

Machine Learning I

Lecture 21: Validation and Model Selection

Nathaniel Bade

Northeastern University Department of Mathematics

Table of contents

1. Hyperparameter Tuning: Algorithms
2. Managing Accuracy For Large Numbers of Models
3. Comparing Model Performance Using Statistical Tests
4. Bayesian Optimization

Hyperparameter Tuning: Algorithms

Hyperparameter Tuning

Most machine learning algorithms come with a large number of settings that we must pick ourselves. These **hyperparameters** are the numerical class parameters that the training of a model depends on:

- The k in k nearest neighbors.

- The depth of a decision tree, as well as the pruning algorithm.

- The number and size of hidden layers in a neural network.

- The type of activation function in a perceptron.

- The size η of the training shrinkage.

To tune hyperparameters, we fit a large set of models and rank them according to our estimate of how well they generalize.

Hyperparameter Tuning

“... hyper-parameter optimization should be regarded as a formal outer loop in the learning process. A learning algorithm, as a functional from data to classifier (taking classification problems as an example), includes a budgeting choice of how many CPU cycles are to be spent on hyper-parameter exploration, and how many CPU cycles are to be spent evaluating each hyper-parameter choice (i.e. by tuning the regular parameters). The results [...] suggest that with current generation hardware such as large computer clusters and GPUs, the optimal allocation of CPU cycles includes more hyper-parameter exploration than has been typical in the machine learning literature” [Bergstra et al, Algorithms for Hyper-Parameter Optimization]

<https://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf>

Goals of Hyperparameter Tuning

The goals of hyperparameter tuning are roughly threefold:

We want to estimate the **generalization accuracy**, or **true accuracy** of our model, that is the predictive performance of our model on unseen data.

We want to increase the predictive performance of our model.

We want to identify the algorithm that is best suited for the project at hand. That is we, want to compare not only the same model with different hyperparameters, but different models.

Common Algorithm Families

The three most common methods of organizing the hyperparameters θ of a hypothesis class are via

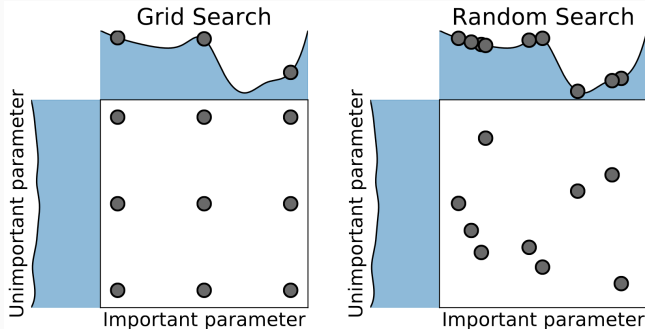
Grid Search: Specify a discrete range of values for each parameter $\theta_\ell \in \{\theta_{\ell,1}, \dots, \theta_{\ell,M_\ell}\}$. We then train $M = \prod_{\ell=1}^L M_\ell$ models by an exhaustive search over all tuples $(\theta_1, \dots, \theta_L)$.

Random Search: Test hyperparameters are drawn from a (discrete or continuous) range of values $\theta_\ell \in [\theta_{\ell,min}, \theta_{\ell,max}]$ and a distribution over each range.

Population Based Algorithms: Generate a population of models. Analyze the population to generate new model parameters.

Bayesian Optimization: Construct a simpler function that estimates the best fit parameters. Iteratively use this function to estimate the new hyperparameters, and evaluate the model fitted with the hyperparameters.

Holdout Method



Between grid search and random search, random search tends to be preferred for a completely open problem spaces as it is more likely to find a “best minimum“ when the variable scale is unknown. Libraries to help automate both can be found in Sci-kit learn. (Image Source)

Population Based Algorithms

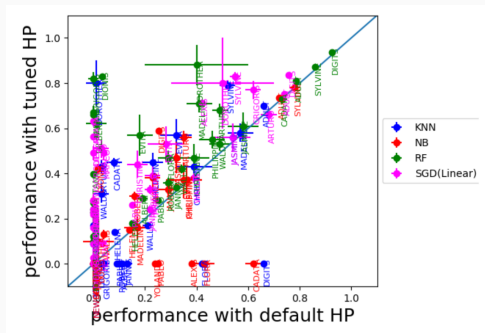
Grid Search and random search can be extended in a number of ways:

Multiple Level Search: After the first parameter search, use the best models to form a better guess for the parameter ranges and search again.

