STAT 535: Statistical Machine Learning **Winter 2019**

Lecture 5: Classification

Instructor: Yen-Chi Chen

5.1 Introduction

Classification is one of the most important data analysis problems. Much early work on this topic was done by statisticians but in the past 20 years, computer science and machine learning communities have made much much more progress on this topic.

Here are some classical applications of classification.

- **Email spam.** Email service provider such as Google often faced the problem of classifying a new email. The problem they want to address is like: given an email, how do you decide if it is a spam, an ordinary email, or an important one?
- **Image classification.** If you have used Facebook, you may notice that whenever your photo contains a picture of one of your friends, Facebook may ask you if you want to tag your friend, even if you did not manually tell the computer that this is your friend. How do they know that picture is a human and that guy is your friend?

We consider a simple scenario – binary classification. Namely, there are only two possible classes that we will consider. We will just denote the two classes as 0 and 1.

The classification problem can be formalized as follows. Given a feature vector x_0, we want to create a **classifier** c that maps x_0 into 0 or 1. Namely, we want to find a function $c(x_0)$ that outputs only two possible number: 0 and 1. Moreover, we want to make sure that our *classification error* is small. Let y_0 be the actual label of x_0. We measure the classification error using a **loss function** L such that the the loss of making a prediction $c(x_0)$ when the actual class label is y_0 is $L(c(x_0), y_0)$. A common loss function is the **0-1 loss**, which is $L(c(x_0), y_0) = I(c(x_0) \neq y_0)$. Namely, when we make a wrong classification, we loss 1 point and we do not lose anything if we make the correct classification.

How do we find the classifier c? A good news is: often we have a labeled sample (data) $(X_1, Y_1), \cdots, (X_n, Y_n)$ available. Then we will find c using this dataset.

In statistics, we often model the data as an IID random sample from a distribution. We now define several useful distribution functions:

$$\begin{aligned} p_0(x) = p(X = x|Y = 0): &\quad \text{the density of } X \text{ when the actual label is 0,} \\ p_1(x) = p(X = x|Y = 1): &\quad \text{the density of } X \text{ when the actual label is 1,} \\ P(y|x) = P(Y = y|X = x): &\quad \text{the probability of being in the class } y \text{ when the feature is } x, \\ P_Y(y) = P(Y = y): &\quad \text{the probability of observing the class } y, \text{ regardless of the feature value.} \end{aligned} \tag{5.1}$$

Using a probability model, we will define the **risk function**, which is the expected value of the loss function when the input is random. The risk of a classifier c is

$$R(c) = \mathbb{E}(L(c(X), Y)).$$

Ideally, we want to find a classifier that minimizes the risk because such a classifier will minimize our *expected losses*.

Assume that we know the 4 quantities in equation (5.1), what class label will you predict when seeing a feature $X = x$? An intuitive choice is that we should predict the value y that maximizer $P(y|x)$. Namely, we predict the label using the one with highest probability. Such classifier can be written as

$$c_*(x) = \mathsf{argmax}_{y=0,1} P(y|x) = \begin{cases} 0, & \text{if } P(0|x) \geq P(1|x), \\ 1, & \text{if } P(1|x) > P(0|x). \end{cases} \tag{5.2}$$

Is this classifier good in the sense of the classification error (risk)? The answer depends on the loss function. A good news is: this classifier is the optimal classifier for the $0-1$ loss. Namely,

$$R(c_*) = \min_c R(c)$$

when using a $0-1$ loss. However, if we are using other loss function, this classifier will not be the best one (with the smallest expected loss).

Derivation of c_* is optimal under $0-1$ loss. Given a classifier c, the risk function $R(c) = \mathbb{E}(L(c(X), Y))$. Using the property of expectation, we can further write it as

$$R(c) = \mathbb{E}(L(c(X), Y)) = \mathbb{E}(\underbrace{\mathbb{E}(L(c(X), Y)|X)}_{(A)}).$$

For the quantity (A), we have

$$\begin{aligned} \mathbb{E}(L(c(X), Y)|X) &= L(c(X), 1)p(Y = 1|X) + L(c(X), 0)p(Y = 0|X) \\ &= I(c(X) \neq 1)p(Y = 1|X) + I(c(X) \neq 0)p(Y = 0|X) \\ &= \begin{cases} p(Y = 1|X) & \text{if } c(X) = 0 \\ p(Y = 0|X) & \text{if } c(X) = 1. \end{cases} \end{aligned}$$

Thus, seeing a feature X, the expected loss we have when predicting $c(X) = 0$ is $P(Y = 1|X)$ whereas when prediction $c(X) = 1$ is $P(Y = 0|X)$. The optimal choice is predicting $c(X) = 0$ if $P(Y = 1|X) \leq P(Y = 0|X)$ and $c(X) = 1$ if $P(Y = 1|X) > P(Y = 0|X)$ (the equality does not matter), which is the classifier c_*.

When a classifier attains the optimal risk (i.e., having a risk of $\min_c R(c)$), it is called a **Bayes classifier**. Thus, the classifier c_* is the Bayes classifier in $0-1$ loss.

For a classifier c, we define its **excess risk (regret)** as

$$\mathcal{E}(c) = R(c) - \min_c R(c).$$

The excess risk is a quantity that measures how the quality of c is away from the optimal/Bayes classifier. If we cannot find the Bayes classifier, we will at least try to find a classifier whose excess risk is small.

