LEARN DATA SCIENCE

Challenges of Generalization in Machine Learning

Author: Blaine Bateman / Posted on September 17, 2018

In predictive analytics, we want to predict classes for new data (e.g. cats vs. dogs), or predict future values of a time series (e.g. forecast sales for next month). We build models on existing data, and hope they extend, or *generalize*, to the future. In supervised learning, we have data from the past with all the predictor values and the true values we wish to predict. Although defining the business problem, gathering relevant data, cleaning and preparing the data, and building models are all challenging, a significant challenge remains—how to know if the model will predict the future well? Most tutorials talk about k-fold cross validation, splitting data into train, validation, and test sets, and similar topics. In this article, I will

review some limitations of those approaches and some things to think about in your own predictive analytics projects.

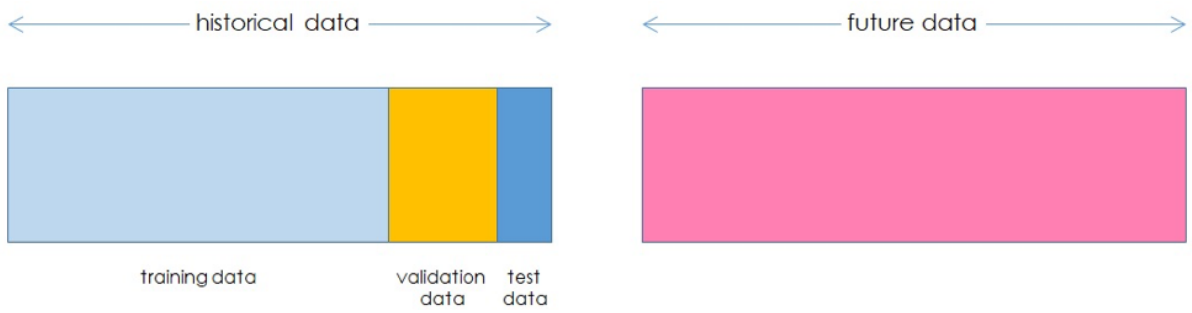

Figure 1: The standard approach to supervised learning is to split the historical data into training, validation, and test data, develop a model, validate performance on k-folds (or splits) of training/validation data, then do a final check on the test data. The results of the k-fold validation are taken as indicative of performance predicting future data.

Selecting the Best Model

It's always a good idea to try many models, within the time and resource constraints of a project. The problem then becomes one of selecting the best model for the desired prediction task. What defines best? Good practice dictates that the models only "see" the *training* data during learning, and the results of the model are compared based on the predictions for the *validation* data. The model giving the best validation results is selected as the best choice. Then, the model is trained on a number (denoted as k) of different subsets of the data, and the expected performance in the future is given by mean and standard deviation of the k validation results on the validation data, and a final check is done with the test data. Test data results are *not* used to tune the model!

K-fold validation seeks to avoid having the random choice of training data vs. validation data affect our expectations of predictive performance. In other words, if we got "lucky" and chose a train/validation split of the data that gave excellent

predictions on the validation, and we reported that, it is likely that future predictions won't be as accurate. To address this, multiple repeats are run and averaged, where the repeats are the "k-folds" of the data. A fold is just one split of the original data into train/validation/test.

How do you know that k-fold validation represents what your model will do in production? One way is to do a final run of the fully trained model on the test data. A challenge for smaller data sets is that you want to retain as much data for training as possible, but you need to hold out some validation data and some test data. A typical breakdown for small data sets might be 70%/20%/10% (train/validation/test) resulting in assessing model performance on small samples. This is mitigated for the validation set by the k-fold validation. If the validation results differ a lot from the final test results, the temptation might be to try to re-optimize for better test results—but that is viewed as poor practice. As soon as you begin taking into account the test data performance in the model development process, you are "leaking" information about the test data into the model. This *invalidates* the approach of using the test results as the final indicator of future performance, and simply trains your model to memorize all the prior data.

So how effective is it to use validation performance to choose a model, and k-fold validation to predict future accuracy? This question is part of a broader topic in machine learning called generalization. Note that generalization is goal-specific and likely project-specific. If you train an image recognition model on zoo animal images, then show it cars and buildings, you would not expect it to generalize. But if you showed it images from other zoos than those used in training, you would want it to perform well. Here I'll show some simple results that I hope will broaden your understanding of this challenge and how to think about it.

The Kaggle Titanic Data Set

I will be using the Kaggle competition Titanic data as the basis for building predictive models. In the Titanic competition, you are provided with data for some of the passengers of the ill-fated Titanic ocean liner. For developing models, data are provided for 891 passengers—in particular, whether they survived or died in the disaster. Data are provided for factors such as age, gender, point of embarkation, ticket class, ticket price, cabin, number of siblings or spouses on board, and number of parents or children on board. Additional data are provided for another 418 passengers, but without the survival status—these are what you are to predict in the competition. As survival is a binary status, the problem is a binary classification task. It is well known that models such as XGBoost and random forest perform well on these data, and that some form of ensemble or bagging can further improve prediction results. I will be comparing two model types—random forest and fully-connected neural networks.

Neural networks can be sensitive to the starting point (i.e. how all the weights are initialized to begin the training) and can produce different results for different initializations. Similar behavior is observed with random forest models due to random effects in searching the space for the model. The selection of folds can also introduce variations from one set of runs to another, if the folds vary. Generally, it is advised to take the folds in a uniform way stepping through the data. In most tutorials, you are advised to fix the "seed" for the random number generator in your programming language to avoid variations when trying to repeat runs. This can be especially useful for initial development until you know everything is working, but my view is that this can cause significant issues in measuring performance. Suppose you chose two seed values (arbitrarily) and ran a 10-fold validation of a model using each seed. Further, let's say in one case you get an expected error of 8% and the validation runs vary across them 3%, and in the other case you get 5% and 2%, respectively. You might be tempted to store the "best" seed as another parameter, but in fact you have no way of really knowing if that will generalize.

Figure 2: In k-fold validation, the validation data are chosen k times from the original data. The average performance and the variation are taken to predict performance on unseen data. The final trained model is used to predict the test data as a check.

