October 19, 2017 Lecture 9, Oct. 4, 2017

Classification

Lecturer: Justin Domke *Scribe: Boya Ren*

1 Summary

Last time we talked about learning of decision trees. Today we will discuss the last topic in regression and forward to classification.

2 Regression with Multiple Outputs

In the previous lectures we usually assume $y \in R$. Then what if y is a vector, i.e. $y \in R^K$? A naive method is called the "Separate Option": process each output independently. Now let's look at the specific scenarios for nearest neighbors, trees and linear regression.

2.1 Nearest Neighbors

Still we will illustrate nearest neighbors with multiple outputs with an example. Suppose $x = (math, python)$ and $y = (grade, like)$, where "like" means how much does the student like the course. The data points as illustrated in the figure below.

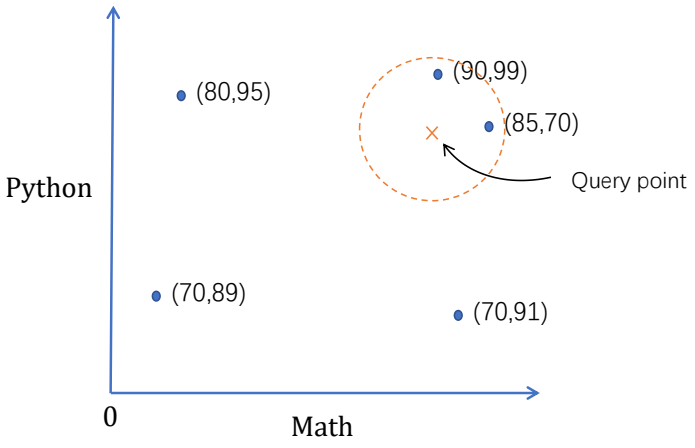

Figure 2.1: An Example of Multiple-Output KNN

We set the K for KNN to be 2, and the query point as shown in a cross will be predicted as $(87.5, 84.5)$ by averaging the output of its 2 nearest neighbor points. This is the same as separate option and since it does not require extra training, this approach actually saves computation time.

2.2 Regression Trees

The problem of learning regression trees is to find the best split each time and recursively grow a deep tree. If multiple outputs, similarly we still try to fit a "regression stump" with one split and use it recursively for deep trees. Recall that the decision tree regression can be modeled as the mapping

$$f(x) = \sum_{m=1}^{M} c_m I[x \in R_m]$$

For a single output, we use $c_m \in R$, while for multiple outputs we have $c_m \in R^K$. We still use the "component-wise" average $c_m = average(y_i | x_i \in R_m)$. Our learning target is to choose splits to achieve the following minimization problem

$$\min_{j,s} \left(\sum_{x_i \in R_1(j,s)} \|y_i - \hat{c}_1\|_2^2 + \sum_{x \in R_2(j,s)} \|y_i - \hat{c}_2\|_2^2 \right)$$

Here $\hat{c}_1$ and $\hat{c}_2$ are averages. It can be seen that this is not the same as separate option since all output components share the same splitted regions.

2.3 Linear Regression

With multiple outputs, the k^{th} component of linear regression can be expressed as

$$f_k(x) = \beta_{0k} + \sum_{j=1}^{p} x_j \beta_{jk}$$

Or in the matrix form $f(x) = B^T X : \ X^p \to X^K$, where $B \in R^{K \times (p+1)}$ and $X \in R^{p+1}$ by adding the constant feature. This is the same as separate option if no regularization. But with regularization, the B learned can be different.

3 Classification

In regression, the output y is usually continuous. In classification, however, we assume there are K classes, with class label $1, 2, \cdots, K$. The classes could be colors, countries, semantic labels and so on, and the labels are not ordered.

Consider the following scenario. Suppose X is user data + movie data, and y is the number of stars the user gives the movie. Should we use regression or classification for this problem? This is actually a hard choice. If we only want to distinguish between $0/1$ star, we will choose classification. But if we want to get the result of one of $0, 1, \cdots, 1000$ stars, you definitely need to use regression.

Actually, the biggest message on classification is that, "it is mostly like regression".

3.1 KNN

Again, our input is math and python skills and output classes are listed below. The data points and one query point is listed in Figure 3.1.