5.2 Regression Approach

If we know the $P(y|x)$, we can build the Bayes classifier and this classifier is the optimal one in terms of the risk function. However, $P(y|x)$ is a population quantity, which is often unknown to us. All we have is a random sample $(X_1, Y_1), \cdots, (X_n, Y_n)$. So the question becomes: how do we estimate $P(y|x)$ using the data?

It turns out that this is a problem we know have to solve. Here is just one hint: because the response variable Y only takes two possible values $\{0, 1\}$, it is actually a Bernoulli random variable! Thus, $\mathbb{E}(Y) = P(Y = 1)$, which implies

$$\mathbb{E}(Y|X = x) = P(Y = 1|X = x) = P(1|x). \tag{5.3}$$

Namely, $P(1|x)$ is the regression function! Using the fact that $P(0|x) + P(1|x) = 1$, an estimator of $P(1|x)$ leads to an estimator of $P(y|x)$ for both $y = 0$ and $y = 1$.

Thus, as long as we have a regression estimator, we can convert it into a classifier. Here is one example of using kernel regression. Let

$$\widehat{m}_K(x) = \frac{\sum_{i=1}^n Y_i K\left(\frac{X_i - x}{h}\right)}{\sum_{i=1}^n K\left(\frac{X_i - x}{h}\right)}$$

be the kernel regression. Then

$$\widehat{P}_K(1|x) = \widehat{m}_K(x), \quad \widehat{P}_K(0|x) = 1 - \widehat{m}_K(x).$$

Thus, a classifier based on kernel regression is

$$\begin{aligned}
\widehat{c}_K(x) &= \begin{cases} 0, & \text{if } \widehat{P}_K(0|x) \geq \widehat{P}_K(1|x), \\ 1, & \text{if } \widehat{P}_K(1|x) > \widehat{P}_K(0|x) \end{cases} \\
&= \begin{cases} 0, & \text{if } 1 - \widehat{P}_K(1|x) \geq \widehat{P}_K(1|x), \\ 1, & \text{if } \widehat{P}_K(1|x) > 1 - \widehat{P}_K(1|x). \end{cases} \\
&= \begin{cases} 0, & \text{if } \widehat{P}_K(1|x) \leq \frac{1}{2}, \\ 1, & \text{if } \widehat{P}_K(1|x) > \frac{1}{2}. \end{cases} \\
&= \begin{cases} 0, & \text{if } \widehat{m}_K(x) \leq \frac{1}{2}, \\ 1, & \text{if } \widehat{m}_K(x) > \frac{1}{2}. \end{cases}
\end{aligned}$$

Namely, the classifier will output 1 whenever the estimated regression function is greater than half and 0 otherwise.

Will this classifier be a good one? Intuitively, it should be true. If we have a good regression estimator, the corresponding classifier should also be good. In fact, we have the following powerful result linking the quality of a regression estimator and the excess risk.

Theorem 5.1 *Assume we use the* $0 - 1$ *loss. Let* $\widehat{m}$ *be a regression estimator and* $\widehat{c}_m$ *be the corresponding classifier. Then*

$$\mathcal{E}(\widehat{c}_m) \leq 2 \int |\widehat{m}(x) - m(x)| dP(x) \leq 2\sqrt{\int |\widehat{m}(x) - m(x)|^2 dP(x)}.$$

Namely, if we have a regression estimator whose overall quality is good, the corresponding classifier will also have a small excess risk (i.e., perform comparably well compared to the optimal classifier).

5.3 Density Estimation Approach (Naive Bayes)

In addition to using a regression function to construct a classifier, we can use a density estimator for classification. This approach is often known as the *naive Bayes* approach.

A key insight is from the Bayes rule:

$$P(y|x) = P(Y = y|X = x) = \frac{p(x,y)}{p(x)} = \frac{p(x|y)P(Y=y)}{p(x)} = \frac{p_y(x)P_Y(y)}{p(x)},$$

where $p(x) = \sum_y p(x,y) = p(x,0) + p(x,1) = p_0(x)P_Y(0) + p_1(x)P_Y(1)$. Thus, the Bayes classifier can be written as

$$\begin{aligned} c_*(x) &= \begin{cases} 0, & \text{if } P(0|x) \geq P(1|x) \\ 1, & \text{if } P(1|x) > P(0|x) \end{cases} \\ &= \begin{cases} 0, & \text{if } \frac{p_0(x)P_Y(0)}{p(x)} \geq \frac{p_1(x)P_Y(1)}{p(x)} \\ 1, & \text{if } \frac{p_1(x)P_Y(1)}{p(x)} > \frac{p_0(x)P_Y(0)}{p(x)} \end{cases} \\ &= \begin{cases} 0, & \text{if } p_0(x)P_Y(0) \geq p_1(x)P_Y(1) \\ 1, & \text{if } p_1(x)P_Y(1) > p_0(x)P_Y(0) \end{cases}. \end{aligned}$$

Thus, if we can estimate $p_0(x), p_1(x)$, and $P_Y(y)$, we can construct a classifier.

$P_Y(y)$ is very easy to estimate. It is the probability of seeing an observation with label y. As a result, a simple estimator is to use the ratio of observations with this label. Namely,

$$\widehat{P}_Y(y) = \frac{1}{n}\sum_{i=1}^n I(Y_i = y).$$

$p_y(x)$ is just the conditional density of X given the label being y. Thus, we can simply apply a density estimator to those observations with a class label y.