A more rigorous approach would be to make many validations using random initialization and random splits, and use the average as the likely accuracy. This is typically not done on large data or complex models (like neural networks with millions of nodes) due to the computational cost. In fact, for very large data even k-fold validation is often not performed. If your data are small enough and you have time and resources, you should investigate the full behavior of your system as I'll demonstrate here. In cases where this isn't practical, it is good to keep these behaviors in mind.

Neural Network Results

I used Keras in R Studio to run TensorFlow models to classify the Titanic data. In my code, I parameterized most of the hyperparameters: batch size, learning rate, decay, L1 and L2 regularization, number of hidden layers, number of units, and in some cases the activation function and the optimizer. The train, validation, and test data sets were held constant to eliminate that source of variation (note the

difference from cross-validation, which would vary the train/val sets). Runs with identical parameters were also repeated in the experiments. Running an n-dimensional (n being the number of hyperparameters) grid-search generated the following data, which show the accuracy on the validation data vs. the accuracy on the training data.

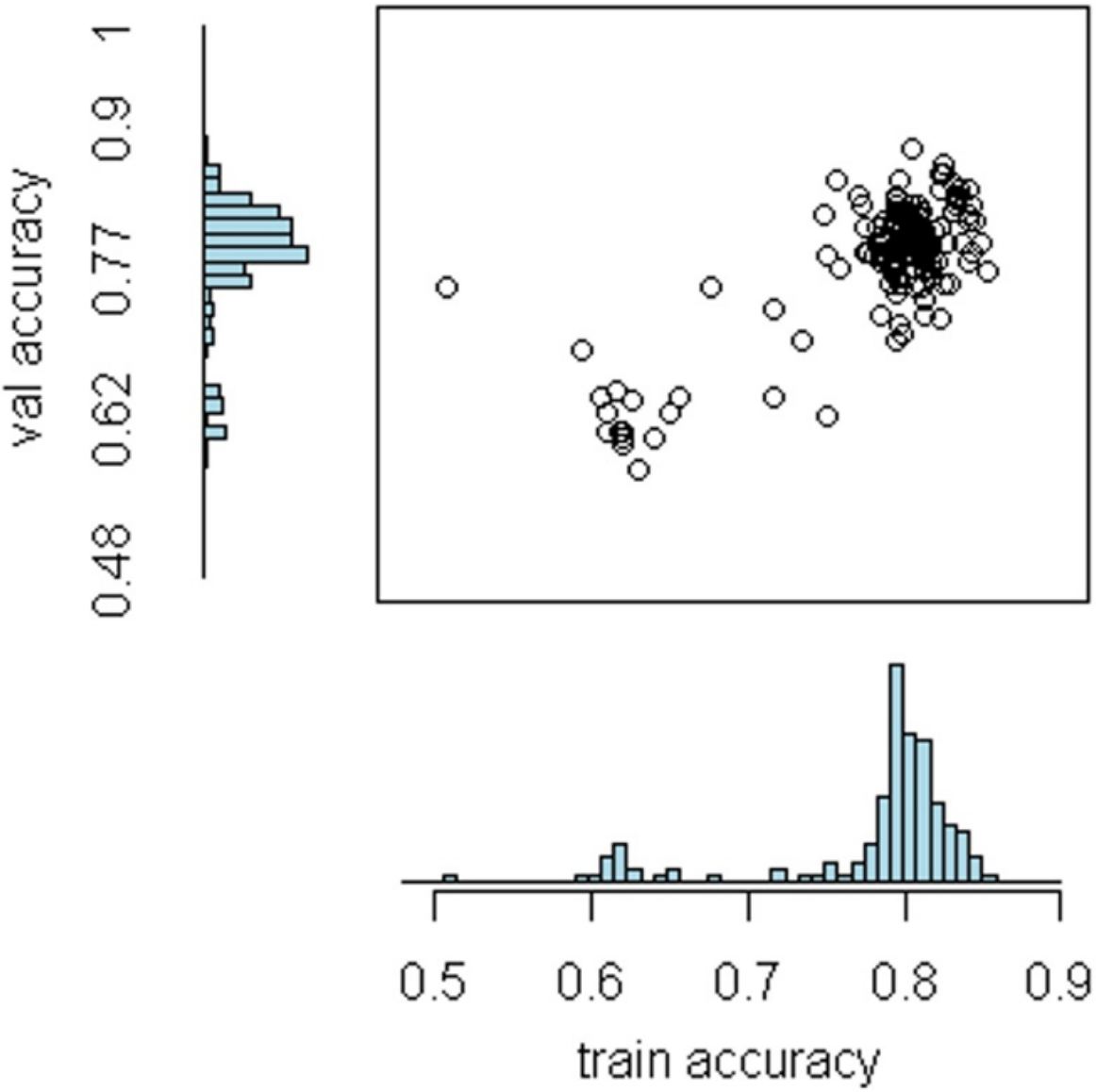

Figure 3: Validation accuracy vs. train accuracy for a grid search of hyperparameters of a neural network classification model of the Titanic data set. Repeat runs of some parameter sets are also included.

A couple concerns are immediately apparent. First of all, both distributions are bimodal, so we need to understand what the smaller cluster of points means. In addition, it is evident that instead of a smooth trend, we have a cloud of results,

meaning the validation accuracy does not smoothly track the training accuracy. Actually, the latter isn't an issue per-se; we don't use train accuracy as an indicator of model performance. Let's look at test accuracy vs. validation accuracy. Recall that the test results are data never seen by the model during training, and we do not optimize test results—only validation results.