Simulated Annealing: Instead of picking a random values in parameter space, begin at a random point $\theta^{(0)}$ in parameter space sequentially jump to $\theta^{(m)} = \theta^{(m-1)} + \lambda\epsilon$, accepting the new value with some probability depending on λ . The annealing comes from slowing taking λ to 0, which freezes the parameter values.

Genetic Algorithms: A genetic algorithm proceeds by combining the best solutions and dropping the worst from the population. After initializing a population of L models, the best K models are randomly combined by choosing pairs and probabilistic averaging (or drawing) new parameters from the old. Some mutation is often added to avoid local minima.

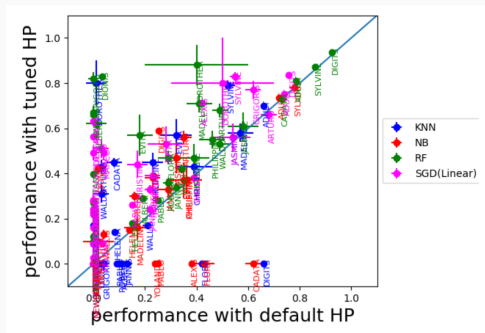
Population Based Algorithms



Hyperparameter tuning is very much a developing field. For example, AutoML found in their hyperparameter tuning competition that most of the successful methods searched over all available models in the sci-kit learn library using default values. Once a “best model” was found, flat tuning methods tended to be used afterwards.

<https://www.automl.org/wp-content/uploads/2018/12/challenge.pdf>.

Population Based Algorithms



The chart above displays the results of the hyperparameter tuning challenge on different types of data sets. Predictors that failed to return results in the allotted time are set to 0. We see that SGD and random forests (RF) benefit the most from hyperparameter tuning, but that naive Bayes (NB) and K nearest neighbors can also be improved, sometimes drastically.

Managing Accuracy For Large Numbers of Models

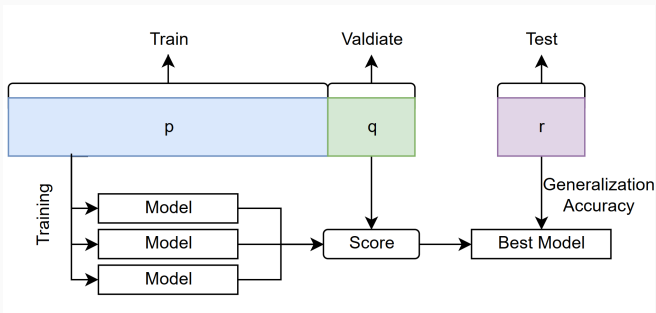
Holdout Method

Consider the following situation: You hold a contest for a position at your new firm where you ask 1000 people to submit algorithms forecasting whether the top 5 stocks will go up or down. You estimate there is $\frac{1}{2}$ chance for each stock to raise or fall so the probability of a person who picks the right result having gotten it by random chance is $(.5)^5 = 0.03125$, correct?

No, the probability of any one person getting a correct by chance is 0.03125. In fact we would expect 31 people to get the result by random. This is something we have to be careful of when hyper parameter tuning, if we throw too many models at a training set, we will expect some to fit by chance.

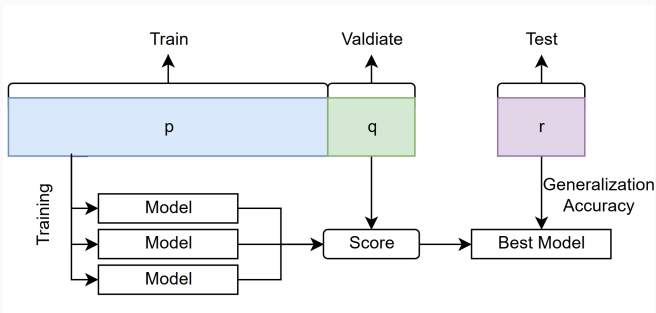
The **holdout method** and **K-folds cross validation** both try to rectify the over fitting problem. The source for this part of the talk is (Rashka, 2016).

Holdout Method



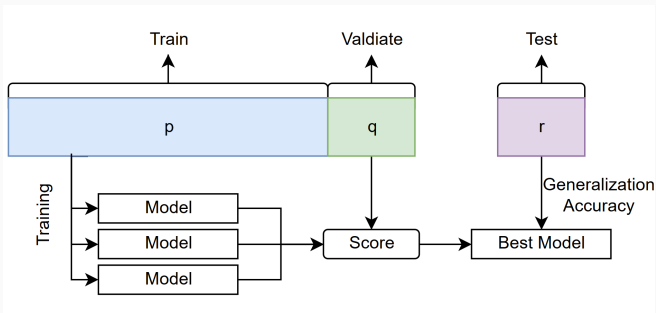
In the **three way holdout method**, or **train-test-validation split**, we separate off both a training and validation portion of the data. For each set of hyperparameters, we train on the training data, and score them on the validation data. Finally, we pick the best model and evaluate it on the test data.