Example: kernel density estimator. Using a kernel density estimator (KDE), we obtain

$$\widehat{p}_{y,kde}(x) = \frac{1}{n_y h}\sum_{i=1}^n I(Y_i = y)K\left(\frac{X_i - x}{h}\right),$$

where $n_y = \sum_{i=1}^n I(Y_i = y)$ is the number of observations with label being y. Note that $\widehat{P}_Y(y) = \frac{n_y}{n}$. Thus, a classifier based on a KDE is

$$\begin{aligned} \widehat{c}_{\mathsf{KDE}}(x) &= \begin{cases} 0, & \text{if } \widehat{p}_{0,kde}(x)\widehat{P}_Y(0) \geq \widehat{p}_{1,kde}(x)\widehat{P}_Y(1) \\ 1, & \text{if } \widehat{p}_{1,kde}(x)\widehat{P}_Y(1) > \widehat{p}_{0,kde}(x)\widehat{P}_Y(0) \end{cases} \\ &= \begin{cases} 0, & \text{if } \sum_{i=1}^n I(Y_i = 0)K\left(\frac{X_i - x}{h}\right) \geq \sum_{i=1}^n I(Y_i = 1)K\left(\frac{X_i - x}{h}\right) \\ 1, & \text{if } \sum_{i=1}^n I(Y_i = 1)K\left(\frac{X_i - x}{h}\right) > \sum_{i=1}^n I(Y_i = 0)K\left(\frac{X_i - x}{h}\right) \end{cases}. \end{aligned}$$

The classifier $\widehat{c}_{\mathsf{KDE}}(x)$ is also called the kernel classifier.

Example: density basis approach. We can use the basis approach as well. Assume that we consider M basis and we use the cosine basis $\{\phi_1(x), \phi_2(x), \cdots\}$. The estimator of $p_y(x)$ will be the density estimator using only those observations with a label y. Let

$$\widehat{\theta}_{y,\ell} = \frac{1}{n_y}\sum_{i=1}^n I(Y_i = y)\phi_\ell(X_i)$$

be the estimator of the ℓ-th coefficient using only those observations with a label y. The corresponding density estimator is

$$\widehat{p}_{y,M}(x) = \sum_{\ell=1}^M \widehat{\theta}_{y,\ell}\phi_\ell(x) = \sum_{\ell=1}^M \frac{1}{n_y}\sum_{i=1}^n I(Y_i = y)\phi_\ell(X_i)\phi_\ell(x) = \frac{1}{n_y}\sum_{i=1}^n I(Y_i = y)\sum_{\ell=1}^M \phi_\ell(X_i)\phi_\ell(x).$$

Thus, the corresponding classifier is

$$\begin{aligned}\widehat{c}_{\mathsf{M}}(x) &= \begin{cases} 0, & \text{if } \widehat{p}_{0,M}(x)\widehat{P}_Y(0) \geq \widehat{p}_{1,M}(x)\widehat{P}_Y(1) \\ 1, & \text{if } \widehat{p}_{1,M}(x)\widehat{P}_Y(1) > \widehat{p}_{0,M}(x)\widehat{P}_Y(0) \end{cases} \\ &= \begin{cases} 0, & \text{if } \sum_{i=1}^n I(Y_i = 0) \sum_{\ell=1}^M \phi_\ell(X_i)\phi_\ell(x) \geq \sum_{i=1}^n I(Y_i = 1) \sum_{\ell=1}^M \phi_\ell(X_i)\phi_\ell(x) \\ 1, & \text{if } \sum_{i=1}^n I(Y_i = 1) \sum_{\ell=1}^M \phi_\ell(X_i)\phi_\ell(x) > \sum_{i=1}^n I(Y_i = 0) \sum_{\ell=1}^M \phi_\ell(X_i)\phi_\ell(x) \end{cases} \end{aligned}.$$

5.4 Confusion Matrix

Given a classifier and a set of labeled data, we can illustrate the quality of classification using a confusion matrix. In binary classification, a confusion matrix is a 2×2 matrix (you can view it as a contingency table) as follows:

	Actual label: 0	Actual label: 1
Predicted label: 0	n_{00}	n_{01}
Predicted label: 1	n_{10}	n_{11}

n_{ij} is the number of instances/observations where the predicted label is i and actual label is j.

The quantity

$$\frac{n_{10} + n_{01}}{n_{00} + n_{01} + n_{10} + n_{11}},$$

is called the misclassification rate and is an empirical estimate of the risk of the classifier.

If the class label 0 stands for 'normal case' while the label 1 stands for 'anomaly', then we can interpret the confusion matrix as

	Actual label: 0	Actual label: 1
Predicted label: 0	True negative	False negtaive
Predicted label: 1	False positive	True positive

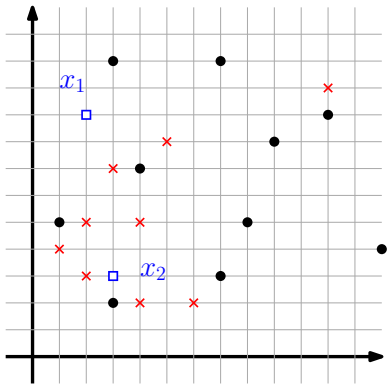
This interpretation is commonly used in engineering problem and medical research for detecting abnormal situation.