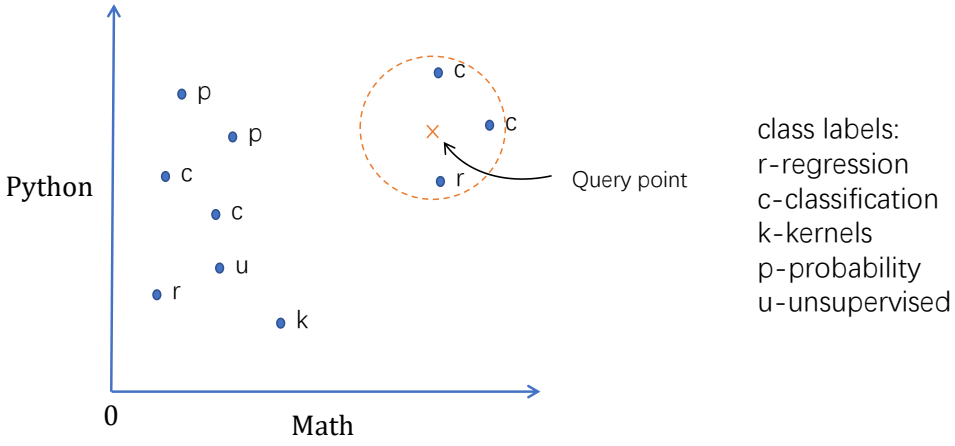

Figure 3.1: An Example of KNN Classification

If we set the K for KNN to be 3, the output for the query point would be "c". The algorithm is simply to return the most common label.

3.2 Classification Trees

By slightly adjusting the formula of regression tree, we get the prediction function for classification tree as below.

$$f(x) = \sum_{m=1}^{M} c_m I[x \in R_m]$$

Now c_m is not a real number anymore but a class label $c_m \in \{1, 2, \cdots K\}$. For fixed splits, c_m is set to be the most common label in region m. For convenience, we will note this most common label as $k(m)$. Then since we can not use MSE here, how can we find the splits?

First, we write the fraction of data in region m with label k to be

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

Our intuition is to let $\hat{P}_{mk}$ have low randomness. We have the following choices of measure randomness.

- Misclassification Error $1 - \hat{P}_{mk}$. This is rarely used since it is hard to optimize.
- Gini Index

$$\sum_{k=1}^{K} \sum_{k' \neq k} \hat{P}_{mk} \hat{P}_{mk'} = \sum_{k=1}^{K} \hat{P}_{mk}(1 - \hat{P}_{mk})$$

- Cross Entropy (from information theory)

$$-\sum_{k=1}^{K} \hat{P}_{mk} \log \hat{P}_{mk}$$

In practice, Gini Index and Cross Entropy usually have similar effects.

3.3 Binary Linear Classification

From now on, we assume the label $y \in \{-1, +1\}$. Our goal is to train a model $f(x) = \beta_0 + \sum_{j=1}^{p} \beta_j x_j$ to predict y. A hard problem is, we need a loss function. The intuition is, we want $f(x)$ to have the same sign as y, i.e. $yf(x) > 0$. More specifically, we want to fit β to minimize $\sum_{i=1}^{N} L(y_i, f(x_i))$ and wonder what L should be.

3.3.1 0-1 Loss

Intuitively, we might use $L(y, f(x)) = I[yf(x) < 0]$.
Good:

- If you can optimize it and you have lots of data, this will give you the best β.

Problems:

- Hard to train.
- Discontinuous.
- May generalize poorly.

Because of the hardness, 0-1 loss is rarely used.

3.3.2 Hinge Loss (Binary)

The main idea of hinge loss is to upper bound 0-1 loss as the following

$$L(y, f(x)) = max\{0, 1 - yf(x)\}$$

The comparison between 0-1 loss and hinge loss functions as shown in he figure below.

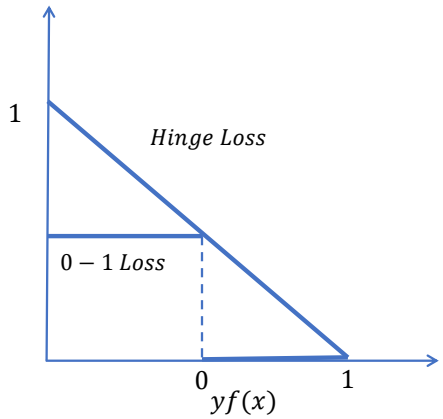

Figure 3.2: Comparison of 0-1 and Hinge Loss