CS229 Lecture notes

原作者：[Andrew Ng](http://cs229.stanford.edu/)（[吴恩达](http://open.163.com/movie/2008/1/M/C/M6SGF6VB4_M6SGHFBMC.html)）

翻译：[CycleUser](https://zhuanlan.zhihu.com/python-kivy)

Part VII 正则化（Regularization）与模型选择（model selection）

设想现在有一个机器学习的问题，我们要从一系列不同的模型中进行挑选。例如，我们可能是用一个多项式回归模型（polynomial regression model）hθ(x) = g(θ0 + θ1x + θ2x2 + ··· + θkxk)，然后想要判定这里的多项式次数 k 应该是多少，0， 1，...，或者 10。那我们怎么才能自动来选择一个能够在偏差（bias）/方差（variance）之间进行权衡的模型呢？1

或者换一个说法，假如我们希望能够自动选出来一个带宽参数（bandwidth parameter） τ 来用于局部加权回归（locally weighted regression，所谓为 LWR，参考 note1的第2节），或者要自动选出一个参数 C 用于拉格朗日正则化的支持向量机算法（l1-regularized SVM）。怎么来实现呢？

为了具体一些，咱们这一系列讲义中都假设备选集合的模型个数有限M = {M1,...,Md}。例如，在我们上面刚刚随便举的本章第一个例子中，Mi 就是一个 i次多项式拟合模型（i-th order polynomial regression model）。（其实把 M 扩展到无穷集合也不难的。2）换个说法就是，如果我们要从支持向量机算法（SVM）、神经网络算法（neural network）、逻辑回归算法（logistic regression）当中三选一，那么这里的 M 就应该都包含了这些模型了。

1Given that we said in the previous set of notes that bias and variance are two very different beasts, some readers may be wondering if we should be calling them “twin” evils here. Perhaps it’d be better to think of them as non-identical twins. The phrase “the fraternal twin evils of bias and variance” doesn’t have the same ring to it, though. 考虑到前面的讲义中我们已经提到过偏差（bias）/方差（variance）这两个家伙有很大区别，有的读者可能觉得是不是应该把它们叫做一对“孪生（twin）”魔鬼（evils）。或许可以把它们俩当做是一对异卵双胞胎（non-identical twins）。理解概念差别就好了，怎么说什么的都不要紧的。

2If we are trying to choose from an infinite set of models, say corresponding to the possible values of the bandwidth τ ∈ R+, we may discretize τ and consider only a finite number of possible values for it. More generally, most of the algorithms described here can all be viewed as performing optimization search in the space of models, and we can perform this search over infinite model classes as well. 如果我们要从一个无穷的模型集合中进行选取一个，假如说要选取一个带宽参数 τ ∈ R+ （正实数）的某个可能的值，可以将 τ 离散化，而只考虑有限的一系列值。更广泛来说，咱们要讲到的大部分算法都可以看做在模型空间（space of models）中进行优化搜索（performing optimization search）的问题，这种搜索也可以在无穷模型类（infinite model classes）上进行。

# 1 Cross validation

Let’s suppose we are, as usual, given a training set S. Given what we know about empirical risk minimization, here’s what might initially seem like a algorithm, resulting from using empirical risk minimization for model selection:

1. Train each model Mi on S, to get some hypothesis hi. 2. Pick the hypotheses with the smallest training error.

This algorithm does not work. Consider choosing the order of a poly- nomial. The higher the order of the polynomial, the better it will fit the training set S, and thus the lower the training error. Hence, this method will always select a high-variance, high-degree polynomial model, which we saw previously is often poor choice.

Here’s an algorithm that works better. In hold-out cross validation (also called simple cross validation), we do the following:

1. Randomly split S into Strain (say, 70% of the data) and Scv (the remain- ing 30%). Here, Scv is called the hold-out cross validation set.
2. Train each model Mi on Strain only, to get some hypothesis hi.
3. Select and output the hypothesis hi that had the smallest error εˆScv (hi) on the hold out cross validation set. (Recall, εˆScv (h) denotes the empir- ical error of h on the set of examples in Scv.)

By testing on a set of examples Scv that the models were not trained on, we obtain a better estimate of each hypothesis hi’s true generalization error, and can then pick the one with the smallest estimated generalization error. Usually, somewhere between 1/4 − 1/3 of the data is used in the hold out cross validation set, and 30% is a typical choice.

Optionally, step 3 in the algorithm may also be replaced with selecting the model Mi according to arg mini εˆScv (hi), and then retraining Mi on the entire training set S. (This is often a good idea, with one exception being learning algorithms that are be very sensitive to perturbations of the initial conditions and/or data. For these methods, Mi doing well on Strain does not necessarily mean it will also do well on Scv, and it might be better to forgo this retraining step.)

The disadvantage of using hold out cross validation is that it “wastes” about 30% of the data. Even if we were to take the optional step of retraining

2

3

the model on the entire training set, it’s still as if we’re trying to find a good model for a learning problem in which we had 0.7m training examples, rather than m training examples, since we’re testing models that were trained on only 0.7m examples each time. While this is fine if data is abundant and/or cheap, in learning problems in which data is scarce (consider a problem with m = 20, say), we’d like to do something better.