5.5 k-NN Approach

The k-NN approach can be applied to classification as well. The idea is very simple – for a given point x_0, we find its k-th nearest data points. Then we compare the labels of these k points and assign the label of x_0 as the majority label in these k points.

Take the data in the following picture as an example. There are two classes: black dots and red crosses. We are interested in the class label at the two blue boxes (x_1 and x_2).

Assume we use a 3-NN classifier $\widehat{c}_{3-NN}$. At point x_1, its 3-NN contains two black dots and one red cross, so $\widehat{c}_{3-NN}(x_1)$ = black dot. At point x_2, its 3-NN has one black dots and two red crosses, so $\widehat{c}_{3-NN}(x_1)$ = red cross. Note that if there are ties, we will randomly assign the class label attaining the tie.



The k-NN approach is simple and easy to operate. It can be easily generalize to multiple classes – the idea is the same: we assign the class label according to the majority in the neighborhood.

How do we choose k? The choice of k is often done by a technique called cross-validation. The basic principle is: we split the data into two parts, use one part to train our classifier and evaluate the performance on the other part. Repeating the above procedure multiple times and applying it to each k, we can obtain an estimate of the performance. We then choose the k with the best performance. We will talk about this topic later.

5.6 Logistic Regression

The logistic regression is a regression model that is commonly applied to classification problems as well. Like the method of converting a regression estimator to a classifier, the logistic regression use a regression function as an intermediate step and then form a classifier. We first talk about some interesting examples.

Example. In graduate school admission, we are wondering how a student's GPA affects the chance that this applicant received the admission. In this case, each observations is a student and the response variable Y represents whether the student received admission ($Y = 1$) or not ($Y = 0$). GPA is the covariate X. Thus, we can model the probability

$$P(\mathsf{admitted}|\mathsf{GPA} = x) = P(Y = 1|X = x) = q(x).$$

Example. In medical research, people are often wondering if the heretability of the type-2 diabetes is related to some mutation from of a gene. Researchers record if the subject has the type-2 diabetes (response) and measure the mutation signature of genes (covariate X). Thus, the response variable $Y = 1$ if this subject has the type-2 diabetes. A statistical model to associate the covariate X and the response Y is through

$$P(\text{subject has type-2 diabetes}|\text{mutation signature} = x) = P(Y = 1|X = x) = q(x).$$

Thus, the function $q(x)$ now plays a key role in determining how the response Y and the covariate X are associated. The logistic regression provides a simple and elegant way to characterize the function $q(x)$ in a 'linear' way. Because $q(x)$ represents a *probability*, it ranges within $[0, 1]$ so naively using a linear regression will not work. However, consider the following quantity:

$$O(x) = \frac{q(x)}{1 - q(x)} = \frac{P(Y = 1|X = x)}{P(Y = 0|X = x)} \in [0, \infty).$$

The quantity $O(x)$ is called the *odds* that measures the contrast between the event $Y = 1$ versus $Y = 0$. When the odds is greater than 1, we have a higher change of getting $Y = 1$ than $Y = 0$. The odds has an interesting asymmetric form– if $P(Y = 1|X = x) = 2P(Y = 0|X = x)$, then $O(x) = 2$ but if $P(Y = 0|X = x) = 2P(Y = 1|X = x)$, then $O(x) = \frac{1}{2}$. To symmetrize the odds, a straight-forward approach is to take (natural) logarithm of it:

$$\log O(x) = \log \frac{q(x)}{1 - q(x)}.$$

This quantity is called *log odds*. The log odds has several beautiful properties, for instance when the two probabilities are the same ($P(Y = 1|X = x) = P(Y = 0|X = x)$), $\log O(x) = 0$, and

$$P(Y = 1|X = x) = 2P(Y = 0|X = x) \Rightarrow \log O(x) = \log 2$$
$$P(Y = 0|X = x) = 2P(Y = 1|X = x) \Rightarrow \log O(x) = -\log 2.$$

The logistic regression is to impose a linear model to the log odds. Namely, the logistic regression models

$$\log O(x) = \log \frac{q(x)}{1 - q(x)} = \beta_0 + \beta^T x,$$

which leads to

$$P(Y = 1|X = x) = q(x) = \frac{e^{\beta_0 + \beta^T x}}{1 + e^{\beta_0 + \beta^T x}}.$$

Thus, the quantity $q(x) = q(x; \beta_0, \beta)$ depends on the two parameter β_0, β. Here β_0 behaves like the intercept and β behaves like the slope vector (they are the intercept and slope in terms of the log odds).

When we observe data, how can we estimate these two parameters? In general, we will use the maximum likelihood approach to estimate them. You can view the (minus) likelihood function as the loss function in the classification (actually, we will use the log-likelihood function as the loss function). And the goal is to find the parameter via minimizing such a loss.

