Classification: Logistic Regression, Classification Tree, Boosting, Random Forest

Aramayis Dallakyan[1] [2]

[1]Agribusiness Teaching Center
Armenia

2019

[2]All errors are my own. armopost@yahoo.com

Outline

Introduction

Introduction

- The models considered in last classes, assumes that the response variable Y is quantitative. But in many situations, the response variable is instead *qualitative* or *categorical*.
- In this class, we study approaches for predicting qualitative responses, a process that is known as *classification*.
- Predicting a qualitative response for an observed classification can be referred to as classifying that observation, since it involves assigning the observation to a category, or class.
- We discuss three of the most widely-used classifiers: logistic regression, Classification trees and random forest.

Introduction

Examples of classification problem are:

- An online banking service must be able to determine whether or not a transaction being performed on the site is fraudulent, on the basis of the user's IP address, past transaction history, and so forth.
- An online store should determine the probability whether the customer will purchase the recommended product or not.
- A person arrives at the emergency room with a set of symptoms that could possibly be attributed to one of three medical conditions. Which of the three conditions does the individual have?

- Just in the regression settings, in the classification settings we have a set of observations $(x_1, y_1), \ldots, (x_n, y_n)$ that we can use to build a classifier.
- Ideally, we want our classifier to perform well not only on the training data, but also on test observations that were not used to train the classifier.
- In this class, we are going to use real dataset to predict the sales of a carseat based on some variables described later.
- For homework you are going to predict whether an individual will default on his/her credit card payment.

Why not Linear Regression

- Intuitive question is, why not linear regression?
- The answer can be best given using example.
- Suppose that we are trying to predict the medical condition of a patient in the emergency room on the basis of her symptoms. In this simplified example, there are three possible diagnoses: $Y = \{\text{stroke} = 1, \text{ drug overdose} = 2, \text{ and epileptic seizure} = 3\}$.
- Let fit the least squares on the bases of a set of predictors $X_1, \dots, X_p$. What can go wrong?

- The issue is , OLS will understand that Y has ordering and it will put *drug overdose* in between *stroke* and *epileptic seizure*, and insisting that the difference between *stroke* and *drug overdose* is the same as the difference between *drug overdose* and *epileptic seizure*.
- Which leads to wrongful estimation. Even if the natural ordering exists, OLS is not recommended.

Logistic Regression

Logistic Regression

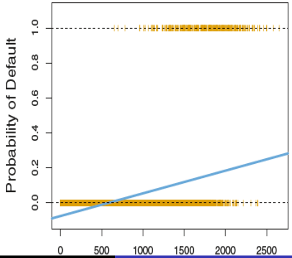
- Consider predicting the credit card default, where $Y = 0$ and 1, respectively.
- Rather then model the response Y directly, logistic regression models the *probability* that Y belongs to a particular category.
- For example $P(\text{default} = 1|X)$, what is the range of probability?
- Then for given X, we can predict probability of default. For example, we may choose to predict $\textit{default} = 1$ if $P > 0.5$.
- Note that 0.5 is arbitrary here and one may be more conservative and choose lower value, let say 0.1.

Logistic Regression

- The next question we answer is **How should we model the relationship between** $p(X) = P(Y = 1|X)$ and X?
- Intuitively one may propose

$$p(X) = \beta_0 + \beta_1 X$$

- What is the problem in this case, Hint look on the figure below, where prediction of *default* variable is based on *Balance* variable.

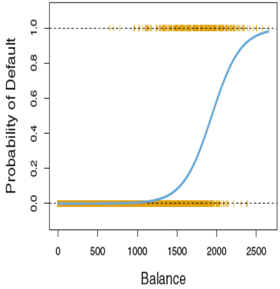


- In order to avoid the problem, we must model $p(X)$ using a function that gives outputs between 0 and 1 for all values of X.
- Many functions meet the description. In logistic regression, we use the **logistic function**

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta^0+\beta_1 X}}$$

- To fit the model we use method called **maximum likelihood**, which we will discuss later.

- The fit of the logistic regression looks as follows



- Note that now the probability is between 0 and 1. The logistic function will always produce an $S-$ shaped curve, which helps to capture the range of probabilities better than the linear model.

- After a bit manipulation of logistic function, we can find

$$\frac{p(X)}{1-p(X)} = e^{\beta_0+\beta_1 X}$$

- The quantity $p(X)/1-p(X)$ is called the *odds*, and can take on any value between 0 and ∞. Values closer to 0 indicates lower probabilities and vice versa for ∞.
- For example, on average nine out of every ten people with an odds of 9 will default, since $p(x) = 0.9$ implies that an odds of $\frac{0.9}{1-0.9} = 9$

- By taking the logarithm of both sides, we arrive at

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X$$

  The left-hand side is called the *log-odds* or *logit*.
- What is the relation between a logit and X?t
- In contrast to linear regression, here increasing X by one unit changes the log odds by β_1, or equivalently it multiplies the odds by e^{β_1}.
- Note that the relationship between $p(X)$ and X is not a straight line, thus β_1 does not correspond to the change in $p(X)$ associated with a one-unit increase in X.

