A Gentle Introduction to Random Forests, Ensembles, and Performance Metrics in a Commercial System

Posted on November 10, 2012 by Dan Benyamin

This is the first in a series of posts that illustrate what our data team is up to, experimenting with, and building 'under the hood' at CitizenNet.

Dr. Arshavir Blackwell is CitizenNet's resident Data Scientist. He has been involved in web-scale machine learning and information retrieval for over 10 years.

Hot tip: Click on the images for a larger view.

One of the first posts we published spoke at a high level of the technical problem CitizenNet is trying to solve. In essence, we are trying to predict what combinations of demographic and interest targets will be interested in some piece of content.

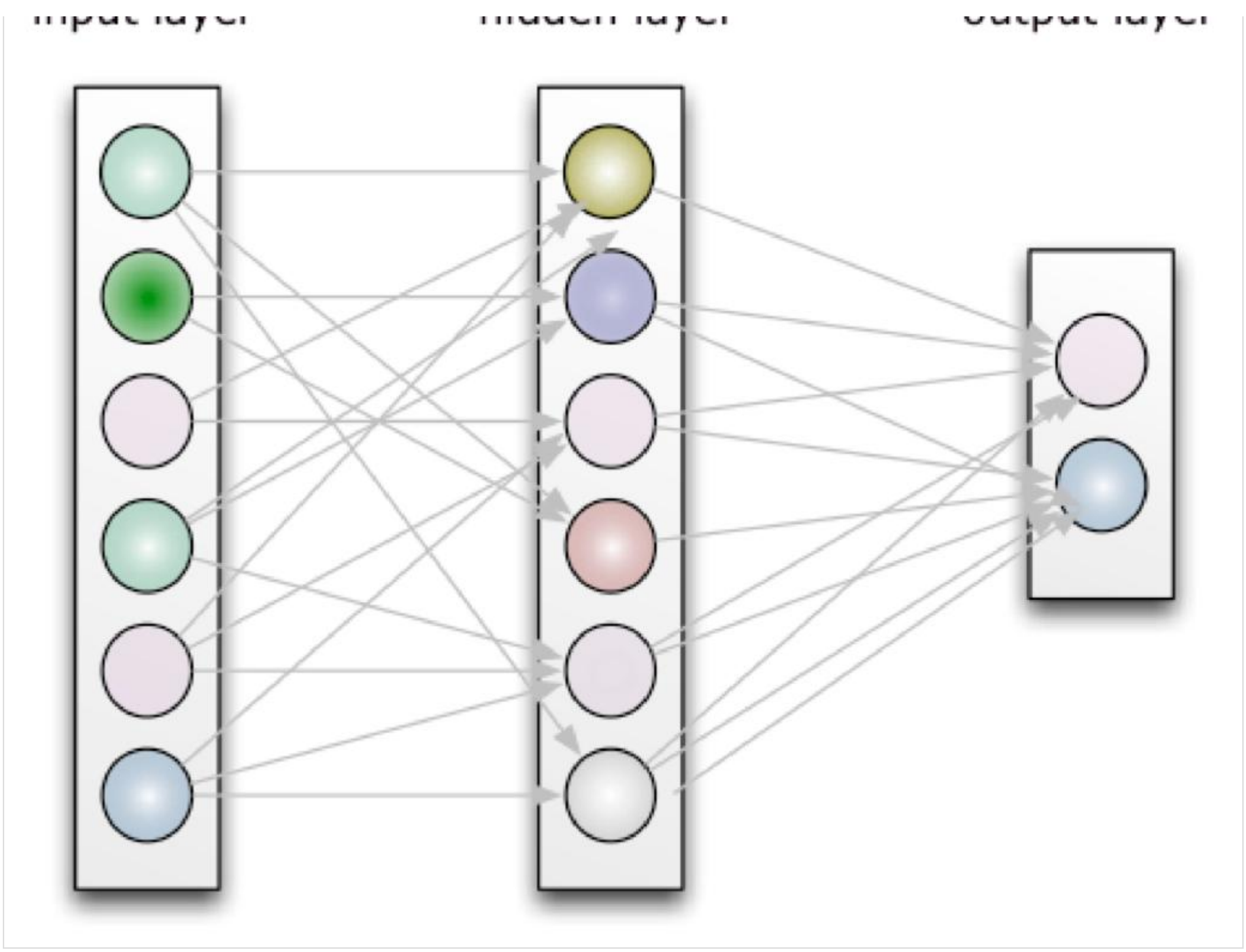
On the CitizenNet platform, a user would create a project that would define (broadly) the target audience, the pieces of Facebook content they are looking to promote, and other campaign and financial information.