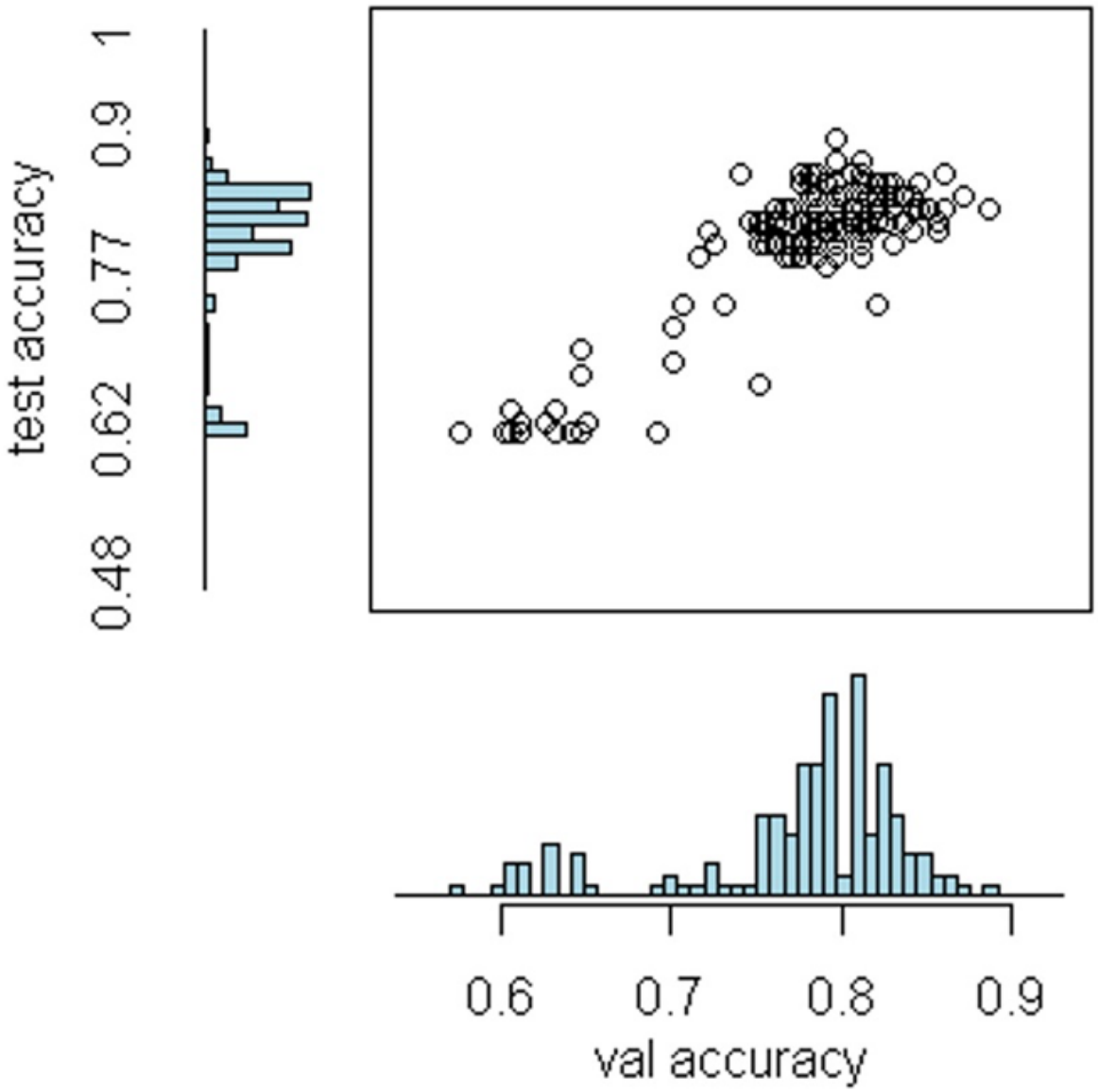

Figure 4: The test set results vs. validation results for the same runs as the previous figure. Although there is a bit more of a trend, we still see random distribution in both dimensions.

This chart shows somewhat more of a trend, but we still see the bimodal characteristics. Investigating that, I found that in some instances, the neural network would converge to a local optimum that was really far from the global

optimum, and thus perform poorly. The chart on the left shows the learning curves for the train/val data for a "normal" experiment; on the right are curves for a poorly converged experiment. This is a challenge in training neural networks in general.

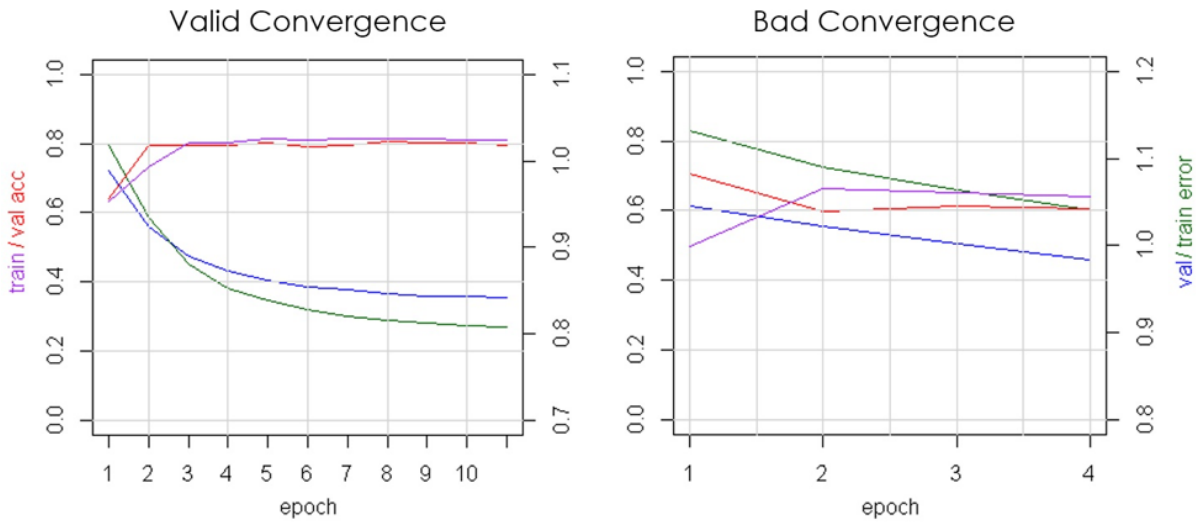

Figure 5: Learning curve for two types of convergence of the neural network model. On the left, the error decreases smoothly and the accuracy plateaus to about 0.8. On the right, the validation accuracy decreases then plateaus, indicating issues with the solution.

Removing all the results below accuracy about 0.7 gives the following results.

