CS 578 Class Notes

2019.01.16

Siqi Liang

0 Review

Note 1. In last lecture, we defined the *loss function* $\mathcal{L}(h(X), Y)$ for penalizing errors in prediction and our true objective is finding h s.t. $\min_{h \in \mathcal{H}} \mathbb{E}_{(X,Y) \in Pr(X,Y)} \mathcal{L}(h(X), Y)$. And usually, we will use an estimate $\min_{h \in \mathcal{H}} \frac{1}{N} \sum_{(X_i, Y_i) \in \mathcal{D}^{Train}} \mathcal{L}(h(X_i), Y_i)$ to approximate the true objective.

Note 2. Recipe:

- For each hyper-parameter:
  - Train using $\mathcal{D}^{Train}$.
  - Eval using $\mathcal{D}^{Test}$.
- Choose the model with the smallest $\mathcal{L}_{Val}$.
- Report the *loss* using $\mathcal{D}^{Test}$.

1 Cross Validation

Example 1.1. 4-fold
Randomly split the dataset into 4 parts. A single subset is retained as the validation data, and the remaining 3 subset are used as training data. The cross-validation process is then repeated 4 times, with each of the 4 subset used exactly once as the validation data (Figure 1).

Figure 1: 4-fold cross validation

For the i_{th}-fold, we can get the loss for training set $\mathcal{L}^i_{Train}$ and validation set $\mathcal{L}^i_{Val}$, respectively. Then the 4 results can then be averaged to produce a single estimation:

$$\mathcal{L}_{Val} = \frac{1}{4}(\mathcal{L}^1_{Val} + \cdots + \mathcal{L}^4_{Val}) \tag{1}$$

Finally, choose the model that has the smallest $\mathcal{L}_{Val}$ (average validation loss).

However, sometimes it might not be a good idea to randomly shuffle the data.

Example 1.2. Suppose you're building a model to predict stock returns, and you have data on the daily returns for every S&P500 stock over several years.

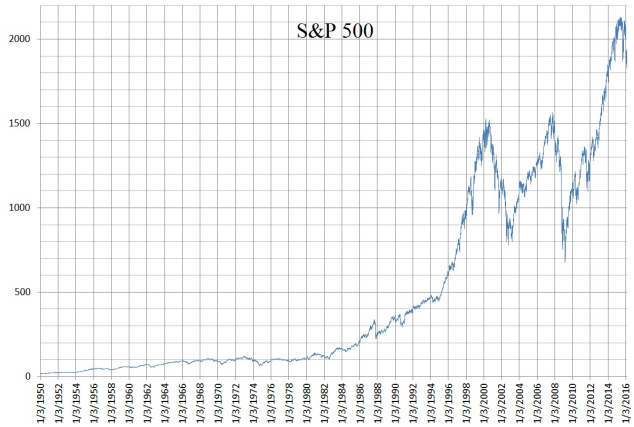

Figure 2: S&P500 index

Now split the data randomly, and train a model to predict next-day returns as a function of date, plus whatever else you think is relevant(interest rate, gold price, previous date S&P500 index etc.). If the S&P500 index has gone up over your sample period, then the model will favor the highest beta stocks. If the S&P500 went down, then your model will favor the lowest beta stocks. This will look great "out-of-sample", but only because you've implicitly told your model how the S&P500 performed over the entire sample period, so it already knows the right answer for your out-of-sample data. Therefore, in this case, it's not a good idea to split the data randomly.
Even if we remove the date, we still cannot randomly shuffle the data. Because the interest rate is hardly depend on date. The model may learn the information of date from the interest rate and back to the previous problem.

Example 1.3. Domain Adaption
Suppose we are given 50 data points which follows the same distribution. Among them, 10 data points are labeled and we want to make prediction on the remaining 40 unlabeled data point. At the same time, we got 1000 data points which follow a different distribution. Then how should we split these data into train, validation and test set?

The answer is that we should treat the 1000 data points which are from another distribution as the training set, treat the 10 labeled data points as the validation set and treat the remaining 40 unlabeled data.

2 K-Nearest Neighbors

There are two species of Iris – Setosa and Versicolor. Two features were measured from each sample: the length of the sepals and the width of the petals, in centimeters. Based on the combination of these two features, we want to developed a model to distinguish one species from the other. That is to say, given $\mathcal{D}^{Train} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, where $x_i \in \mathbb{R}^D, y_i \in [C] = 1, 2, \ldots, C$. Our goal is finding : $f : \mathbb{R}^D \to [C]$.