Behind the scenes, a robust prediction system builds the targets for the project. This prediction system is trained on past behavior, which it then uses to predict how a future (unknown) project may be best targeted. One of the components of the prediction system is a classifier, which is a currently an ensemble of both Neural Networks and Random Forest classifiers. This blog post aims to illustrate the basics of these methods.

Neural Network

Neural networks have long been used in problems such as this, with a lot of data, many variables, and the possibility of noise in the data.

Each input point is a high-dimensional vector. The neural network is organized in a series of layers (see figure), where the input vector enters at the left side of the network, which is then projected to a "hidden layer." Each unit in the hidden layer is a weighted sum of the values in the first layer. This layer then projects to an output layer, which is where the desired answer appears.



The network is trained with the input and the desired output, which in our case is whether the point represents a low click-through-rate (CTR) or high CTR keyword. Then training occurs: weights at both the hidden and output layers are adjusted so that the actual output corresponds to the desired output. Once trained, the neural network should take a new data point and output either a zero (low CTR) or one (high CTR). If the classifier is uncertain it will produce a value somewhere in between.

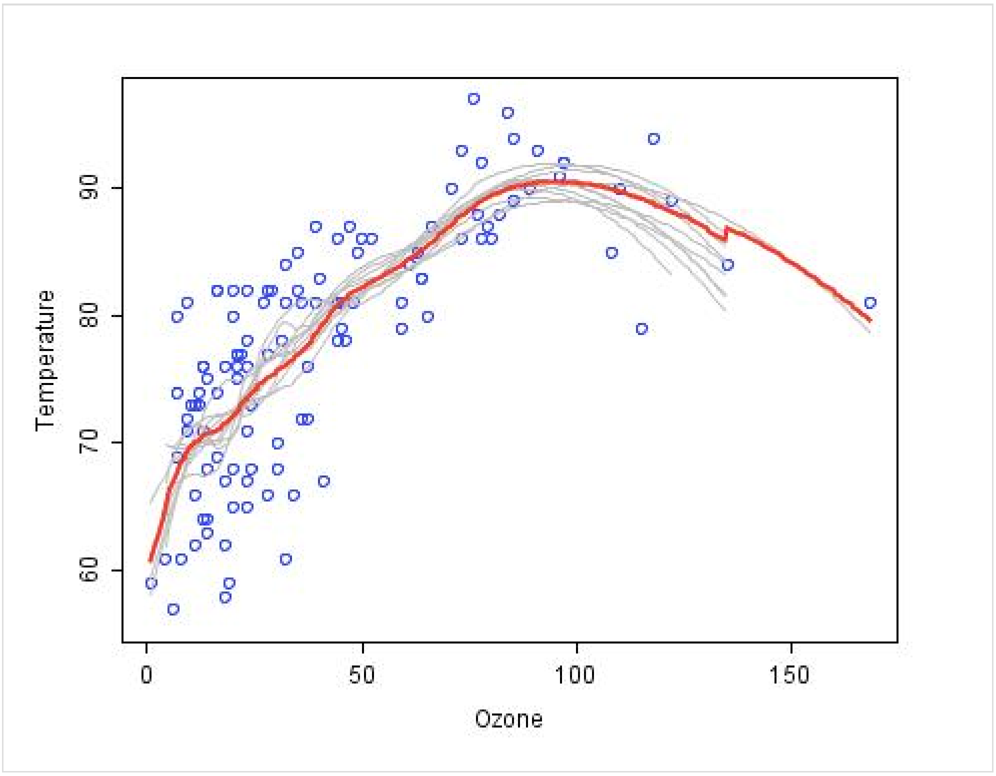
Strengths and weaknesses. Neural networks have quite a few advantages. If we have a lot of input and output data to learn, but no idea what the function mapping the two together is, the network can learn this function without our having to explicitly provide it. Neural networks are also good with data sets that are noisy or where some inputs have missing variables. However, neural nets also have a key disadvantage of many other approaches: the answer that emerges from a neural network's weights can be difficult to understand (it may work, but we don't know how), and the network's training can take longer than certain other methods of machine learning, such as random forests, to which we now turn to.

