes rule, which is essentially about conditional distributions, it looks like this: $Pr(\theta|Z) = \frac{1}{Pr(Z)}Pr(Z|\theta)Pr(\theta)$. The second term in the numerator is called prior and the term on the left hand side is called

[3]This function is inversely related to λ.

posterior. This is because, we know that θ is distributed according to $Pr(\theta)$ before we observe Z (hence is called prior), and is distributed according to $Pr(\theta|Z)$ after observing Z (hence is called posterior). The term $Pr(Z|\theta)$ is called the likelihood!

Example 1. Say $Y \sim N(X^T\beta, \sigma^2)$ and $\beta \sim N(0, \tau^2 I)$. Then, assuming known τ^2, σ^2, the log posterior of β is proportional to the RSS(λ, β) above where $\lambda = \sigma^2/\tau^2$ (we will not show this here).

2.1.1 Interpretation using Singular Value Decomposition

Let the singular value decomposition (SVD) of matrix $\mathbf{X}$ be $\mathbf{X} = \mathbf{U}\,\mathbf{D}\,\mathbf{V}^T$, where $\mathbf{D}$ is a diagonal ($d_1 \geq d_2 \geq ...$), $\mathbf{U}$ and $\mathbf{V}$ are orthogonal. Let us not worry about how this is computed. The thing to notice here is that:

- The columns of $\mathbf{U}$ span the column space of $\mathbf{X}$.
- The columns of $\mathbf{V}$ span the row space of $\mathbf{X}$.

Okay, assuming the decomposition, the training predictions by the ridge solution looks like:

$$\begin{aligned}\mathbf{X}\,\widehat{\beta} &= \mathbf{U}\,\mathbf{D}(\mathbf{D}^2 + \lambda I)^{-1}\,\mathbf{D}\,\mathbf{U}^T\,\mathbf{y} \\ &= \sum_{j=1}^{p} \mathbf{u}_j\,\frac{d_j^2}{d_j^2 + \lambda}\,\mathbf{u}_j^T\,\mathbf{y},\end{aligned}$$

where $\mathbf{u}_j$ is the j^{th} column of $\mathbf{U}$.

Note 4. The term $\frac{d_j^2}{d_j^2+\lambda} \leq 1$.

Like linear regression, ridge regression computes the coordinates of $\mathbf{y}$ with respect to the orthonormal basis $\mathbf{U}$. It then shrinks these coordinates by factors $\frac{d_j^2}{d_j^2+\lambda}$. There is a greater amount of shrinkage to coordinates of basis vectors with smaller d_j^2.

What does a smaller d_j^2 mean? Well, for that, lets understand the (unnormalized) sample covariance matrix $\mathbf{X}^T\,\mathbf{X} = \mathbf{V}\,\mathbf{D}^2\,\mathbf{V}^T$. The vectors $\mathbf{v}_j$s are the *principal component directions* of $\mathbf{X}$.

The first principal component direction $\mathbf{v}_1$ has the property that $\mathbf{z}_1 = \mathbf{X}\,\mathbf{v}_1$ has the largest sample variance among all linear combinations of columns of $\mathbf{X}$. This variance is equal to $\frac{d_1^2}{N}$.

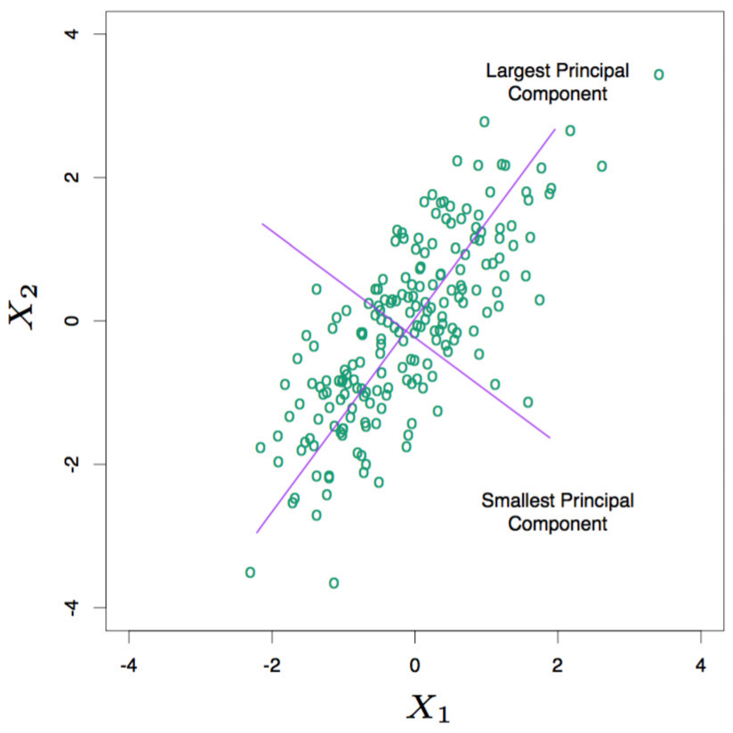


Figure 6: Principal components: illustration.

Note 5. $\mathbf{z}_1 = \mathbf{X}\,\mathbf{v}_1 = d_1\,\mathbf{u}_1$ is called the first principal component of $\mathbf{X}$. Subsequent principal components $\mathbf{z}_j$ have variance $\frac{d_j^2}{N}$, and are orthogonal to others.

Example 2. Figure 6 shows the principal components of a simulated 2-dimensional data.

Small singular values d_j correspond to directions in the column space of $\mathbf{X}$ that have small variance, and ridge regression shrinks these directions the most.

Note 6. When there are large number of input variables, another way to approach the dimensionality issue is to produce a small number of linear combinations $Z_m, m = 1, ..., M$ of the original inputs X_j and use these for regression. When Z_m are the principal components, the approach is called *principal component regression*. While ridge regression shrinks the coefficients of the principal components, principal component regression discards the $p - M$ smallest eigenvalue components.

2.2 LASSO

LASSO is short for *Least Absolute Shrinkage and Selection Operator*. Thats a mouthful!

It is very slightly different from ridge regression:

$$RSS(\lambda, \beta) = (\mathbf{y} - \mathbf{X}\,\beta)^T(\mathbf{y} - \mathbf{X}\,\beta) + \lambda \sum_{j=1}^{p} |\beta_j|.$$

The second term is called an ℓ_1-*penalty*[4].

Note 7. There is no closed form expression for the LASSO estimate $\widehat{\beta}$ in general.

Example 3. Figure 7 shows how $\widehat{\beta}_j$s vary for LASSO as (a function of) λ is varied.

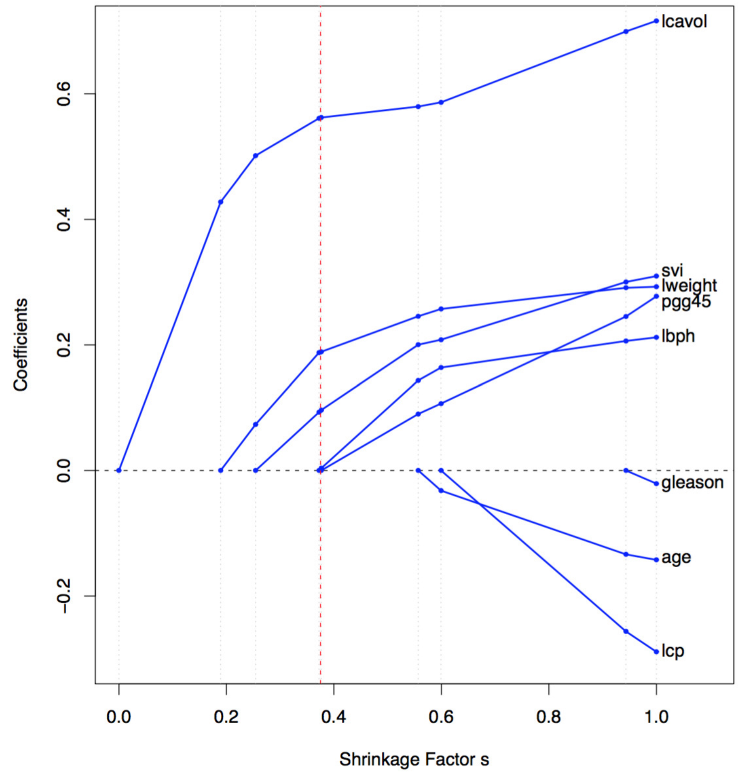


Figure 7: LASSO Coefficients as λ is varied (shrinkage factor above is inversely related to λ).

Lets contrast LASSO with ridge regression. See Figure 8 for an example 2-dimensional input setting. The residual sum of squares (without the penalty term) is depicted as elliptical contours (these are the level sets, where points on the curve lead to the same value). The LASSO level set looks like a diamond. Both methods find the first point where the elliptical contours hit the penalty regions. Since the diamond has corners, if the ellipse hits the penalty region at the corner, some $\widehat{\beta}_j$s will be zero!

Note 8. We have ignored how to compute LASSO model. The search for the best model is not difficult because the problem is *convex*. We may revisit this aspect later.

[4]Sometimes also denoted as L_1-penalty.

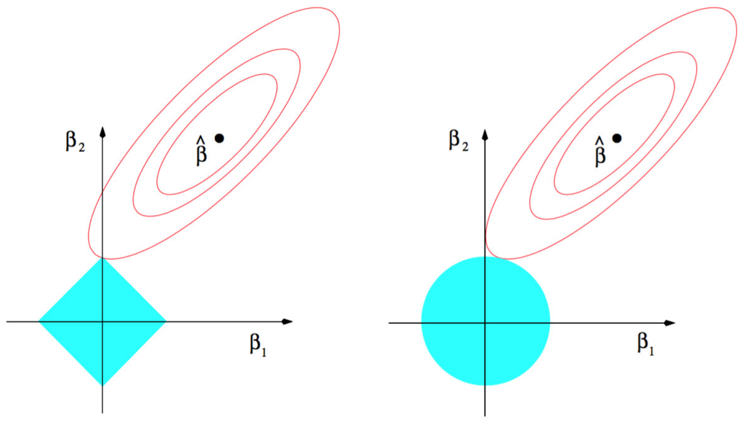


Figure 8: LASSO versus ridge regression: contour plots of the residual sum of squares term and the penalty terms.

3 Summary

We learned the following things:

- Ridge regression and reasoned what the penalty does from the SVD point of view.
- LASSO.

In the next lecture, we will discuss the nuances of classification. If time permits, we will also look at model assessment and selection: (a) the bias-variance decomposition, (b) Bayesian Information Criterion (BIC), (c) cross-validation, and (d) bootstrap methods.

A Sample Exam Questions

1. How does cross-validation improve over just breaking training data further into a validation set and using the remaining for actual training?

2. Why is validation necessary? Why can't we just use test set?

3. What is the motivation between ridge regression and LASSO? How do they relate to subset selection?

Lecture 5

IDS575: Statistical Models and Methods
Theja Tulabandhula

Notes derived from the book titled "Elements of Statistical Learning [2nd edition] (Sections 4.1-4.4, 7)

1 Classification using Linear Regression

Lets start thinking of classification now. Here the output variable G takes discrete values.

Say, hypothetically, you divide the input region into a collection of regions and label them according to the true classification. These decision boundaries can be jagged or nice (e.g., smooth/linear). There are methods that can output classifiers that only lead to linear decision boundaries. These are the ones we will start looking at.

As we saw before, we could use linear regression for classification. To do so, from G, make a K dimensional output variable Y (this is also called *one-hot encoding*). And fit a linear model. More precisely, predict the k^{th} indicator output variable using

$$\widehat{\mathrm{f}}_k(X) = \widehat{\beta}_k^T X.$$

The decision boundary between any two classes, say class k and l, would be given by $\widehat{\mathrm{f}}_k(X) = \widehat{\mathrm{f}}_l(X)$. This is a hyperplane[1]. Hence the input space is divided into regions with linear decision boundaries.

How does one classify using $\widehat{\mathrm{f}}_k, k = 1, ..., K$? Well, we can simply take the class corresponding to the largest one. These are generally called *discriminant functions.* We will see other discriminant functions later (LDA and logistic regression) soon.

Example 1. Linearity decision boundaries may seem more restrictive than necessary. But it turns out that certain nonlinear boundaries, such as quadratic, can be thought of as linear boundaries in a higher dimension. This is illustrated in Figure 1, where the original data is 2-dimensional.

Here are the steps for using linear regression for classification:

[1]Given constants a (p-dimensional) and c (1-dimensional), a hyperplane is a set of points z that obey $a^T z = c$.

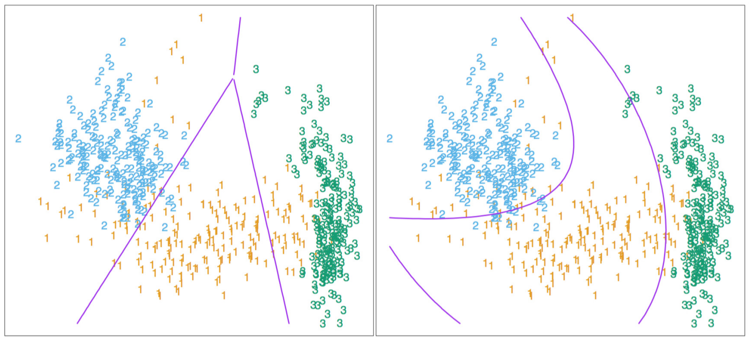


Figure 1: Linear decision boundaries in 2 and 5 dimensions (in the latter case, visualized in the original 2-dimensional space).

1. Each response category ($k = 1, ..., K$) is coded as an *indicator variable* (this is a K-dimensional vector with all zeros except the location/coordinate corresponding to category/level k.)
2. Collect them together to get the indicator response matrix $\mathbf{Y}$, which is $N \times K$-dimensional. Each row has a single 1 in it.
3. Fit a linear regression model to each of the columns of $\mathbf{Y}$ simultaneously, which is $\widehat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$. Here $\widehat{\beta}$ is $p \times K$-dimensional.
4. A new observation x_0 is classified by:
   - Compute $x_0^T\widehat{\beta}$, which is a K-dimensional row vector.
   - Identify the largest component, and report its index as the category/level.

Regression estimates conditional expectation. For Y_k, $E[Y_k|X = x] = Pr(G = k|X = x)$, hence this is reasonable, assuming $Pr(G = k|X = x)$ are linear functions of x.

Note 1. Some coordinates of $x_0^T\widehat{\beta}$ can be negative or greater than one, which is a bad approximation to $Pr(G = k|X = x)$.

Note 2. There is another issue with using regression, especially when $K \geq 3$: classes can get *masked* by other classes.

Example 2. Figure 2 represents an example of class masking. Here the three classes can be separated easily by linear boundaries, but linear regression is unable to do so! To understand this, look at Figure 3. The data has been projected onto the line joining the three centroids. The three regression lines are also plotted, and we can see that the line corresponding to the middle class is horizontal/never-dominant. Although using quadratic features help in this example, there is no easy solution in general.

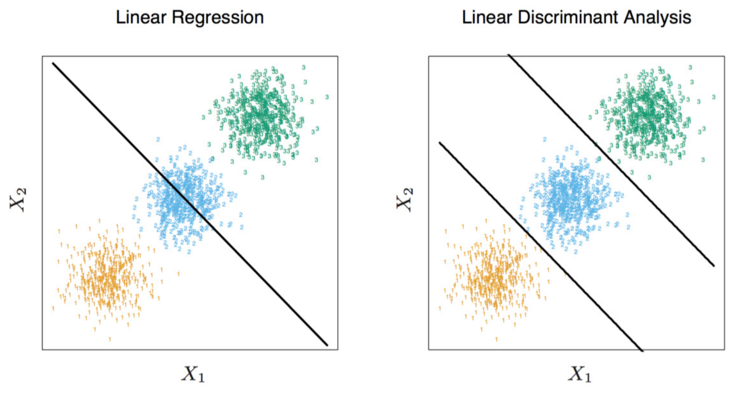


Figure 2: Class masking when linear regression is used for classification (left). Another method LDA (right) is able to circumvent this issue.

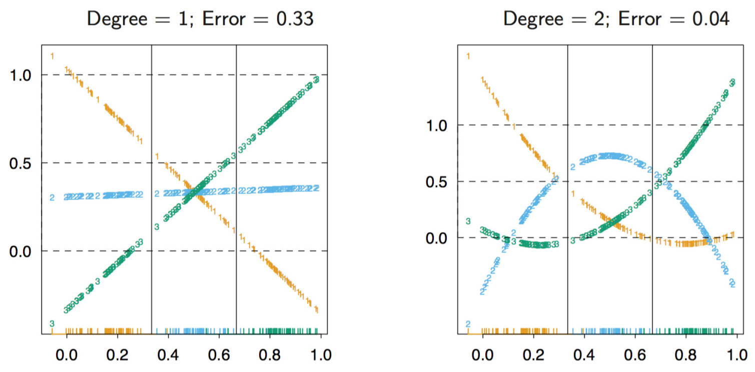


Figure 3: Class masking is minitgated using quadratic derived features (right) in this example. Without them, the $\widehat{\mathrm{f}}$ for one of the class never dominates (left).

2 Classification using Linear Discriminant Analysis (LDA)

If we recall statistical decision theory that we discussed before, then we know that $P(G = k|X = x)$ gives us the best classification (for the zero-one loss). Lets try to get to that quantity by using Bayes rule. Here's how:

1. Let the class conditional density of X be $f_k(X), k = 1, ..., K$
2. Let the class prior be $\pi_k, k = 1, ..., K$
3. Then the class posterior density is

$$P(G = k|X = x) = \frac{f_k(x)\pi_k}{\sum_{l=1}^{K} f_l(x)\pi_l}.$$

The Bayes rule above tells us that choosing a model for $f_k(X)$ can give us the posterior distribution. There are many methods that model these densities, including LDA and *Naive Bayes* and Quadratic Discriminant Analysis (QDA). Lets focus on LDA for now.

LDA assumes that $f_k(X) \sim N(\mu_k, \Sigma_k)$ and $\Sigma_k = \Sigma$ for all $k = 1, .., K$.

Thus, when we compare two classes via the log-ratio of their posteriors we get:

$$\begin{aligned}\log \frac{Pr(G=k|X=x)}{Pr(G=l|X=x)} &= \log \frac{f_k(X)}{f_l(X)} + \log \frac{\pi_k}{\pi_l} \\ &= \log \frac{\pi_k}{\pi_l} - \frac{1}{2}(\mu_k + \mu_l)^T \Sigma^{-1}(\mu_k - \mu_l) + x^T \Sigma^{-1}(\mu_k - \mu_l).\end{aligned}$$

This is linear in x!

The linear log-odds ratio implies that the decision boundary for any pair of *classes* is linear (the points x at which $Pr(G=k|X=x) = Pr(G=l|X=x)$).

Example 3. Figure 4 illustrates these boundaries for a simulated 2-dimensional data involving three classes.

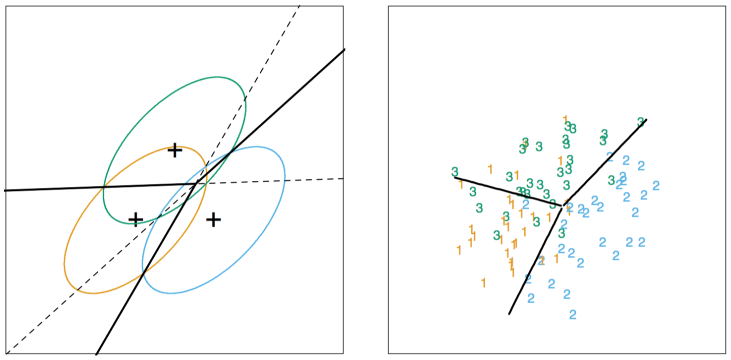


Figure 4: LDA boundaries for a simulated data with the same covariance for the three classes. Left plot shows the lines correspond to the Bayes decision boundaries. The right plot shows the LDA decision boundaries using training data.

The LDA decision rule is to pick the class with the maximum among the following *discriminant functions*: $\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2}\mu_k^T \Sigma^{-1} \mu_k + \log \pi_k$.

The parameters of the discriminant functions are estimated from training data as:

- $\widehat{\pi}_k = \frac{N_k}{N}$, where N_k is the number of class-k observations
- $\widehat{\mu}_k = \sum_{g_i=k} x_i / N_k$
- $\widehat{\Sigma} = \frac{1}{N-K} \sum_k \sum_{g_i=k} (x_i - \widehat{\mu}_k)(x_i - \widehat{\mu}_k)^T$.

Note 3. Gaussian assumption is critical here. Would it hold if one of the features was categorical?

Note 4. Quadratic Discriminant Analysis(QDA): If you relax the assumption that all class densities do not have the same Σ, we end up with discriminant functions that are quadratic. This also leads to quadratic decision boundaries.

Example 4. Figure 5 shows how QDA varies from LDA for a 2-dimensional dataset. For LDA, interactions terms were accounted for to fit a model in 5 dimensions (see Figure 1). As you can see, there is not much difference between them for this dataset. Since QDA needs to estimate different covariance matrices, the number of parameters is much larger.

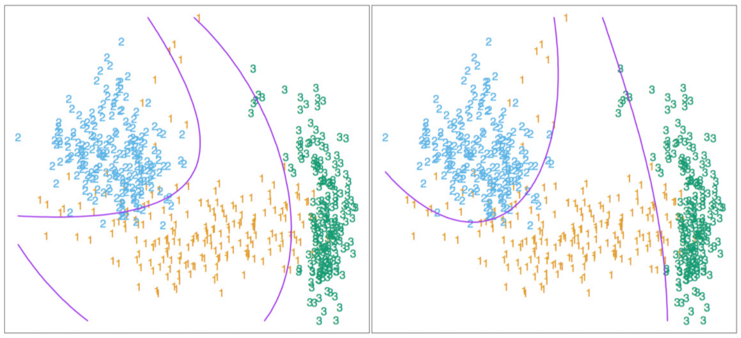


Figure 5: QDA (right) versus LDA (left).

3 Classification using Logistic Regression

Logistic regression model is defined using log-ratios of posterior probabilities of classes given feature vector. That is,

$$\log \frac{Pr(G = l|X = x)}{Pr(G = K|X = x)} = \beta_l^T x,$$

for $l = 1, ..., K - 1$. The log ratio is also called a logit transformation, hence the name.

From these $K - 1$ ratios, we can deduce that for each k

$$Pr(G = k|X = x) = \frac{\exp(\beta_k^T x)}{1 + \sum_{l=1}^{K-1} \exp(\beta_l^T x)}.$$

We will use maximum likelihood method to fit the model parameters. The log-likelihood given a training set of size N ($\{x_i, g_i\}_{i=1}^N$) is

$$l(\{\beta_1, ..., \beta_{K-1}\}) = \sum_{i=1}^{N} \log Pr(G = g_i|X = x_i)$$

Example 5. When $K = 2$, we can use variables y_i instead of g_i to write down the log-likelihood in a nicer form.

- Let $y_i = 1$ when $g_i = 1$ and let it be 0 when g_i is 2.
- Let $Pr(G = 1|X = x) = p$ (just a shorthand notation)

Then,

$$\begin{aligned} l(\beta_1) &= \sum_{i=1}^{N} \Big\{ y_i \log p + (1 - y_i) \log(1 - p) \Big\} \\ &= \sum_{i=1}^{N} \Big\{ y_i \beta^T x_i - \log(1 + \exp(\beta^T x)) \Big\}. \end{aligned}$$

Maximizing the log-likelihood turns out to be a convex[2] problem and can be solved using several off-the-shelf solvers.

Note 5. There is also another way to solve for β_1 that involves repeatedly solving linear regression problems. The method is called IRLS (Iteratively Reweighted Least Squares) that solves the following in each iteration:

$$\beta^{new} \leftarrow \arg\min_{\beta} (\mathbf{z} - \mathbf{X}\,\beta)^T (\mathbf{z} - \mathbf{X}\,\beta),$$

where $\mathbf{z} = \mathbf{X}\,\beta^{old} + \mathbf{W}^{-1}(\mathbf{y} - \mathbf{p})$, $\mathbf{W}$ is a $N \times N$ diagonal matrix with entries $Pr(G = 1|X = x_i)(1 - Pr(G = 1|X = x_i))$ and $\mathbf{p}$ is a N-dimensional vector with entries $Pr(G = 1|X = x_i)$, all evaluated at β^{old}. Finally, note that this is not super interesting unless one cares about optimization and is working in a large scale setting.