Recall that we observe IID random sample:

$$(X_1, Y_1), \cdots, (X_n, Y_n).$$

Let $p_X(x)$ denotes the probability density of X; note that we will not use it in estimating β_0, β. For a given pair X_i, Y_i, recalled that the random variable Y_i given X_i is just a Bernoulli random variable with parameter $q(x = X_i)$. Thus, the PMF of Y_i given X_i is

$$\begin{aligned}
\mathcal{L}(\beta_0, \beta|X_i, Y_i) = P(Y = Y_i|X_i) &= q(X_i)^{Y_i}(1 - q(X_i))^{1-Y_i} \\
&= \left(\frac{e^{\beta_0 + \beta^T X_i}}{1 + e^{\beta_0 + \beta^T X_i}}\right)^{Y_i} \left(\frac{1}{1 + e^{\beta_0 + \beta^T X_i}}\right)^{1-Y_i} \\
&= \frac{e^{\beta_0 Y_i + \beta^T X_i Y_i}}{1 + e^{\beta_0 + \beta^T X_i}}.
\end{aligned}$$

Note that here we construct the likelihood function using only the conditional PMF because similarly to the linear regression, the distribution of the covariate X does not depends on the parameter β_0, β. Thus, the

log-likelihood function is

$$\begin{aligned}\ell(\beta_0,\beta|X_1,Y_1,\cdots,X_n,Y_n) &= \sum_{i=1}^{n}\log\mathcal{L}(\beta_0,\beta^T|X_i,Y_i)\\ &= \sum_{i=1}^{n}\log\left(\frac{e^{\beta_0 Y_i+\beta^T X_i Y_i}}{1+e^{\beta_0+\beta^T X_i}}\right)\\ &= \sum_{i=1}^{n}\beta_0 Y_i+\beta^T X_i Y_i-\log\left(1+e^{\beta_0+\beta^T X_i}\right).\end{aligned}$$

Our estimates are

$$\begin{aligned}\widehat{\beta}_0,\widehat{\beta} &= \underset{\beta_0,\beta}{\mathsf{argmax}}\ \ell(\beta_0,\beta|X_1,Y_1,\cdots,X_n,Y_n)\\ &= \underset{\beta_0,\beta}{\mathsf{argmin}}\ -\ell(\beta_0,\beta|X_1,Y_1,\cdots,X_n,Y_n)\\ &= \underset{\beta_0,\beta}{\mathsf{argmin}}\ \underbrace{\frac{1}{n}\sum_{i=1}^{n}\underbrace{-\ell(\beta_0,\beta|X_i,Y_i)}_{\text{loss function}}}_{\text{empirical estimate of the loss function}},\end{aligned}$$

where the loss function is

$$-\ell(\beta_0,\beta|X_i,Y_i)=\beta_0 Y_i+\beta^T X_i Y_i-\log\left(1+e^{\beta_0+\beta^T X_i}\right).$$

$\widehat{\beta}_0,\widehat{\beta}$ does not have a closed-form solution in general so we cannot write down a simple expression of the estimator. Despite this disadvantage, such a log-likelihood function can be optimized by a gradient ascent approach such as the Newton-Raphson[1].

5.7 Decision Tree

Decision tree is another common approach in classification. A decision tree is like a regression tree – we partition the space of covariate into rectangular regions and then assign each region a class label. If the tree is given, the class label at each region is determined by the majority vote (using the majority of labels in that region).

The following picture provides an example of a decision tree using the same data as the one in the previous section.

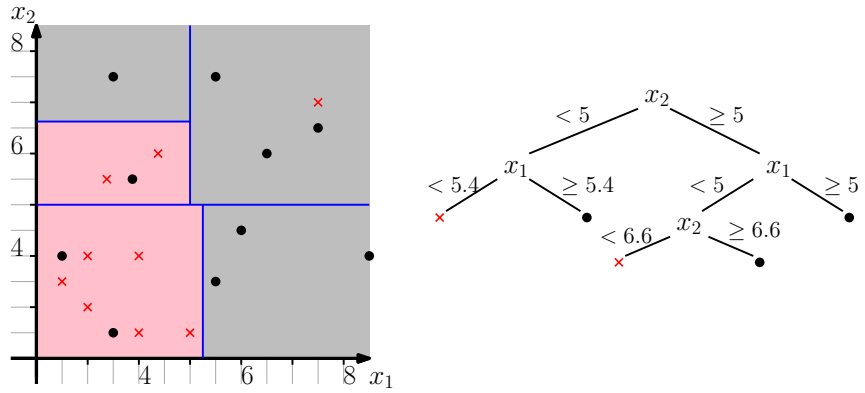
In the left panel, we display the scatterplot and regions separated by a decision tree. The background color denotes the estimated label. The right panel displays the tree structure.

In the case of binary classification (the class label $Y=0$ or 1), the decision tree can be written as follows. Let $(X_1,Y_1),\cdots,(X_n,Y_n)$ denotes the data and $R_1,\cdots,R_k$ is a rectangular partition of the space of the covariates. Let $R(x)$ denotes the rectangular regions where x falls within. The decision tree is

$$\widehat{c}_{DT}(x)=I\left(\frac{\sum_{i=1}^{n}Y_i I(X_i\in R(x))}{\sum_{i=1}^{n}I(X_i\in R(x))}>\frac{1}{2}\right).$$

Here, you can see that the decision tree is essentially a classifier converted from a regression tree.

[1] some references can be found: https://www.cs.princeton.edu/~bee/courses/lec/lec_jan24.pdf



5.8 Random Forest

A modification of the decision tree is called the *Random Forests*, which is a *bagging* approach. The idea of bagging is as follows. Suppose that we have one method of constructing a classifier. Given the original data set, we perform the bootstrap–sampling with replacement from the original data– to generate several bootstrap samples. Suppose that we have generated B bootstrap samples. If we train the classifier on each of the B bootstrap sample, we obtain a classifier. Thus, by training the classifier independently over each bootstrap sample, we obtain $\widehat{c}_1, \cdots, \widehat{c}_B$, B classifiers. We then combine these B classifiers to form our final classifier using the majority vote, i.e,,

$$\widetilde{c}(x) = \begin{cases} 1, & \text{if } \frac{1}{B}\sum_{b=1}^{B} \widehat{c}_b(x) \geq \frac{1}{2} \\ 0, & \text{otherwise} \end{cases} .$$

The formal name of bagging is *bootstrap aggregation*.