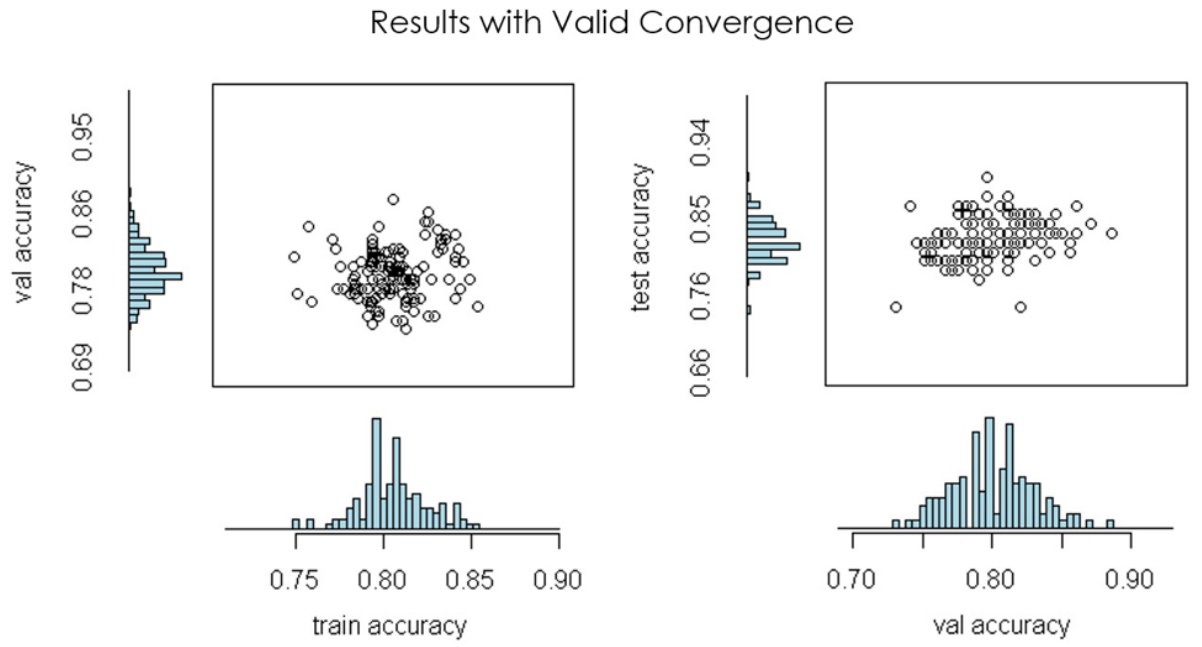

Figure 6: Prediction results using only valid convergence results from the same runs as previous.

We can now see that the test accuracy has a weak trend vs. the validation accuracy, but it is very clear that the best test accuracy is not achieved at the best validation accuracy. Given that conventional guidance is to select the model with the best validation accuracy, what should be done here? Before addressing this question, it could be criticized that using accuracy is usually not a good metric for evaluating model performance in classification problems, especially if the classes are unbalanced. In the Titanic training data, there are 342 survivors out of 891 passengers, or about 38 percent. This means if we simply predicted that everyone died, we could achieve an accuracy of 62 percent; hence the criticism of using accuracy to measure model performance. In this instance, I have used accuracy so far as that is the metric actually used in the Kaggle competition, but let's take a look at another metric, the F1 ratio. The F1 ratio is defined in terms of Precision and Recall as follows:

$$Precision = \frac{TP}{TP + FP}$$ Of the things I predicted positive, what fraction were correct?

$$Recall = \frac{TP}{TP + FN}$$ What fraction of the actual positives did I predict?

$$F1 = \frac{2 * P * R}{P + R}$$

The F1 ratio therefore strikes a balance between different measures of the prediction. If we then look at the F1 ratio of the test data vs. the F1 ratio of the validation data for the same experiments with the neural networks as above, we get:

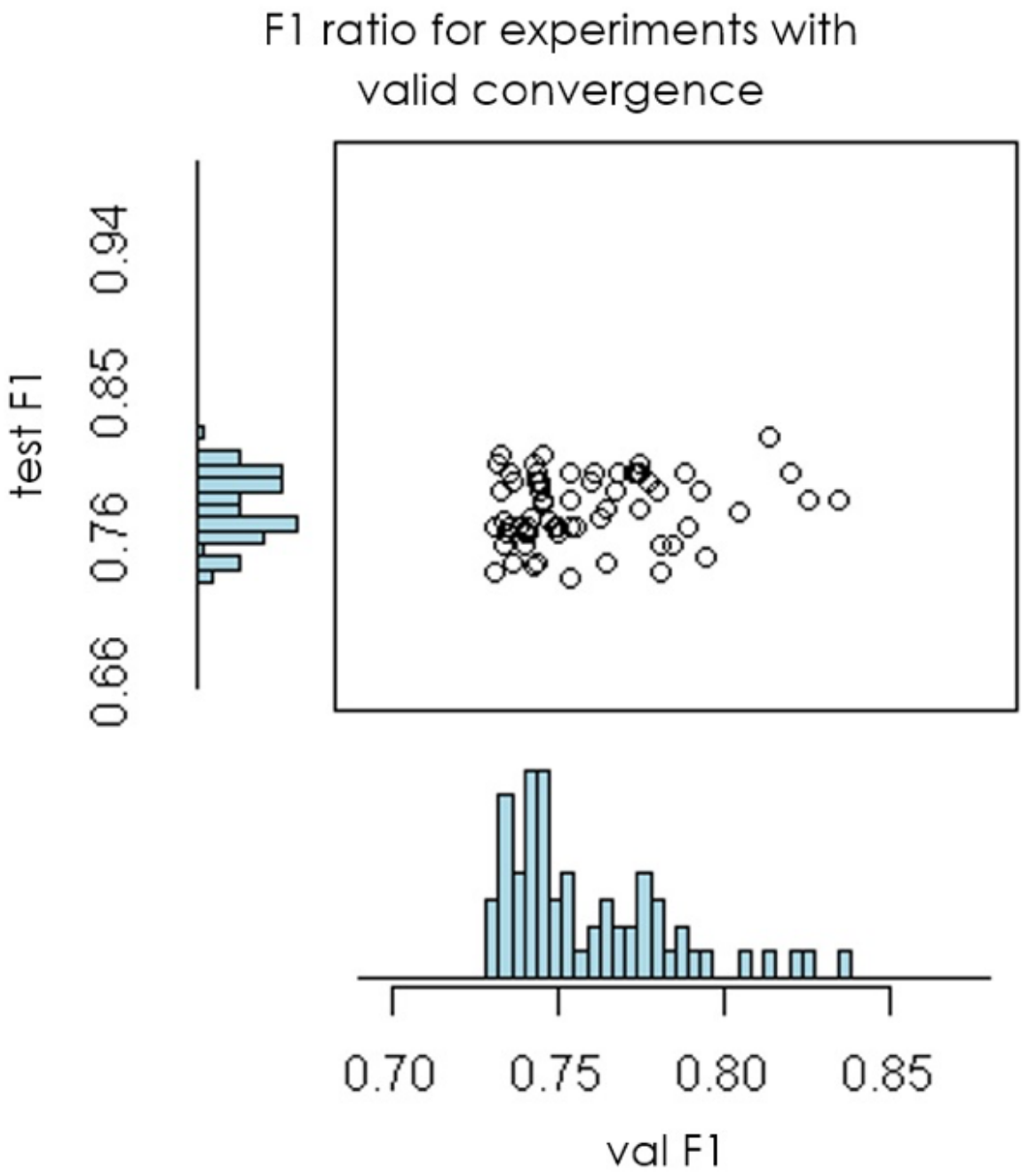

Figure 7: The same results as before but reflected as F1 instead of accuracy. The distribution over both dimensions remain.

It is evident that while again there is a weak trend of test F1 results vs. validation F1 results, there is a lot of noise and the highest test values do not occur at the highest validation values.

Random Forest Results

I repeated a similar process using a random forest classifier, taking the same approach holding constant train, validation, and test data sets, and running a grid of other parameters. Random forest can generate different results due to the algorithm randomly searching through decision trees for good solutions. As before, the random number generator seed was not held constant due to the potential to generate reproducible but misleading results. The test accuracy vs. validation accuracy is shown here:

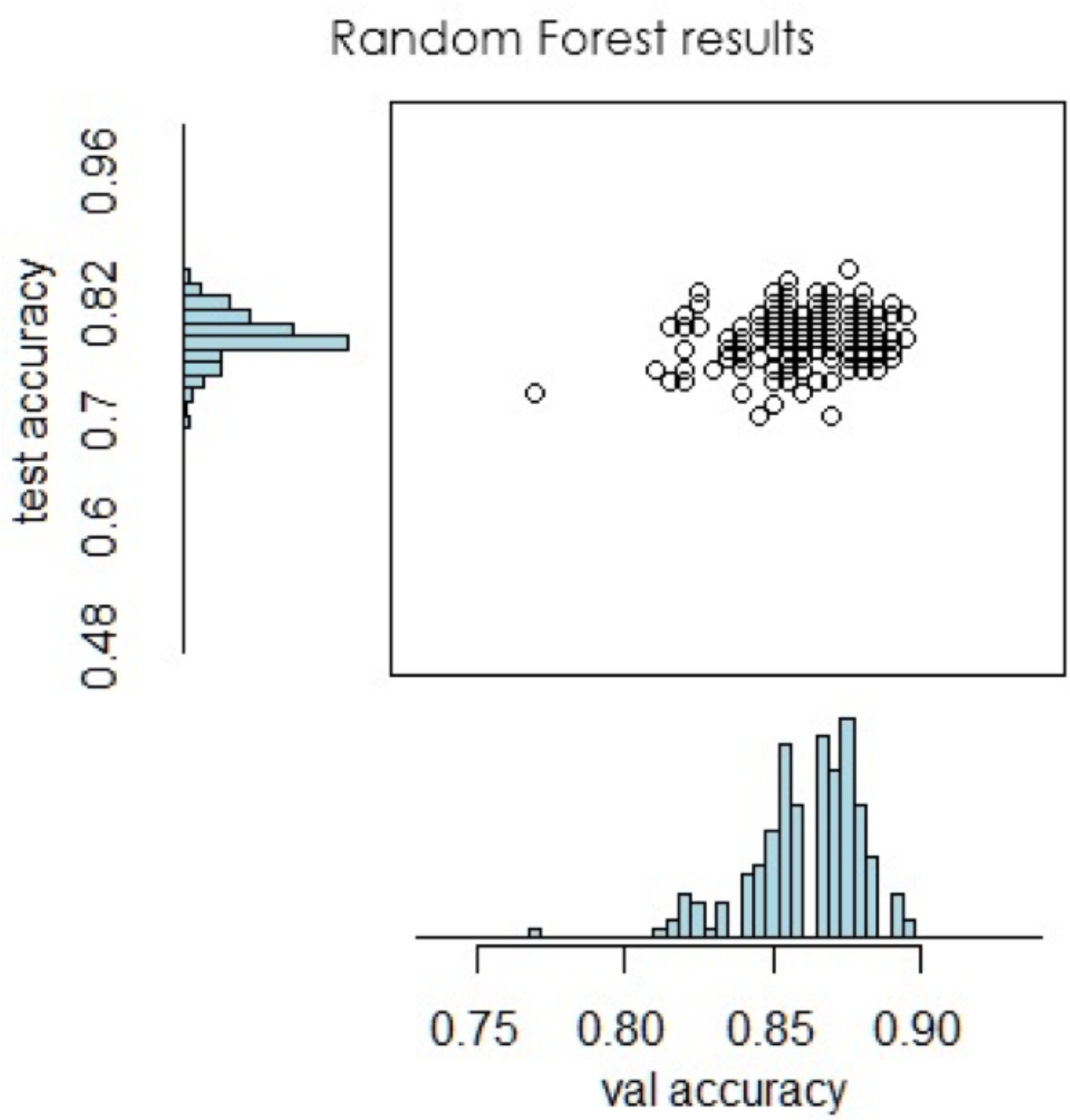

Figure 8: Prediction results using a Random Forest classifier on the Titanic data set, across a range of model parameters and including some repeat runs.