3.1 Logistic Regression versus LDA

Since the logistic model and the LDA have very similar forms for the log-ratios of probabilities, they may seem similar. They do in fact have the same linear form. But there are key differences in how the coefficients are estimated.

- Logistic regression makes lesser assumptions. Specifically, it does not impose that the marginal density of X is a mixture of Gaussian distributions.
- LDA on the other hand makes Gaussian assumptions. It is also estimated easily because of this.
- If indeed data was such that $P(X)$ was a mixture of Gaussians, then LDA would need lesser data to reach the same level of estimation accuracy.

So if you are unsure about which to use, side with logistic regression.

4 Model Selection and Assessment

Now that we have seen several methods: (a) linear regression with subset selection, ridge and LASSO penalties, (b) k-nearest neighbor methods, (c) linear discriminant analysis, and (d) logistic regression, lets discuss how to pick the most promising model.

We want to know the generalization performance: the prediction capability on independent test data.

This is the problem of *model selection and assessment.* We have already seen cross-validation as a way to assess any choice that we make while doing model search. There are other methods, some simpler than other that help us assess performance. Before doing that, lets understand *model complexity.*

4.1 Model Complexity, Model Bias and Model Variance

As before, let $\tau = \{(x_1, y_1), ..., (x_i, y_i), ..., (x_N, y_N)\}$ represent the training set, and $L(Y, \widehat{\mathrm{f}}_\alpha(X))$ denote the loss function for measuring errors (α denotes a tunable parameter or choice that you make). We have already seen the expected prediction error:

$$Err = E_{\tau,Pr(X,Y)}[L(Y, \widehat{\mathrm{f}}_\alpha(X))] = E_\tau[Err_\tau],$$

where $Err_\tau = E_{Pr(X,Y)}[L(Y, \widehat{\mathrm{f}}_\alpha)|\tau]$. This latter quantity is also what we want to know, but is not easy to estimate, so we focus on Err instead. Figure 6 depicts both of these for 100 simulated trainings sets with 50 observations each, and using LASSO.

Training error is the average loss over the training sample (RSS is a special case), and we will denote it as $\overline{err} = \frac{1}{N}\sum_{i=1}^N L(y_i, \widehat{\mathrm{f}}_\alpha(x_i))$. In general, negative log-likelihoods can be used to define the loss function as well.

The intuition for model complexity is this: as the model becomes more complex (more parameters), it will be able to adapt to more underlying structure in the data. The model bias will decrease, but the model variance will increase.

Note 6. Although we discuss quantitative outputs here, most of our discussion can also encompass the qualitative output setting easily.

Note 7. We want to set the tuning parameter α such that we get the minimum Err.

Note 8. There are two related but distinct goals: (a) model selection that involves estimating different models (corresponding to different αs) to pick the best one; and (b) model assessment that involves estimating the Err for a given model.

Example 6. Say we have lots of data. Then we can do the following:

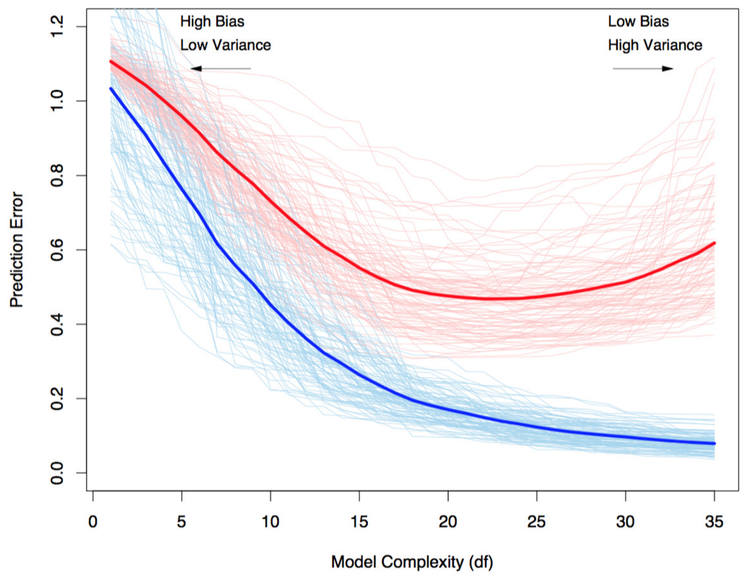


Figure 6: Illustration of expected prediction error Err (solid red) vs $Err_{\mathcal{T}}$ (light red) for LASSO. Training error $\overline{err}$ (blue) decreases as model complexity increases.

- Randomly divide dataset into three parts as shown in Figure 7.
- Fit models using train.
- Use validation to estimate Err for model selection.
- Finally, use test set to estimate Err of the one chosen model (model assessment). Do not use test set multiple times!

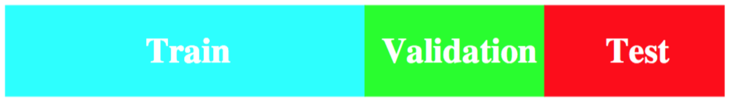


Figure 7: Train, validation and test sets.

You have to make a choice about what proportion of your data will be validation and test.

If we do not have lots of data, then we can attempt to use other methods discussed here, viz., AIC, BIC and cross-validation.

4.2 Bias Variance Decomposition

Consider the data distribution $Pr(X,Y)$ such that $Y = f(X) + \epsilon, E[\epsilon] = 0, Var(\epsilon) = \sigma^2$, and X fixed. In this case, recall that the EPE (which is the same as Err for the squared loss) can be written at a point x_0 as:

$$Err(x_0) = \sigma^2 + (E_\tau \widehat{\mathrm{f}}(x_0) - f(x_0))^2 + E_\tau(\widehat{\mathrm{f}}(x_0) - E_\tau \widehat{\mathrm{f}}(x_0))^2.$$

The first term is the variance of the output variable around its true mean $f(x_0)$. The second term is the squared model bias and the third term is model variance.

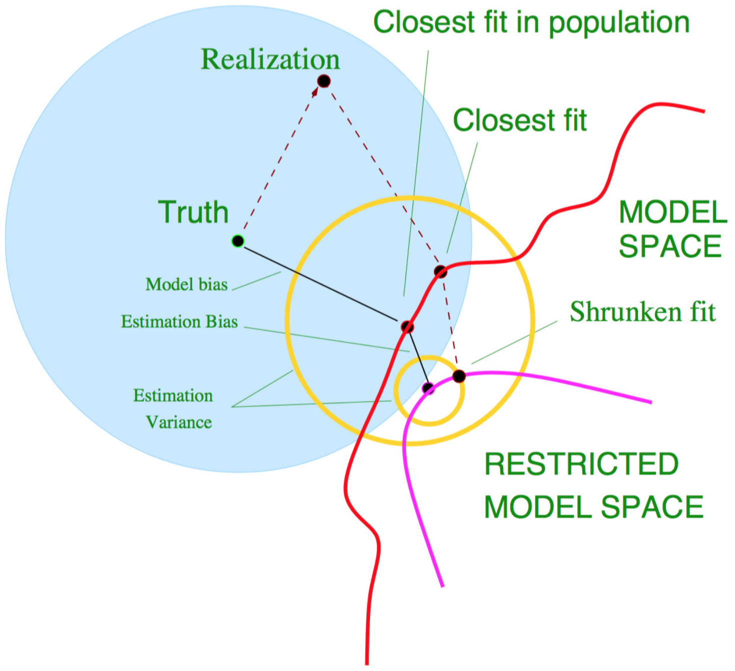


Figure 8: An example illustration of bias-variance decompostion. Blue region indicates σ^2. Red boundary represents linear models. Purple boundary represents ridge models. Yellow circles represent model variance (larger one for linear regression and smaller one for ridge).

The rule of thumb is that if a model is complex, then its bias will be low, but variance will be high.

Example 7. For k-nearest neighbor:

- Bias: $(f(x_0) - \frac{1}{k}\sum_{l=1}^{k} f(x_{(l)}))^2$.
- Variance: σ^2/k.

This means that if k is chosen to be high, the bias is high and the variance is low. When k is small, $\widehat{\mathrm{f}}(x)$ can adapt itself better to the underlying $f(x)$.

Example 8. For a p-dimensional linear model, the variance term for $Err(x_0)$ is equal to $\|\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}x_0\|_2\sigma^2$. There is a slightly intuitive version for the in-sample model variance term for $\frac{1}{N}\sum_i Err(x_i)$, which is $p\sigma^2/N$. Figure 8 shows an illustration of bias-variance decomposition for this class of models.

Note 9. Bias variance decomposition behaves differently for different loss functions. This implies different tuning parameters for different settings.

4.3 Model Selection Basics

Note 10. Intuitively, training error $\overline{err}$ will be less than the expected error Err because the same data is used to fit the model as well as assess its error.

Methods such as AIC and BIC try to estimate the optimism in training error $\overline{err}$ and add a corrective term to get an estimate of Err. Thus, they use training data only, and mostly make sense[3] for linear models. On the other hand, methods such as cross-validation directly estimate Err, but use 'validation' data.

4.4 AIC: Akaike Information Criterion

The AIC tells us to pick the model with the smallest AIC over the set of models considered.

AIC for regression models is defined as follows. Let $\overline{err}_\alpha$ and $d(\alpha)$ be the training error and number of parameters of model $\widehat{\mathrm{f}}_\alpha(x)$. Then, AIC is:

$$AIC = \overline{err}_\alpha + 2\frac{d(\alpha)}{N}\widehat{\sigma}^2,$$

where $\widehat{\sigma}^2$ is the estimated noise variance.

AIC for classification models such as logistic regression is defined as:

$$AIC = -\frac{2}{N}\sum_{i=1}^{N}\log Pr_{\widehat{\theta}}(y_i) + 2\frac{d}{N}.$$

Although it is a useful way to perform model selection, it may be inapplicable for certain choices of loss/likelihood functions. Further it is sensitive to $d(\alpha)$.

4.5 BIC: Bayesian Information Criterion

BIC is an alternative to AIC and is applicable when the loss function is defined in terms of a likelihood function (so we are maximum likelihoods here). The general formula is:

$$BIC = -2\sum_{i=1}^{N}\log Pr_{\widehat{\theta}}(y_i) + d\log N.$$

Under simple Gaussian assumptions, it simplifies to:

$$BIC = \frac{N}{\sigma^2}\overline{err} + d \log N.$$

Because of the $\log N$ term, it penalizes more complex models more heavily than AIC.

Although the formula looks very similar, BIC is motivated from a Bayesian perspective. Say we have multiple models. Say we have the same prior for each model. Then the posterior given data can be written as the likelihood times the prior. It turns out that the (negative of) log of this posterior can be approximated using the equations defined above.

Note 11. There is an additional benefit of BIC over AIC because of the Bayesian perspective: we can compare two models easily because we have the posterior probability of each model.

4.6 Nuances of Cross-Validation

Cross-validation (CV) estimates Err directly. It uses part of training data to fit models, and a different part to estimate Err, and repeats this by exchanging the parts. For example, $K = 5$ parts look as shown in Figure 9, where we have chosen the third part to estimate Err in this turn.

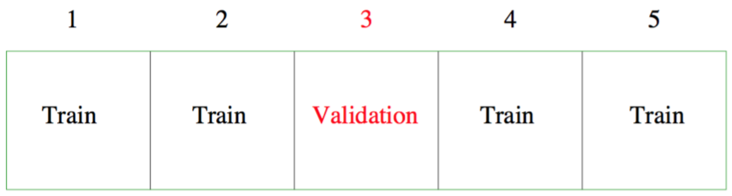


Figure 9: An example CV data split.

When $K = N$, we call this version of CV *leave-one-out* cross-validation.

How to pick K, the cross-validation parameter? Again it is a choice. Lets see how it impacts estimation of Err.

Example 9. One of the factors that influences how K-CV performs is the performance of the learning methods as training data increases. Figure 10 shows an illustration of the *learning curve* of a classifier. The classifier performance increases as training set size N increases up to about 100 observations, beyond which the performance improvement is small.

If our training data had 200 observations to work with. Then choosing $K = 5$ would give us 160 observations to fit a model, whose performance would be similar to the model fit using the full 200 observations. On the other hand, if our training data had 50 observations, then $K = 5$ would give us only 40 observations to fit a model, which would give us an incorrect (lower) estimate of $1 - Err$.

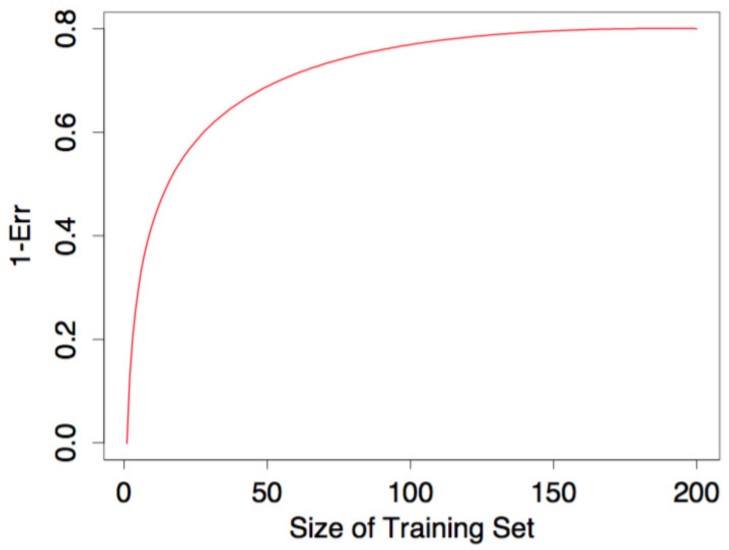


Figure 10: Learning curve as a function of training set.

4.7 Incorrect Use of CV

Cross validation should always be applied before the entire sequence (if any) of choices are made. For example, folds must be created before any filtering/data-processing steps are applied. And these filtering/data-processing steps must be applied in turn within each run of CV.

Note 12. If you ignore the realizations of the output variable, then you can process the remaining data (predictors) without involving cross-validation here.

Example 10. Say you plot scatterplots of predictors with the output and pick a subset of predictors (say 100). Using these, you build your classifier. And use cross-validation to estimate the error of the final model. *This is an incorrect application of cross-validation!*

Consider setting where $N = 50, p = 5000$ and all the input variables are standard Gaussians that are independent of class labels. Then the true $Err = 50\%$. If we use the above process for 1-nearest neighbor, we get a CV error rate as 3% (50 simulations, see Figure 11). The reason for this discrepancy is that the chosen input variables have an unfair advantage because they were chosen based on all data. Leaving data out after that does not let us estimate its Err correctly because these variables have "seen the left out data" as well. In Figure 11, we chose a random set of 10 observations (because we are using 5-fold CV) and computed the correlation between the pre-selected 100 input variable coordinates and the class labels of just these 10 samples. These correlations are non-zero!

So what is the correct way? First divide data into K folds. For each fold, find input coordinates that correlated with labels using all data except the k^{th} fold. Build a model using all data except the k^{th} fold. Get its Err estimate. Repeat this over all folds in turn. Average the estimates. The bottom panel in Figure 11 shows the correlation of class labels with chosen predictors in a typical fold, which is zero on average.

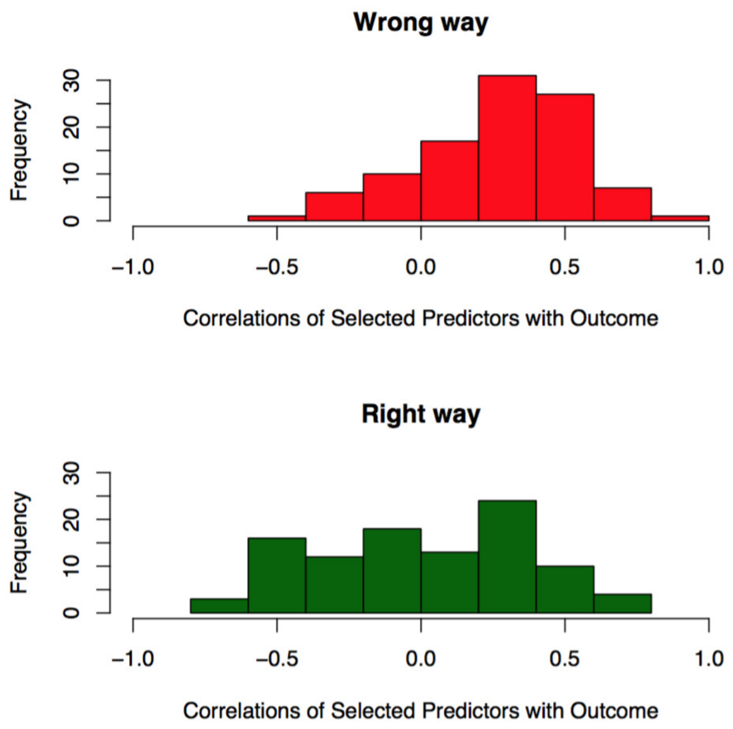


Figure 11: Incorrect vs correct use of cross-validation.

Note 13. Models must always be retrained from scratch for each turn of the K-CV.

Note 14. It is also useful to report the standard error of the CV estimate of Err to get an idea about the variance of the estimator.

5 Summary

We learned the following things:

- Classification via regression, linear discriminant analysis and logistic regression.
- Model selection and assessment using: (a) the bias-variance decomposition, (b) Akaike and Bayesian Information Criterion (AIC, BIC), and (c) cross-validation.

A Sample Exam Questions

1. What is class masking?
2. What are the similarities and differences between LDA and logistic regression?
3. What is the difference between Err and Err_τ?
4. How do you find the final model using cross-validation? What data will be used to find it?

Lecture 6

IDS575: Statistical Models and Methods
Theja Tulabandhula

Notes derived from the book titled "Elements of Statistical Learning [2nd edition] (Sections 8.2)

1 Beyond Supervised Learning

There are very interesting and realistic ways in which statistical models and methods influence several fields. We have seen such examples in previous lectures. Here is one more:

Example 1. A very visual example of the potential of using statistical models is illustrated in a webapp, made by a few companies (Stoj, UseAllFive, Google).

The setting is supervised learning. It happens that they use a certain family of models called neural networks (instead of the familiar LDA and logistic regression) to perform a classification task. The number of classes is 3. You need to generate visual input variable realizations (images!) using the browser/webcam. Once trained on examples from each class, the classifier can classify any new visual input to one of the three classes. Each prediction triggers a gif to be played, giving a cool overall effect.

We will now look at the general landscape of statistical modeling and a couple of key methods to model and *infer*.

Previously:

- We have been studying the supervised learning task. This involved minimizing the squared loss or the *cross-entropy* loss (this is defined for a classification model as $\sum_{i=1}^{N} \log Pr_\theta(G = g_i|X = x_i)$, where θ is the parameter of the model. Classification is made at a new point x_0 as $\arg\max_{k=1,..,K} Pr_\theta(G = k|X = x_0)$).
- We connected these loss functions the method of maximizing likelihood estimation (MLE).
- The MLE method (and others) are applicable to statistical problems beyond supervised learning.

So, we will look at a couple of these general statistical problems and methods that solve them.

In general, in a statistical modeling task, the inference problem is to estimate either the unknown parameters of the model or a distribution over them. This is called *inference*.

Before discussing inference, lets look at a tool called the *bootstrap*, a straightforward process to estimate prediction uncertainty.

2 The Bootstrap for Capturing Prediction Uncertainty

The bootstrap is a general process for capturing prediction uncertainty, used throughout statistics. Consider data $\mathbf{Z} = (z_1, ..., z_N)$. For instance, this can be training data ($z_i = (x_i, y_i)$). The idea is to sample new datasets with replacement from Z, where each new dataset is of size N (there are variations where this is not necessary). Let the number of new datasets is B. This is illustrated in Figure 1.

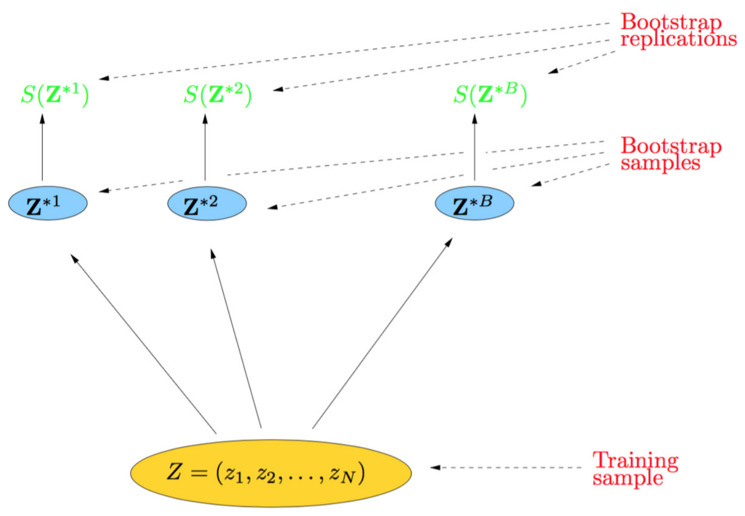


Figure 1: The bootstrap process. The new datasets are denoted $\mathbf{Z}^{*b}, b = 1, ..., B$.

We fit a model to each of the dataset and look at the performance of the models across the B replications. Let $S(\mathbf{Z})$ be a function of data (for example, it can be the linear regression model making a prediction at a fixed point x_0).

Using bootstrap, we can estimate any aspect of the distribution of $S(\mathbf{Z})$. It is like a *Monte-Carlo estimation*[1] of any function of $S(\mathbf{Z})$ under sampling from the empirical distribution function for data $\mathbf{Z} = (z_1, ..., z_N)$.

Below, we list two example ways to quantify prediction uncertainty.

Example 2. How do we use the bootstrap for say *estimating Err*? We can do the following:

- For each observation, we record the predictions from bootstrap samples that do not contain this observation.
- Then we average over the performance of these predictions as: $\frac{1}{N}\sum_{i=1}^{N}\frac{1}{C^{-i}}\sum_{b\in C^{-i}} L(y_i, \widehat{\mathrm{f}}^{*b}(x_i))$.

Here, C^{-i} is the set of bootstrap samples b that do not contain observation i, and $|C^{-i}|$ is their number. $\widehat{\mathrm{f}}^{*b}(x)$ is the model fit using the b^{th} bootstrap sample.

Just like cross-validation, training size may impact how well the bootstrap estimates Err.

Example 3. How do we use the bootstrap to get *confidence bands* around predictions? We can do the following:

- For each bootstrap sample, fit the prediction function.
- Get the 95% (in general $1-\alpha\%$) point-wise confidence band from the percentiles at each x. Say, for a 95% with $B = 200$, the $2.5\% \times 200$ percentile on each side would correspond to the fifth largest and smallest value at each x.

See Figures 2 and 3 for an illustration. Here, $N = 50$ and 7 pre-determined basis functions of the original 1-dimensional input variable were used. The model is nothing but a vector of coefficients $\widehat{\beta}$ that linearly combines the basis functions.

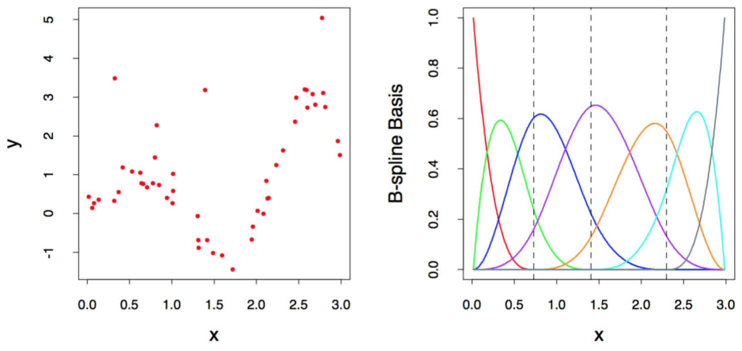


Figure 2: Dataset and the basis functions.

Thus, the bootstrap gives a direct computational way to assess prediction uncertainty.

[1]Monte-Carlo estimate corresponds to taking an average of a bunch of points. We will discuss this later.