Holdout Method



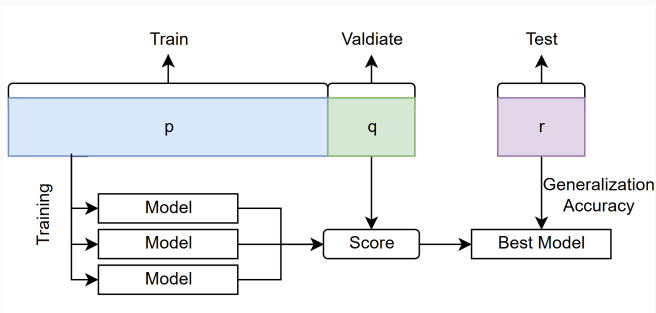
If we reuse the test set multiple times, we effectively start to train against it and the test set is said to “leak information.” The three way holdout method allows us to both fine tune the generalization accuracy our model, and to test the generalization accuracy of the fine tuned model in a way that avoids overfitting.

Holdout Method



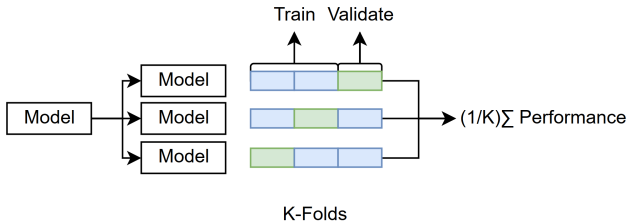
Once we have found a best fit $\hat{\theta}$ for our hyper parameters, and verified the generalization accuracy of best fit model, we may recombine all of the training data and train the model up on the whole dataset before using it in production.

Holdout Method



The holdout method is probably the best the use provided you have enough data. With enough data, you get genuine generalization information without any worry that you've introduced over fitting. However, often in machine learning we have nowhere near enough data. In these cases we may not be able to afford training on a fraction of our data set.

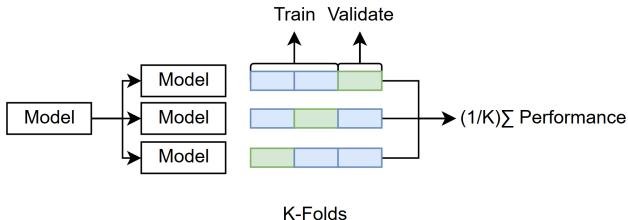
K-Fold Cross-Validation



The most common technique for model validation is called **K-fold cross validation**. The idea behind cross validation is to split the dataset into K subsets and train a model on each combination of $K - 1$ subsets, using the last for validation.

This has the advantage that every piece of data is used in training.

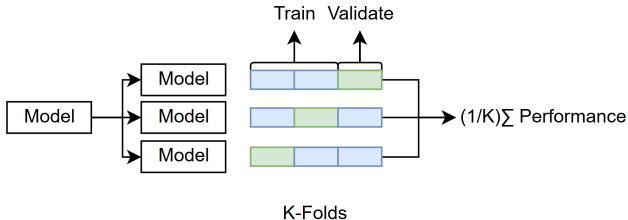
K-Fold Cross-Validation



In hyperparameter tuning, for each set of trial values θ_m we fit h_{θ_m} on all K -folded training sets. This results in K models $h_{\theta_m,k}$. The performance is evaluated as the average performance across all training folds

$$\text{Performance}(h_{\theta_m}) = \frac{1}{K} \sum_{k=1}^K \text{Performance}(h_{\theta_m,k}).$$

K-Fold Cross-Validation



The main advantage of K-fold cross validation is that all data is used for both training and testing. This allows us to train without having to lose a large amount of data to validation and testing. It also reduces overall variance by ensuring that all of our data is accounted for at some point during the training-validation process, so our best model is more likely to reflect the whole data set.

K-Fold Cross-Validation

Of course K is now a parameter we must decide. Usually, K is kept low $k \approx 5$ for training speed. There are two special cases that are worth mentioning:

For $K = 2$, this is almost like a train-test split, except that we fit models to both the “training” and “test” set. This is commonly used especially when the cost of training is high.