Random Forests, an Ensemble Method

The random forest (Breiman, 2001) is an ensemble approach that can also be thought of as a form of nearest neighbor predictor.

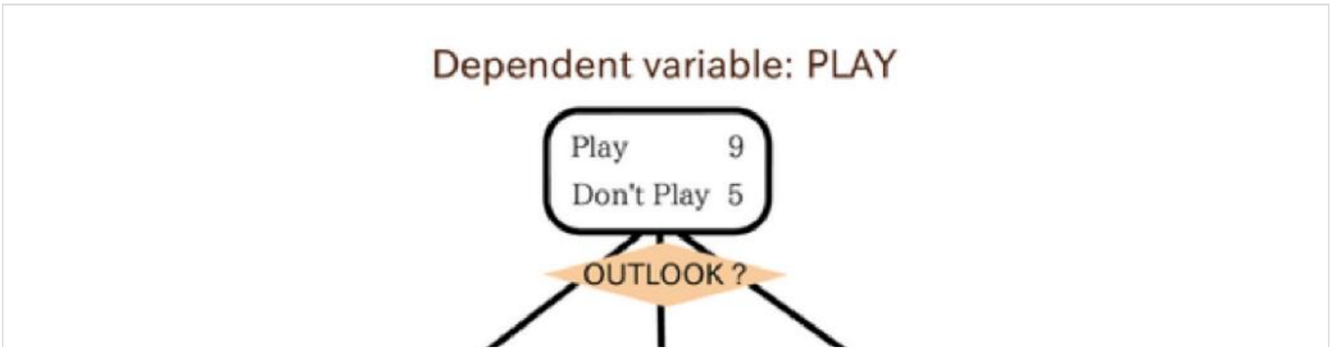
Ensembles are a divide-and-conquer approach used to improve performance. The main principle behind ensemble methods is that a group of "weak learners" can come together to form a "strong learner". The figure below (taken from here) provides an example. Each classifier, individually, is a "weak learner," while all the classifiers taken together are a "strong learner".

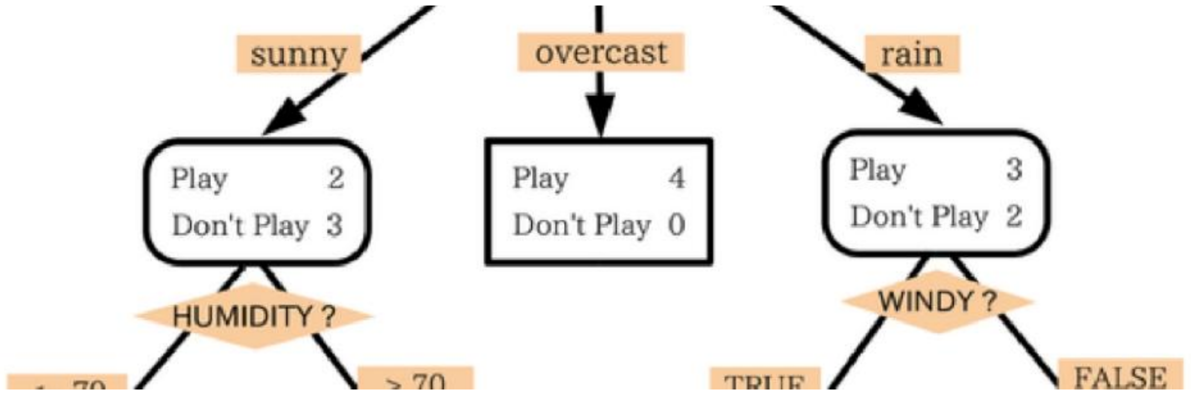
The data to be modeled are the blue circles. We assume that they represent some underlying function plus noise. Each individual learner is shown as a gray curve. Each gray curve (a weak learner) is a fair approximation to the underlying data. The red curve (the ensemble "strong learner") can be seen to be a much better approximation to the underlying data.



