dummies
A Wiley Brand

10 Ways to Improve Your Machine Learning Models

SHARE

By John Paul Mueller, Luca Massaron

Now that you're machine learning algorithm has finished learning from the data obtained using Python or R, you're pondering the results from your test set and wondering whether you can improve them or have really reached the best possible outcome. There are a number of checks and actions that hint at methods you can use to improve machine learning performance and achieve a more general predictor that's able to work equally well with your test set or new data. This list of ten techniques offers you opportunities to improve the outcome achieved using machine learning algorithms.

Studying learning curves

As a first step to improving your results, you need to determine the problems with your model. Learning curves require you to verify against a test set as you vary the number of training instances. You'll immediately notice whether you find much difference between your in-sample and out-of-sample errors. A wide initial difference is a sign of estimate variance; conversely, having errors that are both high and similar is a sign that you're working with a biased model.

TIP

Python helps you easily draw learning curves using the Scikit-learn() function. You can also easily achieve the same result using R with custom functions, as described by the Revolution analytics blog.

Using cross-validation correctly

Seeing a large difference between the cross-validation (CV) estimates and the result is a common problem that appears with a test set or fresh data. Having this problem means that something went wrong with the cross-validation. Beyond the fact that CV isn't a good performance predictor, this problem also means that a misleading indicator has induced you to model the problem incorrectly and achieve unsatisfactory results.

REMEMBER

Cross-validation provides you with hints when the steps you take are correct. It's important, but not critical, that CV estimates precisely replicate out-of-sample error measurements. However, it is crucial that CV estimates correctly reflect improvement or worsening in the test phase due to your modeling decisions. Generally, there are two reasons that the cross-validation estimates can vary from the true error results:

- Snooping
- Incorrect sampling

Python offers a **stratified-k-folds CV sampler**. R can stratify samples using the **createFolds method of the caret library** when you provide the y parameter as a factor.

Choosing the right error or score metric

Trying to optimize an error metric based on the median error by using a learning algorithm based on the mean error won't provide you with the best results unless you manage the optimization process in a fashion that works in favor of your chosen metric. When solving a problem using data and machine learning, you need to analyze the problem and determine the ideal metric to optimize.

Examples can help a lot. You can get many of them from academic papers and from public machine learning contests that carefully define specific problems in terms of data and error/score metric. Look for a contest whose objective and data are similar to yours, and then check the requested metric.

Searching for the best hyper-parameters

Most algorithms perform fairly well out of the box using the default parameter settings. However, you can always achieve better results by testing different hyper-parameters. All you have to do is to create a grid search among possible values that your parameters can take and evaluate the results using the right error or score metric. The search takes time, but it can improve your results.

TIP

When a search takes too long to complete, you can often achieve the same results by working on a sample of your original data. Fewer examples chosen at random require fewer computations, but they usually hint at the same solution. Another trick that can save time and effort is to do a randomized search, thus limiting the number of hyper-parameter combinations to test.

Testing multiple models

As a good practice, test multiple models, starting with the basic ones — the models that have more bias than variance. You should always favor simple solutions over complex ones. You may discover that a simple solution performs better.

TIP

Representing the performance of different models using the same chart is helpful before choosing the best one to solve your problem. You can place models used to predict consumer behavior, such as a response to a commercial offer, in special gain charts and lift charts. These charts show how your model performs by partitioning its results into deciles or smaller parts.

Because you may be interested only in the consumers who are most likely to respond to your offer, ordering predictions from most to least likely will emphasize how good your models are at predicting the most promising customers. These Quora answers help you see how gain and lift charts work: **What's ROC Curve?** and

What’s Lift Curve?.

Testing multiple models and *introspecting* them can also provide suggestions as to which features to transform for feature creation, or which feature to leave out when you make feature selections.

Averaging models

Machine learning involves building many models and creating many different predictions, all with different expected error performances. It may surprise you to know that you can get even better results by averaging the models together. The principle is quite simple: Estimate variance is random, so by averaging many different models, you can enhance the *signal* and rule out the noise that will often cancel itself.

REMEMBER

Sometimes the results from an algorithm that performs well, mixed with the results from a simpler algorithm that doesn’t work as well, can create better predictions than using a single algorithm. Don’t underestimate contributions delivered from simpler models, such as linear models, when you average their results with the output from more sophisticated algorithms, such as gradient boosting.

Stacking models

For the same reasons that averaging works, stacking can also provide you with better performance. In stacking, you build your machine learning models in two stages. Initially this technique predicts multiple results using different algorithms, with all of them learning from the features present in your data. During the second phase, instead of providing features that a new model will learn, you provide that model with the predictions of the other, previously trained models.

Using a two-stage approach is justified when guessing complex target functions. You can approximate them only by using multiple models together and then by combining the result of the multiplication in a smart way. You can use a simple logistic regression or a complex tree ensemble as a second-stage model.

TIP

The Netflix competition provides evidence and a detailed illustration about how heterogeneous models can be stacked together to form more powerful models. However, implementing this solution as a working application can be quite cumbersome.

Applying feature engineering

If you believe that bias is still affecting your model, you have little choice but to create new features that improve the model's performance. Every new feature can make guessing the target response easier.

Automatic feature creation is possible using polynomial expansion or the support vector machines class of machine learning algorithms. Support vector machines can automatically look for better features in higher-dimensional feature spaces in a way that's both computationally fast and memory optimal.

However, nothing can really substitute for your expertise and understanding of the method needed to solve the data problem that the algorithm is trying to learn. You can create features based on your knowledge and ideas of how things work in the world. Humans are still unbeatable in doing so, and machines can't easily replace them.

Selecting features and examples

If estimate variance is high and your algorithm is relying on many features, you need to prune some features for better results. In this context, reducing the number of features in your data matrix by picking those with the highest predictive value is advisable.

When working with linear models, linear support vector machines, or neural networks, regularization is always an option. Both L1 and L2 can reduce the influence of redundant variables or even remove them from the model. Stability selection leverages the L1 ability to exclude less useful variables. The technique resamples the training data to confirm the exclusion.

TIP

You can learn more about stability selection by viewing the example on the **Scikit-learn website**. In addition, you can practice using the RandomizedLogisticRegression and RandomizedLasso Scikit-learn functions in the linear_model module.

Looking for more data

After trying all the previous suggestions, you may still have a high variance of predictions to deal with. In this case, your only option is to increase your training set size. Try increasing your sample by providing new data, which could translate into new cases or new features.

If you want to add more cases, just look to see whether you have similar data at hand. If you want to add new features, locate an open source data source, if possible, to match your data with its entries. Another great way to obtain both new cases and new features is by scraping the data from the web. Often, data is available between different sources or through an application programming interface (API). For instance, **Google APIs** offer many geographical and business information sources.

About the Book Author

John Paul Mueller is a prolific freelance author and technical editor. He's covered everything from networking and home security to database management and heads-down programming.

Luca Massaron is a data scientist who specializes in organizing and interpreting big data, turning it into smart data with data mining and machine learning techniques.