When $K = N$, we called the training **leave-one-out** cross validation (**LOOCV**). While computationally expensive, LOOCV can work very well on small datasets or easy to train models.

What is the best method for choosing K ?

In 2003, Bengio and Grandvalet showed that there is a form of the no free lunch theorem for the variance of K -fold cross validation.

Practically, broke the they variance down into a total variance σ , an in K -block covariance ω and a between K -block covariance γ . The then demonstrated that no unbiased estimator can be simultaneously constructed from the errors $L(y_i, \hat{y}_i)$.

However, we can still use cross validation for any K that is computationally feasible. Given the task, the sweet spot is probably the largest K for which the tuning is computationally feasible.

No Free Lunch

(ESLII) If $K = N$ the cross validation estimator is approximately unbiased, but has a high variance since each of the N training sets is so similar. There is also a considerable computational burden. In general, as we increase k ,

The bias decreases but the variance increases (we get a better fit but we start to overfit).

The computational cost increases due to the larger number of training steps on a larger training set.

On the other hand, if N is small and K is small (2 or 3) we may increase the variance and the bias due to random sampling effects.

Cross Validation vs Holdout Method

Which method should be used in practice? The mainstream opinion is to use cross validation whenever computationally feasible, and certainly on small datasets. Both ESLII and Raschka 2016 demonstrate the increase in accuracy using synthetic datasets.

On the other hand, if a sufficient amount of data is available the holdout method may be preferred for its computational simplicity. Since the expensive step is training the model, there is a tradeoff to the number of validation steps and the number of hyperparameter values we can test.

Finally, if computational time is really no issue one can get even more robust results by repeating the k-fold fitting with different random seeds. This lies somewhere in the forest between bootstrapping and validation, and we would expect it to lead to more robust tests of accuracy.

Occam's Razor and One Standard Error

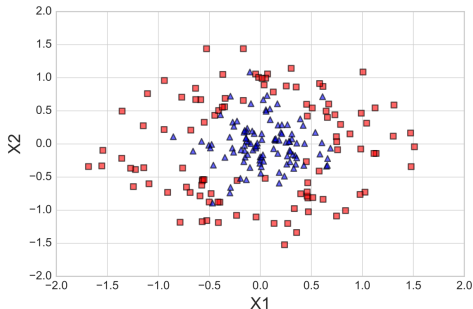
After generating hundreds of models and carefully gathering information on their performance, do we simply pick the one with the highest accuracy? The statistician will tell us to be patient and compute the confidence intervals. In fact, it may turn out that many models are within one standard error of the best fit. It's common to hearken back to Occam's Razor:

"Among competing hypotheses, the one with the fewest assumptions should be selected."

Practically, this can be implemented by choosing the simplest model within one standard error of the best model. Here, simplest may mean least degrees of freedom, most interpretable, easiest to implement, or least computationally expensive. It could also mean lowest VC dimension.

Lets look at an example due to Raschka.

Occam's Razor and One Standard Error

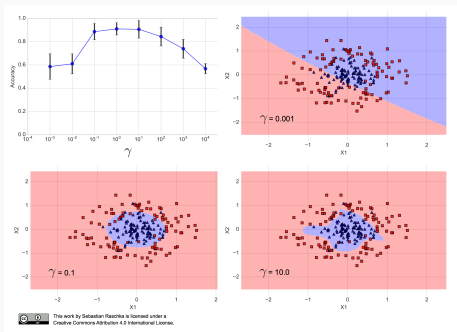


To see the one standard error method in practice, we generate 300 points of concentric circle data with a 70% 30% training test split. We want to optimize the Gaussian RBF kernel

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2), \quad \gamma > 0,$$

by fitting γ using grid search over $\gamma \in \{10^k\}$ for $k = -3, \dots, 4$.

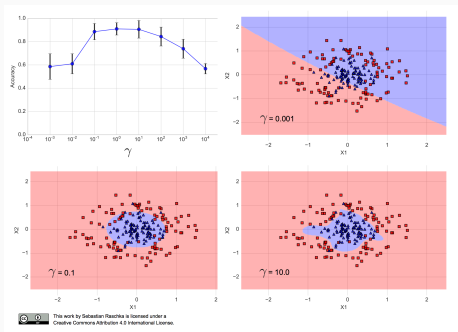
Occam's Razor and One Standard Error



The results of 10-fold cross validation, with 1 standard error bars is displayed in the top left chart. The point with the highest accuracy is $\gamma = 1$, with $\gamma = 10$ a close second and $\gamma = .1$ third but still within 1 standard error.