The corresponding chart for the F1 ratio is similar:

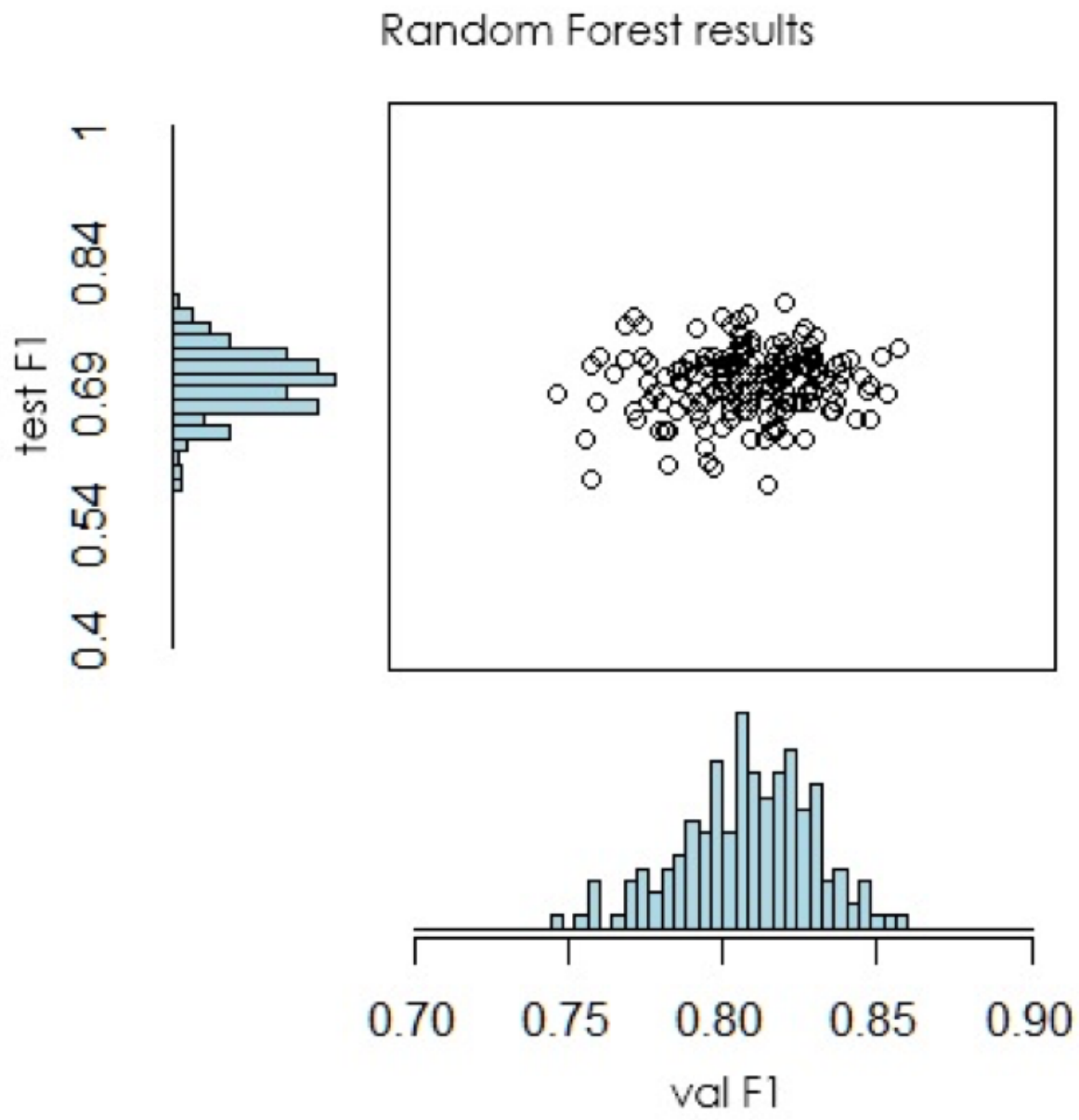

Figure 9: Prediction results using a random forest classifier on the Titanic data set, reflected as F1 instead of accuracy.

Cross Validation

What I have shown so far, is that doing a large number of repeated experiments with both a neural network classifier and a random forest classifier leads to apparently independent random distributions of the classification performance

metrics between the train, validation, and test data sets. Cross validation, the standard approach to finding the best model and set of parameters, won't improve the situation. To show that, one set of parameters was run 25 times using a particular random forest model, with new random selection of the test and validation data sets each time; essentially a 25-fold cross validation.