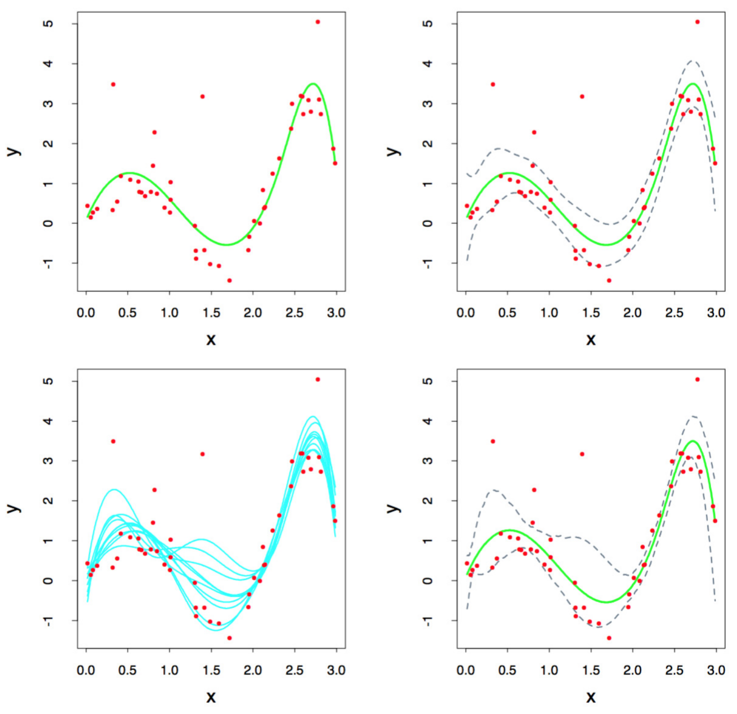


Figure 3: Confidence intervals ($\pm 1.96\times$ standard error) using linear regression (top right) and confidence bands using the bootstrap (bottom right).

3 Inference using MLE

Now, lets focus on a general statistical task: say $Z \sim g_\theta$. That is, there is a *parameter* vector θ that specifies a distribution function (more precisely, the density function) for the random variable Z.

Example 4. If $Z \sim N(\mu, \sigma^2)$, then $\theta = [\mu, \sigma^2]^T$.

The likelihood function is $L(\theta; \mathbf{Z}) = \prod_{i=1}^{N} g_\theta(z_i)$, which captures the probability of observed data under model g_θ. We maximize the likelihood, keeping $\mathbf{Z}$ fixed and varying θ. While maximizing a function, taking its log does not change the maximum point, so we maximize $l(\theta; \mathbf{Z}) = \sum_{i=1}^{N} \log g_\theta(z_i)$.

Say the maximum is attained at $\widehat{\theta}$. We can create a confidence bands for this estimate as well. How? With our familiar tool: the bootstrap!

Note 1. Just like in cross validation, choices should be made for each bootstrap sample separately, to get the right confidence bands.

4 Summary

We learned the following things:

- Learning problems that are broader in scope than supervised learning.
- Learning/inference in these setting can be achieved using maximum likelihood.

A Sample Exam Questions

1. What are the different ways to use the bootstrap in a statistical modeling task?

B List of Topics

1. Linear models
2. k-nearest neighbors
3. Comparison between nearest neighbor and linear methods
4. Statistical decision theory
5. Curse of dimensionality
6. Supervised learning
7. Linear regression
8. Bias-variance trade-off
9. Subset selection
10. Cross validation
11. Ridge regression
12. LASSO
13. Classification using linear regression
14. Linear discriminant analysis
15. Logistic regression
16. Model selection
17. Maximum likelihood method
18. Bootstrap

Lecture 7

IDS575: Statistical Models and Methods
Theja Tulabandhula

Notes derived from the book titled "Elements of Statistical Learning [2nd edition] (Sections 8.5-8.6, 9.1-9.2)

1 Inference using Expectation Maximization

The Expectation Maximization (EM) algorithm is a very useful tool to simplify MLE problems that are difficult to solve.

Lets understand it for a specific problem first: the problem of density estimation.

Density estimation is the problem of estimating the distribution function (density function) using a dataset.

Consider the 1-dimensional data ($N = 20$) shown in Figure 1 and enumerated in Figure 2.

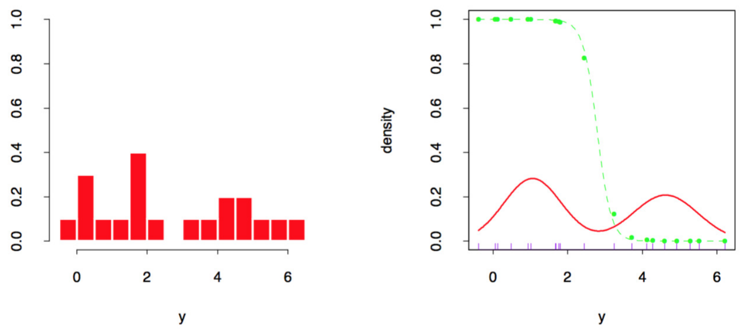


Figure 1: Dataset of 20 1-dimensional points, drawn as a histogram on the left. MLE fit on the right (solid red, more explanation in text).

Eyeballing the data, it looks like there is *bi-modality*, so we hypothesize that the data came from a *mixture* of two Gaussian distributions in the following way:

1. $Y_1 \sim N(\mu_1, \sigma_1^2)$

-0.39	0.12	0.94	1.67	1.76	2.44	3.72	4.28	4.92	5.53
0.06	0.48	1.01	1.68	1.80	3.25	4.12	4.60	5.28	6.22

Figure 2: Data used for the histogram in Figure 1.

2. $Y_2 \sim N(\mu_1, \sigma_1^2)$
3. $\Delta \sim$ Bernoulli(π)
4. $Y = (1-\Delta)Y_1 + \Delta Y_2$.

The above is called a *generative* description. We flip a coin with bias π and then depending on the outcome, make Y equal to Y_1 or Y_2. This is a very popular model in statistics and machine learning.

The density of Y is $g_Y = (1-\pi)N(Y;\mu_1,\sigma_1^2) + \pi N(Y;\mu_2,\sigma_2^2)$ and the parameter vector is $\theta = [\pi, \mu_1, \sigma_1^2, \mu_2, \sigma_2^2]^T$.

The log likelihood is

$$l(\theta; \mathbf{Y}) = \sum_{i=1}^{N} \log[(1-\pi)N(y_i;\mu_1,\sigma_1^2) + \pi N(y_i;\mu_2,\sigma_2^2)].$$

This optimization problem is difficult because there is an additive term inside the log. It turns out that there are multiple *local maxima* for this function[1].

Lets introduce the *unobserved/latent* variables δ_i (realizations of $\Delta \sim$ Bernoulli(π) taking values 0 and 1 resp.). Then the new log-likelihood would be

$$l_0(\theta; \mathbf{Y}, \mathbf{\Delta}) = \sum_{i=1}^{N} \left\{(1-\delta_i)\left(\log N(y_i;\mu_1,\sigma_1^2) + \log(1-\pi)\right) + \delta_i\left(\log N(y_i;\mu_2,\sigma_2^2) + \log\pi\right)\right\}.$$

Maximizing the above likelihood is easy because the terms involving one parameter can be decoupled with terms involving other parameters.

Since δ_i are not observed, we could substitute their conditional expected values, which are defined as:

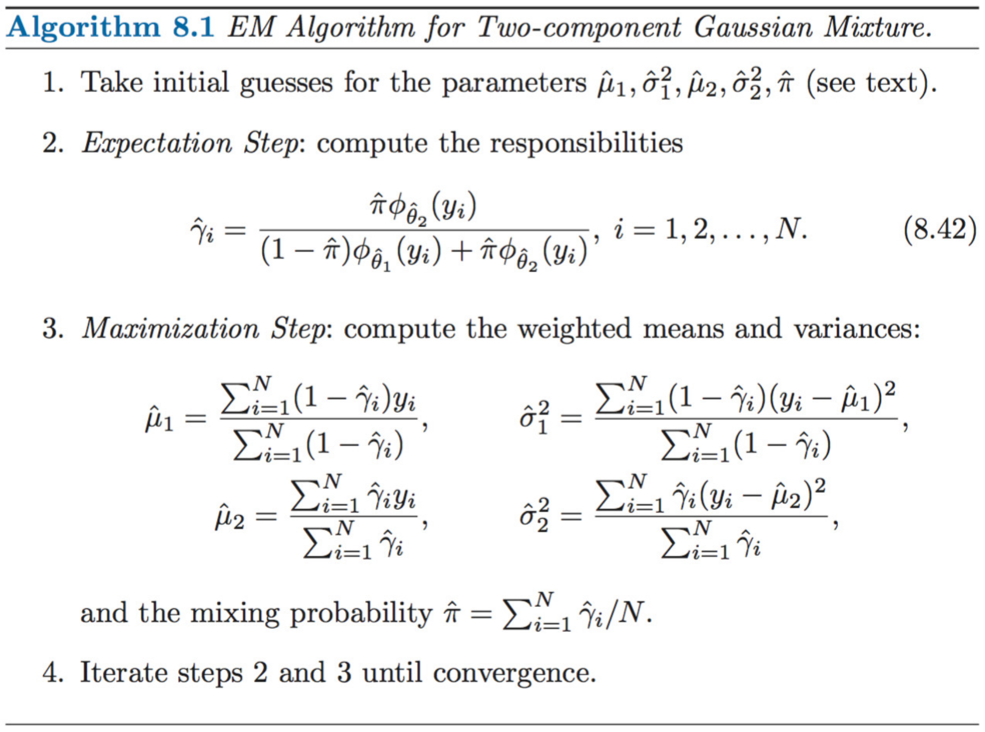
$$\gamma_i = E[\Delta_i|\theta, \mathbf{Y}],$$

giving us the idea for an iterative algorithm shown in Figure 3.

In each iteration, there are two steps:

- *The expectation step* (E): We find γ_i, a soft assignment of each y_i to one of the two Gaussians.

[1]A point θ is a local maximum if it maximizes the value of the given function in its neighborhood. It is a global maximum if it maximizes the function across all possible points.



Algorithm 8.1 *EM Algorithm for Two-component Gaussian Mixture.*

1. Take initial guesses for the parameters $\hat{\mu}_1, \hat{\sigma}_1^2, \hat{\mu}_2, \hat{\sigma}_2^2, \hat{\pi}$ (see text).
2. *Expectation Step*: compute the responsibilities

$$\hat{\gamma}_i = \frac{\hat{\pi}\phi_{\hat{\theta}_2}(y_i)}{(1-\hat{\pi})\phi_{\hat{\theta}_1}(y_i) + \hat{\pi}\phi_{\hat{\theta}_2}(y_i)}, \; i = 1, 2, \ldots, N. \qquad (8.42)$$

3. *Maximization Step*: compute the weighted means and variances:

$$\hat{\mu}_1 = \frac{\sum_{i=1}^N (1-\hat{\gamma}_i) y_i}{\sum_{i=1}^N (1-\hat{\gamma}_i)}, \qquad \hat{\sigma}_1^2 = \frac{\sum_{i=1}^N (1-\hat{\gamma}_i)(y_i - \hat{\mu}_1)^2}{\sum_{i=1}^N (1-\hat{\gamma}_i)},$$

$$\hat{\mu}_2 = \frac{\sum_{i=1}^N \hat{\gamma}_i y_i}{\sum_{i=1}^N \hat{\gamma}_i}, \qquad \hat{\sigma}_2^2 = \frac{\sum_{i=1}^N \hat{\gamma}_i (y_i - \hat{\mu}_2)^2}{\sum_{i=1}^N \hat{\gamma}_i},$$

and the mixing probability $\hat{\pi} = \sum_{i=1}^N \hat{\gamma}_i / N$.

4. Iterate steps 2 and 3 until convergence.

Figure 3: The Expectation-Maximization algorithm for the 2-component mixture model.

- *The maximization step* (M): Substituting these γ_i in place of δ_i and maximize $l_0(\theta; \mathbf{Y}, \mathbf{\Delta})$.

Example 1. For the 1-dimensional data above, starting with several initial values for θ, the EM algorithm was run. Figure 4 shows one of these runs. Figure 5 shows how the estimate $\widehat{\pi}$ changes over iterations. Finally, the right panel of Figure 1 shows the fitted Gaussians.

1.1 The General Trick

The idea behind EM is to make the likelihood maximization problem easier by enlarging the data with latent data. This latent data can be any data that was missing, corrupted, or just unobserved. The general algorithm is shown in Figure 6. Let the observed data be $\mathbf{Z}$ and the latent data be $\mathbf{Z}^m$ and let $\mathbf{T} = (\mathbf{Z}, \mathbf{Z}^m)$. Let the original and new log-likelihoods be $l(\theta; \mathbf{Z})$ and $l_0(\theta; \mathbf{T})$. Then, in the j^{th} iteration, we do:

- The E step: compute $Q(\theta', \widehat{\theta}(j))$ by computing the conditional distribution of the latent variables given the observed and the current parameter $\widehat{\theta}(j)$ and substituting these in the expression for $l_0(\theta'; \mathbf{T})$ (call it $Q(\theta', \widehat{\theta}(j))$).

- The M step: Maximize the expression $Q(\theta', \widehat{\theta}(j))$ over θ' to get $\widehat{\theta}(j+1)$. And repeat.

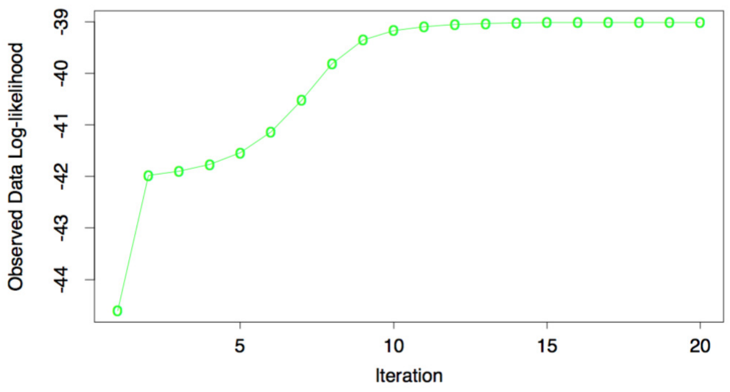


Figure 4: Progress of the Expectation-Maximization algorithm for the 1-dimensional dataset.

Iteration	$\hat{\pi}$
1	0.485
5	0.493
10	0.523
15	0.544
20	0.546

Figure 5: Estimates of π across select iterations of the Expectation-Maximization algorithm for the 1-dimensional dataset.

1.2 Why does EM Work?

Lets look at the following conditional distribution relation, that related the two likelihoods:

$$\begin{aligned} Pr(\mathbf{Z}\,|\theta') &= \frac{Pr(\mathbf{T}\,|\theta')}{Pr(\mathbf{Z}^m\,|\,\mathbf{Z},\theta')} \\ \Rightarrow l(\theta';\mathbf{Z}) &= l_0(\theta';\mathbf{T}) - l_1(\theta';\mathbf{Z}^m\,|\,\mathbf{Z}). \end{aligned}$$

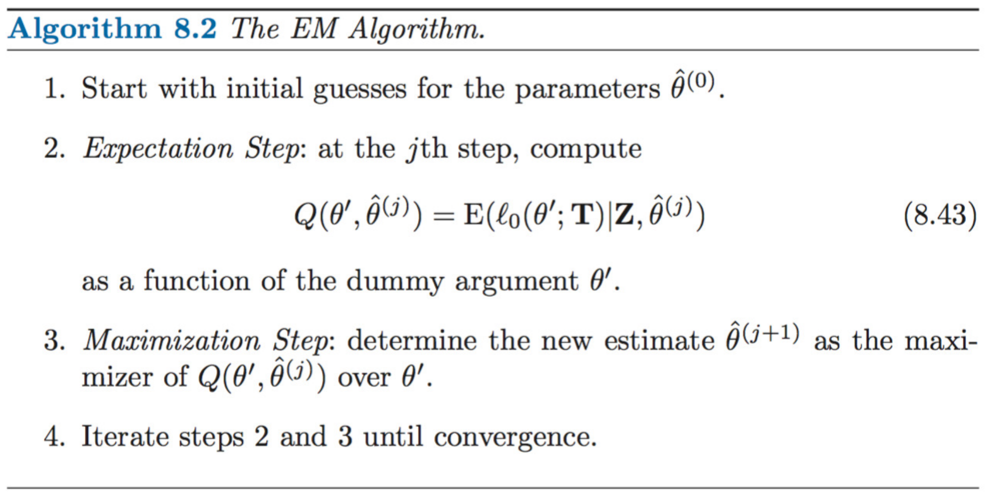
Take conditional expectation of the above equality with respect to $Pr(\mathbf{T}\,|\,\mathbf{Z},\theta)$, we get:

$$\begin{aligned} l(\theta';\mathbf{Z}) &= E[l_0(\theta';\mathbf{T})|\,\mathbf{Z},\theta] - E[l_1(\theta';\mathbf{Z}^m\,|\,\mathbf{Z})|\,\mathbf{Z},\theta] \\ &= Q(\theta',\theta) - R(\theta',\theta). \end{aligned}$$

So in the M step, we maximize the first term above with respect to θ'. But we still succeed in maximizing $l(\theta';\mathbf{Z})$, which is our original log-likelihood.

This is because the second term is maximized[2] when $\theta = \theta'$. What does this imply, lets look at two different parameter values, θ' and θ, where θ' maximizes $Q(\theta',\theta)$. Then, the

[2]We will assume that this is true here. You can verify this claim yourself.


Algorithm 8.2 *The EM Algorithm.*

1. Start with initial guesses for the parameters $\hat{\theta}^{(0)}$.
2. *Expectation Step*: at the jth step, compute
$$Q(\theta', \hat{\theta}^{(j)}) = \mathrm{E}(\ell_0(\theta'; \mathbf{T})|\mathbf{Z}, \hat{\theta}^{(j)}) \tag{8.43}$$
as a function of the dummy argument θ'.
3. *Maximization Step*: determine the new estimate $\hat{\theta}^{(j+1)}$ as the maximizer of $Q(\theta', \hat{\theta}^{(j)})$ over θ'.
4. Iterate steps 2 and 3 until convergence.

Figure 6: The Expectation-Maximization algorithm in general.

difference in the original log-likelihood will be:

$$\begin{aligned} l(\theta'; \mathbf{Z}) - l(\theta; \mathbf{Z}) &= Q(\theta', \theta) - R(\theta', \theta) - (Q(\theta, \theta) - R(\theta, \theta)) \\ &= (Q(\theta', \theta) - Q(\theta, \theta)) - (R(\theta', \theta) - R(\theta, \theta)) \\ &\geq 0. \end{aligned}$$

The above inequality just means that we are always improving our original objective values in each iteration. Hence, EM will at least get to the local maxima.

In summary, EM is a very powerful technique to fit complex models to many different type of data: from fitting mixture models to community detection in social network analysis. for every application, the E step and the M steps may need to be customized.

2 Sampling from the Posterior

Lets go back to Bayes rule. if you remember MAP estimation, it includes the prior on the parameter θ and computes the mode of the posterior distribution over θ given data.

Here we discuss ways to compute the mode, or other functions of the posterior via the technique of *sampling*.

Example 2. Say you have a random variable Z. Then one way to estimate $E[Z]$ is to sample say N realizations and take a *Monte Carlo average* $\frac{1}{N}\sum_{i=1}^{N} z_i$. Of course, $E[Z]$ can be computed analytically given the distribution/density if the latter is a well known distribution/density like the Multinomial or the Gaussian.

Sampling from certain distributions is easy, especially if you have access to samples from the uniform distribution $U[0, 1]$.

Example 3. Say we want to sample from a continuous increasing distribution function F which is defined as $F(z) = Pr(Z \leq z)$. Let F^{-1} be its inverse function. Then, Z can be sampled in two steps:

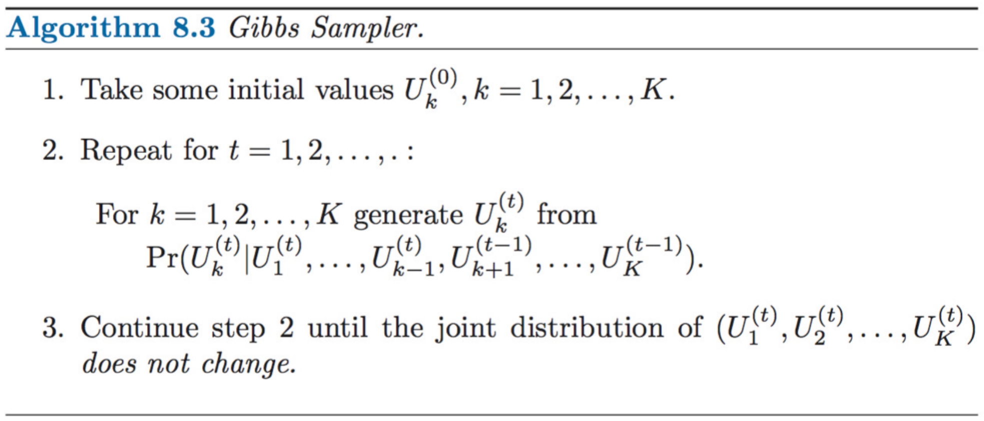
- Sample U from $U[0, 1]$.
- Return $Z = F^{-1}(U)$.

For example, $Z = -\frac{1}{\lambda}\log(U)$ is exponentially distributed with parameter λ.

If we wish to sample from a more complex distribution, say that of a p-dimensional random vector $U = [U_1, ..., U_p]^T$, there are a family of techniques called Markov Chain Monte Carlo, one of which we will discuss: the Gibbs Sampler.

The key assumption for the Gibbs sampler is that it is easy to sample from $P(U_j|U_1, ..., U_{j-1}, U_{j+1}, ..., U_p)$ for every $j = 1, ..., p$.

The key idea of Gibbs sampling is to sample from each of the conditional distributions in some sequence, many times, and finally bundle p of them together to get a sample. This is illustrated in Figure 7. It turns out that this repeated sampling is related to an object called the *Markov chain.* And the relation is this: when we are sampling coordinate U_j, it depends on $p-1$ samples that we obtained previously and everything before that is irrelevant (this is called the Markov property). Gibbs sampler stabilizes after some time and actually produces realizations u that are from the distribution $P(U_1, ..., U_p)$ [3].



Algorithm 8.3 *Gibbs Sampler.*

```
1. Take some initial values $U_k^{(0)}, k = 1, 2, \dots, K$.
2. Repeat for $t = 1, 2, \dots, $ :
     For $k = 1, 2, \dots, K$ generate $U_k^{(t)}$ from
         $\Pr(U_k^{(t)}|U_1^{(t)}, \dots, U_{k-1}^{(t)}, U_{k+1}^{(t-1)}, \dots, U_K^{(t-1)})$.
3. Continue step 2 until the joint distribution of $(U_1^{(t)}, U_2^{(t)}, \dots, U_K^{(t)})$
   does not change.
```

Figure 7: The Gibbs Sampler.

Note 1. There are many other sampling techniques, and Gibbs sampler is just one of them. They are very relevant in Bayesian statistical modeling.

[3]One can verify this claim, although we will skip this here.

In the next several sections/lectures, we will get back to supervised learning. In particular, we will look at classification and regression methods that work with different (sometimes lesser) assumptions than linearity, but still give great predictive performance. These are:

1. Generalized Additive Models (GAMs),
2. *(future)* Tree-based methods,
3. *(future)* Multivariate Adaptive Regression Splines (MARS),
4. *(future)* Adaboost and Gradient Boosting Methods,
5. *(future)* Random Forest, and
6. *(future)* Support Vector Machines.

3 Generalized Additive Models

In many situations, Y does not depend linearly on X. To counter this, a GAM assumes the following:

$$E[Y|X] = \alpha + f_1(X_1) + f_2(X_2) + ... + f_p(X_p),$$

where f_j can be specified arbitrarily. For example, each such function can be a cubic smoothing spline[4].

Example 4. The GAM version of logistic regression for two classes would be:

$$\log(\frac{P(G=1|X)}{1-P(G=1|X)}) = \alpha + f_1(X_1) + f_2(X_2) + ... + f_p(X_p).$$

It turns out that $P(G=1|X) = \mu(X)$ is the conditional mean of response Y. In general, this mean is related to the right hand side above via a *link* function. Example link functions include:

- Identity link: $g(\mu) = \mu$.
- Logit link: $g(\mu) = \log(\frac{\mu}{1-\mu})$.
- Probit link: $g(\mu) = \Phi^{-1}(\mu)$.
- log-linear link: $g(\mu) = \log(\mu)$.