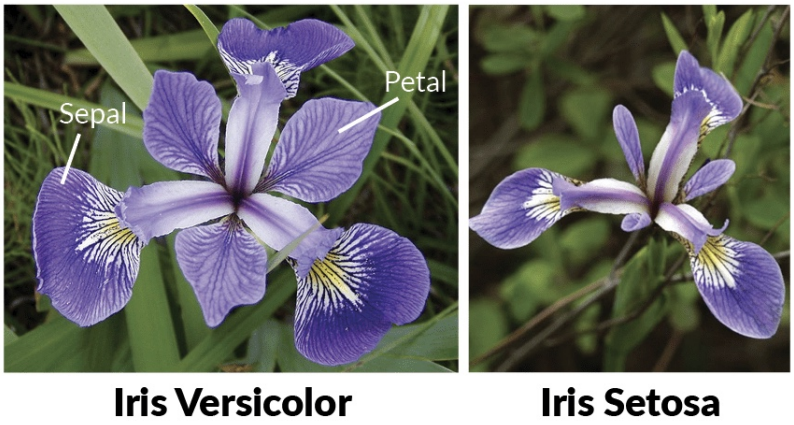

Figure 3: Two Types of Iris

2.1 1-Nearest Neighbor

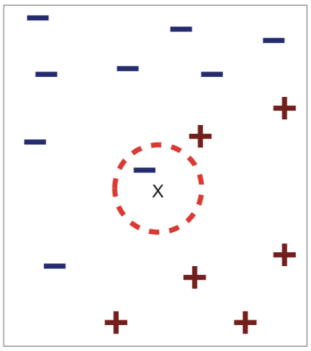

Figure 4: 1-Nearest Neighbor

Predict the same class as the nearest instance in the training set. Define

$$nn(x) = \arg\min_{n \in [N]} ||x - x_n||_2 = \sqrt{\sum_{d=1}^{D} (x_d - x_{nd})^2} \tag{2}$$

where $||\cdot||_2$ is the l_2 norm and it can be replaced with different distance measure like $||\cdot||_1, ||\cdot||_0, ||\cdot||_\infty$. The classification rule is $y = h(x) = y_{nn(x)}$.

2.2 K-Nearest Neighbors

Instead of deciding the label depend on one nearest neighbor, use the sorted distances to select the K nearest neighbors and then use majority rule to determine the label.

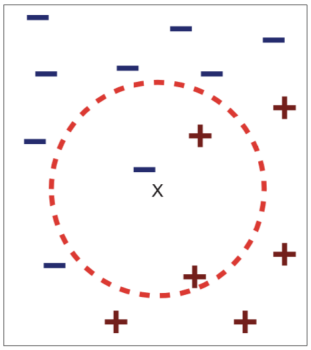

Figure 5: K-Nearest Neighbors

Therefore, the KNN algorithm will be:

Algorithm 1 KNN algorithm

1. Find k data points x_i^*, y_i^* closest to the test instance x.
2. Classification output is majority class

$$y = \arg\max_{y_0^* \in \{y_i^*\}_{i=1}^k} \sum_{i=1}^{k} \delta(y_0^*, y_i^*) \tag{3}$$

Note 3. It's important to normalize the data!

2.3 Asymptotics for KNN

The expected risk is

$$R(h) = \mathbb{E}_{(X,Y)\in Pr(x,y)}\mathcal{L}(h(X), Y) \tag{4}$$

where $\mathcal{L}(h(X), Y) = \mathbb{1}(h(X) \neq Y)$

Suppose $Pr(Y|X)$ is given, the **Bayes optimal classifier** try to find a classifier f^* s.t. $f^*(x) = \arg\max_{c\in[C]} Pr(c|x)$. The Bayes optimal classifier is the optimal classifier that one can hope to learn i.e $R(f^*) \leq R(f)\forall f$. And

$$\begin{aligned} R(f^*) &= \mathbb{E}_{X\in Pr(x)}\mathbb{E}_{Y\in Pr(y|x)}[\mathbb{1}(f^*(X) \neq Y)] \\ &= \mathbb{E}_{X\in Pr(x)}[1 - \max_{c\in[C]} \mathbb{P}(c|X)] \end{aligned} \tag{5}$$

Note 4. NNC is almost Bayesian optimal classifier

Thm (Corer&Hart) In a two-class classification problem, suppose f_N is the 1-Nearest Neighbor classifier with N training data points. Then

$$R(f^*) \leq lim_{n\to\infty}\mathbb{E}[R(f_N)] \leq 2R(f^*) \tag{6}$$