- The coefficients β_0 and β_1 are unknown. The big question is how to estimate them?
- We use **maximum likelihood**(MLE) approach. The intuition is following , we try to find $\hat{\beta}_0$ and $\hat{\beta}_1$ such that plugging these estimates into the model for $p(X)$ yields a number close to one for all individuals who defaulted, and a number close to 0 for all individuals who did not.

- Mathematically, using the definition of joint independent Bernoulli probabilities:

$$\begin{aligned}\ell(\beta_0, \beta_1) &= \log(\prod_{i:y_1=1} p(x_i) \prod_{i':y_i'=0} (1 - p(x_i'))\\ &= \sum_{i=1}^{N} y_i \log p(x_i) + (1 - y_i) \log(1 - p(x_i))\\ &= \sum_{i=1}^{N} y_i \beta^t x_i - \log(1 + e^{\beta^t x_i})\end{aligned}$$

- Then by taking derivative with respect of β will find the MLE.

Classification Tree and Random Forest

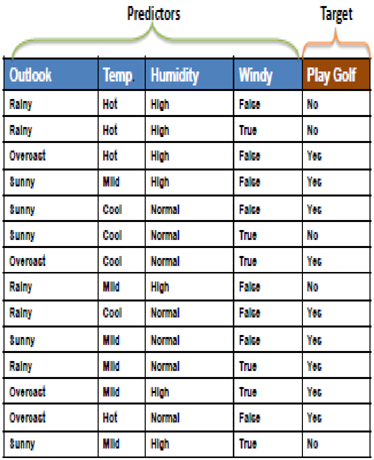
Classification Trees

- A *classification tree* is very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one.
- Here the target is a classification outcome taking values $k = 1, 2, \ldots, K$, the only changes needed in the tree algorithm concerns how to split node and prune the tree.
- Recall that for a regression tree, the predicted response for an observation is given by the mean response of the training observations that belong to the same terminal node.
- In contrast, for a classification tree, we predict that each observation belongs to the *most commonly occurring class* of training observations in the region to which it belongs.

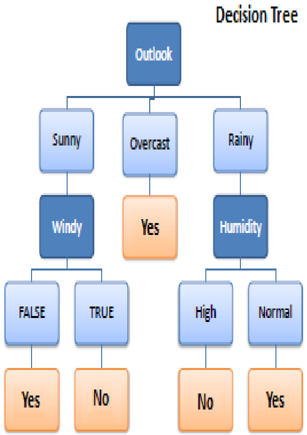
- The procedure of growing the classification or implementing random forest is very similar to continuous case, so we are not going to spend much time to develop it.
- One major difference is that we cannot use RSS as a criterion for making the binary splits.
- A natural alternative to RSS is the *classification error rate*.
- Since our task is to assign an observation in a given region to the most commonly occurring class of training observations in that region.
- In node m, representing a region R_m with N_m observations we define

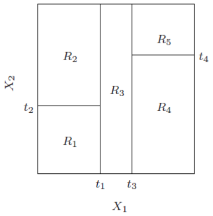
$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{i \in R_m} I(y_i = k)$$

the proportion of class k observations in node m.


Predictors
Target

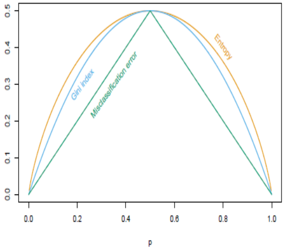
Outlook	Temp.	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No





- We classify the observations in node m to class $k(m) = \arg\max_k \hat{p}_{mk}$, the majority class in node m.
- Instead of RSS, we use the following three measures $Q_m(T)$
- Classification error rate $\frac{1}{N_m}\sum_{i\in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{mk}$,
- *Gini index* $\sum_{k=1}^{K} \hat{p}_{mk}(1-\hat{p}_{mk})$
- and Cross-entropy $-\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$

- For two classes, if p is the proportion in the second class, these three measures are
- $1 - max(p, 1-p)$, $2p(1-p)$, and $-p\log p - (1-p)\log(1-p)$, respectively.