[4]Such a function is the minimizer (over the class of twice differentiable functions) of $\sum^N (y_i - f(x_i))^2 + \lambda \int f''(x)^2 dx$.

These (and other links) define the family of Generalized Linear Models (GLMs) when $f_j(X_j) = X_j$.

Note 2. We can have nonlinear functions of more than one X_j as well.

The way these functions are estimated from data, is by solving the following optimization problem:

$$PRSS(\alpha, f_1, ..., f_p) = \sum_{i=1}^{N}(y - \alpha - \sum^{p} f_j(x_i j))^2 + \sum_{j=1}^{p} \lambda_j \int f_j''(t_j)^2 dt_j,$$

where x_{ij} is the j^{th} component of the i^{th} observation. It turns out that this is not very difficult, although we will skip it to focus on other models.

Example 5. An example set of functions for a spam classification task is shown in Figure 8.

Note 3. Unequal Loss Functions: In a spam classification task, it may be more important to not classify a genuine email as spam, compared to classifying a spam as a genuine email. In order to do so, the $0-1$ loss function can be changed to penalize one type of mis-classification more than the other.

4 Summary

We learned the following things:

- Learning/inference in general settings can be achieved using maximum likelihood and the expectation maximization (EM) methods.
- Sampling as a way to infer/estimate quantities of interest, especially when the probability distribution function is complex.
- More models for regression and classification including: Generalized Additive Models and Tree-based methods.

A Sample Exam Questions

1. When is the EM algorithm useful? How is it different from MLE?
2. In what situations would sampling be useful?
3. What are the key differences between General Additive Models and linear models? Which family is more flexible?

B Sensitivity and Specificity

In certain 2-class classification applications (say, disease (1) and no-disease (0)), one is not only interested in a single mis-classification metric, but in two additional related metrics:

1. Sensitivity: It is the probability of the model predicting disease when the true label is disease.

2. Specificity: It is the probability of the model predicting no-disease when the true label is no-disease.

These can be empirically computed over training/validation data just like any other performance measure. By introducing unequal loss values (say $L_{kk'} \neq L_{k'k}$; here $L_{kk'}$ is the loss for predicting true label k as label k') we can change these numbers. For example, if we want specificity to be high, then we can set L_{01} high.

The receiver operating characteristic curve (ROC) is used to shown this trade-off between sensitivity and specificity. It plots sensitivity vs specificity.

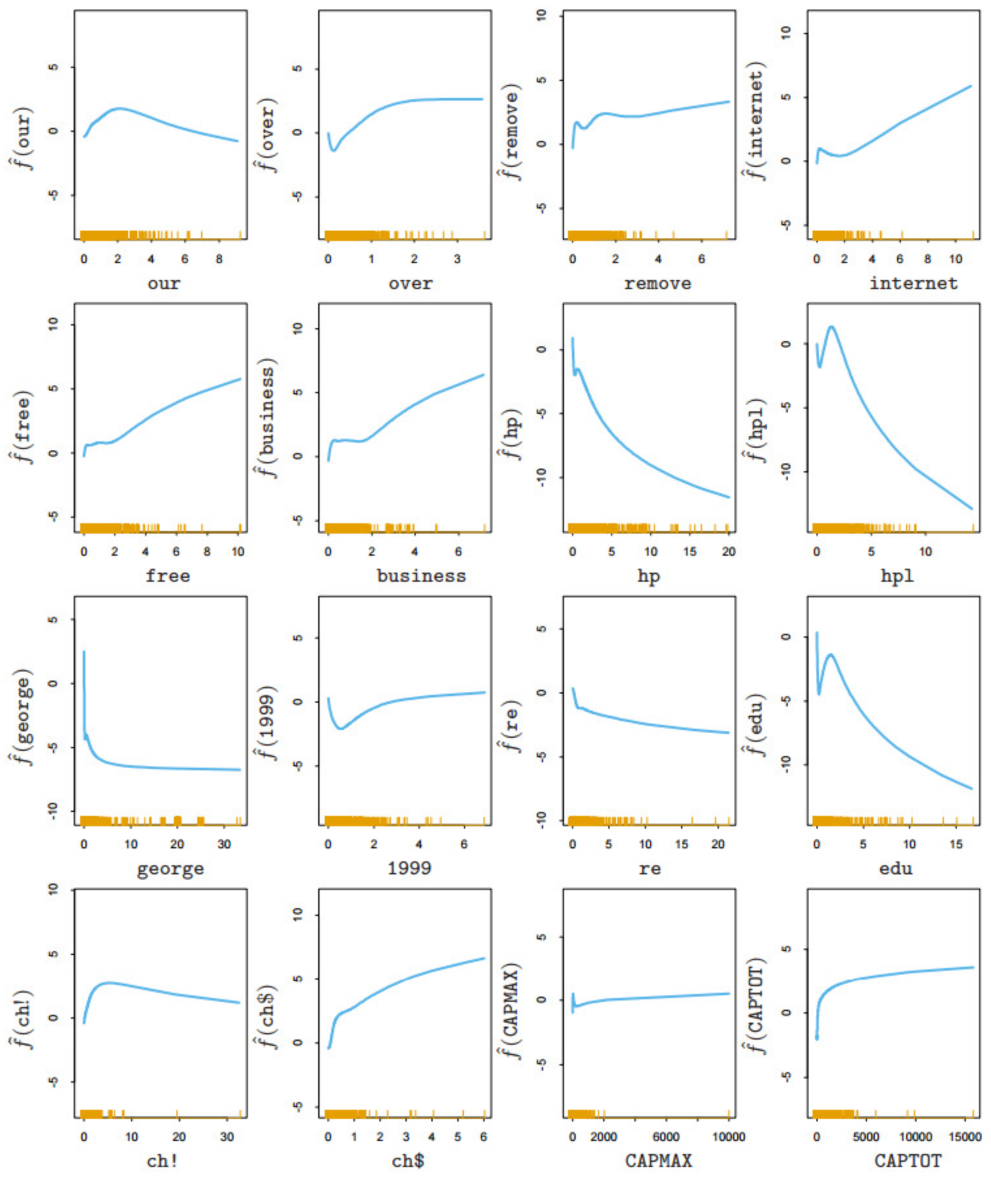


Figure 8: Estimated functions for a GAM model for a spam classification dataset.

Lecture 8

IDS575: Statistical Models and Methods
Theja Tulabandhula

Notes derived from the book titled "Elements of Statistical Learning [2nd edition] (Sections 9.2, Chapter 10)

Continuing from before, we will look at classification and regression methods that work with different (sometimes lesser) assumptions than linearity, but still give great predictive performance:

1. *(previously)* Generalized Additive Models (GAM),
2. *(today)* Tree-based methods,
3. *(today)* Adaboost and Gradient Boosting Methods,
4. *(future)* Random Forests,
5. *(future)* Multivariate Adaptive Regression Splines (MARS), and
6. *(future)* Support Vector Machines.

1 Tree-based Methods

These are methods that partition the input space into a set of rectangles and for each rectangle, make a constant prediction.

We will discuss a popular technique known as *CART (Classification and Regression Tree)* that can be used for regression and classification.

Example 1. An illustration of how a tree model looks like is given in Figure 1 for a 2-dimensional regression dataset.

Here, we are recursively doing a binary partitioning of the input space. First, we break it into two parts and model the response by the mean of y_i in each region. The choice of the

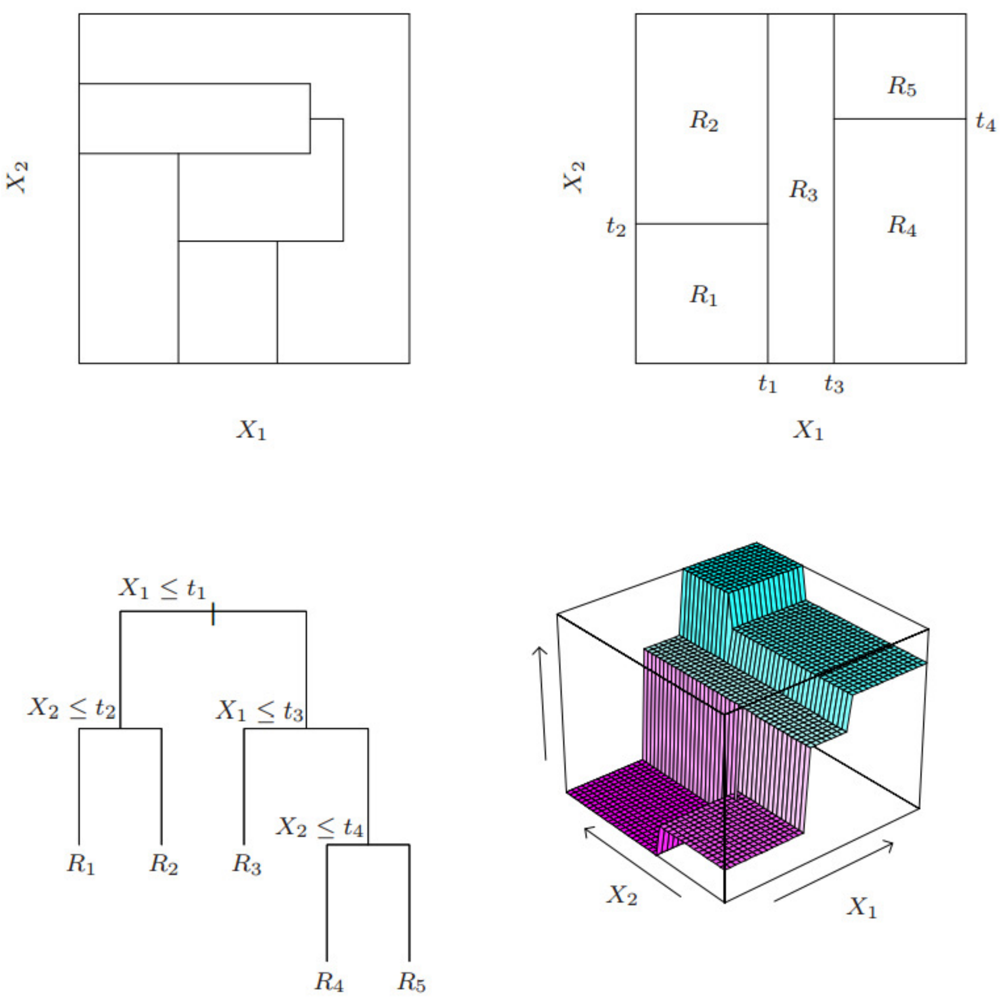


Figure 1: Partitioning of the input space in general (top left), partitioning using CART (top right), equivalent tree representation (bottom left), and predictions in each region (bottom right).

variable to split and where to split is based on some criteria (we will describe soon). Then we repeat the partitioning in these regions as long as a stopping condition is not met. In the figure, we get five regions. Our prediction function is:

$$\widehat{\mathrm{f}}(x) = \sum_{m=1}^{5} c_m 1[x \in R_m],$$

where c_ms are the constant predictions in each region.

One of the advantages of recursive binary tree is *interpretability*, even when we have $p > 2$ dimensions.

1.1 Regression Trees

Computing the best binary partition that minimizes the residual sum of squares (for example) is hard. So there is a greedy approach:

- Start with all data. Consider a split variable j and split point s. This defines regions $R_1(j,s) = \{X : X_j \leq s\}$ and $R_2(j,s) = \{X : X_j > s\}$. We can optimize for j, s by solving the following problem:

$$\min_{j,s} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right].$$

- For a choice of (j,s), the optimal $\hat{c}_k = \text{avg}(y_i | x_i \in R_k(j,s))$ for $k = 1, 2$. Optimizing over this choice is not very difficult.

- We repeat the splitting for each of the regions obtained above.

- We stop the process when some criteria is met (see below).

Note 1. There are a couple of ways to stop the tree expansion process:

- We can stop if the decrease in the sum of squared errors is small after a split compared to before. This doesn't seem to work well in practice.

- We can stop if the number of observations in the current node is less than a number (say 5). Such a tree T_0 may be too big. This can be pruned using a strategy that is called *cost-complexity pruning.*

  - Notation: Let $T \subset T_0$ be a pruning of T_0 by collapsing any number of its internal nodes. Let the number of terminal nodes of any tree T be $|T|$. Let $N_m = |\{x_i \in R_m\}|$, $\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i$ and *node-impurity function* $Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2$.
  - Define the cost-complexity function as $C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|$.
  - We can minimize this[1] for different α values that trade off tree size and its fit to training data (α itself can be chosen via K-fold cross-validation).

Note 2. The size of the tree is a design choice that controls predictive performance. A very large tree will potentially overfit the training data. Whereas a small tree may underfit.

[1] One can successively collapse the internal node that produces the smallest increase in $\sum_m N_m Q_m(T)$.

1.2 Classification Trees

When we have qualitative variable G, then we just need to replace squared loss and $Q_m(T)$ functions. Lets define the proportion of class k observations in a region R_m (node m) as:

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} 1[y_i = k].$$

With this, a new observation in node m can be classified as $k(m) = \arg\max_k \hat{p}_{mk}$. The choice for $Q_m(T)$ can be any of the following:

- Mis-classification: $1 - \hat{p}_{mk(m)}$.
- Gini index: $\sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$.
- Cross-entropy: $-\sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk})$.

Example 2. Here is a visualization in Figure 2 of how these look like when $K = 2$ and $(p_{m1}, p_{m2}) = (1 - p, p)$. if you can find a split such that p is closet to 0 or 1, the lower are these node-impurity functions.

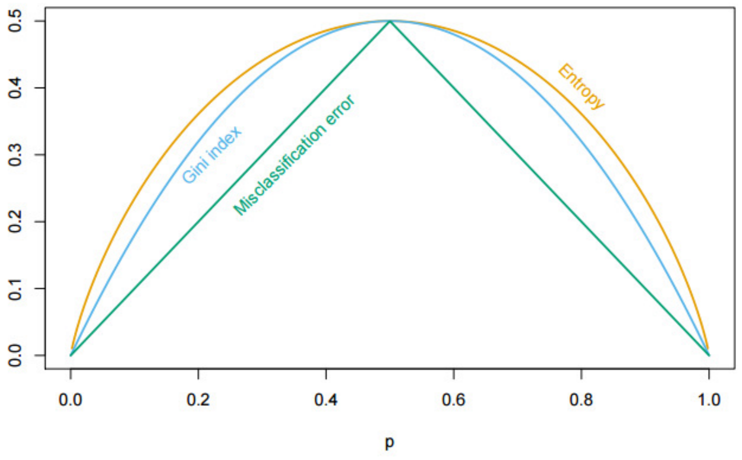


Figure 2: Node-impurity functions are higher when a split leads to higher error (p close to 0.5).

Note 3. There are many variants of tree-based methods such as C4.5, CART etc and we have only looked at one of them, namely, CART.

Example 3. Consider the spam classification problem. The performance of tree-based classifiers is shown in Figure 3. The pruned tree is shown in Figure 4.

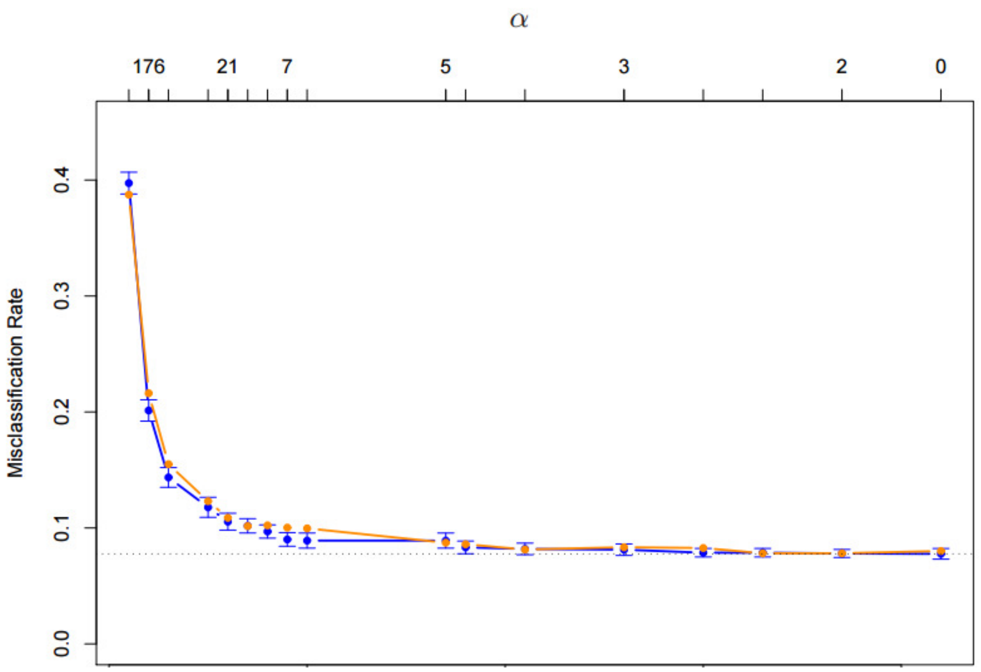


Figure 3: Spam classification: 10-fold CV performance is plotted (blue curve) as a function of α (top label). The orange curve is the test error.

1.3 Issues with Trees

There are several issues to be aware of when using tree-based methods for regression or classification tasks. These are:

- *Categorical predictors*: When X_j takes q values, then there are $2^{q-1} - 1$ partitions into two groups, which is a lot of choices. The impact of this is that the tree building process will be splitting on these choices more than the other variables. There is also a possibility of overfitting because of so many choices.

- *Binary splitting vs continuous splitting*: Although we could do multi-way splits, it tends to break data too quickly leading to worse predictive performance. One could also think of a split such as $\sum_j a_j X_j \geq s$, but these become non-interpretable very quickly.

- *Stability*: Often a small change in data will change the whole tree, especially if there is a change at the beginning stages of tree-building process.

- *Smoothness*: When the underlying regression function $E[Y|X = x]$ is smooth, then region-wise constant function fitting may not be a good idea.

Finally, lets also look at the issue of missing data deeply below, as it not only affects tree-based methods, but many other methods for supervised learning.

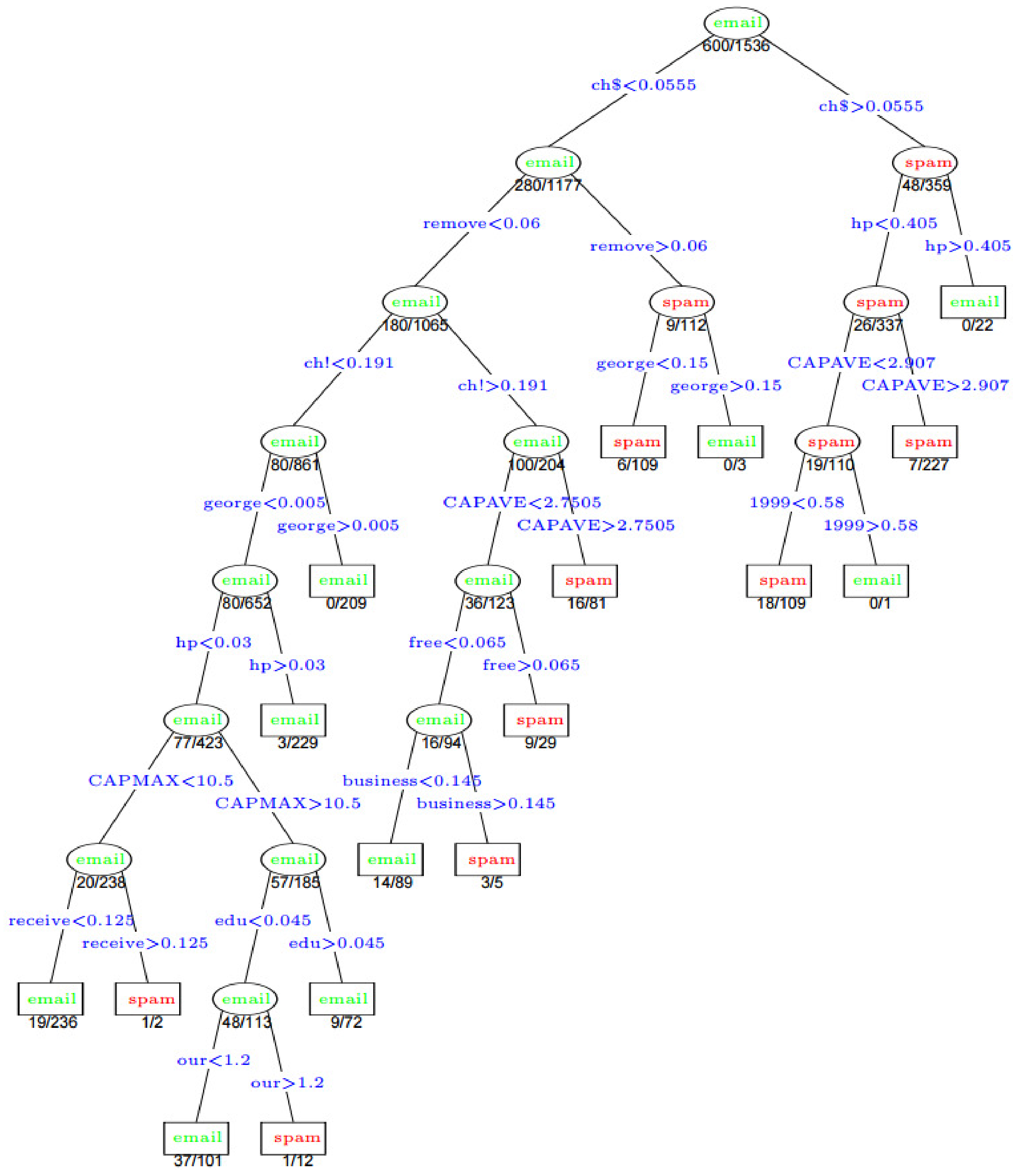


Figure 4: Spam classification: the pruned tree. The majority class is shown in each node. The ratio under the nodes indicate misclassification rates on test data.

1.4 Missing Data

Most importantly, we need to hypothesize whether the missing data influences the observed data in any way before choosing a solution approach.

Example 4. As a toy example, consider that physical measurements were made of Olympic athletes and the latter information was not recorded. Then any analysis based on this sample will be subject to selection bias.

Let $\mathbf{X}_{\text{obs}}$ be the observed entries in $\mathbf{X}$ and let $Z = (\mathbf{y}, \mathbf{X}), Z_{\text{obs}} = (\mathbf{y}, \mathbf{X}_{\text{obs}})$. Let $\mathbf{R}$ be an indicator matrix with ij^{th} entry 1 if x_{ij} is missing.

Data is *missing at random* (MAR) if $Pr(\mathbf{R} \,|\, \mathbf{Z}, \theta) = Pr(\mathbf{R} \,| Z_{\text{obs}}, \theta)$, where θ is a parameter for the distribution of $\mathbf{R}$.

Data is *missing completely at random* (MCAR) if $Pr(\mathbf{R} \,|\, \mathbf{Z}, \theta) = Pr(\mathbf{R} \,| \theta)$, where θ is again a parameter for the distribution of $\mathbf{R}$.

Example 5. If a patient's sickness measurement was not taken because she was too sick, then that observation would not be MAR or MCAR.

Here are some workarounds if data is MCAR:

- If some of the predictors have missing entries, then we can discard the corresponding observations if they are relatively small in number.
- Another option is to *impute*: fill in some reasonable values at these locations in the dataset. This is a popular approach in many settings.
- Depend on an algorithm such as EM to deal with missing values in the training phase.

Note 4. For tree-based methods, if the predictor with missing entries X_j is categorical, then we can just make a new category called 'missing' and proceed. Another way to deal with missing entries is to not consider them in the binary partitioning step. Further, during tree construction, we can track surrogate variable and split choices that mimic the original splits. This helps when during prediction, the new feature vector also has missing entries, then the surrogate variables and splits can be used.

2 Adaboost and Gradient Boosting

This is a method that combines the outputs of many *weak classifiers* in a way that has very good predictive performance, both for regression and classification. A weak classifier is one which is only slightly better than random guessing.

Lets directly jump into a boosting algorithm called *Adaboost.M1*. Let there be 2 classes such that $Y \in \{-1, 1\}$. We will use $G(X)$ to represent a classifier[2].

