Train, Test, Validation

Intro

In software products, test cases verify the correctness of the software and flesh out any bugs before they make it to production. In a similar sense, given a dataset to train your machine learning model, there is a need to verify if our machine learning model is generalizing correctly and not overfitting or underfitting to the data. For example, if you are deploying a classifier that differentiates between various types of animals, it is worth testing if the classifier can differentiate between house cats and wild cats like cheetahs. On the flip-side, it is important to test if the model is able to classify house cats *as house cats* despite differences in fur and size. In the following sections, we will go through a basic overview of how machine learning models are trained and tested to ensure optimal performance.

When developing a model, your dataset is split into two distinct parts: train and test. Usually, the split is 80% train and 20% test. The 80-20 split is no hard-fast rule and can be adjusted based on your needs like the size of data, etc. AutoAI allows you to customize this split when the data is loaded.

Terms:

1. Test Set/Holdout set)
2. Cross Validation
3. Overfitting (High Variance) - when the model fits the training data too well but can't generalize to new, unseen data. An easy way to tell this happening is if training error is low, but test error is high.
4. Underfitting (High Bias) - when the model fits poorly to the train data. Underfitting occurs when there are high training and testing error.

1. Test / Holdout Set

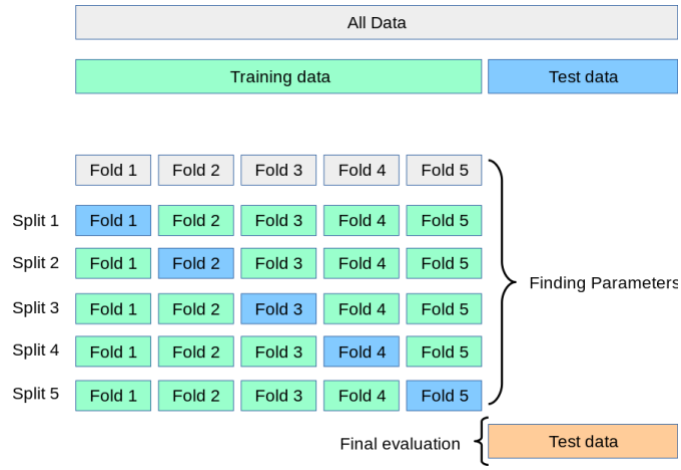
The test set is a blind set. You are **not allowed** to look at the data inside the test set so that you reduce the possibility of bias creeping into your model. After your model is trained, you run it against the test set to see how the model performs against data it has never seen before. If the model performs well, then congrats! Your model generalized well. If it did not perform well, then that's ok! Train a different type of model (Random Forest instead of Linear Regression, for example) or fine-tune the hyper-parameters. Be careful of repeatedly testing the model against the test set because you run the risk of overfitting to the test set. A good rule of thumb is to test your model only when you feel that it is 100% ready. In fact, in many data science competitions such as the ones hosted on Kaggle, you are only allowed to submit your model for evaluation against their test set three times per day.

For example, given our classifier to differentiate between various animals, we run our model against the test set. It performs poorly, we can't look inside the test set since that defeats the purpose of the test set being blind, so we go back and add more training examples and run it again. It performs better, but not up to par so we repeat this process and test it again. Eventually, our classifier performs amazingly on the test set. We deploy the model and see that the model starts to perform horribly. What went wrong? Our model performed well on the *test set* but failed to generalize to the distribution of the data. The test set is a very useful tool to measure the performance of your model, but if used carelessly it can be detrimental to your machine learning model.

This presents a question, if you are only allowed to test your model against the test set a few times, how can you continuously improve your model? Enter cross-validation.

2. Cross Validation

Cross-validation has similar goals to the holdout set, except the approach is slightly different. The model is split into k-folds. The model is trained on k-1 folds and evaluated on the remaining fold. This is done repeatedly until every fold has appeared in the test set. Typical values for k are 3, 5, or 10; however, k can be anything you want. In AutoAI, k is three.



In our animal classifier example, we take our training data and split it into 5 folds. We do 5 iterations, and in each iteration, a different fold is chosen as the validation/test set, and the remaining is training set. The cross-validation score is the average of the performance of the model after k-iterations. Cross-validation is very important because you can quickly detect signs of overfitting and underfitting. Here you can see if our animal classifier is able successfully differentiate between house casts and wild cats for example because you are allowed to look inside. If the classifier is struggling to do that, then taking steps like adding more training data would improve performance.

Your cross-validation score is the average of the performance of the model against the k-test sets. The performance of the model is dependent on the metric you have chosen to optimize for given your problem. For regression, the most common performance measures are mean-squared-error, mean-absolute-error. For classification, common metrics include

precision, recall, or AUC. Details on metrics can be found in the *performance_metrics* blog post

Grid Search

Another use of cross-validation is to test various combinations of hyper-parameters (*see post on hyper-parameters)*. This use of cross validation is more commonly called grid-search because you are searching across a *grid* of hyper-parameters to find the best combination. Hyper-parameters are equivalent to settings of a model. Many models have multiple hyper-parameters, such as learning rate, regularization, etc. Often choosing the right value for hyper-parameters is guesswork where various values are tried until an optimal value is found. With cross-validation, you can pass a range of values for hyper-parameters. Through cross-validation, various combinations of the hyper-parameters are tested. The combination that results in the best cross-validation score is the best combination of hyper-parameters. See this post for more information on hyper-parameters.

Putting together Grid Search and Cross Validation

In our animal classifier, we are going perform 5 fold cross-validation. Our animal classifier also has 3 hyper-parameters that need to be set. For each hyper-parameter you test 4 different values.

Model A Grid Search

For Grid-search you train 64 unique configurations (4 x 4 x 4) , and on each configuration you train a model using 5-fold cross validation. This means to find the model with the optimal configuration, you will have trained 320 (4*4*4*5) models! Choosing k and testing the number of hyper-parameters is all a matter of tradeoffs. Choosing more hyperparameters to test, choosing a higher k will lead to better performance of the model at the cost of computing time and power. Finding the correct balance depends on the amount of resources available, budget, and performance expectations.

Cross-validation is useful for validating and testing your model and selecting the best hyper-parameters for a particular model.

Conclusion

Developing an effective evaluation methodology for developing and testing machine learning can mitigate against failures in production. We talked about splitting the data into training and testing, and using the training data for cross validation to iteratively improving your model while using the test set sparingly as a sanity check.

This is by no means a comprehensive list other specific testing measures exist for certain types of problems , but they are all built from these foundations

When your model is deployed into production, it is still not the end of the story! Machine learning relies on input from the outside world, and the outside world is always changing. Monitoring your models performance in production and making changes if necessary is just as important when developing the model.