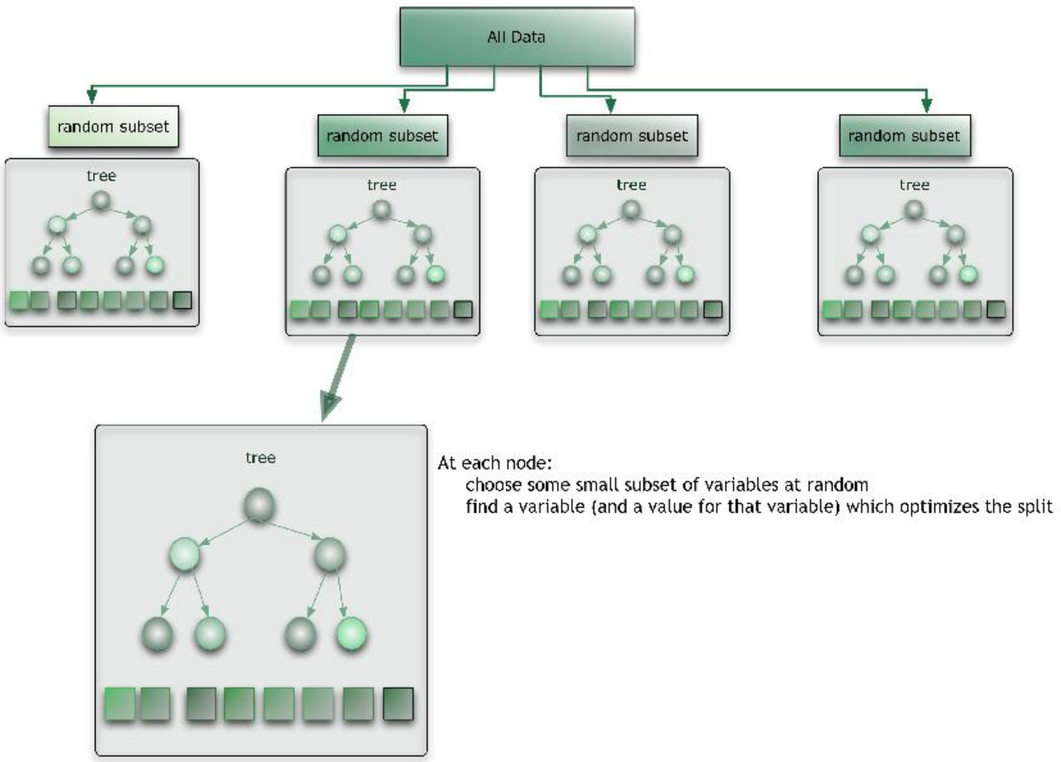
Trees and Forests. The random forest starts with a standard machine learning technique called a "decision tree" which, in ensemble terms, corresponds to our weak learner. In a decision tree, an input is entered at the top and as it traverses down the tree the data gets bucketed into smaller and smaller sets. For details see here, from which the figure below is taken.

In this example, the tree advises us, based upon weather conditions, whether to play ball. For example, if the outlook is sunny and the humidity is less than or equal to 70, then it's probably OK to play.