Intuitively boosting in this setting applies the weak classification algorithm on modified copies of the original training data producing classifiers $G_m(x)$ for $m = 1, ..., M$. The final classifier is $G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$. See Figure 5 for a schematic and Figure 6 for the algorithm.

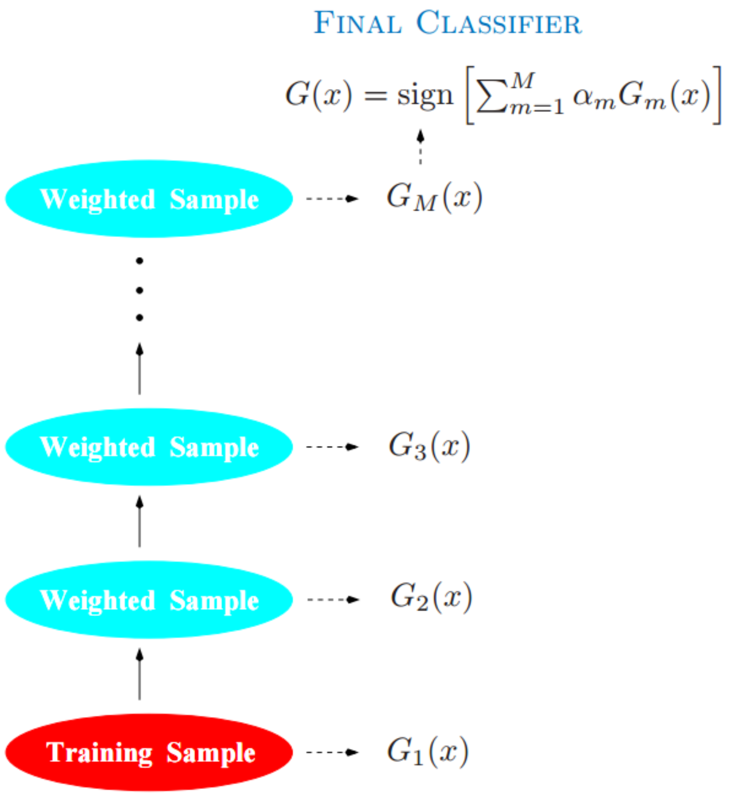
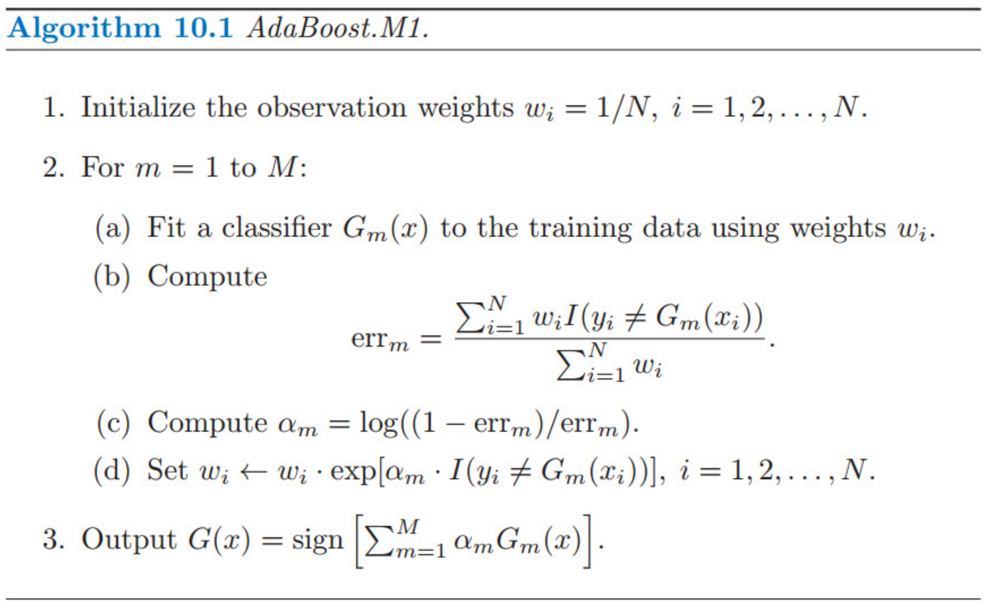


Figure 5: Adaboost.

[2]Notation: we had used G to represent qualitative output variable before, here we are using it as a classification function.



Algorithm 10.1 *AdaBoost.M1.*

```
1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to $M$:
    (a) Fit a classifier $G_m(x)$ to the training data using weights $w_i$.
    (b) Compute
        $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
    (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
    (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.
```

Figure 6: Adaboost.M1 algorithm.

Example 6. Consider a synthetic data setup: $Y = 1$ if $\sum_{j=1}^{10} X_j^2 > 9.3$ and 0 otherwise. The X_js are independent standard Gaussians. Let $N = 2000$. The weak classifier is a two-node classification tree (also called a *stump*). As shown in Figure 7, the performance of the stump is 45.8%, and a 244-node tree is 24.7%, whereas boosting reaches 5.8% in 400 iterations.

2.1 Why does Boosting Work?

Boosting works because it fits an additive model with exponential loss.

In particular, it sequentially adds a new basis function (G_m) without adjusting the parameters and coefficients of those that have already been added (α_j, G_j for $j = 1, .., m-1$).

A general version of this stagewise additive model fitting is shown in Figure 8.

For Adaboost.M1, $L(y, f(x)) = \exp(-yf(x))$. Hence, we solve

$$(\beta_m, G_m) = \arg\min_{\beta, G} \sum_{i=1}^{N} \exp(-y_i f_{m-1}(x_i)) \exp(-\beta y_i G(x_i)).$$

From manipulating this expression a bit (we will not do that here), Adaboost.M1 can be seen as equivalent to forward stagewise additive modeling, which we know are good predictive models. A slight difference is that Adaboost is not minimizing training-set misclassification error (see Step 2(a) in figure 6 where no loss criteria is specified).

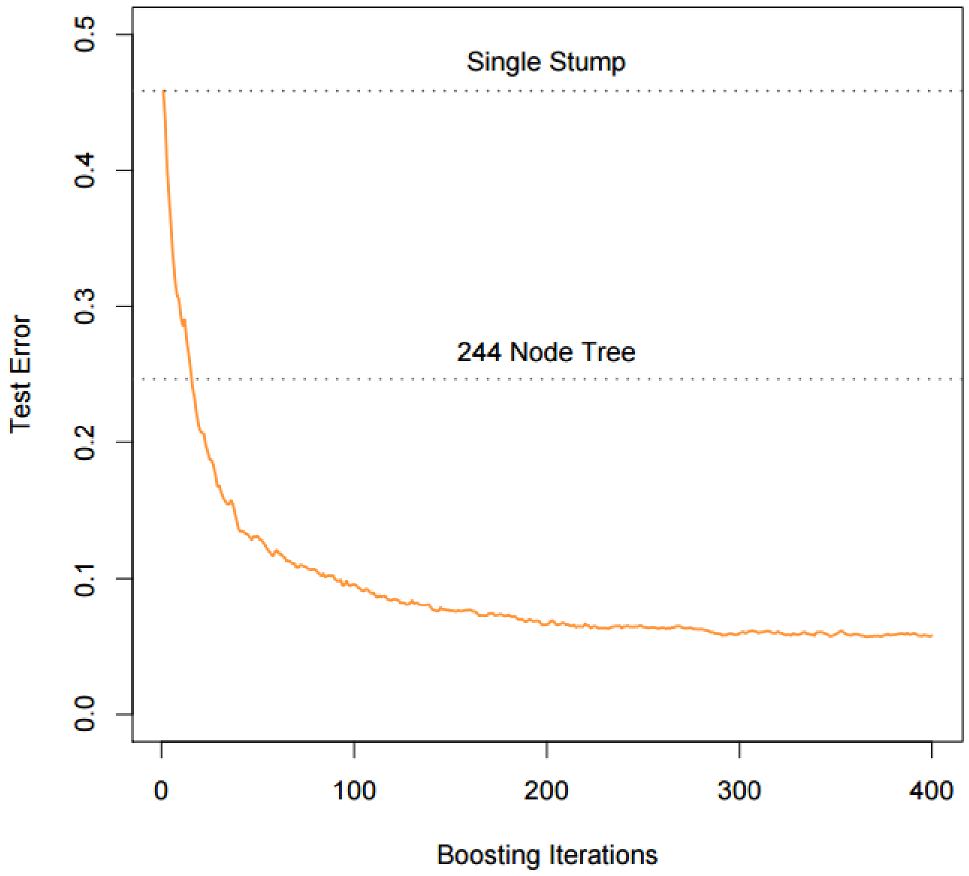


Figure 7: Boosting on toy data.

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

```
1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to $M$:
    (a) Compute
        $$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$
    (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.
```

Figure 8: General stagewise fitting. Here $b(x_i; \gamma)$ is a basis function.

Example 7. A boosted tree is a sum of scaled trees:

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m).$$

We would like to solve the following step in the forward stagewise procedure:

$$\Theta_m = \arg\min_{\Theta_m} \sum_{i=1}^{M} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)),$$

where $\Theta_m = \{R_{jm}, \gamma_{jm}\}_1^{J_m}$ corresponds to the regions and the constants of the next tree given the current model $f_{m-1}(x)$. Finding the regions is difficult, while finding constants is relatively straightforward:

$$\hat{\gamma}_{jm} = \arg\min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm}). \tag{1}$$

We get Adaboost with trees (Figure 6) when the problem is a 2-class classification problem with exponential loss function.

Some advantages of tree methods are sacrificed by boosting: computational speed, interpretability and robustness against mislabeling of training data. Gradient boosted (tree) models attempt to mitigate these issues.

In fact, we will focus on speed below. When the loss function is general (not just least squares or exponential), performing stagewise boosting is computationally difficult, so we use gradient boosting as a way to get around this.

2.2 Gradient Boosted Models (GBMs)

Lets now motivate gradient boosting by thinking about gradients in a generic problem. Say $L(f_m) = \sum_{i=1}^{N} L(y_i, f_m(x_i))$. Instead of thinking of function f_m, lets think of vector $\mathbf{f}_m \in \mathbb{R}^N$, where $\mathbf{f}_m = [f_m(x_1), ..., f_m(x_N)]^T$.

We can think of these as N numbers that need to be set to minimize $L(f_m)$. And there is a family of methods (actually gradient based methods) that do this by iteratively updating $\mathbf{f}_m$ from $\mathbf{f}_{m-1}$ using gradient information.

For example, there is a numerical method called steepest descent which does the following:

- Computes $\mathbf{g}_m \in \mathbb{R}^N$ such that

$$g_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i) = f_{m-1}(x_i)}.$$

- Computes *step length* ρ_m such that

$$\rho_m = \arg\min_{\rho} L(\mathbf{f}_{m-1} - \rho\, \mathbf{g}_m).$$

- Then updates $\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m\, \mathbf{g}_m$ and repeats.

The intuition behind steepest descent is that $-\mathbf{g}_m$ is the local direction in $\mathbb{R}^N$ that decreases the objective $L(f_m)$ from the current point $\mathbf{f}_{m-1}$.

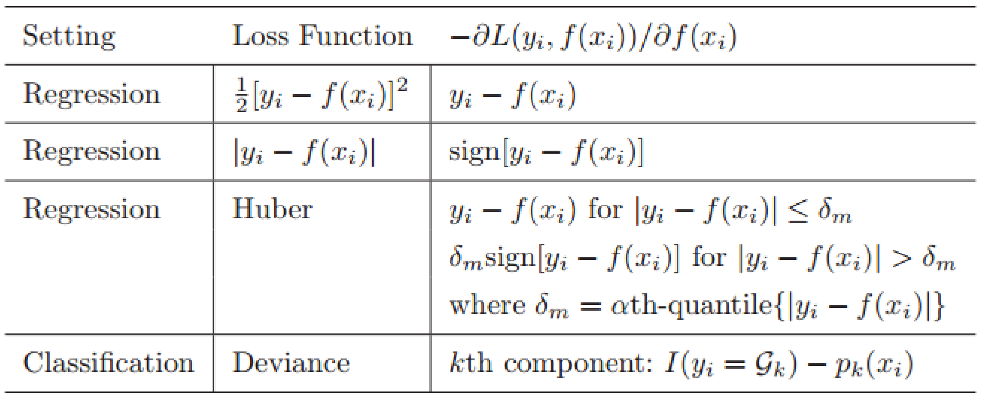
Of course, these N numbers in vector $\mathbf{f}_m$ are dependent on each other through a model, such as a tree. This brings us to the next idea: building a tree (or any other model) that approximates these numbers ($\mathbf{f}_m$).

Tree predictions $T(x_i; \Theta_m)$ are analogous to components of the negative gradient. And thus the N numbers are constrained to be predictions of a tree.

If we find such a tree, then finding the γ_{jm} is similar to finding ρ_m above. In the former, we can think of doing J_m line searches, one for each region.

So the key idea is to induce a tree $T(x; \Theta_m)$ at the m^{th} iteration whose predictions are close to the negative gradient , say by using a least squares fit. Once the tree is fit, the constants in each region are fit using Equation 1.

Gradients for commonly used loss functions are given in Figure 9.



Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\|y_i - f(x_i)\|$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\|y_i - f(x_i)\| \le \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\|y_i - f(x_i)\| > \delta_m$ where $\delta_m = \alpha$th-quantile$\{\|y_i - f(x_i)\|\}$
Classification	Deviance	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

Figure 9: Gradients.

Figure 10 presents the gradient tree-boosting algorithm for regression, which summarizes what we discussed above.

Note 5. The key parameters for this method are: (a) the number of iterations M, and (b) the size of trees $J_m, m = 1, .., M$.

```
Algorithm 10.3 Gradient Tree Boosting Algorithm.

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to $M$:
    (a) For $i = 1, 2, \ldots, N$ compute
            $r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$.
    (b) Fit a regression tree to the targets $r_{im}$ giving terminal regions
        $R_{jm}$, $j = 1, 2, \ldots, J_m$.
    (c) For $j = 1, 2, \ldots, J_m$ compute
            $\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right)$.
    (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.
```

Figure 10: Gradient boosting algorithm for trees.

2.3 Gradient Boosting for Classification

For classification, K trees are fitted at each iteration. Tree T_{km} is fit to its negative gradient vector $\mathbf{g}_{km}$ which component wise is given as:

$$-g_{ikm} = \left[\frac{\partial L(y_i, f_1(x_i), ..., f_K(x_i))}{\partial f_k(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}.$$

3 Summary

We learned the following things:

- Tree-based methods.
- Idea of boosting to get better predictive performance.

A Sample Exam Questions

1. What is Gini index in the context of a classification-tree?

2. What are the different models of missing data?

3. Describe the Gradient Boosted Tree algorithm. What is the use of gradients?

B Loss Functions

In the classification setting, loss function L should ideally penalize incorrect classifications. A generalized notion of this is to penalize negative *margins* (margin is defined as $yf(x)$) instead of misclassifications. See Figure 11 for some examples.

Example 8. Misclassification loss $L(y, f(x)) = 1[yf(x) < 0]$ where $1[\cdot]$ is an indicator function (takes value 1 when the argument is true).

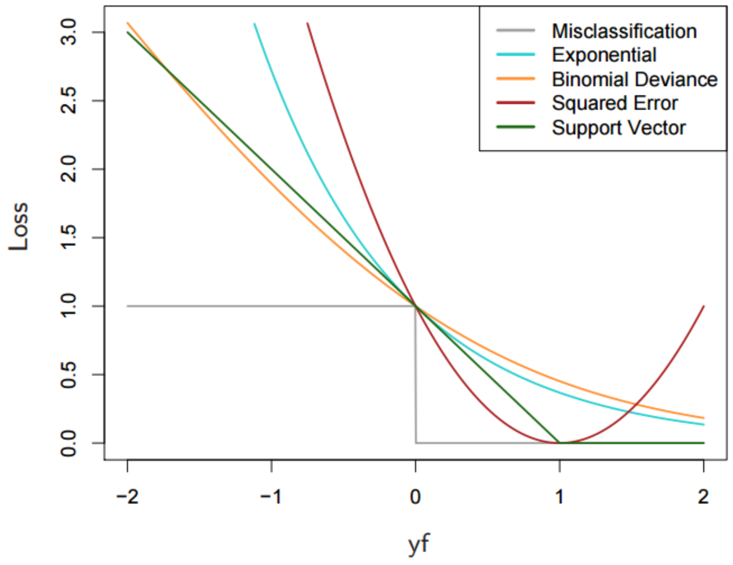


Figure 11: Loss functions for classification vs margin.

A similar set of loss functions for regression are shown in Figure 12.

C Prediction Models

Here is a table (Figure 13) that shows the relative merits of various prediction models.

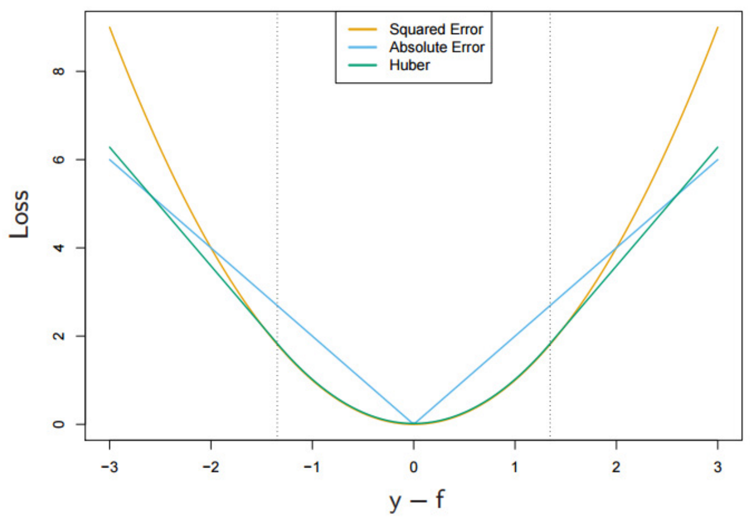


Figure 12: Loss functions for regression vs $y - f(x)$.

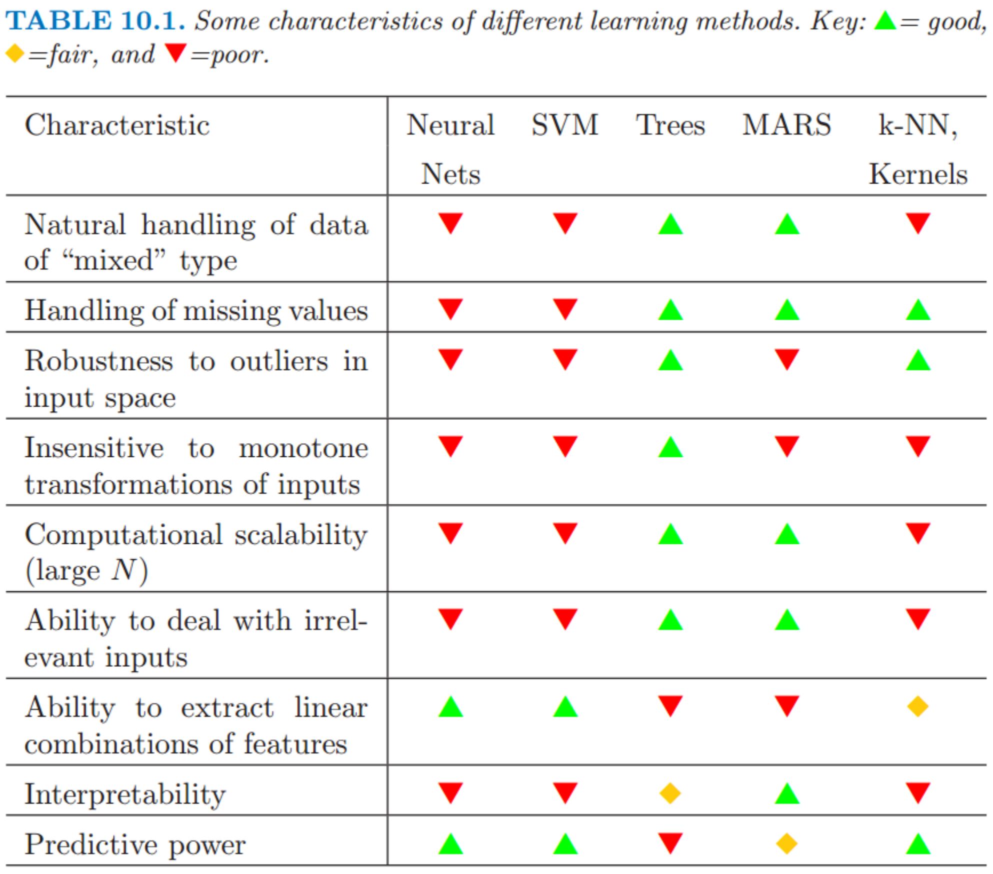


TABLE 10.1. *Some characteristics of different learning methods. Key:* ▲*= good,* ◆*=fair, and* ▼*=poor.*

Characteristic	Neural Nets	SVM	Trees	MARS	k-NN, Kernels
Natural handling of data of "mixed" type	▼	▼	▲	▲	▼
Handling of missing values	▼	▼	▲	▲	▲
Robustness to outliers in input space	▼	▼	▲	▼	▲
Insensitive to monotone transformations of inputs	▼	▼	▲	▼	▼
Computational scalability (large N)	▼	▼	▲	▲	▼
Ability to deal with irrelevant inputs	▼	▼	▲	▲	▼
Ability to extract linear combinations of features	▲	▲	▼	▼	◆
Interpretability	▼	▼	◆	▲	▼
Predictive power	▲	▲	▼	◆	▲

Figure 13: List of prediction models.

Lecture 9

IDS575: Statistical Models and Methods
Theja Tulabandhula

Notes derived from the book titled "Elements of Statistical Learning [2nd edition] (Section 9.4, Chapters 12 and 15)

We continue our foray into more supervised learning methods, viz., Random Forests, Multivariate Adaptive Regression Splines and Support Vector Machines.

1 Random Forests

The Random Forests method improves on *bagging* by reducing the correlation between the sampled trees. Below is a brief description of bagging.

1.1 Bagging

Bagging (bootstrap aggregation) improves the performance of a classifier through averaging predictions across bootstrapped models. For each bootstrap sample $Z^{*b}, b = 1, ..., B$, we fit our model giving prediction $\widehat{\mathrm{f}}^{*b}(x)$ at some new point x. The bagging prediction is:

$$\widehat{\mathrm{f}}_{\mathrm{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B}\widehat{\mathrm{f}}^{*b}(x).$$

Note 1. Each bootstrap tree could involve different features than the original and may have different number of terminal nodes.

Note 2. Bagged estimates for classification cannot be used as estimates of the true conditional distribution of the class given input.

In regression with squared loss, bagging helps smooth out the high variance in predictions, especially when inputs are highly correlated.

1.2 Details of the Random Forests (RF) Method

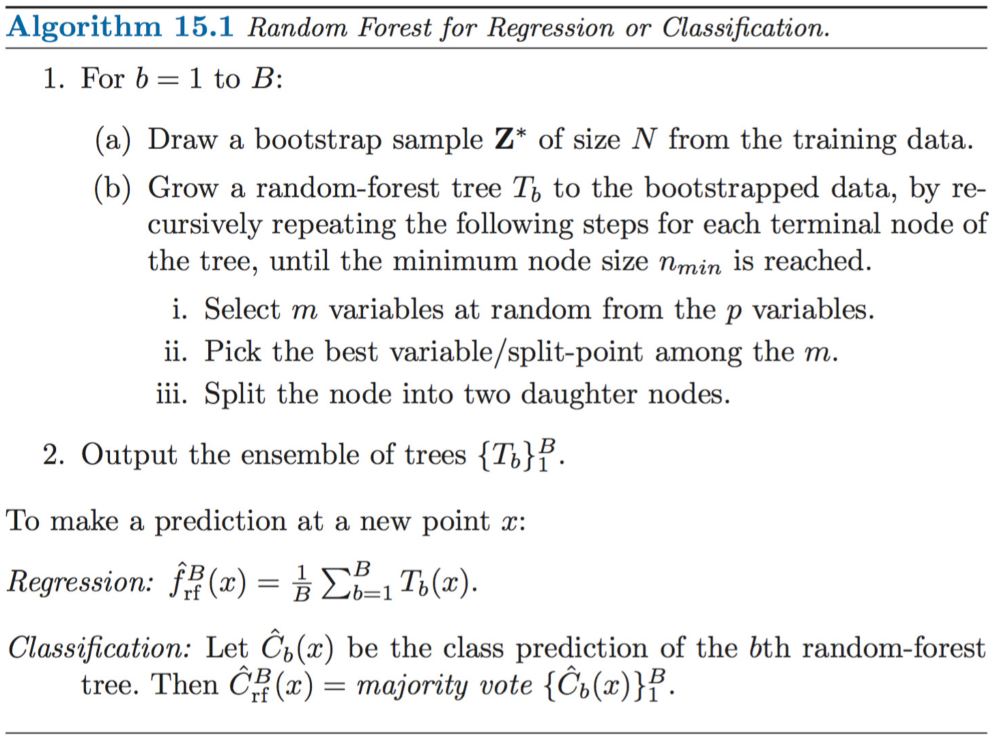
The key idea in RF is to build a large collection of *de-correlated* trees and then taking averages.