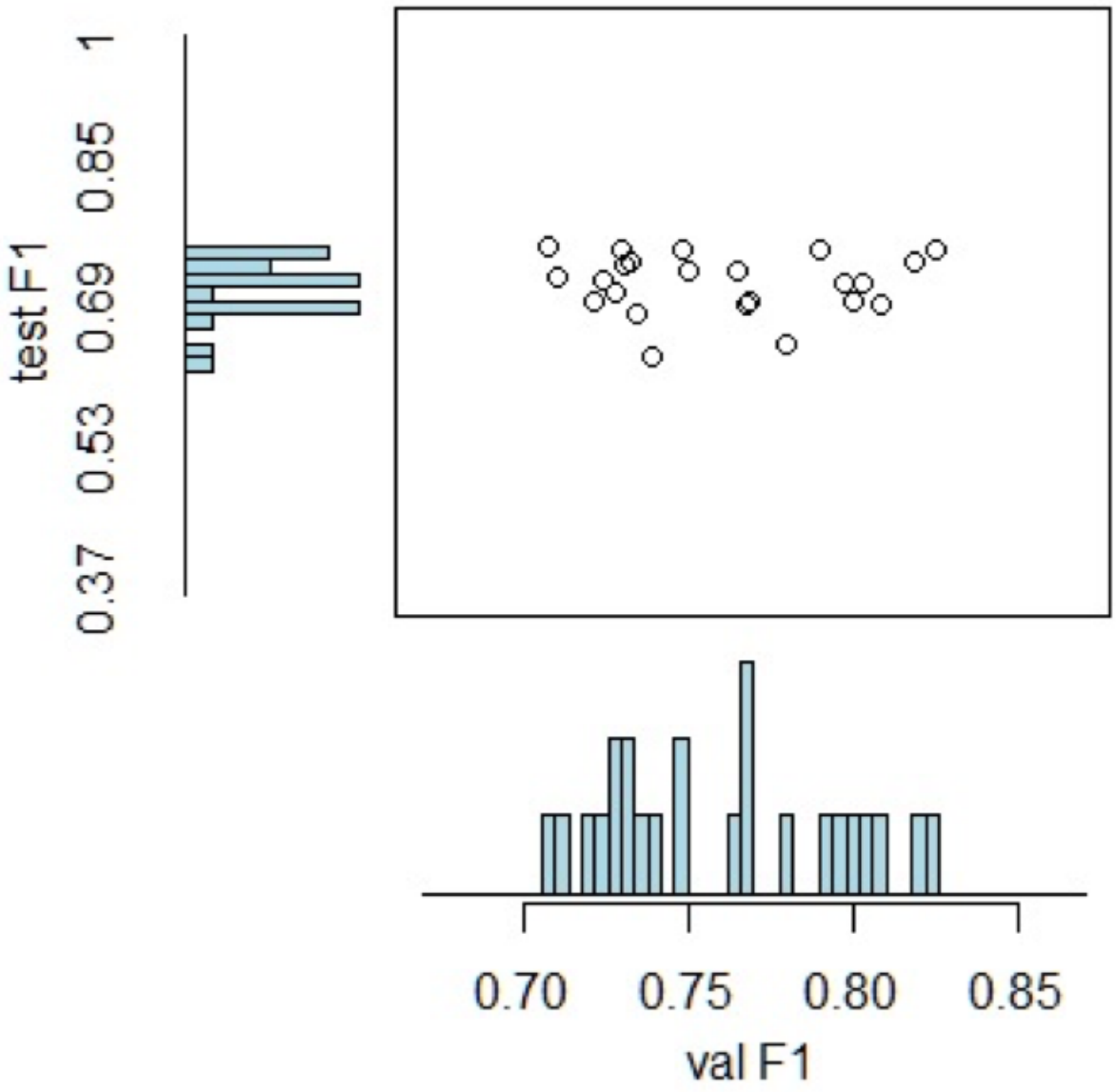

Figure 10: Prediction results using a random forest classifier on the Titanic data set, for 25 runs with randomly selected train/validation splits, for one set of model parameters.

From these data, we would expect the population of F1 values on future data to cover a range of about 20% (+/- 3 std. dev.), around a mean of 0.71. If we look at

the previous results where we held the sample constant and varied model parameters, we estimate a range of more than 2x this result. Why is that? It indicates that the model parameters do impact test F1, but given the uncertainties (ranges of possible results), is it valid to pick the parameters producing the best validation accuracy then run cross validation to estimate future performance? The problem with this approach is that the choice you make may not be statistically different from another choice, and therefore the improvement you expect may not be realized.

Repeating the 25-fold experiment with another set of model parameters gives a similar result, but a different mean:

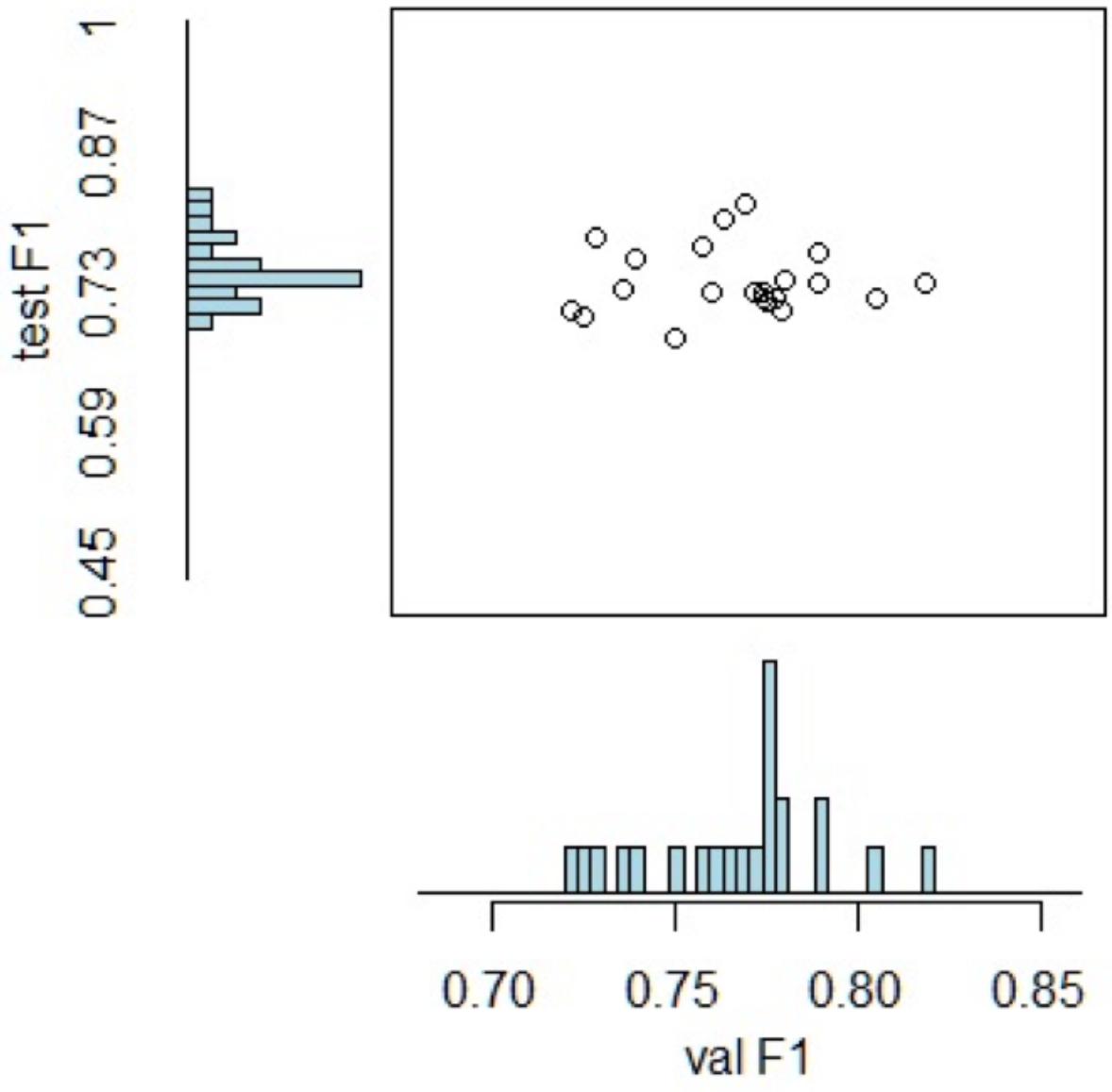

Figure 11: Prediction results using a Random Forest classifier on the Titanic data set, for 25 runs with randomly selected train/validation splits, for another set of model parameters.

