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。

1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。

2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 交叉验证（Cross validation）

一如既往，假如我们得到了一个训练集 S。我们已经了解了经验风险最小化（empirical risk minimization，缩写为 ERM），那么接下来就要通过使用 ERM 来进行模型选择来推导出一种新的算法：

1. 对训练集 S 中的每一个模型（model） Mi 进行训练，得到某假设类(hypothesis) hi。
2. 从这些假设中选取训练误差最小的假设（hypotheses）。

上面这个算法是行不通的。比如考虑要选择多项式的阶（最高次项的次数）的情况。多项式的阶越高，对训练集 S 的拟合程度就越好，训练误差自然也就更小。然而，这个方法选出来的总是那种波动非常强（high-variance）的高次多项式模型（high-degree polynomial model），这种情况我们之前就讲过了，通常都是很差的选择。

下面这个算法就更好一些。这个方法叫做保留交叉验证（hold-out cross validation），也叫做简单交叉验证（simple cross validation），步骤如下：

1. 随机拆分训练集 S 成 Strain (例如，可以选择整体数据中的 70% 用于训练) 和 Scv (训练集中剩余的 30%用于验证)。这里的 Scv 就叫做保留交叉验证集（hold-out cross validation set）。
2. 只对集合 Strain 中的每一个模型 Mi 进行训练，然后得到假设类（hypothesis） hi。
3. 筛选并输出对保留交叉验证集有最小误差 εˆScv (hi) 的假设hi 。（回忆一下，这里的 εˆScv (hi) 表示的是假设 h 在保留交叉验证集 Scv 中的样本的经验误差（empirical error）。）

这样通过在一部分未进行训练的样本集合 Scv 上进行测试，我们对每个假设 hi 的真实泛化误差（generalization error）就能得到一个比上一个方法更好的估计，然后就能选择出来一个有最小估计泛化误差（smallest estimated generalization error）的假设了。通常可以选择 1/4 到 1/3 的数据样本用来作为保留交叉验证集（hold out cross validation set），30% 是一个很典型的选择。

还有另外一种备选方法，就是在第三步的时候，也可以换做选择与最小估计经验误差 εˆScv (hi) 对应的模型 Mi ，然后对整个训练样本数据集 S 使用 Mi 来进行再次训练。（这个思路通常都不错，但有一种情景例外，就是学习算法对初始条件和数据的扰动（perturbations of the initial conditions and/or data）非常敏感的情况。在这样的方法中，适用于Strain 的模型未必就能够同样适用于 Scv，这样就最好还是放弃再训练的步骤（forgo this retraining step）。）

使用保留交叉验证集（hold out cross validation set）的一个弊端就是“浪费（waste）”了训练样本数据集的 30% 左右。甚至即便我们使用了备选的那个针对整个训练集使用模型进行重新训练的步骤，也还不成，因为这无非是相当于我们只尝试在一个 0.7 m 规模的训练样本集上试图寻找一个好的模型来解决一个机器学习问题，而并不是使用了全部的 m 个训练样本，因为我们进行测试的都是每次在仅 0.7 m 规模样本上进行训练而得到的模型。当然了，如果数据非常充足，或者是很廉价的话，也可以用这种方法，而如果训练样本数据本身就很稀缺的话（例如说只有 20 个样本），那就最好用其他方法了。

下面就是一种这样的方法，名字叫k-折交叉验证(k-fold cross validation)，这样每次的用于验证的保留数据规模都更小：

1. 随机讲训练集 S 切分成 k 个不相交的子集，其中每一个子集的规模为 m/k 个训练样本。咱们就把这些子集称为 S1,...,Sk。
2. 对每个模型 Mi，我们都按照下面的步骤来进行评估（evaluate）：对 j = 1, ..., k，在 S1 ∪···∪Sj−1 ∪Sj+1 ∪···Sk上（也就是除了 Sj 之外的其他所有数据）对模型 Mi 进行训练，然后得到假设 hij 。接下来针对 Sj 使用假设 hij 进行测试，得到经验误差 εˆSj (hij )。对 εˆSj (hij) 取平均值（也就是对所有的 j 都计算然后取平均值），计算得到的值就当做是模型 Mi 的估计泛化误差（estimated generalization error）。
3. 选择具有最小估计泛化误差（lowest estimated generalization error）的模型 Mi 的，然后在整个训练样本集 S 上重新训练该模型。这样得到的假设（hypothesis）就可以输出作为最终结果了。

通常这里进行折叠的次数（number of folds）k 一般是 10，即 k = 10。这样每次进行保留用于验证的数据块就只有 1/k ，这就比之前的 30% 要小多了，当然这样一来这个过程也要比简单的保留交叉验证方法消耗更多算力成本，因为现在需要对每个模型都进行 k 次 训练。

虽然通常选择都是设置 k = 10，不过如果一些问题中数据量确实很匮乏，那有时候也可以走一点极端，设 k = m，这样是为了每次能够尽可能多地利用数据，尽可能少地排除数据。这种情况下，我们需要在训练样本集 S 中除了某一个样本外的其他所有样本上进行训练，然后在保留出来的单独样本上进行检验。然后把计算出来的 m = k 个误差放到一起求平均值，这样就得到了对一个模型的泛化误差的估计。这个方法有专门的名字；由于每次都保留了一个训练样本，所以这个方法就叫做弃一法交叉验证（leave-one-out cross validation）。