The decision boundary for $\gamma = 10$ is fairly complex, while the boundary for $\gamma = .1$ is simple, and likely to have lower variance.

Occam's Razor and One Standard Error



The one standard error rule tells us to use $\gamma = .1$.

The 1-SE rule is fine for a small number of hyper parameters, but once we have many models and large amounts of data we want to do more than eyeball the results. We will now discuss some ways to compare model accuracy statistically.

Comparing Model Performance Using Statistical Tests

Difference of Proportions

The simplest test is to perform a **z-score test** on the difference of proportion of accuracy. The pooled z-score for two models with accuracy $p_1 \in [0, 1]$ and $p_2 \in [0, 1]$ respectively is

$$z = \frac{p_1 - p_2}{\sqrt{2\sigma^2}} .$$


Here, since p_i is a proportion the variance is estimated from the pooled accuracy $p_{12} = (p_1 + p_2)/2$ as

$$\sigma^2 = \frac{p_{12}(1 - p_{12})}{2N}$$

The z-score is then compared to the z score of the standard confidence level and rejected if it falls outside it. Unfortunately, empirical tests show that the z-test tends to have a high false positive rate, often reporting classifiers as different when in fact they are not.

McNemar Test

		Model 2 correct	Model 2 wrong
Model 1 correct	A	B	
Model 1 wrong	C	D	

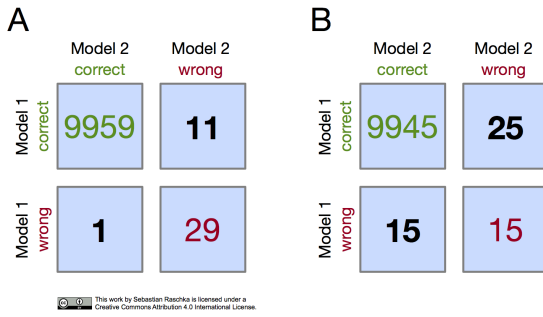
 This work by Sebastian Raschka is licensed under a Creative Commons Attribution 4.0 International License.

Instead of using the z-test on proportions, (Dietterich, 1998) found that **McNemar test** gives better results. For two models, a joint accuracy table is computed (see above). The individual model accuracy is

$$p_1 = \frac{A + B}{N}, \quad p_2 = \frac{C + D}{N}.$$

We want to compare the off diagonal elements to see how the fits differ.

McNemar Test

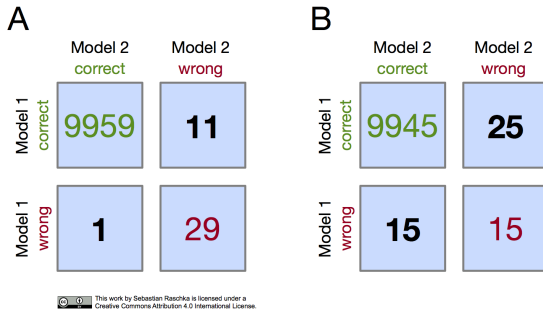


For example, in both cases above, $p_1 = .997$ while $p_2 = .996$. The **McNemar test** statistic compares them by evaluating

$$\chi^2 = \frac{(B - C)^2}{B + C},$$

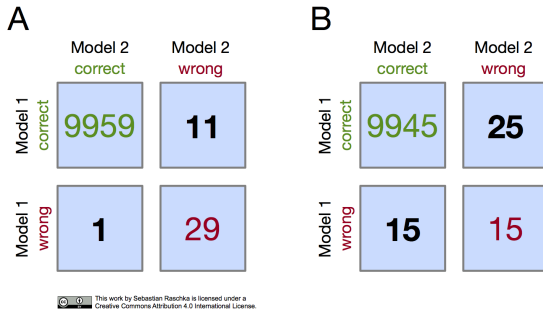
according to a χ^2 distribution with 1 degree of freedom. In the example above, $\chi_A^2 = 8.3$ has a p_A value of 0.0039 while $\chi_B^2 = 2.5$ has $p_B = 0.1138$.

McNemar Test



The **McNemar test** has the advantage of being able to see more than the accuracy but joint breakdown of misclassification. The test sees model 1 in A as being different from model 2 because model 2 misclassifies what model 1 does (e.g. noise) and a proportionally significant number of additional things.

McNemar Test



In B, they are missclassifying largely disjoint sets. It is hard to know whether 1 performs better than 2 in general, or just on this data.

Multiple Hypotheses Testing

When we have large numbers of models whose performance is close enough to need numerical analysis, the standard statistical practice is to perform an **omnibus** test to determine with the null hypothesis that there is no significant difference between models.