The random forest is the bagging decision tree with one additional modification– when training each classifier, instead of using all features, we only use a randomly selected subset of features. Informally, you can say

Random Forest = Decision Tree + Bagging + Random features.

There will be three tuning parameters in training a random forest: the number of leaves in the decision tree k, the bootstrap sample number B, and the number of randomly selected features d_0. Note that sometimes instead of using the bootstrap to generate a bootstrap sample, people would use subsampling method that randomly select only a fraction of the original sample in the bagging stage (each bagging sample has different subsets of the original data). If we use subsampling method, there will be one additional tuning parameter– the size of subsample.

The random forest can be applied to not only classification problem but also the regression problem. In the regression problem, the bagging stage will average the regression function from each bootstrap sample and the averaged value will be the final regression estimator. Namely,

$$\widetilde{m}(x) = \frac{1}{B}\sum_{b=1}^{B} \widehat{m}_b(x),$$

where $\widehat{m}_1(x), \cdots, \widehat{m}_B(x)$ are the regression tree estimator of each bootstrap sample.

5.8.1 Some theory about random forest

A useful tutorial summarizing recent advances in the theoretical behaviors of random forest can be found in

> Biau, G., & Scornet, E. (2016). A random forest guided tour. *Test*, 25(2), 197-227.

One useful theoretical result on random forest is the following theorem:

Theorem 5.2 (Biau 2012) *Suppose the regression function m is Lipschitz and only depends on a subset of features $S \subset \{1, \cdots, d\}$ and the probability of selecting a feature $j \in S$ is $\frac{1}{\|S\|}(1 + o(1))$, where $\|S\|$ is the cardinality of S. Then when the number of leaves $k_n \asymp n^{4S\log 2/(4S\log 2+3)}$,*

$$\mathbb{E}(|\widetilde{m}(X) - m(X)|^2) = O\left(\frac{1}{n}\right)^{\frac{3}{4\|S\|\log 2+3}}.$$

This is from

> Biau, G. (2012). Analysis of a random forests model. *Journal of Machine Learning Research*, 13(Apr), 1063-1095.

When using a subsampling method in training the random forest, here is an improved theoretical result that also regularizes the tuning parameters under the additive model.

Theorem 5.3 (Scornet, Biau and Vert (2015)) *Suppose that $Y = \sum_j m_k(X_j) + \epsilon$ where $X \sim \mathsf{Uni}[0,1]^d, \epsilon \sim N(0, \sigma^2)$ and each m_j is continuous. Assume that the split is chosen using the maximum drop in sums of squares. Let t_n be the number of leaves on each tree and a_n be the size of each subsample. If $t_n, a_n \to \infty$ and $\frac{t_n \log^9 a_n}{a_n} \to 0$ then*

$$\mathbb{E}(|\widetilde{m}(X) - m(X)|^2) \to 0.$$

The above result is from

> Scornet, E., Biau, G., & Vert, J. P. (2015). Consistency of random forests. *The Annals of Statistics*, 43(4), 1716-1741.

Random forest is also related to the kNN approach; see the following paper

> Lin, Y. and Jeon, Y. (2006). Random Forests and Adaptive Nearest Neighbors. *Journal of the American Statistical Association*, 101, p 578.

It is possible to use the random forest to perform statistical inference(e.g., hypothesis test and/or confidence intervals). See the following two papers:

> Mentch, L., & Hooker, G. (2014). Ensemble trees and clts: Statistical inference for supervised learning. *stat*, 1050, 25.
> Wager, S., & Athey, S. (2018). Estimation and inference of heterogeneous treatment effects using random forests. *Journal of the American Statistical Association*, 113(523), 1228-1242.

5.8.2 Kernel based random forest

Recently, there is a new approach that combines random forest and kernel regression. For a point x, let $A_j(x)$ be the cell such that x falls in j-th regression tree. The b-th regression tree estimator can be written as

$$\widehat{m}_b(x) = \frac{\sum_{i=1}^n Y_i I(X_i \in A_b(x))}{\sum_{i=1}^n I(X_i \in A_b(x))}.$$

Thus, the random forest regression function is

$$\begin{aligned}
\widetilde{m}(x) &= \frac{1}{B}\sum_{b=1}^B \widehat{m}_b(x) \\
&= \frac{1}{B}\sum_{b=1}^B \sum_{i=1}^n \frac{I(X_i \in A_b(x))}{\sum_{\ell=1}^n I(X_\ell \in A_b(x))} Y_i \\
&= \frac{1}{B}\sum_{i=1}^n \sum_{b=1}^B \frac{I(X_i \in A_b(x))}{N_b(x)} Y_i \\
&= \frac{1}{B}\sum_{i=1}^n \sum_{b=1}^B W_{bi} Y_i \\
&= \sum_{i=1}^n W_i Y_i
\end{aligned}$$

where and $N_b(x) = \sum_{i=1}^n I(X_i \in A_b(x))$ is the total number observations of b-th sample in the cell $A_b(x)$ and $W_{bi} = \frac{I(X_i \in A_b(x))}{N_b(x)}$, $W_i = \sum_{b=1}^B W_{bi}/B$ are both weights.