最后总结一下，咱们讲了不同版本的交叉验证，在上文中是用来作为选择模型的方法，实际上也可以更单纯地用来对一个具体的模型或者算法进行评估。例如，如果你已经实现了某中学习算法，然后想要估计一下针对你的用途这个算法的性能表现（或者是你创造了一种新的学习算法，然后希望在技术论文中报告你的算法在不同测试集上的表现），交叉验证都是个很好的解决方法。

# 2 特征选择（Feature Selection）

模型选择（model selection）的一个非常重要的特殊情况就是特征选择（feature selection）。设想你面对一个监督学习问题（supervised learning problem），其中特征值的数量 n 特别大（甚至可能比训练样本集规模还大，即n ） m），然而你怀疑可能只有一小部分的特征（features）是与学习任务“相关（relevant）”的。甚至即便是针对 n 个输入特征值使用一个简单的线性分类器（linear classifier，例如感知器 perceptron），你的假设类（hypothesis class）的 VC 维（VC dimension）也依然能达到 O(n)，然后这样的话就很有过拟合（overfitting）的潜在风险，除非训练样本集也足够巨大（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:

在这样的一个背景下，你就可以使用一个特征选择算法，来降低特征值的数目。

1. 初始化（Initialize）一个集合为空集 F = ∅.
2. 循环下面的过程 {  (a) 对于 i =1, ..., n ，如果i̸∈F, 则令 Fi =F∪{i}，然后使用某种交叉验证（cross validation）来评估特征 Fi。（也就是说，仅仅使用 Fi 当中的特征来训练你的学习算法，然后估计一下泛化误差（generalization error）。）  (b) 将 F 设为步骤 (a) 中的最佳特征子集（best feature subset）。}
3. 整个搜索过程中筛选出来了最佳特征子集（best feature subset），将其输出。

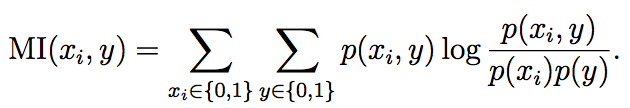
上面这个算法最外层的循环体（outer loop）可以在 F = {1, ... , n} 达到全部特征规模的时候终止（terminated），或者也可以在 |F | 超过某个预设阈值的情况下终止（这种情况对应的就是例如你可能对算法要用到的特征数量设定了最大值）。

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 probabilities 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 information can also be expressed as a Kullback-Leibler (KL) divergence:



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

6

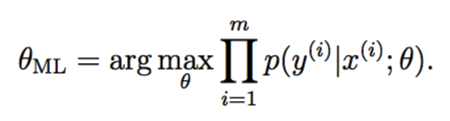
p(xi,y) and p(xi)p(y) are. If xi and y are independent random variables, then we would have p(xi, y) = p(xi)p(y), and the KL-divergence between the two distributions will be zero. This is consistent with the idea if xi and y are independent, then xi is clearly very “non-informative” about y, and thus 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

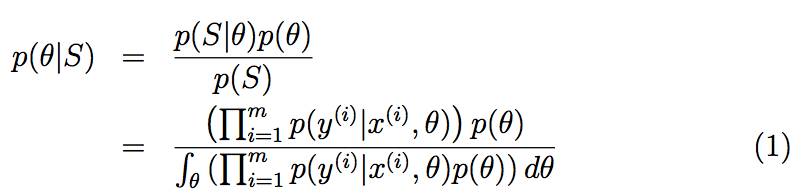
In this section, we will talk about one more tool in our arsenal for our battle against overfitting.

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



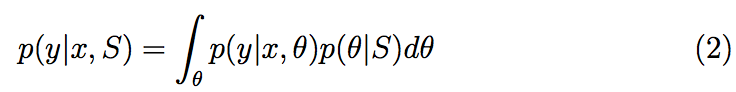
Throughout our subsequent discussions, we viewed θ as an unknown parameter of the world. This view of the θ as being constant-valued but unknown is taken in frequentist statistics. In the frequentist, this view of the world, θ is not random—it just happens to be unknown—and it’s our job to come up with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



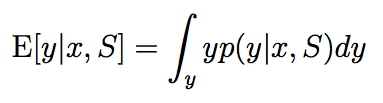
In the equation above, p(y(i)|x(i),θ) comes from whatever model you’re using for your learning problem. For example, if you are using Bayesian logistic regression, then you might choose p(y(i)|x(i), θ) = hθ(x(i))y(i) (1−hθ(x(i)))(1−y(i)), where hθ(x(i)) = 1/(1 + exp(−θT x(i))).3

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

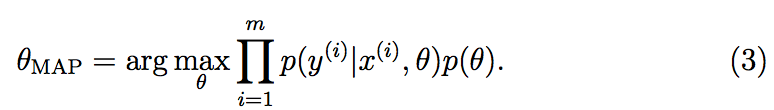






The procedure that we’ve outlined here can be thought of as doing “fully Bayesian” prediction, where our prediction is computed by taking an average with respect to the posterior p(θ|S) over θ. Unfortunately, in general it is computationally very difficult to compute this posterior distribution. This 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.