If an omnibus test determines there is a significant difference, it does not tell you which model performs differently, just that (at least) one does. To determine the odd model out, you have to perform posthoc testing, comparing each model (or subsets of models).

Multiple Hypotheses Testing

Cochran's Q test is an omnibus generalization of McNemars test. It is similar to ANOVA but works for categorical data. It again is a χ^2 test with $M - 1$ degrees of freedom where M is the number of models. The statistic is given by

$$Q = (M - 1) \frac{M(\sum_{m=1}^M G_m^2) - T^2}{MT - \sum_{i=1}^n M_j^2}$$

Here, T is the total number of correct votes by all classifiers, G_m is the correct number of votes for model m , and M_j is the number of classifiers that classified x_i correctly. Indeed, in the notation before,

$T = 2A + B + C$, $G_1 = A + B$, and $G_2 = A + C$, so when $M = 2$,

$$Q = \frac{(B - C)^2}{B + C}.$$

Dietterich's 5×2 -Fold Cross-Validated Paired t -Test

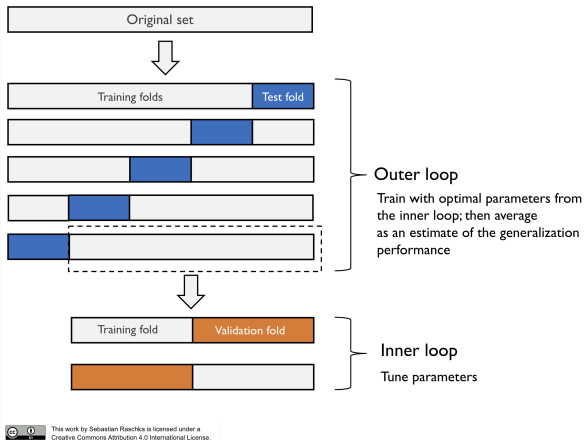
All of the test discussed so far assume that we have a relatively large, labeled test set on which to compare the models. If we do not, we can again use resampling and cross validation to our advantage. The simplest idea would be to use a paired t -test for small data sizes, but cross validation and re-sampling badly violate the independence assumption for t -test.

The 5×2 -Fold Cross-Validated Paired t -Test (Dietterich, 1998) dodges this as follows: In step A, split the test and training sets in half $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$, training model 1 on \mathcal{T}_1 and model 2 on \mathcal{T}_2 . We then validate by swapping models. In step B, rotate the sets, training model 1 on \mathcal{T}_2 . We perform 5 such splittings to compute the variance.

The t -statistic is

$$t = \frac{p_A^{(1)} + p_B^{(1)}}{\sqrt{\frac{1}{5} \sum_{i=1}^5 s_i^2}}.$$

Nested Cross-Validation



A popular modern extension of 5×2 -fold cross validation is to mix it with k fold cross validation. This works especially well on small datasets and is thought to reduce the variance in the estimation.

Breakdown of Cross Validation Algorithms

Raschka has the following breakdown the algorithms we've discussed in this section:

McNemar's test: low false positive rate, fast, only needs to be executed once.

The difference in proportions test: high false positive rate, cheap to compute, well known outside of machine learning.

5x2cv paired t-test: low false positive rate (similar to mcnemar), slightly more powerful than McNemar's test; recommended if computational efficiency is not an issue.

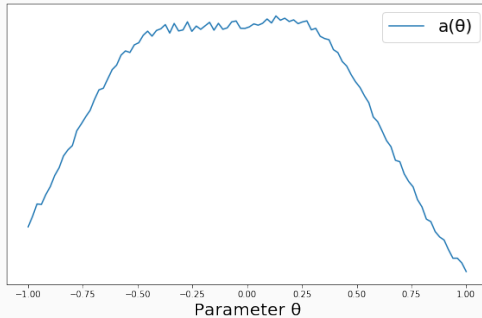
Nested Cross-Validation: Computationally intense, but the most robust test for small amounts of data.

Cochran's Q Test: Omnibus test for comparing large numbers of models.

These are implemented in the mlxtend library.