In this case, the expected range is still about 20%, but the mean is 0.75. Is the second model better? In this case, a statistical difference does exist between the two experiments (determined by calculating the P value of a t-test between the two means). However, the ranges overlap, which means if we had not run so many experiments, it is very possible we could have concluded the first model was better (i.e. if we happened to get F1 = 0.75 in the first case, and 0.73 in the second case, we would conclude the first was better). A bigger problem is that the validation F1 values, which are what we would use to choose the model, have fairly large ranges as well, and overlap. In fact, the validation result means are not significantly different, so we would have no basis to choose between the two models.

The Misleading Fixed Seed

If we repeat the above experiment but used a fixed seed before the modeling steps, the results are significantly more tightly grouped in the test dimension:

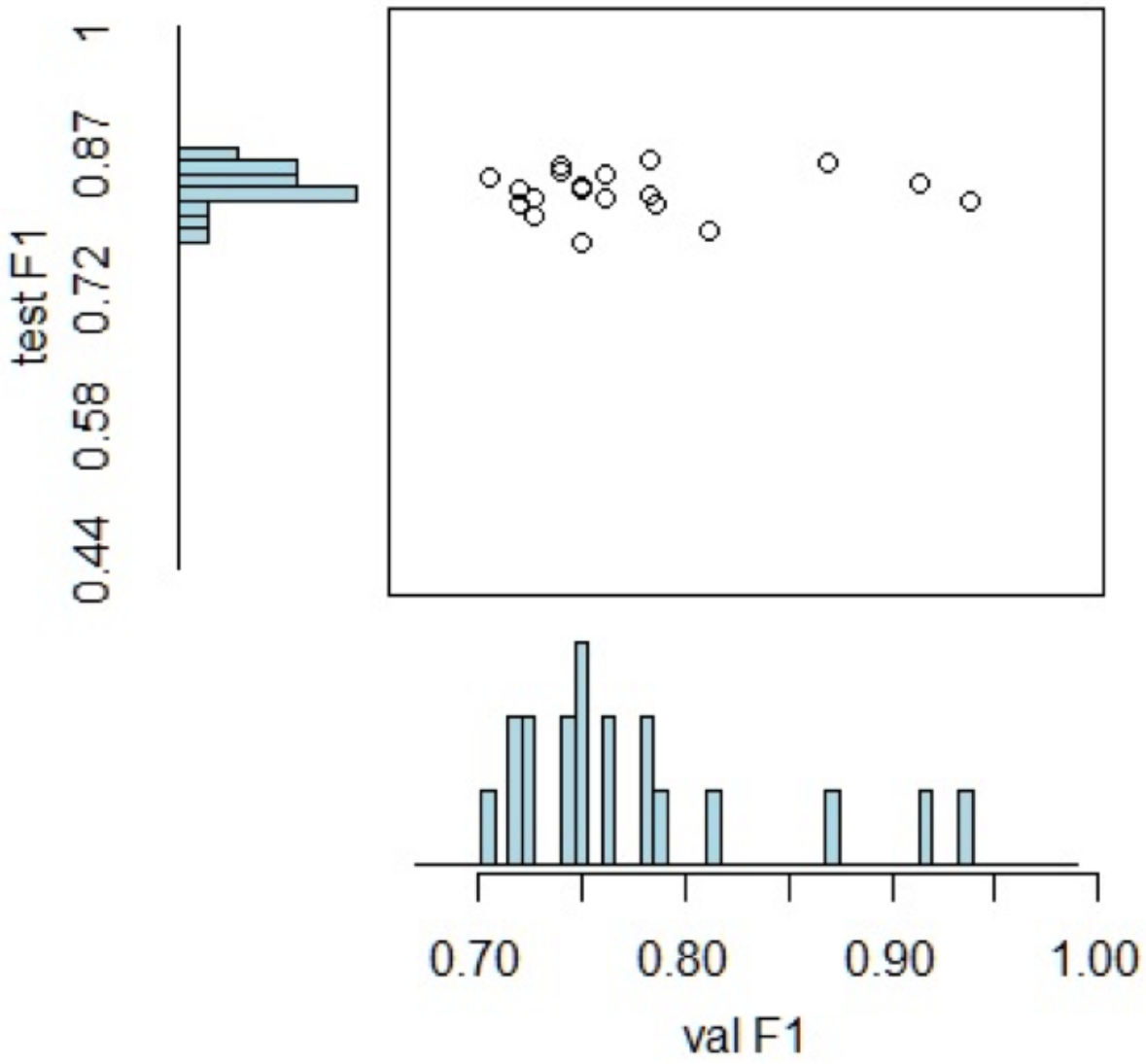

Figure 12: Prediction results using a random forest classifier on the Titanic data set, for 25 runs with randomly selected train/validation splits, with the random number generator seed fixed for every model run.

In this case, the expected range of the test F1 has reduced to 13% from around 20%, yet the range of the validation results is slightly greater than before. Using this result to predict future performance could be very misleading.

Conclusion

Admittedly, it may not be practical to do such an exhaustive number of experiments to validate predictive model performance. However, these results can provide some guidance:

- If you use a fixed seed in development, disable that before doing cross validation to estimate future performance.
- Run as many cross validation folds as you can afford.
- Don't assume test (and future) results will linearly follow validation results; be prepared for noise.
- Re-evaluate models on new data as you obtain it to ensure performance is still what you think it is; if not, re-optimize and re-train.
- Keep in mind the caveat used throughout the financial industry "past performance is no guarantee of future results."

Author **Blaine Bateman**

Blaine leads EAF LLC with 35+ years of international experience, consulting in business analytics/machine learning, strategy, and market analysis. He graduated with Special Honors in ChE and has a Certificate in Quality Management (CU Boulder) and Machine Learning (Stanford/Coursera).