The random forest (see figure below) takes this notion to the next level by combining trees with the notion of an ensemble. Thus, in ensemble terms, the trees are weak learners and the random forest is a strong learner.


All Data
random subset
random subset
random subset
random subset
tree
tree
tree
tree
tree
At each node:
choose some small subset of variables at random
find a variable (and a value for that variable) which optimizes the split

Here is how such a system is trained; for some number of trees T:

1. Sample N cases at random with replacement to create a subset of the data (see top layer of figure above). The subset should be about 66% of the total set.
2. At each node:
   A. For some number m (see below), m predictor variables are selected at random from all the predictor variables.

B. The predictor variable that provides the best split, according to some objective function, is used to do a binary split on that node.
C. At the next node, choose another m variables at random from all predictor variables and do the same.

Depending upon the value of m, there are three slightly different systems:

- Random splitter selection: m =1
- Breiman's bagger: m = total number of predictor variables
- Random forest: m << number of predictor variables. Brieman suggests three possible values for m: ½√m, √m, and 2√m

Running a Random Forest. When a new input is entered into the system, it is run down all of the trees. The result may either be an average or weighted average of all of the terminal nodes that are reached, or, in the case of categorical variables, a voting majority.

Note that:

- With a large number of predictors, the eligible predictor set will be quite different from node to node.
- The greater the inter-tree correlation, the greater the random forest error rate, so one pressure on the model is to have the trees as uncorrelated as possible.
- As m goes down, both inter-tree correlation and the strength of individual trees go down. So some optimal value of m must be discovered.

Strengths and weaknesses. Random forest runtimes are quite fast, and they are able to deal with unbalanced and missing data. Random Forest weaknesses are that when used for regression they cannot predict beyond the range in the training data, and that they may over-fit data sets that are particularly noisy. Of course, the best test of any algorithm is how well it works upon your own data set.

Testing our Classifiers

To understand how we test the classifier, we must explain several concepts:

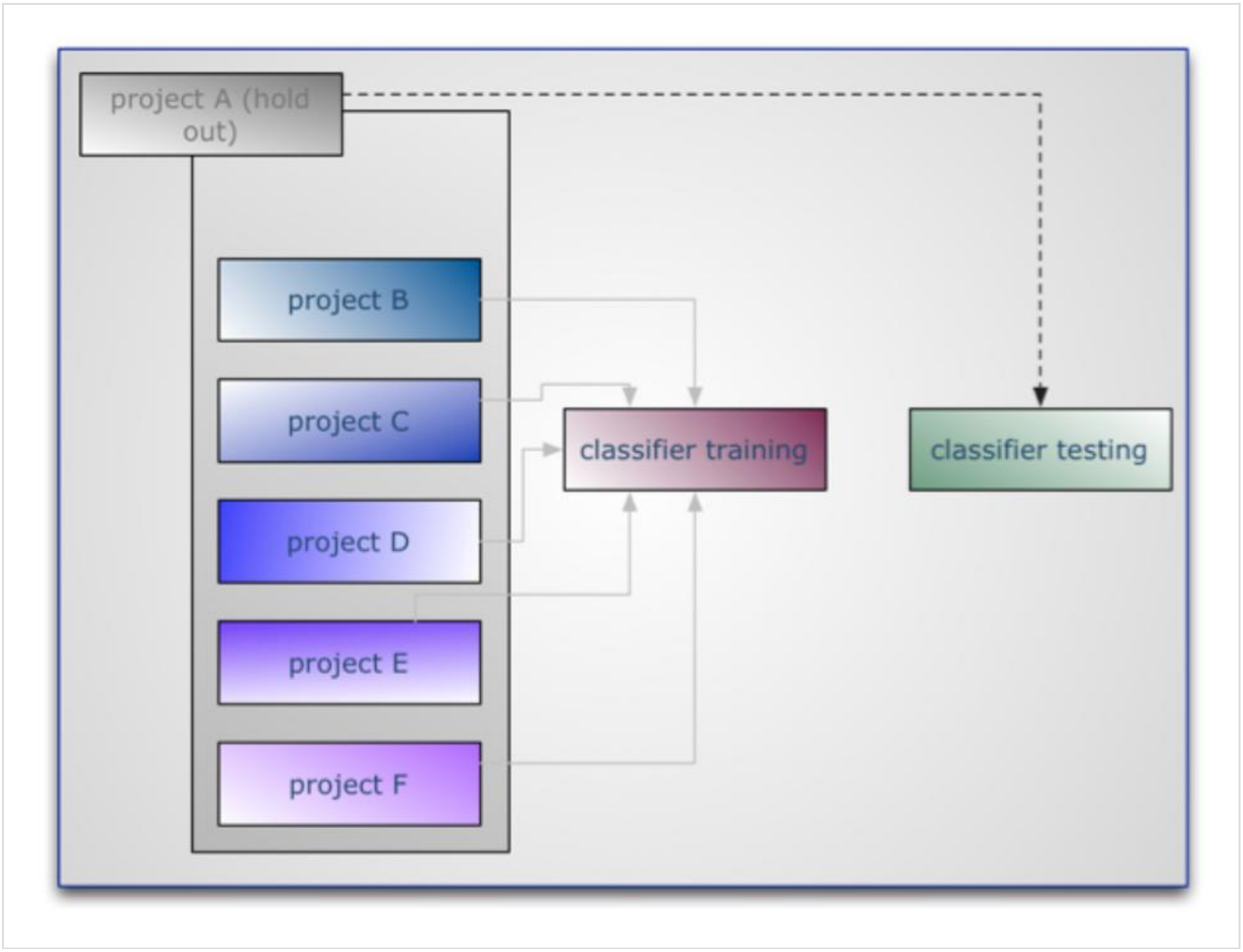
- cross-validation
- thresholds
- mean precision
- precision above chance

CROSS-VALIDATION

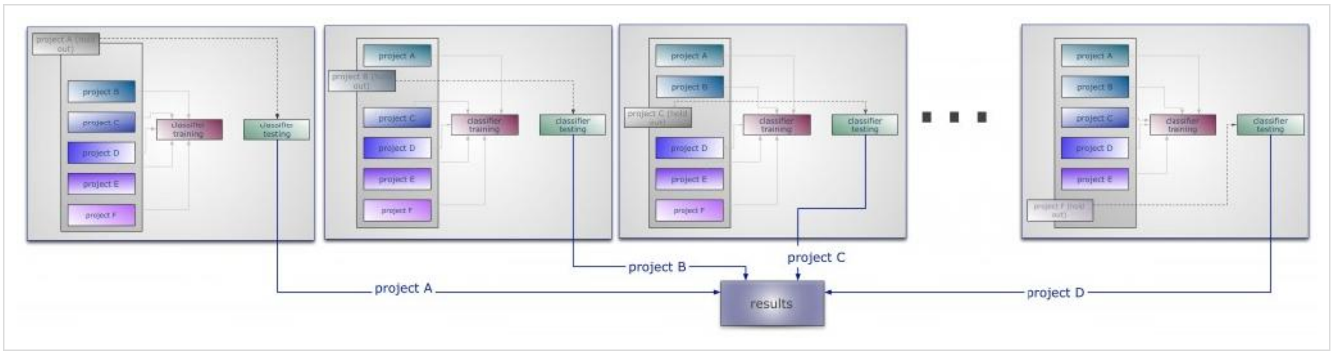
Our classifier test should be ecologically valid. It should prove our classifier under conditions as close as possible to the production environment.

In production the classifier is trained on many projects where we know the CTRs, but really we would like to know how the classifier predicts CTRs for a new project it has never seen before.

We test our classifier using a technique called "cross-validation": train the classifier on all projects except for one. Of course, we also know the correct CTRs for this held-out project, so we can see how well the classifier does on it, without cheating by training the classifier on the hold-out. We do this in turn with each project. See figure below.