The shortcoming of bagging is that the expectation of the average of B trees is the same as any single tree. This means that error can only be decreased via variance reduction. But since each of the trees are built using the same data, they are correlated with each other.

Example 1. If B i.i.d. random variables are averaged the variance of the average is $\frac{1}{B}\sigma^2$. On the other hand, if they are pairwise correlated, then the variance of the average is $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$. So as B increases there is no further reduction in the first term.

RF tries to improve on bagging by reducing the correlation between trees without increasing the variance too much.

The RF algorithm is shown in Figure 1. The key feature is that before each split, we select $m \leq p$ of the input variables randomly as candidates for splitting, while building a tree with each bootstrapped sample. Smaller m reduces correlation across trees.



Algorithm 15.1 *Random Forest for Regression or Classification.*

```
1. For $b = 1$ to $B$:
    (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size $N$ from the training data.
    (b) Grow a random-forest tree $T_b$ to the bootstrapped data, by re-
        cursively repeating the following steps for each terminal node of
        the tree, until the minimum node size $n_{min}$ is reached.
          i. Select $m$ variables at random from the $p$ variables.
         ii. Pick the best variable/split-point among the $m$.
        iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.
```

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

Figure 1: The RF algorithm.

Example 2. In Figure 2, RF is compared to GBM with shrinkage[1]for the California housing dataset. From the plot, we can see that RF stabilizes with about 200 trees whereas boosting continues to improve as more trees are added.

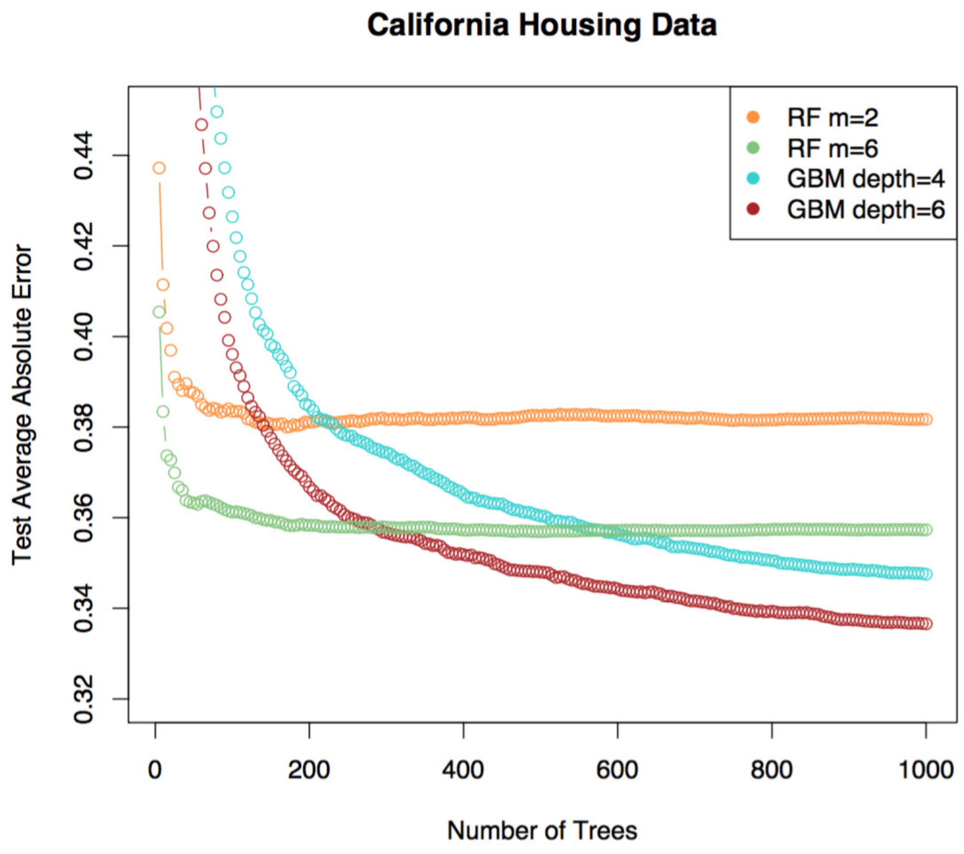


Figure 2: The RF algorithm performance on a housing dataset.

Note 3. RF takes an average for regression, and it takes the majority vote for classification.

Note 4. Out-of-Bag (OOB) samples are used to compute validation performance: for each (x_i, y_i), only those trees that did not use this data are averaged. This can be used in lieu of K-fold cross validation. OOB samples can be used to estimate variable important estimates as well, although we will not discuss this further here.

1.3 Interpretation

We will look at a couple of ways to get interpretability with this class of models. First is via relative importance of variables, and the second is via partial dependence plots.

[1]Shrinkage is an additional parameter that scales down the contribution of each tree while being added to the sum.

The idea here is to isolate those input variables that are the most relevant to the prediction task.

For a single tree, let the measure of (squared) relevance for variable X_l be denoted by $I_l^2(T) = \sum_{t=1}^{J-1} \widehat{\text{i}}_t^2 \, 1[v(t) = l]$. Here, the sum is over the $J-1$ internal nodes. We are checking if the variable $v(t)$ used to split the node was l or not, and if it is, we are adding the improvements denoted as $\widehat{\text{i}}_t^2$. While constructing the tree, the variable that is chose gives the maximal estimated improvement $\widehat{\text{i}}_t^2$ in squared error over that for a constant over the entire region.

For boosted tree models, the formula for importance is simply $I_l^2 = \frac{1}{M} \sum_{m=1}^{M} I_l^2(T_m)$.

Note 5. Since these measures are relative, one can scale the largest variable's score to 100 and rescale others respectively.

Note 6. For classification, this is very similar, and we add up the K trees for each m.

Example 3. Figure 3 shows the relative importance of certain variables in a prediction task involving California housing dataset.

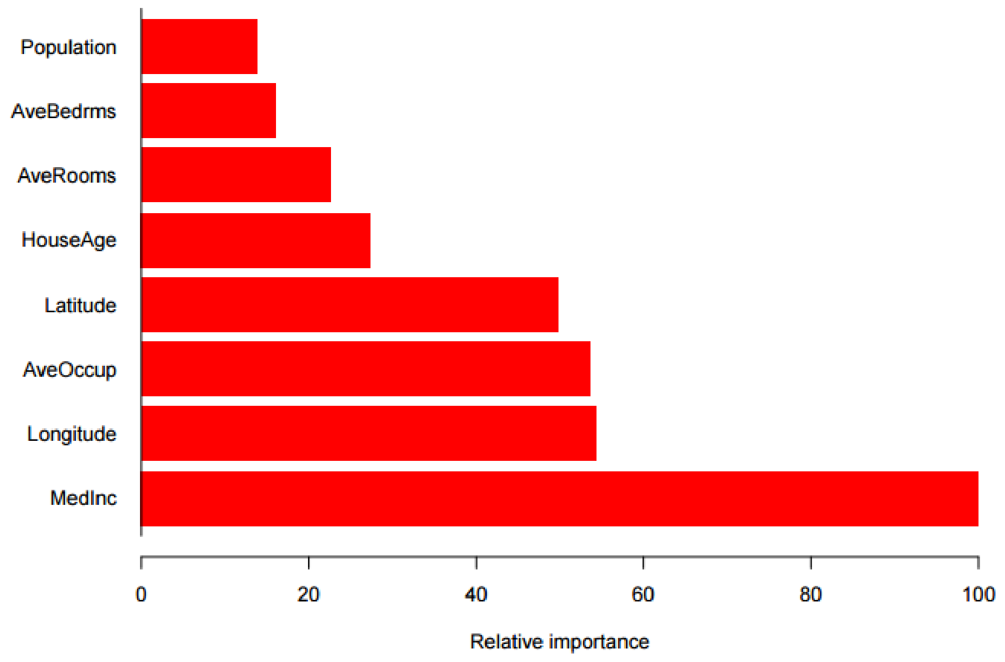


Figure 3: Example relative importance plot.

Partial dependence plots help understand how $f(X)$ depends on some of the input variables (up to 2 variables). Let X_S denote a subset of $l < p$ variables (and X_C its complement). Let $f(X) = f(X_S, X_C)$. Then, the *partial dependence* of $f(X)$ on X_S is given as:

$$f_S(X_S) = E_{X_C} f(X).$$

They can be estimated using data as: $f_S(X_S) = \frac{1}{N}\sum_{i=1}^{N} f(X_S, x_{iC})$. This is true for both regression and classification.

2 Multivariate Adaptive Regression Splines (MARS)

MARS gives us a regression model that is composed of 1-dimensional basis functions defined below:

$$(x-t)_+ = \begin{cases} x-t & \text{if } x > t \\ 0 & \text{if } x \le t \end{cases}, \text{ and}$$

$$(t-x)_+ = \begin{cases} t-x & \text{if } x < t0 \\ 0 & \text{if } x \ge t \end{cases}.$$

Here, t is called a *knot*.

Example 4. An example plot of these 1-dimensional functions is shown in Figure 4.

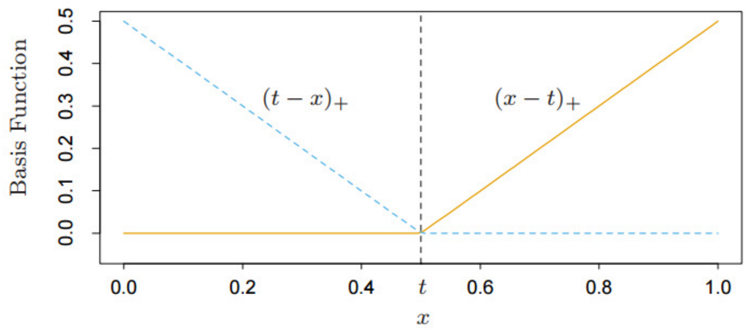


Figure 4: The basis functions (for $t = 0.5$).

The idea in MARS is to use such pairs of basis functions, defined for each x_{ij} in the input data matrix $\mathbf{X}$ (lets call this set of functions $\mathcal{C}$).

Example 5. We will deonte the functions of the j^{th} coordinate and x_ij using $h_1(X) = (X_j - x_{ij})_+$ and $h_2(X) = (x_{ij} - X_j)_+$. We use generic notations $h_i(X)$ to think of these basis functions as functions of all coordinates notationally.

The MARS model will be of the form

$$f(X) = \beta_0 + \sum_{m=1}^{M} \beta_m h_m(X),$$

where each h_m would either be one of the basis functions above or a product of such basis functions. This model is built as follows:

- Say we have some functions $\mathcal{M}$ already chosen (say this number is M). At the beginning, assume $\mathcal{M}$ has the function $h_0(X) = 1$ (we find β_0 by minimizing RSS).
- To add a new function we do the following. For every function $h_l \in \mathcal{M}$ and every function[2] in $\mathcal{C}$, we consider adding that derived function:

$$\widehat{\beta}_{M+1}\, h_l(X) \cdot (X_j - t)_+ + \beta_{M+2} h_l(X) \cdot (t - X_j)_+,$$

  which gives the largest decrease in the RSS (all coefficients are re-estimated).
- Do this till we reach a maximum number of functions (a design choice). Then, if needed, delete some of these functions incrementally that show the lowest change in a cross-validation score[3].

Example 6. When $\mathcal{M}$ just has one function $h_0(X) = 1$, we consider adding functions of the form $\widehat{\beta}_1 \cdot 1 \cdot (X_j - t)_+ + \beta_2 \cdot 1 \cdot (t - X_j)_+$. There are Np such functions. We get the best one among these and add it to $\mathcal{M}$. Say it was the one corresponding to x_{72}. Then $\mathcal{M}$ has three functions now: $\{h_0(X), h_1(X) = (X_2 - x_{72})_+, h_2(X) = (x_{72} - X_2)_+\}$. Next stage, we may add functions that may for example be $h_3(X) = (X_1 - x_{51})_+ \cdot (x_{72} - X_2)_+$ etc.

Why these functions? When these functions get multiplied, they they tend to be non-zero in a small region of the input space. Thus they can model local aspects of the regression problem well.

Another interesting restriction for MARS is that higher order products can only come in as functions if the lower order functions are already in $\mathcal{M}$.

3 Support Vector Machines

The big idea with this class of methods is to produce non-linear decision boundaries by constructing a linear boundary in a large transformed version of the input space.

Lets look at the support vector classifier first (we'll bring in the term *machine* a bit later).

As usual, let $\{x_i, g_i\}_{i=1}^n$ be our training data. Let $g_i \in \{-1, 1\}$. Let a hyperplane be defined as $\{x : f(x) = x^T\beta + \beta_0 = 0\}$.

(Linear) SV classifier is defined as: $G(x) = \text{sign}(f(x))$ (basically the hyperplane splits the two classes by a linear decision boundary).

[2] We avoid using the function involving x_{ij} if h_l already involves x_{ij}.

[3] Actually, there is an estimate called the Generalized Cross Validation (GCV) estimate that is used here, very similar to the least squared loss estimate.

Note 7. If the classes are separable, then it is possible to find a $f(x)$ such that $y_i f(x_i) > 0$ for all $i = 1, .., N$. In fact there may be many such classifiers.

Which classifier should we choose: we will choose the classifier that maximizes the geometric *margin* M between the training data for class -1 and 1. The optimization problem written below captures this:

$$\max_{\beta,\beta_0,M} 2M \text{ such that}$$
$$y_i\left(x_i^T \frac{\beta}{\|\beta\|_2} + \frac{\beta_0}{\|\beta\|_2}\right) \geq M, i = 1, ..., N.$$

See the left panel of Figure 5. The band/margin is M units from points on either side and is 2M wide.

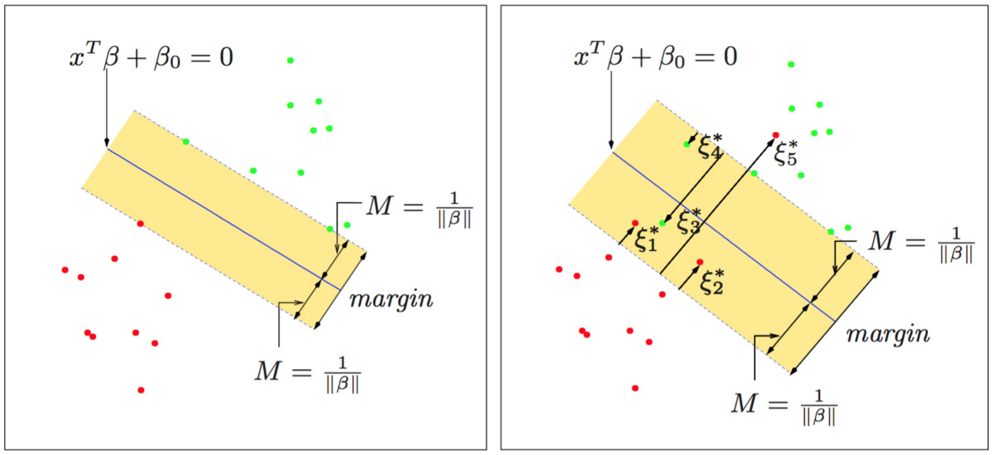


Figure 5: Linear support vector classifier. Left: separable instance. Decision boundary is shown as a solid line. We can choose margin $M = \frac{1}{\|\beta\|_2}$ without loss of generality. Right: the non-separable case, with ξ_i^* labeled points are on the wrong side of the margin by a multiplicative factor of the margin.

We can set $M = \frac{1}{\|\beta\|_2}$ without any loss of generality because margin depends on the scale of the hyperplane normal vector β, which itself does not affect classification accuracy. Thus we get:

$$\max_{\beta,\beta_0} \frac{2}{\|\beta\|_2} \text{ such that}$$
$$y_i(x_i^T\beta + \beta_0) \geq 1, i = 1, ..., N,$$

which can be equivalently written as[4]:

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|_2^2 \text{ such that}$$
$$y_i(x_i^T\beta + \beta_0) \geq 1, i = 1, ..., N.$$

Note 8. The above formulation is a convex optimization problem.

When classes overlap, we allow for some training observations to be on the wrong size of the margin. Lets define *slack* variables ξ_i and modify the above problem to:

$$\min_{\beta,\beta_0,\xi_i} \frac{1}{2}\|\beta\|_2^2 + C\sum_{i=1}^{N}\xi_i \text{ such that}$$
$$y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i, i = 1, ..., N,$$
$$\xi_i \geq 0, i = 1, ..., N.$$

One key observation from the above formulation is that training data observations that are far away from the boundary and in their correct class do not affect the choice of the boundary.

4 Summary

We learned the following things:

- The Random Forests method.
- A model for regression called MARS.
- A model for classification based on the idea of margins called Support Vector Machine.

A Sample Exam Questions

1. Compare and contrast Random Forests with Bagging.
2. What are the key characteristics of a MARS model?
3. What is the idea of margins?

[4]Notice the square!

B Proportional Decrease in Model Error (R^2)

R^2 is defined as:

$$R^2 = \frac{\text{MSE}_0 - \text{MSE}}{\text{MSE}_0},$$

where $\text{MSE}_0 = \text{ave}_{x \in \text{Test}}(\bar{y} - y^{true})^2$ and $.rmMSE = \text{ave}_{x \in \text{Test}}(\widehat{\text{f}}(x) - y^{true})^2$

Lecture 10

IDS575: Statistical Models and Methods
Theja Tulabandhula

Notes derived from the book titled "Elements of Statistical Learning [2nd Ed.] (Chapter 12 and Section 14.2)

We will finish Support Vector Machines and discuss unsupervised learning methods.

1 Support Vector Machines

Continued.

1.1 Computing the Dual Formulation

Since the optimization problem is a convex program, we can write another equivalent optimization problem with a different set of variables, called the *dual*.

The motivation for doing so is that in this dual formulation, data appears in a nice way (inner products) which lets us relax linearity. Giving us the chance to use a trick called the *kernel trick*.

Lets first write the Lagrangian:

$$L_P = \frac{1}{2}\|\beta\|_2^2 + C\sum_{i=1}^N \xi_i - \sum_{i=1}^N \alpha_i(y_i(x_i^T\beta + \beta_0) - 1 + \xi_i) - \sum_{i=1}^N \mu_i\xi_i,$$

where C, α_i, μ_i are the non-negative Lagrange multipliers. Taking the derivative with respect to β, β_0 and ξ_i we get:

$$\beta = \sum_i \alpha_i y_i x_i, \text{ (this is a key equation)}$$

$$0 = \sum_{i=1}^N \alpha_i y_i$$

$$\alpha_i = C - \mu_i.$$

We can now replace the occurrence of β, β_0 and ξ_i in L_P to get the dual problem, which will just be a function of α_i (μ_i is exactly $C - \alpha_i$). The problem is to maximize

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j,$$

subject to $0 \leq \alpha_i \leq C$ and $0 = \sum_{i=1}^{N} \alpha_i y_i$. These α_i are called the dual variables. At its maximum, $L_D = L_P$ (this is called strong duality in the optimization literature).

It turns out[1] that for the optimal $\{\alpha_i\}_{i=1}^{N}$ and β, β_0 and $\{\xi_i\}_{i=1}^{N}$, the following additional conditions hold:

$$\begin{aligned} \alpha_i(y_i(x_i^T\beta + \beta_0) - 1 + \xi_i) &= 0, \text{ and} \\ (C - \alpha_i)\xi_i &= 0. \end{aligned}$$

Since $\beta = \sum_{i=1}^{N} \alpha_i y_i x_i$, we can try interpreting what αs mean. In particular, α_i is non-zero when $y_i(x_i^T\beta + \beta_0) - 1 + \xi_i$ is zero. The observations where this holds are known as *support vectors*[2].

Example 1. An example decision boundary and the support vector boundary is shown for two different values of C in Figure 1. As can be seen margin is larger when C is smaller. In both settings, a linear model may not be appropriate.

1.2 The Kernel Trick and the Support Vector Machine

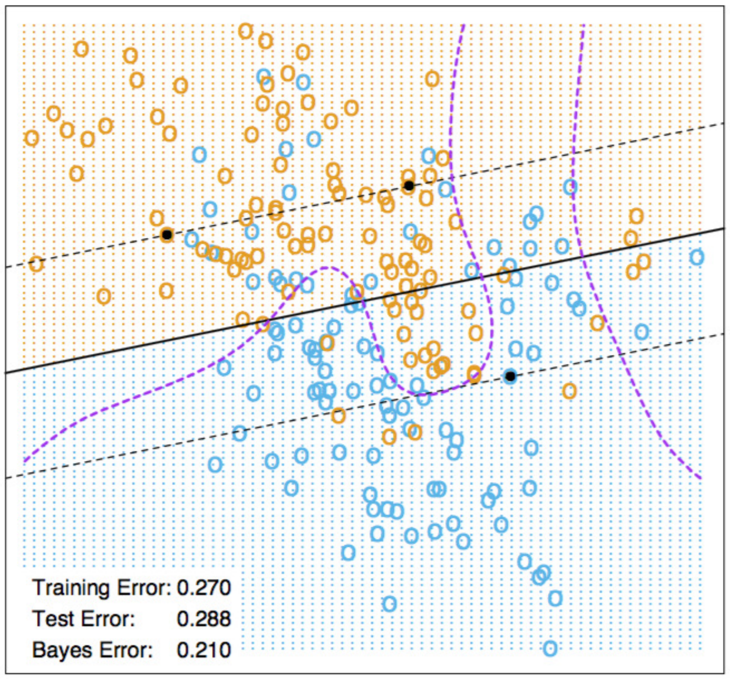
We will now extend the support vector classifier to a more flexible setting where we call the method a *support vector machine*.

The idea here is to work with high-dimensional transformed input spaces in a succinct way using the kernel trick.

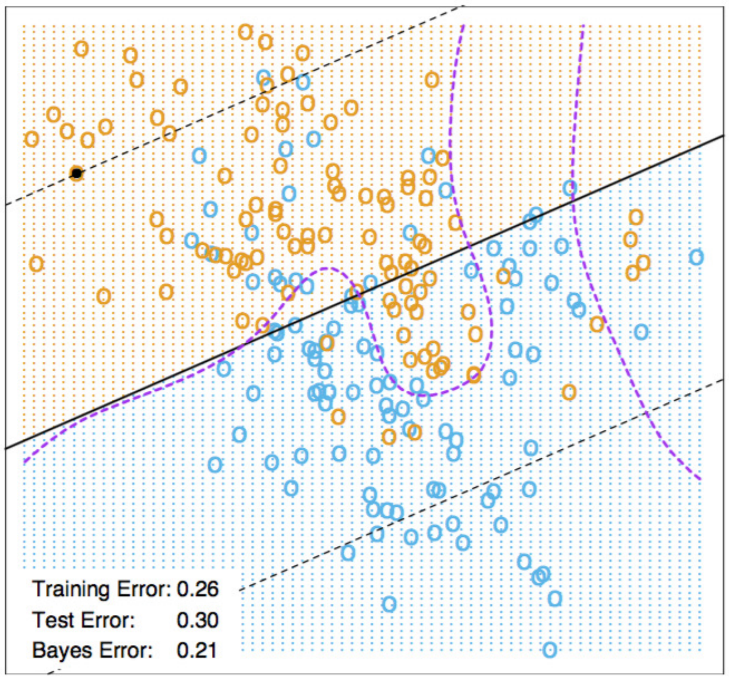
Recall that in the dual formulation, only terms of the form $x_i^T x_j$ appeared. Further, to make a prediction at a point x, we also only need inner products because:

$$f(x) = \beta^T x = \sum_{i=1}^{N} \alpha_i y_i x_i^T x + \beta_0.$$

[1]These are part of what are known as the KKT conditions.
[2]Hence the name!