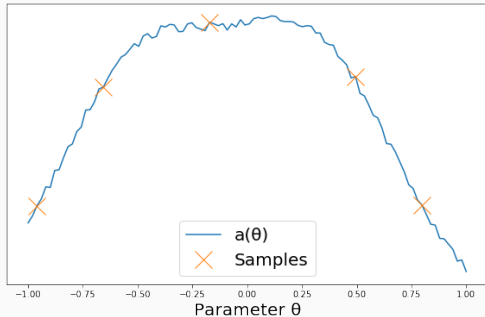
Bayesian Optimization

Bayesian Optimization



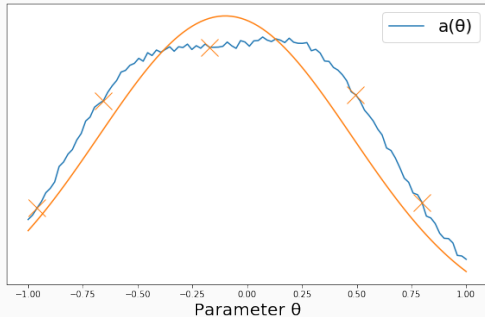
Imagine the following situation: we are trying to optimize a function $a(\theta)$ without access to derivatives, and where $a(\theta)$ is expensive to compute.

Bayesian Optimization



Imagine the following situation: we are trying to optimize a function $a(\theta)$ without access to derivatives, and where $a(\theta)$ is expensive to compute. As a result we can only draw a few samples $a(\theta)$ on which to base our decision.

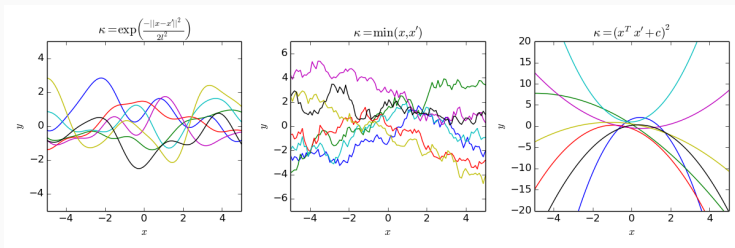
Bayesian Optimization



One way to proceed is to assume a simple model on the samples that's relatively easy to compute, and use the model to make an educated guess for the next best parameter $\theta^{(n)}$. We can then check $a(\theta^{(n)})$, update our model, and repeat.

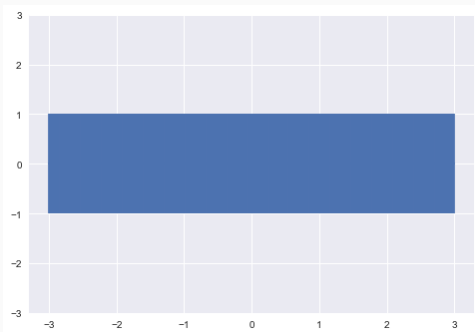
This is the central idea of Bayesian optimization, to estimate an expensive task by an inexpensive one that yields a similar maximum.

Gaussian Process



The most common model in Bayesian optimization is a Gaussian process (GP). Without going too deep, a Gaussian process is a kernel method that assumes that any set of samples (X_1, \dots, X_N) form a multivariate Gaussian random variable. The proceeds is then defined entirely by the covariance between points, some examples of data drawn from a GP are shown above.

Gaussian Process

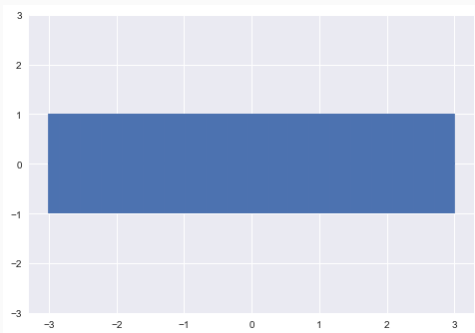


A Gaussian process depends on a pointwise mean $\mu(x)$ and the kernel $K(x, x')$. For example, if $\mu(x) = 0$ and

$$\text{Cov}(x, x') = K(x, x') = \beta_1 \exp\left(-\frac{\beta_2}{2}(x - x')^2\right)$$

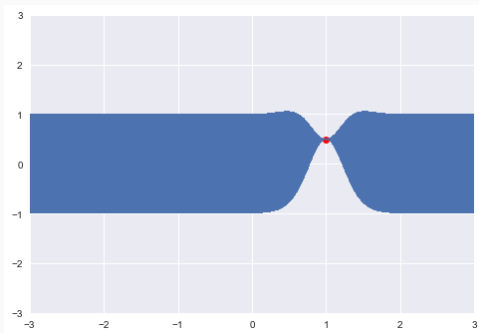
the fit initially assumes a process can be drawn uniformly from a strip as above.