Notice that the weight $W_{bi} \propto I(X_i \in A_b(x))$. This motivates us to construct a kernel function that is proportional to $\sum_{b=1}^B I(X_i \in A_b(x))$, which leads to

$$\widehat{m}_{\mathsf{KeRF}}(x) = \frac{\sum_{i=1}^n Y_i K(x, X_i)}{\sum_{\ell=1}^n K(x, X_\ell)},$$

where

$$K(x, y) = \frac{1}{B}\sum_{b=1}^B I(y \in A_b(x)).$$

This method is called *kernel based random forest (KeRF)*. Note that in KeRF, the kernel function is a random function. Its randomness can be described by the IID random regions

$$A_1, \cdots, A_B \sim G,$$

where G is a distribution of random regions. Although it looks complicated, the randomness of each A_b is determined by 1. the bootstrap sample and 2. the randomly chosen features. So G is not so difficult to analyze.

Warning! The KeRF is different from the random forest although the construction of KeRF is motivated by random forest.

More details about KeRF can be found in the following paper:

> Scornet E. Random forests and kernel methods. (2016). *IEEE Transactions on Information Theory.* 62(3):1485-500.

5.9 Training Classifiers as an Optimization Problem

Even without any probabilistic model, classification problem can be viewed as an optimization problem. The key element is to replace the risk function $\mathbb{E}(L(c(X), Y)$ by the empirical risk (training error)

$$\widehat{R}_n(c) = \frac{1}{n}\sum_{i=1}^{n} L(c(X_i), Y_i).$$

Consider a collection of classifiers $\mathcal{C}$. Then the goal is to find the best classifier $c^* \in \mathcal{C}$ such that the the empirical risk $\widehat{R}_n(c)$ is minimized. Namely,

$$c^* = \underset{c\in\mathcal{C}}{\mathsf{argmin}}\ \widehat{R}_n(c).$$

Ideally, this should work because

$$\mathbb{E}(\widehat{R}_n(c)) = \mathbb{E}\left(\frac{1}{n}\sum_{i=1}^{n} L(c(X_i), Y_i)\right) = \mathbb{E}(L(c(X_1), Y_1)) = R(c) \tag{5.4}$$

is the actual risk function. Namely, $\widehat{R}_n(c)$ is an unbiased estimator of the risk function.

Here are some concrete examples.

Linear classifier. Assume that we consider a linear classifier:

$$c_{\beta_0,\beta}(x) = I(\beta_0 + \beta^T x > 0),$$

where β_0 is a number like the intercept and β is the vector like the slope of each covariate/feature. Namely, how we assign the class label purely depends on the value of $\beta_0 + \beta^T x$. If this value is positive, then we give it a label 1. If the value is negative, then we give it a label 0. Then the set

$$\mathcal{C}_{\mathsf{lin}} = \{c_{\beta_0,\beta} : \beta_0 \in \mathbb{R}, \beta \in \mathbb{R}^d\}$$

is a collection of all linear classifier. The idea of empirical risk minimization is to find the classifier in $\mathcal{C}_{\mathsf{lin}}$ such that the empirical risk $\widehat{R}_n(\cdot)$ is minimized. Because every classifier is indexed by the two quantities (parameters) β_0, β, finding the one that minimizes the empirical risk is equivalent to finding the best β_0, β minimizing $\widehat{R}_n(\cdot)$.

Logistic regression. Similar to the linear classifier, the logistic regression can be viewed as a classifier of the form

$$\widetilde{c}_{\beta_0,\beta}(x) = I\left(\frac{e^{\beta_0+\beta^T x}}{1+e^{\beta_0+\beta^T x}} > \frac{1}{2}\right).$$

Then we can define the set

$$\mathcal{C}_{\mathsf{logistic}} = \{\widetilde{c}_{\beta_0,\beta} : \beta_0 \in \mathbb{R}, \beta \in \mathbb{R}^d\}$$

as the collection of all classifiers from a logistic regression model. The MLE approach becomes an empirical risk minimization method using a particular loss function – the log-likelihood function.

Decision tree (fixed k, fixed leaves). The decision tree classifier $\widehat{c}_{\mathsf{DT}}$ can be viewed as a classifier from the empirical risk minimization as well. For simplicity, we assume that the regions/leaves of the decision trees $R_1, \cdots, R_k$ are fixed. Then any decision tree classifier can be written as

$$c_{\mathsf{DT}}(x) = \sum_{j=1}^{k} \alpha_j I(x \in R_j),$$

where $\alpha_1, \cdots, \alpha_k$ are quantities/parameters that determine how we will predict the class label of region $R_1, \cdots, R_k$, respectively. And the collection of all possible classifiers will be

$$\mathcal{C}_{\mathsf{DT}} = \{c_{\mathsf{DT}}(x) : \alpha_j \in \{0,1\}, j = 1, \cdots, k\}.$$

Note that there are only 2^k classifiers in the set $\mathcal{C}_{\mathsf{DT}}$. The estimator $\widehat{c}_{\mathsf{DT}}$ is just the one that minimizes the empirical loss $\widehat{R}_n(\cdot)$ with a $0-1$ loss.

