University of California, Berkeley Lecturer: Benjamin Recht

CS189 Introduction to Machine Learning

Spring 2018 Notes: Benjamin Recht

Cross Validation and Friends

Probably the most important part of machine learning is understanding how to validate and evaluate models. So I thought it would be worthwhile to type up notes on this topic and to be sure that it's clear how to do this.

The typical situation in machine learning is that you want to make a choice about a model that is hard to directly estimate on the training data. Some key examples include

1. Picking the tuning parameter in ridge regression.
2. Picking which set of features should be used to make a prediction.
3. Choosing the right parameters in feature generating functions.

Let's dive into some standard ways to make such decisions.

1 Basic Setup

In machine learning, we often have algorithms that chew up data and spit out prediction functions. To make this formally precise, we say that a machine learning algorithm that takes as input a dataset of arbitrary size and a parameter file. The data set will be input output pairs $T = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_L, y_L)\}$. The parameter file will be represented by the Greek letter ϑ. ϑ could contain a lot of things! It could contain the parameter in Ridge regression, or it could be the degree of a polynomial fit, or it could be the list of figures to include in the predictor. We assume that the algorithm takes T and ϑ and outputs prediction function f. That is,

$$f = \text{Algorithm}(S, \theta)$$

is a prediction function, and we want to find the setting of the configuration file ϑ that returns the best predictions of y from $\boldsymbol{x}$.

Throughout we assume that we have H candidate parameter configurations $\vartheta_1, \vartheta_2, \ldots, \vartheta_H$. The goal is to find the parameter file that we expect will work best on new data. By work well, we hope that $\text{loss}(f(\boldsymbol{x}), y)$ is small on new test data. We will evaluate this by having a new set of data T' and computing the empirical risk of the function on this new data:

$$R_{T'}[f] = \frac{1}{|T|} \sum_{(\boldsymbol{x},y)\in T} \text{loss}(f(\boldsymbol{x}, y)) \,.$$

The intention of all of these methods is to get a good estimate of the risk associated with a setting of the parameters ϑ.

2 The holdout method

We start with a set of data $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ For the holdout method, we break the training set into disjoint sets T and V with n_T and n_V elements. T is called the *training set* and V is the *holdout set* or *validation set*. Typically, the split is about 80% in the training set and 20% in the holdout set, but this varies from problem to problem.

The holdout method then proceeds as follows:

Algorithm 1 The Holdout Method

Input: Set of candidate parameters $\{\vartheta_k\}_{k=1}^H$, training set T and validation set V.
for $k = 1, \ldots, H$ **do**
  $f \leftarrow \text{Algorithm}(T, \theta_k)$
  $R_k \leftarrow \frac{1}{|V|} \sum_{(\boldsymbol{x},y)\in V} \text{loss}(f(\boldsymbol{x}, y))$
end for
Let $k_\star = \arg\min_k R_k$
$f \leftarrow \text{Algorithm}(S, \theta_{k_\star})$
return f

.

3 Cross Validation

Cross validation attempts to yield a lower variance estimate of the risk by running the holdout method on multiple validation sets. The idea is to partition the data into equal sized subsets, called *folds*. Each fold will function as a validation set. With each iteration, the algorithm is trained on the entire data set minus a single fold, and then the risk of the resulting prediction function is evaluated on the held out fold. The estimate of the risk is then the average over all considered folds. The holdout method is fully described in Algorithm 4.

Algorithm 2 The Cross Validation Method

Input: Set of candidate parameters $\{\vartheta_k\}_{k=1}^{H}$, data set S, number of folds F.
Partition S into F disjoint sets of the same size $S_1, S_2, \ldots, S_F$.
for $k = 1, \ldots, H$ **do**
  for $j = 1, \ldots, F$ **do**
    Let the training set $T = \cup_{i \neq j} S_i$ (that is, $T = S \setminus S_j$).
    $f \leftarrow \text{Algorithm}(T, \theta_k)$
    $R_{kj} \leftarrow \frac{1}{|S_j|} \sum_{(\boldsymbol{x}, y) \in S_j} \text{loss}(f(\boldsymbol{x}, y))$
  end for
  $R_k \leftarrow \frac{1}{F} \sum_{j=1}^{F} R_{kj}$
end for
Let $k_\star = \arg\min_k R_k$
$f \leftarrow \text{Algorithm}(S, \theta_{k_\star})$
return f

.

4 Monte Carlo Cross Validation

A variant of cross validation is to sample a number of holdout sets *with replacement* from S, and to compute the average of the holdout method on each of these sets. This looks a lot like cross validation, except that the folds at each iteration may intersect, and it can be run for an arbitrary number of rounds.

Algorithm 3 The Monte Carlo Cross Validation Method

Input: Set of candidate parameters $\{\vartheta_k\}_{k=1}^{H}$, data set S, holdout size n_h, number of rounds r.
for $k = 1, \ldots, H$ **do**
  for $j = 1, \ldots, r$ **do**
    Sample a holdout set V of size n_h from S. Let the training set $T = S \setminus V$
    $f \leftarrow \text{Algorithm}(T, \theta_k)$
    $R_{kj} \leftarrow \frac{1}{n_h} \sum_{(\boldsymbol{x},y)\in V} \text{loss}(f(\boldsymbol{x}, y))$
  end for
  $R_k \leftarrow \frac{1}{r} \sum_{j=1}^{r} R_{kj}$
end for
Let $k_\star = \arg\min_k R_k$
$f \leftarrow \text{Algorithm}(S, \theta_{k_\star})$
return f

.

5 The Leave-one-out Method

The leave-one-out method is the logical extreme of cross validation. In this algorithm, each data point is a fold. The leave-one out error is likely the most precise estimate of the true risk, but it is also the most computationally expensive.

Algorithm 4 The Leave-one-out Method

Input: Set of candidate parameters $\{\vartheta_k\}_{k=1}^{H}$, data set S.
for $k = 1, \ldots, H$ **do**
  for $j = 1, \ldots, n$ **do**
    Let the training set $T = S \setminus \{(\boldsymbol{x}_j, y_j)\}$.
    $f \leftarrow \text{Algorithm}(T, \theta_k)$
    $R_{kj} \leftarrow \text{loss}(f(\boldsymbol{x}_j, y_j))$
  end for
  $R_k \leftarrow \frac{1}{n} \sum_{j=1}^{n} R_{kj}$
end for
Let $k_\star = \arg\min_k R_k$
$f \leftarrow \text{Algorithm}(S, \theta_{k_\star})$
return f

.