$C = 10000$



$C = 0.01$

Figure 1: Linear support vector classifier on a 2-dimensional dataset.

We replace these inner products with kernel functions $k(x, z)$. These functions implicitly compute inner products in arbitrary transformed spaces.

Example 2. The following are some popular choices for kernels:

- A d^{th}-degree polynomial $k(x, z) = (1 + x^T z)^d$, and
- Radial basis $k(x, z) = \exp(-\gamma \|x - z\|_2^2)$.

For the same 2-dimensional data as before, the decision boundaries and support vector boundaries are shown in Figure 2. These seem to be a better fit.

Note 1. Beyond two classes: there is no really good answer to extending SVMs to setting where the number of classes $K > 2$. One straightforward way is to build multiple two-class SVMs and use them simultaneously.

1.3 Support Vector Regression

There is a *penalization* view of SVM that alllows us to reconcile it with other methods we have seen earlier. This view is as follows:

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|_2^2 + C\sum_{i=1}^{N}[1 - y_i(\beta^T x_i + \beta_0)]_+,$$

where $[\cdot]_+ = max(0, \cdot)$. This formulation looks like a penalty plus a loss function. In particular, the loss function above is known as the *hinge loss*.

If we have a regression setting, the y_i is now going to be a real number. Then, we can suitably change the loss function above to get a version of support vector regression method:

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|_2^2 + C\sum_{i=1}^{N} L(y_i, \beta^T x_i + \beta_0),$$

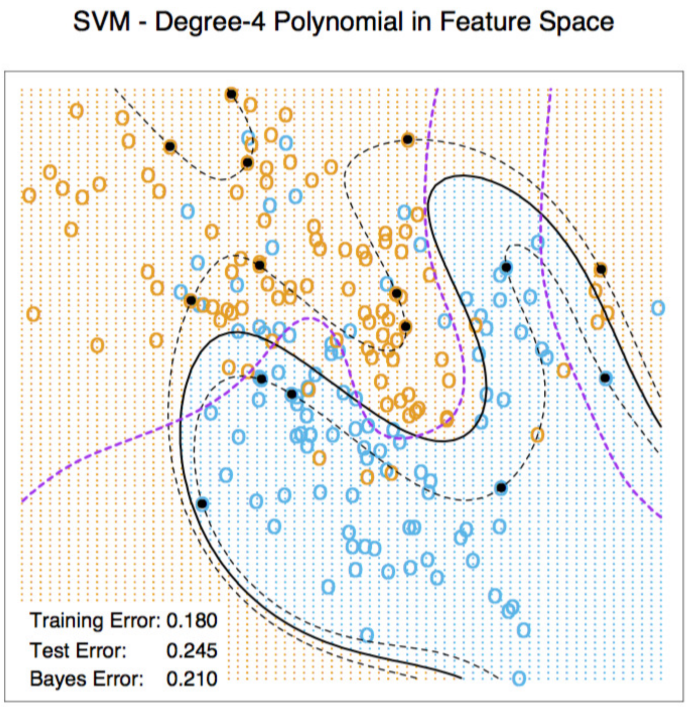
where for example, $L(y, w) = 0$ if $|y - w| < \epsilon$ and $|y - w| - \epsilon$ otherwise. This is called the ϵ-insensitive loss[3].

2 Unsupervised Learning with Association Rules

As the name implies, unsupervised learning is when you don't have an explicit notion of a target or dependent variable. So what is the task? If you recall supervised learning, it was all about the conditional distribution of Y/G given X (or functions of it). Here, assuming data is denoted by X, we are interested in inferring properties of $P(X)$ (the distribution of X).

What properties are we interested in learning? Some examples are:

[3] ϵ is a parameter of the loss function here.



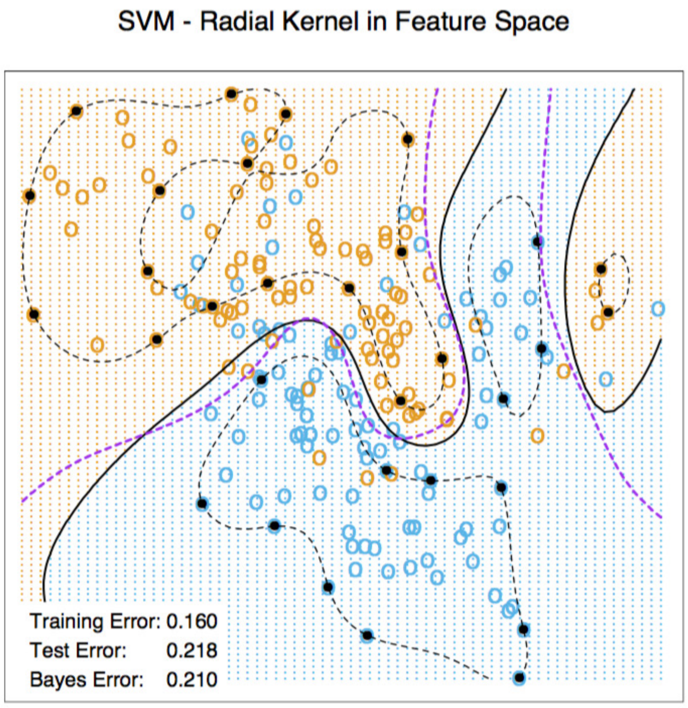


Figure 2: Support vector machine with two different kernels on a 2-dimensional dataset.

- Estimating a mixture of Gaussians model

- Descriptive statistics that characterize $P(X)$.
  - For example, methods such as principal components, multidimensional scaling, self-organizing maps etc (which we will look at later) focus on finding association between variables and reduce dimensionality.
  - Another example is cluster analysis, which finds regions in space that contain modes of $P(X)$.
  - A final example is that of association rules. These try to construct descriptions/rules that describe regions of high density when the data is all binary.

There is no measure of success for unsupervised learning similar to test error notion that we studied for supervised learning. Algorithms are motivated heuristically and the results are also judged heuristically.

2.1 Association Rules

The goal of this method is to find joint values of *subsets* of $X = (X_1, ..., X_p)$ that appear most frequently in *binary* valued datasets.

Example 3. Consider sales transaction records at a retail store. Given lots of observations, you can find out which products are likely to be purchased together. Here p would be the number of all products sold in the store, which can be very large. Finding such patterns can help manage inventory, promotions, customer segmentation and a variety of business tasks.

Note 2. We care about those values in subsets of X which have high probability mass.

Note 3. This is a difficult problem when p is large, because there are exponentially many such subsets. And for each subset, there are exponential number of values X can take.

Let $s_j \subseteq \{0,1\}$ be a subset corresponding to the j^{th} input variable. We want to find $(s_1, ..., s_p)$ such that

$$Pr[\cap_{j=1}^{p}(X_j \in s_j)]$$

is large.

Note 4. If we relax the binary assumption, the above definition and goal still stands. One can always convert data from quantitative and categorical to binary through the notion of *dummy* variables.

Example 4. if each X_j takes values in S_j, then the total number of dummy variabls is $\sum_{j=1}^{p} |S_j|$.

Coming back to the binary setting, let $\mathcal{K}$ be a subset of $\{1, ..., p\}$. Then the probability mass definition above can be rewritten as:

$$Pr[\cap_{j=1}^{p}(X_j \in s_j)] = Pr[\prod_{k \in \mathcal{K}} X_k = 1].$$

The set $\mathcal{K}$ is called an itemset and $|\,\mathcal{K}\,|$ is its size.

Its estimate $\frac{1}{N}\sum_{i=1}^{N}\prod_{k\in\mathcal{K}} x_{ik}$ is called the support of $\mathcal{K}$ and is denoted as $T(\mathcal{K})$.

We want to only identify those itemsets whose support is at least $t > 0$, and this collection can be written using set notation as: $\{\mathcal{K}_j\,|T(\mathcal{K}_j) \geq t\}$.

2.2 Apriori Algorithm

This is one of the first algorithms for finding the collection $\{\mathcal{K}_j\,|T(\mathcal{K}_j) \geq t\}$.

The key idea in the algorithm is that for a set $\mathcal{K} \leq \mathcal{K}$ then $T(\mathcal{K}) \geq T(\mathcal{K})$.

The algorithm works in rounds. In each round, it reads through the dataset once.

- In the first round, compute support of single-item sets. Discard those with $T(\mathcal{K}) < t$.
- In the second, it computes support of two-item sets that can be formed from pairs of single-items left from previous round.
- In round m, an itemset of size m is considered only if *all* its $\binom{m}{m-1}$ subsets have enough support in the previous round and the remaining single-item was left over after the first round.

Till now, we have only mined what are called *frequent itemsets*. These can be turned into *rules* by breaking the itemset into two parts: an antecedent and a consequent and checking for additional desired statistics (e.g., confidence defined below).

An association rule is of the form $p \to q$ where p and q are sets such that $p \cup q = \mathcal{K}$.

- Support of a rule is $T(p \cup q)$.
- Confidence $c(p \to q)$ of a rule is defined as $T(p \cup q)/T(p)$, which can be viewed as an estimate of $Pr(q|p)$, where for any set $\mathcal{K}$, $Pr(\mathcal{K}) = Pr(\prod_{k\in\mathcal{K}} X_k = 1)$.

Note 5. We would like to have rules that have high confidence and support. That is $\{p \to q : T(p \cup q) > t, c(p \to q) > c\}$ for some thresholds t and c.

Some of the issues with association rule mining include:

- Only for binary data.
- Computationally expensive.
- Rules with high confidence but low support will not be discovered. This may be important in some situations.

Example 5. An example rule: {'number in household = 1','number of children = 0'} → {'language in home = English'}.

3 Summary

We learned the following things:

- The kernel trick for Support Vector Machine.
- Statistical basis for association rules.

A Sample Exam Questions

1. What are support vectors? Express the prediction model in terms of the support vectors.
2. What is the kernel trick?
3. Describe the pros and cons of association rules? What is their time complexity?

Lecture 11

IDS575: Statistical Models and Methods
Theja Tulabandhula

Notes derived from the book 'Elements of Statistical Learning' [2nd Ed.] (Sections 14.3,14.5-14.7)

We will discuss two unsupervised learning methods: clustering and principal components.

1 Unsupervised Learning with Clustering

The idea of clustering is to segment observations into separate sets such that the observations in the same set are closer to each other according to a metric/measure compared to observations in other sets.

Note 1. We do not necessarily have to work with observations. Even information about how similar each observation is to other observations is enough to cluster them.

Thus, a notion of *dissimilarity* (or similarity) is key to cluster analysis. This is typically defined by an expert using domain knowledge, and is not data driven.

1.1 Defining Dissimilarity

Dissimilarity can be represented using a $N \times N$ matrix $\mathbf{D}$, where N is the number of observations (e.g., x_i for $i = 1, ..., N$). Each entry of this matrix $d_{ii'}$ is the similarity between observation i and observation i'.

Note 2. If we have the similarity value between two observations, we can convert it to a value dissimilarity by using a function such as $1 - z$. For example, if similarity is $0 \leq s \leq 1$, then dissimilarity can be defined as $1 - s$.

One way to define dissimilarity is to define it coordinate wise. That is, dissimilarity

$$D(x_i, x_i') = d_{ii'} = \sum_{j=1}^{p} d_j(x_{ij}, x_{i'j}),$$

where $d_j(x_{ij}, x_{i'j})$ is the coordinate wise dissimilarity[1].

Example 1. For example, $d_j(x_{ij}, x_{i'j}) = (x_{ij} - x_{i'j})^2$.

Example 2. Define correlation between *standardized* observations x_i and $x_{i'}$ as:

$$\rho = \frac{\sum_{j=1}^p x_{ij} x_{i'j}}{\sqrt{\sum_{j=1}^p x_{ij}^2 \sum_{j=1}^p x_{i'j}^2}}.$$

This correlation (similarity/dissimilarity) information can be used to cluster. In fact, this is equivalent to defining clustering using squared distance because:

$$\sum_{j=1}^p (x_{ij} - x_{i'j})^2 \propto 2(1 - \rho).$$

Example 3. For categorical variables, you can define $d_j(x_{ij}, x_{i'j})$ using exogenous information about how one category is different from another.

Specifying $d_{ii'}$ is generally very important compared to the clustering method of choice.

1.2 Algorithms for Clustering

There are two main ways to cluster:

- optimization formulations (there is no statistical or probability distribution)
- mixture models (like Gaussian mixtures)

Since we have looked at mixture models before, lets investigate the optimization formulation route. Here, we want to find the cluster label of each observation in our dataset given the dissimilarities matrix $\mathbf{D}$. This cluster labeling should minimize some loss.

Example 4. An example loss function is $W(C) = \frac{1}{2} \sum_{k=1}^K \sum_{C(i)=k} \sum_{C(i')=k} d(x_i, x_{i'})$, where we are optimizing for cluster labels given by $C(i)$ for $i = 1, ..., N$. This loss function is high if observations that are not close in similarity are assigned to the same cluster.

Note 3. Optimizing the cluster assignments over a loss function such as the above is typically a difficult problem. Enumeration is not possible for even small datasets. Hence iterative greedy methods are our only computationally tractable choice.

[1] One can question why dissimilarity in each attribute should be added equally. There is no right answer here.

In particular, lets look at the *k-means* clustering method. Assume that all X_j are quantitative. Let $d_{ii'} = \|x_i - x_{i'}\|_2^2$. Then, $W(C)$ can be written as:

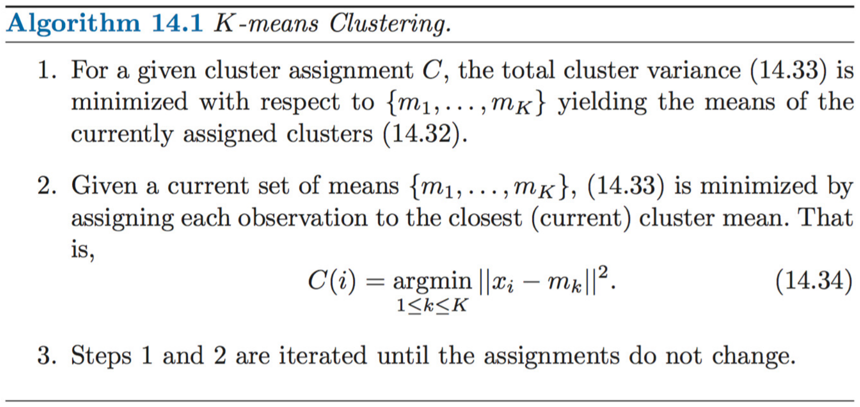
$$W(C) = \sum_{k=1}^{K} N_k \sum_{C(i)=k} \|x_i - \bar{x}_k\|_2^2.$$

Here, $\bar{x}_k$ is the mean of the k^{th} cluster and N_k is the number of observations in the k^{th} cluster.

By noting that $\bar{x}_S = \arg\min_m \sum_{i \in S} \|x_i - m\|_2^2$, we can solve for cluster assignments by solving the following problem:

$$\min_{C, \{m_k\}_1^K} \sum_{k=1}^{K} N_k \sum_{C(i)=k} \|x_i - m_k\|_2^2.$$

The above can be minimized by alternative over C and the $\{m_k\}$ variables, giving us the Algorithm shown in Figure 1.



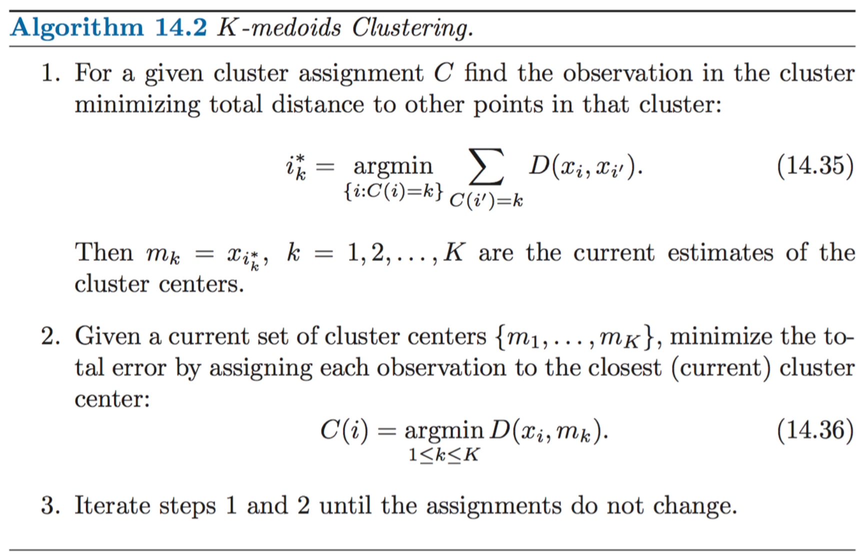
Algorithm 14.1 *K-means Clustering.*

1. For a given cluster assignment C, the total cluster variance (14.33) is minimized with respect to $\{m_1, \ldots, m_K\}$ yielding the means of the currently assigned clusters (14.32).
2. Given a current set of means $\{m_1, \ldots, m_K\}$, (14.33) is minimized by assigning each observation to the closest (current) cluster mean. That is,
$$C(i) = \underset{1 \le k \le K}{\text{argmin}} \, ||x_i - m_k||^2. \qquad (14.34)$$
3. Steps 1 and 2 are iterated until the assignments do not change.

Figure 1: The k-means clustering method.

The EM algorithm for Gaussian mixture model is closely related to the k-means algorithm. The E-step finds soft assignments (recall the $q(|x)$ conditional probability values) of observations to mixture components. The M-step finds the mixture parameters given the soft assignments. If we assume K mixture components with covariances $\sigma^2\mathbf{I}$, then given mixture soft assignments, the likelihood of data is proportional to Euclidean distance to the mixture center. This is very similar to the k-means method above.

A variation of k-means, called k-medoids, extends it to non-Euclidean dissimilarity measures. The Algorithm is given in Figure 2. Other than replacing the squared dissimilarity measure, an additional restriction imposed here is that each cluster center has to be one of the observation itself.



Algorithm 14.2 *K-medoids Clustering.*

1. For a given cluster assignment C find the observation in the cluster minimizing total distance to other points in that cluster:

$$i_k^* = \underset{\{i:C(i)=k\}}{\operatorname{argmin}} \sum_{C(i')=k} D(x_i, x_{i'}). \tag{14.35}$$

Then $m_k = x_{i_k^*}$, $k = 1, 2, \ldots, K$ are the current estimates of the cluster centers.

2. Given a current set of cluster centers $\{m_1, \ldots, m_K\}$, minimize the total error by assigning each observation to the closest (current) cluster center:

$$C(i) = \underset{1\le k\le K}{\operatorname{argmin}} D(x_i, m_k). \tag{14.36}$$

3. Iterate steps 1 and 2 until the assignments do not change.

Figure 2: The k-medoid clustering method.

Note 4. The choice of K: Larger K implies a lower objective value. Hence, as a heuristic, we can search for that K for which the decrease in the objective drastically changes from large values to small values. This is known as identifying a *kink* in the objective versus K plot.

Note 5. One issue with k-means and k-medoids is that empirically, the clusters may change drastically as K is changed. This does not happen for a different clustering method called *hierarchical clustering*.

Hierarchical clustering requires user to specify a dissimilarity measure between groups of observations.

It produces a hierarchical representation of clusters, where clusters at one level are obtained by splitting the clusters one level above (*divisive*) or merging the clusters from one level below (*agglomerative*).

Such cluster representations can be represented as *trees* with nodes representing clusters. Sometimes these hierarchical clusters have a monotonicity property: dissimilarity between merged clusters is monotonically increasing with the level of the merger. In terms of plot, the height of a node can be proportional to the dissimilarity of its children nodes. This is called a *dendrogram*.

Note 6. Hierarchical clustering will impose a hierarchical representation whether such pattern exists in the dataset or not.

For agglomerative clustering, the following are some measures for dissimilarity between groups:

- Single Linkage (SL): $d_{SL}(G, H) = \min_{i \in G, i' \in H} d_{ii'}$.
- Complete Linkage (CL): $d_{CL}(G, H) = \max_{i \in G, i' \in H} d_{ii'}$.
- Group Average (GA): $d_{GA}(G, H) = \frac{1}{N_G N_H} \sum_{i \in G} \sum_{i' \in H} d_{ii'}$.

Note 7. SL has a tendency to combine observations linked by a series of close intermediate observations, and this phenomena is called *chaining*.

2 Principal Components

Recall that we used principal components to understand ridge regression. So, what are these?

Intuitively, they are projections (linear approximations) of data ($x_i \in \mathbb{R}^p$ for $i = 1, .., N$) that are uncorrelated with each other and capture data variance in order.

For simplicity subtract from each observation a constant vector defined as $\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$.

Say we want a rank $q < p$ linear approximation to this data. This means, that we want for each vector x_i an equivalent q-dimensional vector λ_i such that a linear transformation of x_i is close to λ_i for all i. Let $\mathbf{V}_q$ be a $p \times q$-dimensional matrix with q orthogonal unit column vectors.

It turns out that $\mathbf{V}_q \mathbf{V}_q^T$ is called a projection matrix that maps x_i to a rank q reconstruction of itself.

The search for $\mathbf{V}_q$ ($\lambda_i = \mathbf{V}_q^T x_i$) is done by minimizing reconstruction error:

$$\min_{V_q} \sum_{i=1}^{N} \|x_i - \mathbf{V}_q \mathbf{V}_q^T x_i\|_2^2.$$

The matrix $\mathbf{V}_q \mathbf{V}_q^T$ projects x_i onto the columnspace of $\mathbf{V}_q$, which is of rank q.

The solution to the above problem is given by taking the SVD of $\mathbf{X}$ ($N \times p$-dimensional), i.e., $X = \mathbf{U}\,\mathbf{D}\,\mathbf{V}^T$[2] and set $\mathbf{V}_q$ to be equal to the first q columns of $\mathbf{V}$.

[2]Assume that the entries of the diagonal matrix D are ordered $|d_1| \geq |d_2| \geq ....$

The columns of $\mathbf{U}\,\mathbf{D}$ are called the principal components of $\mathbf{X}$.

The N λ_is are given by the first q principal components (the first q columns of $\mathbf{U}\,\mathbf{D}$).

Example 5. One and two dimensional principal components are shown in Figures 3 and 4 respectively.

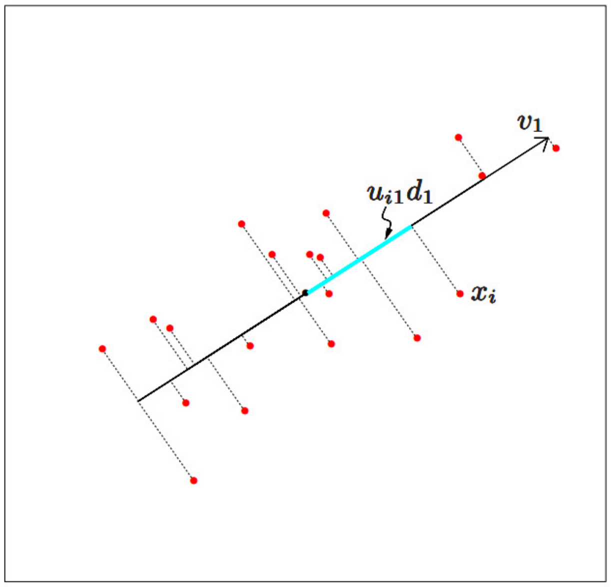


Figure 3: The 1-dimensional principal components of an example 2-dimensional dataset.

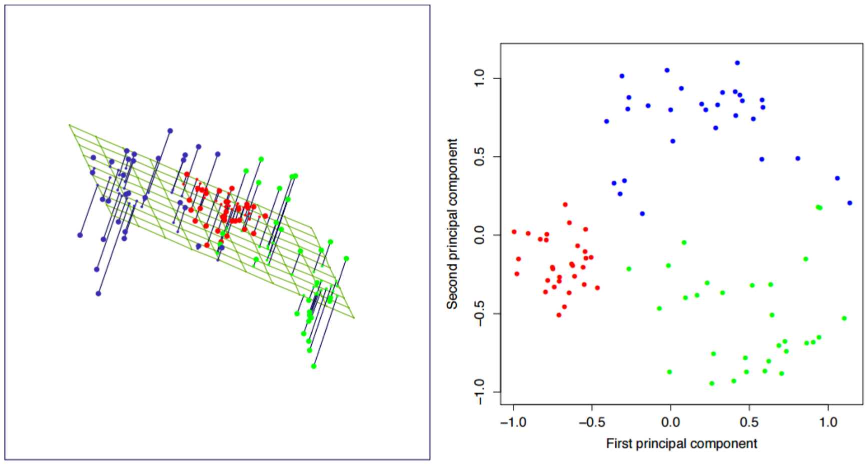


Figure 4: The 2-dimensional principal components (right) of an example dataset.