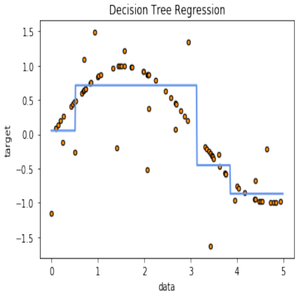
-

- Then similar to regression case, we find for each α, subtree $T_\alpha \subset T$ to minimize

$$\sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T|$$

Other Methods

- Bagging and Random Forest works the same way as described in previous section.
- Now we are going to learn a new method where trees are grown sequentially.
- Recall Avetik's question:

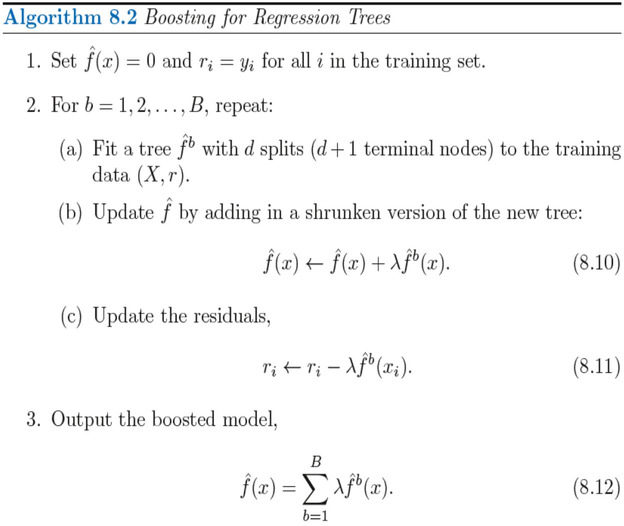


Boosting

- Recall that bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model. Notably, each tree is built on a bootstrap data set, independent of the other trees.
- **Boosting** works in a similar way, except that the trees are grown sequentially: each tree is grown using information from previously grown trees. Boosting does not involve bootstrap sampling; instead each tree is fit on a modified version of the original data set.

- Like bagging, boosting involves combining a large number of decision trees, $\hat{f}_1, \ldots, \hat{f}_B$.
- The big idea is following: Unlike fitting a single large decision tree to the data, which amounts to fitting the data hard and potentially overfitting, the boosting approach instead learns slowly.
- Given the current model, we fit a decision tree to the residuals from the model. That is, we fit a tree using the current residuals, rather than the outcome Y , as the response.
- We then add this new decision tree into the fitted function in order to update the residuals.
- Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm.

Boosting Algorithm



Algorithm 8.2 *Boosting for Regression Trees*

```
1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all $i$ in the training set.

2. For $b = 1, 2, \ldots, B$, repeat:

   (a) Fit a tree $\hat{f}^b$ with $d$ splits ($d + 1$ terminal nodes) to the training
       data $(X, r)$.

   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:

           $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$        (8.10)

   (c) Update the residuals,

           $r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$        (8.11)

3. Output the boosted model,

           $\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$        (8.12)
```

- Note that by fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well.
- The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to attack the residuals.
- In general, statistical learning approaches that learn slowly tend to perform well.
- Note that in boosting, unlike in bagging, **the construction of each tree depends strongly on the trees that have already been grown.**

Boosting has three tuning parameters:

- The number of trees B. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.
- The shrinkage parameter λ, a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.
- The number d of splits in each tree, which controls the complexity of the boosted ensemble. Often d $=$ 1 works well.

Measuring Performance

Suppose we are trying to classify the default of the individual based on credit card payment.

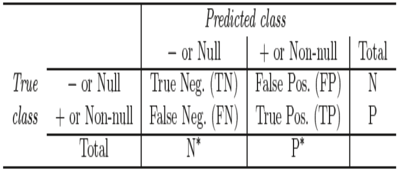
- In practice, binary classifier can make two types of errors:
    1. It can incorrectly assign an individual who defaults to the *no default* category.
    2. It can incorrectly assign an individual who does not default to the *default* category.
- A *confusion matrix, is a convenient way to display this information.*

- The example of confusion matrix is following:

		True default status		
		No	Yes	Total
Predicted default status	No	9,644	252	9,896
	Yes	23	81	104
	Total	9,667	333	10,000

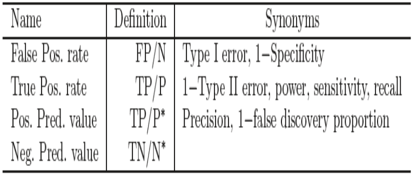
- For example in this case the method predicted that 104 people would default. Of these people, 81 actually defaulted and 23 did not. Hence only 23 out of 9,667 of the individuals who did not default were incorrectly labeled.
- However, of the 333 individuals who defaulted, 252 (75.7%) were missed by the method.
- So while overall rate is low, the error rate among individuals who defaulted is very high.

- Possible results when we apply a classifier to a population is following



		Predicted class		
		− or Null	+ or Non-null	Total
True	− or Null	True Neg. (TN)	False Pos. (FP)	N
class	+ or Non-null	False Neg. (FN)	True Pos. (TP)	P
	Total	N*	P*	

- Then we calculate following measures



Name	Definition	Synonyms
False Pos. rate	FP/N	Type I error, 1−Specificity
True Pos. rate	TP/P	1−Type II error, power, sensitivity, recall
Pos. Pred. value	TP/P*	Precision, 1−false discovery proportion
Neg. Pred. value	TN/N*	