Here is a method, called k-fold cross validation, that holds out less data each time:

1. Randomly split S into k disjoint subsets of m/k training examples each. Let’s call these subsets S1,...,Sk.
2. For each model Mi, we evaluate it as follows: For j = 1,...,k  Train the model Mi on S1 ∪···∪Sj−1 ∪Sj+1 ∪···Sk (i.e., train on all the data except Sj) to get some hypothesis hij.  Test the hypothesis hij on Sj , to get εˆSj (hij ). The estimated generalization error of model Mi is then calculated  as the average of the εˆSj (hij)’s (averaged over j).
3. Pick the model Mi with the lowest estimated generalization error, and retrain that model on the entire training set S. The resulting hypothesis is then output as our final answer.

A typical choice for the number of folds to use here would be k = 10. While the fraction of data held out each time is now 1/k—much smaller than before—this procedure may also be more computationally expensive than hold-out cross validation, since we now need train to each model k times.

While k = 10 is a commonly used choice, in problems in which data is really scarce, sometimes we will use the extreme choice of k = m in order to leave out as little data as possible each time. In this setting, we would repeatedly train on all but one of the training examples in S, and test on that held-out example. The resulting m = k errors are then averaged together to obtain our estimate of the generalization error of a model. This method has its own name; since we’re holding out one training example at a time, this method is called leave-one-out cross validation.

Finally, even though we have described the different versions of cross vali- dation as methods for selecting a model, they can also be used more simply to evaluate a single model or algorithm. For example, if you have implemented

4

some learning algorithm and want to estimate how well it performs for your application (or if you have invented a novel learning algorithm and want to report in a technical paper how well it performs on various test sets), cross validation would give a reasonable way of doing so.

# 2 Feature Selection

One special and important case of model selection is called feature selection. To motivate this, imagine that you have a supervised learning problem where the number of features n is very large (perhaps n ≫ m), but you suspect that there is only a small number of features that are “relevant” to the learning task. Even if you use a simple linear classifier (such as the perceptron) over the n input features, the VC dimension of your hypothesis class would still be O(n), and thus overfitting would be a potential problem unless the training set is fairly large.

In such a setting, you can apply a feature selection algorithm to reduce the number of features. Given n features, there are 2n possible feature subsets (since each of the n features can either be included or excluded from the subset), and thus feature selection can be posed as a model selection problem over 2n possible models. For large values of n, it’s usually too expensive to explicitly enumerate over and compare all 2n models, and so typically some heuristic search procedure is used to find a good feature subset. The following search procedure is called forward search:

1. Initialize F = ∅.
2. Repeat {  (a)Fori=1,...,nifi̸∈F,letFi =F∪{i},andusesomever- sion of cross validation to evaluate features Fi. (I.e., train your learning algorithm using only the features in Fi, and estimate its generalization error.)  (b) Set F to be the best feature subset found on step (a). }
3. Select and output the best feature subset that was evaluated during the entire search procedure.

5

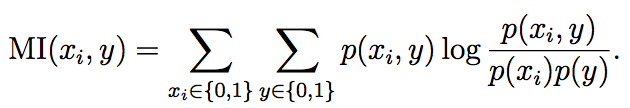
The outer loop of the algorithm can be terminated either when F = {1, . . . , n} is the set of all features, or when |F | exceeds some pre-set thresh- old (corresponding to the maximum number of features that you want the algorithm to consider using).

This algorithm described above one instantiation of wrapper model feature selection, since it is a procedure that “wraps” around your learning algorithm, and repeatedly makes calls to the learning algorithm to evaluate how well it does using different feature subsets. Aside from forward search, other search procedures can also be used. For example, backward search starts off with F = {1, . . . , n} as the set of all features, and repeatedly deletes features one at a time (evaluating single-feature deletions in a similar manner to how forward search evaluates single-feature additions) until F = ∅.

Wrapper feature selection algorithms often work quite well, but can be computationally expensive given how that they need to make many calls to the learning algorithm. Indeed, complete forward search (terminating when F = {1, . . . , n}) would take about O(n2) calls to the learning algorithm.

Filter feature selection methods give heuristic, but computationally much cheaper, ways of choosing a feature subset. The idea here is to compute some simple score S(i) that measures how informative each feature xi is about the class labels y. Then, we simply pick the k features with the largest scores S (i).

One possible choice of the score would be define S(i) to be (the absolute value of) the correlation between xi and y, as measured on the training data. This would result in our choosing the features that are the most strongly correlated with the class labels. In practice, it is more common (particularly for discrete-valued features xi) to choose S(i) to be the mutual information MI(xi, y) between xi and y:



(The equation above assumes that xi and y are binary-valued; more generally the summations would be over the domains of the variables.) The probabil- ities above p(xi,y), p(xi) and p(y) can all be estimated according to their empirical distributions on the training set.