There is an interesting property that principal components capture with regard to *variance* in the data. For example, the N-dimensional vector (linear combination) $\mathbf{X}\,v_1$ has the highest variance among all linear combinations of columns of $\mathbf{X}$. Similarly $\mathbf{X}\,v_2$ has the next highest variance.

Example 6. They capture low dimensional aspects of the data well. For example, consider the 16-dimensional images of the number '3' in Figure 5.

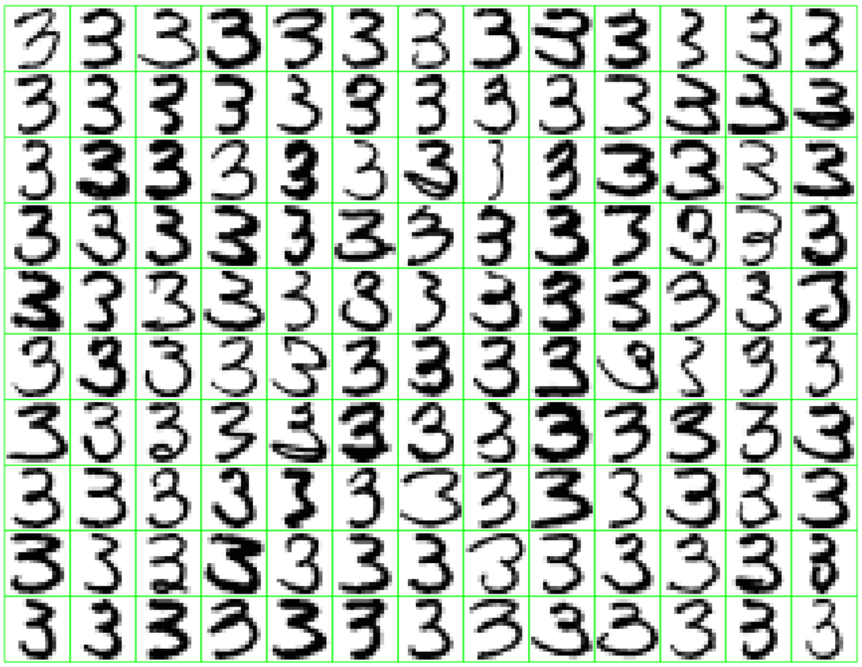


Figure 5: The handwritten '3' dataset.

Note 8. *Kernel PCA* computes principal components with the kernel matrix and is more flexible than using the gram matrix ($\mathbf{X}^T\,\mathbf{X} = \mathbf{U}\,\mathbf{D}^2\,\mathbf{U}^T$).

2.1 Spectral Clustering

The idea here to cluster, taking into account a different metric to define dissimilarity/similarity.

Note 9. The name *spectral* represents the use of eigenvalues of matrices (such as those based on the dissimilarity matrix).

Example 7. For example, it is difficult to cluster the points shown in Figure **??** (top left) using k-means.

Say, we have N observations and let $d_{ii'}$ be the Euclidean distance between observation $x_i \in \mathbb{R}^p$ and observation $x_{i'}$. We will convert the dissimilarity to a similarity metric as: $s_{ii'} = \exp(-d_{ii'}/c)$ for some $c > 0$.

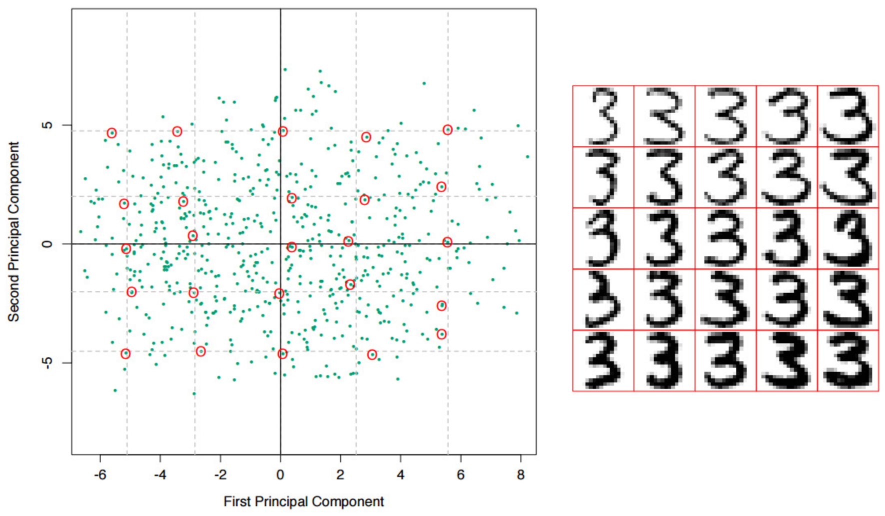


Figure 6: First two principal components and representative observations (corresponding to red circles on the left plot). The first component seems to encourage longer lower tail and the second encourages character thickness.

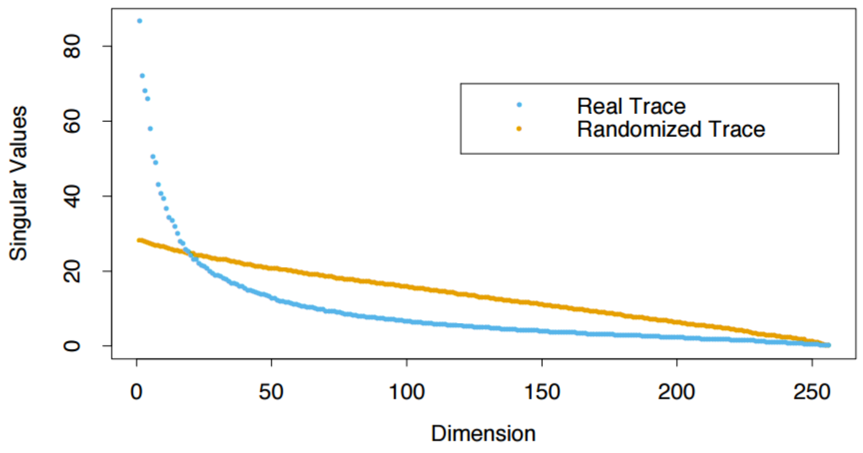


Figure 7: Evidence that principal components capture correlation. If you randomly scramble ech column of $\mathbf{X}$, less variance is captured in the lower principal directions.

We create a graph with observation indices as nodes and edge weights $s_{ii'}$. With this, we phrase a graph partitioning problem: break the graph into pieces such that edges between different clusters have low weight.

We can optionally do some preprocessing on the similarity matrix (such as setting low values to 0). Let this processed matrix be $\mathbf{W}$ (we will call it the *weighted adjacency matrix*). Spectral clustering has the following steps:

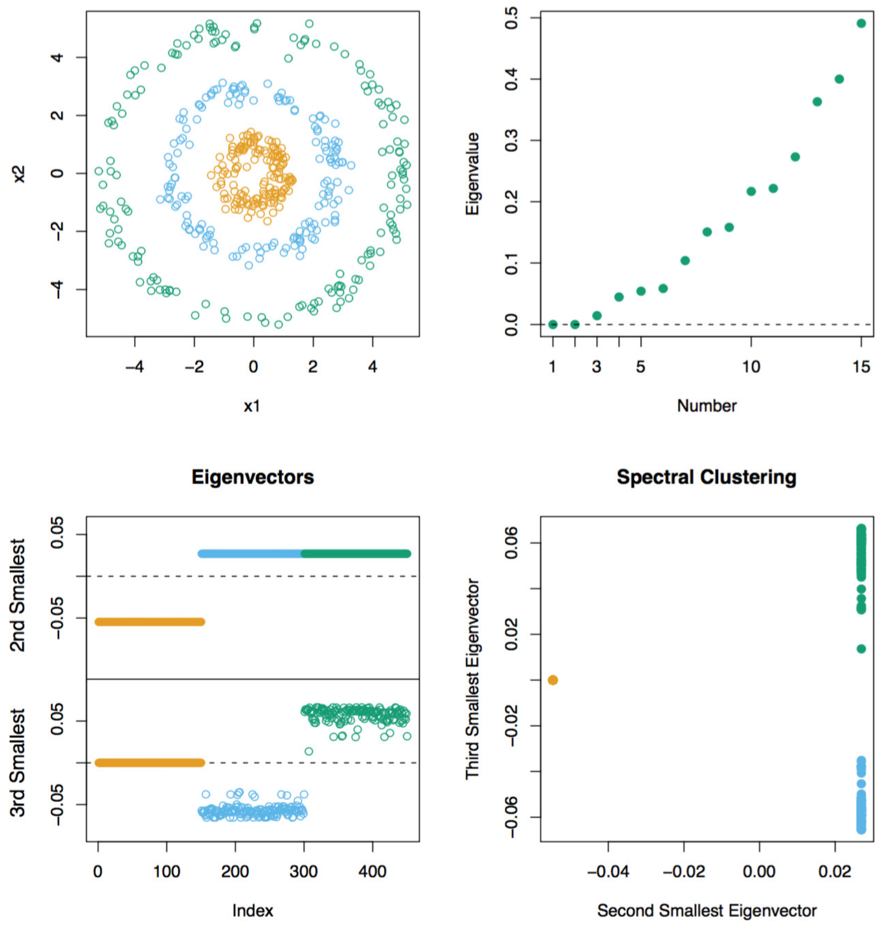


Figure 8: Spectral clustering example containing three concentric circular clusters with 150 points each.

- Compute $\mathbf{G}$ as a diagonal matrix with weighted degree value $g_i = \sum_{i'} w_{ii'}$ for each i.
- Compute the *un-normalized graph Laplacian* as $\mathbf{L} = \mathbf{G} - \mathbf{W}$.
- Find eigenvectors the $\mathbf{Z}$ ($N \times m$ dimensional) corresponding to the m smallest eigenvalues excluding 0.
- Use k-means to cluster the rows of $\mathbf{Z}$ to get a clustering of the original dataset.

Example 8. See Figure 8 for a visualization of how the smallest eigenvectors capture cluster information. The nodes of the graph were only connected to 10 nearest neighbors.

Why does this work? For any vector $\mathbf{f}$, $\mathbf{f}\,\mathbf{L}\,\mathbf{f} = \sum^N g_i f_i^2 - \sum_i^N \sum_{i'}^N f_i f_{i'} w_{ii'} = \frac{1}{2}\sum_i^N \sum_{i'}^N w_{ii'}(f_i - f_{i'})^2$. This takes small values for those vectors whose coordinates have close values if $w_{ii'}$ is large. Eigenvectors can be defined using this product as well. It turns out that if there is a single connected component, then $\mathbf{1}$ vector corresponds to the unique 0 eigenvalue. If there are more than one connected components, then there will be multiple such *indicator* eigenvectors corresponding to the 0 eigenvalue. In reality, this leads to the heuristic of finding smallest eigenvalues.

3 Summary

We learned the following things:

- Clustering methods: k-means and k-medoid.
- Principal components.

A Sample Exam Questions

1. What is the relation between k-means clustering and the EM algorithm for Gaussian mixture models?
2. Why do we cluster the left singular vectors corresponding to large singular values (in principal components) compared to clustering eigenvectors corresponding to small eigenvalues (in spectral clustering)?

Lecture 12

IDS575: Statistical Models and Methods
Theja Tulabandhula

1 Course Evaluation

- You should have received an email with subject line: "UIC Student Evaluation of Teaching [Subject Code and Number] [Instructor Name] [Semester, Year]."
- Some reasons to submit feedback are:
  - Positive feedback helps keep industry relevant content.
  - Helps improve the course offering to meet the needs of the students in the future and strengthen the degree programs at UIC.
- Pleas do submit before the deadline!

Notes derived from the course material titled "Time Series" (2010) and "Applied Time Series Analysis" (2010)

We will look at a brief overview of time series modeling.

2 Time Series and Supervised Learning

The idea of time series is to capture dependence across observations. So far, we have not modeled any dependency across examples (say (x_i, y_i) and (x_j, y_j)) in a dataset. We will now explicitly model this.

Example 1. One way to measure dependency is via correlation.

Note 1. Because we want to model dependency across observations, the ordering of observations becomes important.

Time series models are applicable in many realistic settings, including:

- Demographic projections
- Financial analysis
- Dynamics of viral media (e.g., view-count of a youtube video)
- Speech and video
- ...

While supervised learning is a broad area, the emphasis is typically on predictive ability and not on modeling dependence across observations. One can certainly model dependencies in a supervised learning setting (say by casting an appropriate maximum likelihood problem). Although, the collection of techniques that explicitly do this have been historically put under the time series modeling umbrella.

Note 2. Also, while the concept of input and output variables is important in supervised learning, it is less important, or even ignored, in time series modeling.

Example 2. Time series of lap times of a Nascar driver is shown in Figure 1.

The various phenomena that we care about with time series modeling include:

- finding trends and seasonality,
- dependence across time, and
- properties of random fluctuations around trend and seasonality patterns.

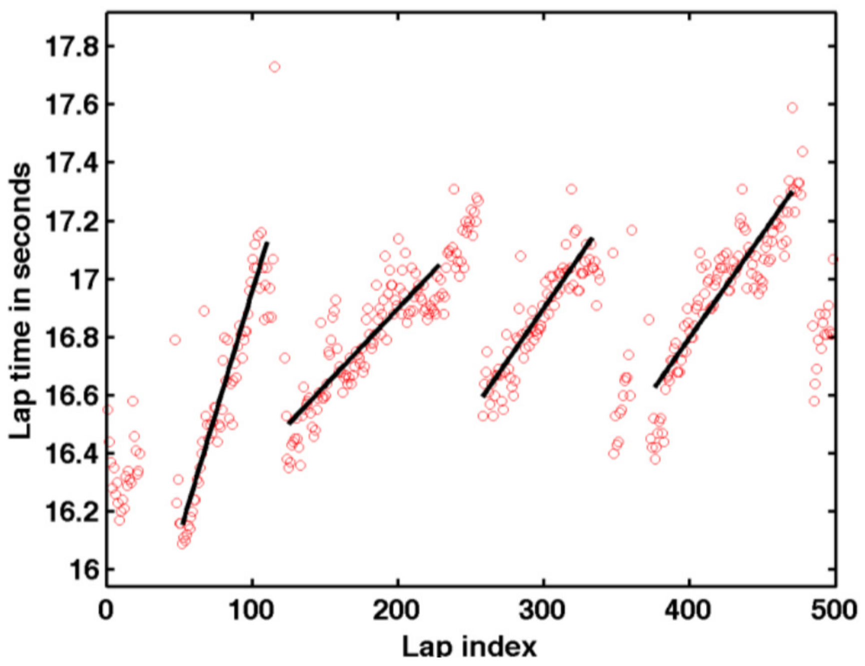


Figure 1: An example of a time series realization.

2.1 Moments vs Distributions

We want to model dependencies across observations. Lets denote random variable W_1 for the first observation, W_2 for the second observation and so on. Say we have N observations.

Note 3. A sequence of random variables is also called a *stochastic process*.

We could define a parametric joint distribution of these random variables as:

$$P_\theta(W_1, .., W_N).$$

The issue with this is that it may be very difficult to specify such a parametric model. A slightly simpler modeling approach is to only focus on *moments*, which are functions such as $E[W_i]$ and $cov(W_{t+\tau}, W_t)$ etc.

A sequence of random variables is called weakly stationary if the following conditions hold:

- $E[W_t] = \mu$ is a constant.
- $cov(W_{t+\tau}, W_t) = \gamma_\tau < \infty$ is a function that only depends on the relative *lag* and not on the absolute index t.

Note 4. These are okay assumptions to make once the predictability using features has already been accounted for. In other words, roughly speaking, first do supervised learning and then apply time series modeling on the residuals.

Note 5. Check: (a) $var(W_t) = \gamma_0$, and (b) $\gamma_{-\tau} = \gamma_\tau$.

The function γ_τ is called the autocovariance function. The function $\rho_\tau = \gamma_\tau/\gamma_0$ is called the autocorrelation function.

Note 6. A sequence $W_1, ..., W_N$ is called white noise if $E[W_t] = 0$ and $E[W_t^2] = \sigma^2$. If the W_t are independent and identically distributed, then they are sometimes called *i.i.d noise*.

Example 3. An example of a non-stationary time series is as follows: Let W_t be a white noise sequence. Let $S_0 = 0$ and $S_t = W_1 + W_2 + \ldots + W_t$. Then the sequence $S_0, S_1, ...$ is called a random walk. We can check that $cov(S_t, S_{t+\tau}) = t\sigma^2$.

Why stationarity? Well, it allows us to average. We are only observing one realization of each random variable. Still, we are allowed to average the values of these realizations across (time).

Example 4. Sample autocorrelation and sample autocovariance functions can be estimated similar to estimating sample mean, which we have seen before.

How do we estimate seasonality and trend effects? Well, there are approaches that filter out "noise" such that trends or seasonality patterns remain. We will skip the details of these techniques for now.

3 The Autoregressive Moving Average (ARMA) Model

First we describe two linear time series models: the autoregressive model and the moving average model.

By linearity, we mean that the random variable at time t is linearly related to other random variables.

3.1 Autoregressive (AR) Model

An autoregressive (AR) model assumes that the random variable at time t is related to the random variables before time t. For instance, an AR(1) model is written as follows:

$$W_t = \phi W_{t-1} + \epsilon_t,$$

where ϵ_t is a white noise random variable with $var(\epsilon_t) = \sigma^2$.

If ϵ_t is independent of $W_{t-1}, W_{t-2}, ...,$ then the sequence is Markovian. That is, $P(W_t|W_{t-1}, W_{t-2}, ...) = P(W_t|W_{t-1})$.

If we repeatedly *back-substitute*, we get the following expression:

$$W_t = \epsilon_t + \phi\epsilon_{t-1} + \phi^2\epsilon_{t-2} + \phi^{t-1}\epsilon_1 + \phi^t W_0.$$

For this sum to be finite for any realization (so that stationarity holds), we need $|\phi| < 1$.

Example 5. The mean of the t^{th} random variable in an AR(1) model is:

$$E[W_t] = \sum_{j=0}^{t} \phi^j E[\epsilon_{t-j}] = 0.$$

The autocovariance function is:

$$\begin{aligned}
\gamma_\tau &= cov(W_{t+\tau}, W_t) \\
&= E[\sum_{j=0}^{t+\tau} \phi^k \epsilon_{t+\tau-j} \sum_{k=0}^{t} \epsilon_{t-k}] \\
&\approx \sigma^2 \sum_{k=0}^{\infty} \phi^{k+\tau}\phi^k \\
&= \sigma^2\phi^\tau \sum_{k=0}^{\infty} \phi^{2k} = \frac{\sigma^2\phi^h}{1-\phi^2}.
\end{aligned}$$

3.2 Moving Average (MA) Model

Say $W_t = c_t\epsilon_t + c_{t-1}\epsilon_{t-1} + ...$, where ϵ_t is white noise. This is called a moving average (MA) model. Then γ_τ for the W_t time series is given by:

$$\gamma_\tau = \sigma^2 \sum_t c_t c_{t+\tau}.$$

More specifically, a MA(q) model would be:

$$W_t = \theta_0\epsilon_t + \theta_1\epsilon_{t-1} + ... + \theta_q\epsilon_{t-q}.$$

The mean of MA(q) would be $E[W_t] = 0$.
The covariance would be:

$$\gamma_\tau = \sigma^2 \sum_{t=0}^{q-\tau} \theta_t\theta_{t+\tau}; \;\; 0 \le \tau \le q.$$

3.3 Combining AR and MA

We can combine to two models above to get the following model:

$$W_t - \phi_1 W_{t-1} - ... - \phi_p W_{t-p} = \epsilon_t + \theta_1\epsilon_{t-1} + ... + \theta_q\epsilon_{t-q},$$

where ϵ_t represents white noise.

Why do we combine? Because, MA and AR(1) capture certain covariances, but not all. For example, for AR(1), the covariances are all determined using a single parameter. For MA(q), the number of parameters is q.

In terms of notation, we can compactly write this as:

$$\phi(B)W_t = \theta(B)\epsilon_t,$$

where $\phi(B)W_t = W_t - \phi_1 W_{t-1} - ... - \phi_p W_{t-p}$, similarly for $\theta(B)\epsilon_t$.

Note 7. B is a shorthand symbol for the backshift operation. For example, $BW_t = W_{t-1}$ and $B^k W_t = W_{t-k}$.

Not all coefficients can go into the above model equation. In particular, we will need the following conditions:

- The roots of the polynomial $\theta(B)$ are different from the roots of the polynomial $\phi(B)$ (if this was not true, terms would cancel out on both sides).
- $\phi(B) \neq 0$ for all $|B| \leq 1$ (allowing B to be a complex number). This ensures a property called causality (W_t only depends on variables before time t)
- $\theta(B) \neq 0$ for all $|B| \leq 1$ (allowing B to be a complex number). This ensures a property called identifiability (otherwise there are multiple θ coefficients that correspond to the same autocovariance values.)
- Ultimately, we are writing $W_t = \psi(B)\epsilon_t$, where $\psi(B) = \theta(B)/\phi(B)$. We need to ensure that this polynomial is nice (it will have infinite terms, but hopefully the coefficients are small).
- We will also assume for simplicity that $\phi(0) = \theta(0) = 1$.

3.4 Estimation

How does one estimate all these parameters? How to determine p and q? The estimation of both these is based on covariance or correlation estimates.

Example 6. For example for MA(q), if you plot the covariance values for different τ values, you will observe that for $\tau > q$, the covariance will be nearly zero.

Example 7. For AR(p), there is another quantity called the partial autocorrelation function, that can be plotted to determine p. This is defined as:

$$\begin{aligned}\phi_{11} &= corr(W_1, W_0)\\ \phi_{\tau\tau} &= corr(W_\tau - W_\tau^{\tau-1}, W_0 - W_0^{\tau-1}),\ \ \tau \geq 2,\end{aligned}$$

where $W_\tau^{\tau-1}$ is the regression of W_τ on $(W_{\tau-1}, ..., W_1)$ and $W_0^{\tau-1}$ is the regression of W_0 on $(W_1, ..., W_{\tau-1})$. Seems complicated, but it just ensures that beyond p, the plot of partial autocorrelation dies down.

Example 8. Estimation of the parameters of, say, an AR(p) model can be done using a specific *method of moments* technique called the system of *Yule-Walker equations.* In general, such estimation can be done using the method of maximum likelihood as well.

Note 8. In the literature, you may come across a methodology called the *Box-Jenkins* approach, which describes the steps relating to estimating p and q and then estimating the model parameters and then testing their validity.

Note 9. There is another model called ARIMA which just means that you may have to *difference* the time series a few times before fitting an ARMA model. Differencing means subtracting successive values.

4 Summary

We learned the following things:

- Relation between time series models and supervised learning.
- The Autoregressive Moving Average (ARMA) model.
- There are many time series specific techniques beyond what we saw here. For instance, time domain and frequency domain methods, methods based on distributional assumptions etc are all very useful to know when dependency between observations is a key aspect of your dataset.

A Sample Exam Questions

1. In time series models, why do we focus on moments of the distribution rather than the distribution itself?
2. What are the key differences between an autoregressive (AR) model and a moving average (MA) model?

B List of Topics

1. Expectation Maximization
2. Sampling from the posterior
3. Generalized additive models
4. Tree based methods
5. Classification and regression trees, their issues
6. Missing data
7. Adaboost

8. Gradient boosting models
9. Interpretation using relative importance and partial dependence plots
10. Random Forests and bagging
11. MARS
12. SVMs and the dual formulation
13. The kernel trick
14. Association rules
15. Clustering: dissimilarity and algorithms for clustering
16. Principal components
17. Spectral clustering
18. Basics of time series