Gaussian Process



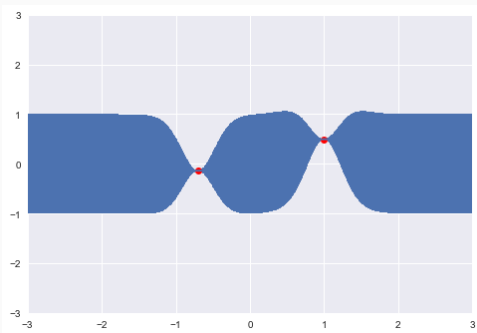
But as we sample points (x_i, y_i) , the probability is restricted around those points by the prescribed covariance $K(x, x')$.

Gaussian Process



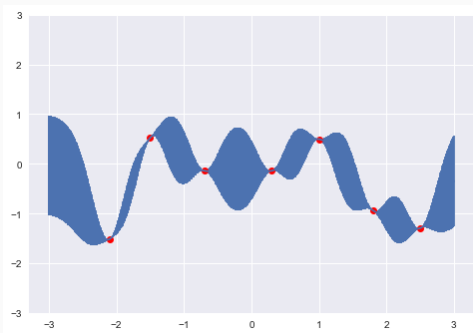
But as we sample points (x_i, y_i) , the probability is restricted around those points by the prescribed covariance $K(x, x')$.

Gaussian Process



But as we sample points (x_i, y_i) , the probability is restricted around those points by the prescribed covariance $K(x, x')$.

Gaussian Process



But as we sample points (x_i, y_i) , the probability is restricted around those points by the prescribed covariance $K(x, x')$.

Sequential Model Base Optimization

Sequential Model Base Optimization (SMBO) uses a Gaussian process as a model to estimate the best new hyperparameter based on previous observations.

Let $a(\theta)$ be the accuracy of a hypothesis class with hyperparameters θ fit to a training set \mathcal{T} . Let \mathcal{X}_θ be the hyperparameter domain and \mathcal{M} be a Gaussian process modeling $a(\theta)$.

Finally, pick an acquisition function S that picks new points based on the fit model \mathcal{M} . S is usually chosen to be the Expected Improvement

$$\mathbf{EI}_{a^*}(\theta) = \int_{\mathbb{R}} \max(a^* - a, 0) p_{\mathcal{M}}(a|\theta) da.$$

The SMBO algorithm picks new model parameters as follows:

$\mathcal{D} = \text{Initial Samples}$

for $|\mathcal{D}| < L$:

$p(a|\theta, \mathcal{D}) = \text{Fit } \mathcal{M} \text{ to } \mathcal{D}.$

$\theta_j = \operatorname{argmax}_{\theta} S(\theta, p(a|\theta, \mathcal{D})).$

$a_j = a(\theta_j)$. This is the expensive step.

$\mathcal{D} = \mathcal{D} \cup (\theta_j, a_j).$

Sequential Model Base Optimization

The SMBO is tuned by varying the regression model \mathcal{M} and the acquisition function S :

Software	Regression Model	Acq. Function
Spearmint	Gaussian Process	Exp. Improv
MOE	Gaussian Process	Exp. Improv
Hyperopt	Tree Parzen Est.	Exp. Improv
SMAC	Random Forest	Exp. Improv

https://app.sigopt.com/static/pdf/SigOpt_Bayesian_Optimization_Primer.pdf

The hyperopt library seems to be the most robust, and has been extended to work with Keras. It also plugs into some visualization libraries: <https://github.com/dvgodoy/deepreplay>

Sequential Model Base Optimization

If you're interested in SMBO, there's a lot of work to be done but you should start with AutoML's paper:

<https://www.automl.org/wp-content/uploads/2018/11/hpo.pdf>

For automated data pipelines with genetic algorithms, take a look at TPOT: <https://github.com/EpistasisLab/tpot>

Finally, for a list of <https://medium.com/@mikkokotila/a-comprehensive-list-of-hyperparameter-optimization-tuning-solutions>

Happy tuning!

References

No Free K-fold Lunch: [http:](http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.9.3582)

[//citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.9.3582](http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.9.3582)

Nested Cross Validation Code:

https://scikit-learn.org/stable/auto_examples/model_selection/plot_nested_cross_validation_iris.html

https://github.com/rasbt/model-eval-article-supplementary/blob/master/code/nested_cv_code.ipynb

List of Libraries for Hyperparameter tuning:

<https://medium.com/@mikkokotila/>

[a-comprehensive-list-of-hyperparameter-optimization-tuning-solu](https://medium.com/@mikkokotila/a-comprehensive-list-of-hyperparameter-optimization-tuning-solutions)

Gaussian Processes: <https://blog.dominodatalab.com/fitting-gaussian-process-models-python/>