Decision tree (fixed k, non-fixed leaves). When the regions/leaves are not fixed, the collection of all possible decision tree classifiers is much more complex. Here is an abstract way of describing such a collection. Recall that at each split of the tree, we pick one region and one feature and a threshold (e.g., $x_1 > 10$ versus $x_1 \leq 10$), so every split can be represented by three indices: the region we are splitting, the feature index (which feature is this split occurs), and the threshold index (what is the split level). Because at the ℓ-th split, there is only ℓ regions, the ℓ-th split is characterized by a triplet (ω, m, λ), where $\omega = \{1, 2, \cdots, \ell\}$ and $m \in \{1, 2, \cdots, d\}$ and $\lambda \in \mathbb{R}$. If a tree has k leaves, there will be $k-1$ splits. So any decision tree classifier is indexed by

$$\alpha_1, \cdots, \alpha_k, (\omega_1, m_1, \lambda_1), \cdots, (\omega_{k-1}, m_{k-1}, \lambda_{k-1}).$$

Namely, given a set of these values, we can construct a unique decision tree classifier. Thus, the collection of all decision tree with k leaves can be written as

$$\begin{aligned}\mathcal{C}_{\mathsf{DT}}(k) = \{c_{\mathsf{DT}}(x) = c_{\alpha,\omega,m,\lambda}(x) : \alpha_j \in \{0,1\},& \\ (\omega_\ell, m_\ell, \lambda_\ell) \in \{1, \cdots, \ell\} \times \{1, 2, \cdots, d\} \times \mathbb{R},& \\ j = 1, \cdots, k, \ell = 1, \cdots, k-1\}.&\end{aligned}$$

In reality, when we train a decision tree classifier with a fixed number k, the regions are also computed from the data. Thus, we are actually finding $\widehat{c}_{\mathsf{DT}}$ such that

$$\widehat{c}_{\mathsf{DT}} = \underset{c \in \mathcal{C}_{\mathsf{DT}}(k)}{\mathsf{argmin}} \ \widehat{R}_n(c).$$

Decision tree (both k and leaves are non-fixed). If we train the classifier with k being un-specified, then we are finding $\widehat{c}_{\mathsf{DT}}$ from $\bigcup_{k \in \mathbb{N}} \mathcal{C}_{\mathsf{DT}}(k)$ that minimizes the empirical risk. However, if we really consider all possible k, such an optimal decision tree is not unique and may be problematic. When $k > n$, we can make each leave contains at most and the predicted label is just the label of that observation. Such a classifier has 0 empirical risk but it may have very poor performance in future prediction because we are **overfitting** the data. Overfitting the data implies that $\widehat{R}_n(c)$ and $R(c)$ are very different, even if $\widehat{R}_n(c)$ is an unbiased estimator of $R(c)$!

Why will this happen? $\widehat{R}_n(c)$ is just the sample-average version of $R(c)$, right? Is this contradict to the law of large number that $\widehat{R}_n(c)$ converges to $R(c)$?

It is true that $\widehat{R}_n(c)$ is an unbiased estimator of $R(c)$ and yes indeed the law of large number is applicable in this case. BUT a key requirement for using the law of large number is that we assume c is fixed. Namely, if the classifier c is fixed, then the law of large number guarantees that the empirical risk $\widehat{R}_n(c)$ converges to the true risk function $R(c)$.

However, when we are finding the best classifier, we are consider many many many possible classifiers c. Although for a given classifier c the law of large number works, it may not work when we consider many classifiers. The empirical risk minimization works if

$$\sup_{c \in \mathcal{C}} \left| \widehat{R}_n(c) - R(c) \right| \xrightarrow{P} 0$$

Namely, the convergence is *uniform* for all classifiers in the collection that we are considering. In the next few lectures we will be talking about how the above uniform convergence may be established.

Remark.

- **Regression problem.** The empirical risk minimization method can also be applied to regression problem. We just replace the classifier by the regression function and the loss function can be chosen as the L_2 loss (squared distance). In this formulation, we obtain

  $$R(m) = \mathbb{E}\left(\|Y - m(X)\|^2\right)$$

  and

  $$\widehat{R}_n(m) = \frac{1}{n}\sum_{i=1}^{n} \|Y_i - m(X_i)\|^2.$$

  Estimating a regression function can thus be written as an empirical risk minimization – we minimizes $\widehat{R}_n(m)$ for $m \in \mathcal{M}$ to obtain our regression estimator. The set $\mathcal{M}$ is a collection of many regression functions. Similar to the classification problem, we need a uniform convergence of the empirical risk to make sure we have a good regression estimator.

- **Penalty function.** Another approach to handle the difference $\sup_{c\in\mathcal{C}} |\widehat{R}_n(c) - R(c)|$ is to add an extra quantity to $\widehat{R}_n(c)$ so that such a uniform difference is somewhat being controlled. Instead of minimizing $\widehat{R}_n(c)$, we minimizes

  $$\widehat{R}_n(c) + \mathcal{P}_\lambda(c),$$

  where $\mathcal{P}_\lambda$ is a penalty function. This is just the penalized regression approach but now applying it to a classification problem. When the penalty function is chosen in a good way, we can make sure the optimal classifier/regression estimator from the (penalized) empirical risk minimization indeed has a very small risk.

5.10 Other approaches

There are many other classification methods that are not covered in our lecture but I will highly recommend you to learn it. Here are some famous methods/keywords: boosting (adaBoost), neural nets (deep learning), ensemble learning.