To gain intuition about what this score does, note that the mutual infor- mation can also be expressed as a Kullback-Leibler (KL) divergence:



You’ll get to play more with KL-divergence in Problem set #3, but infor- mally, this gives a measure of how different the probability distributions

6

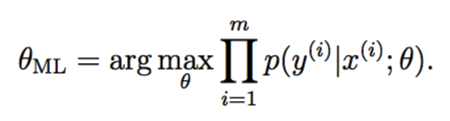
p(xi,y) and p(xi)p(y) are. If xi and y are independent random variables, then we would have p(xi, y) = p(xi)p(y), and the KL-divergence between the two distributions will be zero. This is consistent with the idea if xi and y are independent, then xi is clearly very “non-informative” about y, and thus the score S(i) should be small. Conversely, if xi is very “informative” about y, then their mutual information MI(xi,y) would be large.

One final detail: Now that you’ve ranked the features according to their scores S(i), how do you decide how many features k to choose? Well, one standard way to do so is to use cross validation to select among the possible values of k. For example, when applying naive Bayes to text classification— a problem where n, the vocabulary size, is usually very large—using this method to select a feature subset often results in increased classifier accuracy.

# 3 Bayesian statistics and regularization

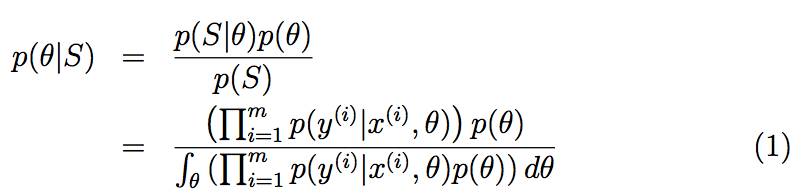
In this section, we will talk about one more tool in our arsenal for our battle against overfitting.

At the beginning of the quarter, we talked about parameter fitting using maximum likelihood (ML), and chose our parameters according to



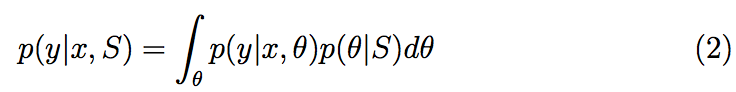
Throughout our subsequent discussions, we viewed θ as an unknown param- eter of the world. This view of the θ as being constant-valued but unknown is taken in frequentist statistics. In the frequentist this view of the world, θ is not random—it just happens to be unknown—and it’s our job to come up with statistical procedures (such as maximum likelihood) to try to estimate this parameter.

An alternative way to approach our parameter estimation problems is to take the Bayesian view of the world, and think of θ as being a random variable whose value is unknown. In this approach, we would specify a prior distribution p(θ) on θ that expresses our “prior beliefs” about the parameters. Given a training set S = {(x(i),y(i))}mi=1, when we are asked to make a prediction on a new value of x, we can then compute the posterior distribution on the parameters



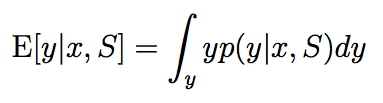
In the equation above, p(y(i)|x(i),θ) comes from whatever model you’re using for your learning problem. For example, if you are using Bayesian logistic re- gression, then you might choose p(y(i)|x(i), θ) = hθ(x(i))y(i) (1−hθ(x(i)))(1−y(i)), where hθ(x(i)) = 1/(1 + exp(−θT x(i))).3

When we are given a new test example x and asked to make it prediction on it, we can compute our posterior distribution on the class label using the posterior distribution on θ:



In the equation above, p(θ|S) comes from Equation (1). Thus, for example, if the goal is to the predict the expected value of y given x, then we would output4

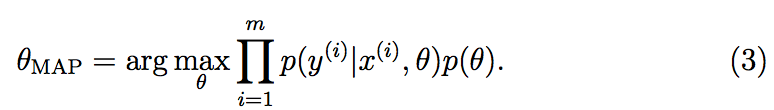






The procedure that we’ve outlined here can be thought of as doing “fully Bayesian” prediction, where our prediction is computed by taking an average with respect to the posterior p(θ|S) over θ. Unfortunately, in general it is computationally very difficult to compute this posterior distribution. This is because it requires taking integrals over the (usually high-dimensional) θ as in Equation (1), and this typically cannot be done in closed-form.

Thus, in practice we will instead approximate the posterior distribution for θ. One common approximation is to replace our posterior distribution for θ (as in Equation 2) with a single point estimate. The MAP (maximum a posteriori) estimate for θ is given by



3Since we are now viewing θ as a random variable, it is okay to condition on it value, and write “p(y|x, θ)” instead of “p(y|x; θ).”

4The integral below would be replaced by a summation if y is discrete-valued.



Note that this is the same formulas as for the ML (maximum likelihood) estimate for θ, except for the prior p(θ) term at the end.

In practical applications, a common choice for the prior p(θ) is to assume that θ ∼ N(0,τ2I). Using this choice of prior, the fitted parameters θMAP will have smaller norm than that selected by maximum likelihood. (See Problem Set #3.) In practice, this causes the Bayesian MAP estimate to be less susceptible to overfitting than the ML estimate of the parameters. For example, Bayesian logistic regression turns out to be an effective algorithm for text classification, even though in text classification we usually have n ≫ m.