Let's say we have projects A through F. We train a classifier on projects A through E, and test on F. Then we train on A and C through F, and test on B. We do this in turn, holding out each project, until we train on A through E and test on F. Each project is a subject in our experiment, with subjects A through F (see figure below).



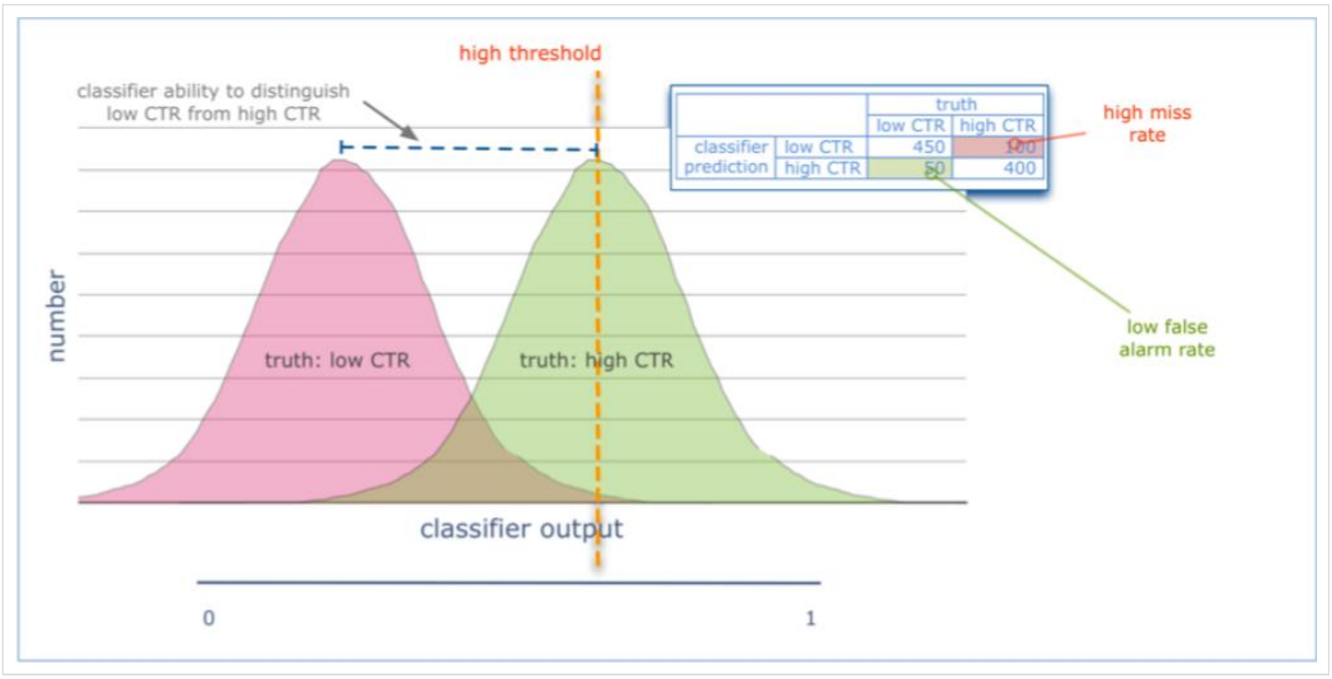
Finally, we collect the results from each cross-validation run for statistical analysis. The experimental question we want to answer: does the classifier give us any more information about the CTR than just guessing?

THRESHOLDS

The classifier predicts whether the input vector would result in a low or high CTR. The classifier’s output is a number from zero to one, give or take. A zero predicts low CTR, a one predicts high CTR. But what do we do if the classifier outputs a value in between, say a 0.4 or a 0.6? We need to choose a threshold. Above this value we treat the classifier output as predicting high CTR. Below this value we treat the classifier output as predicting low CTR.

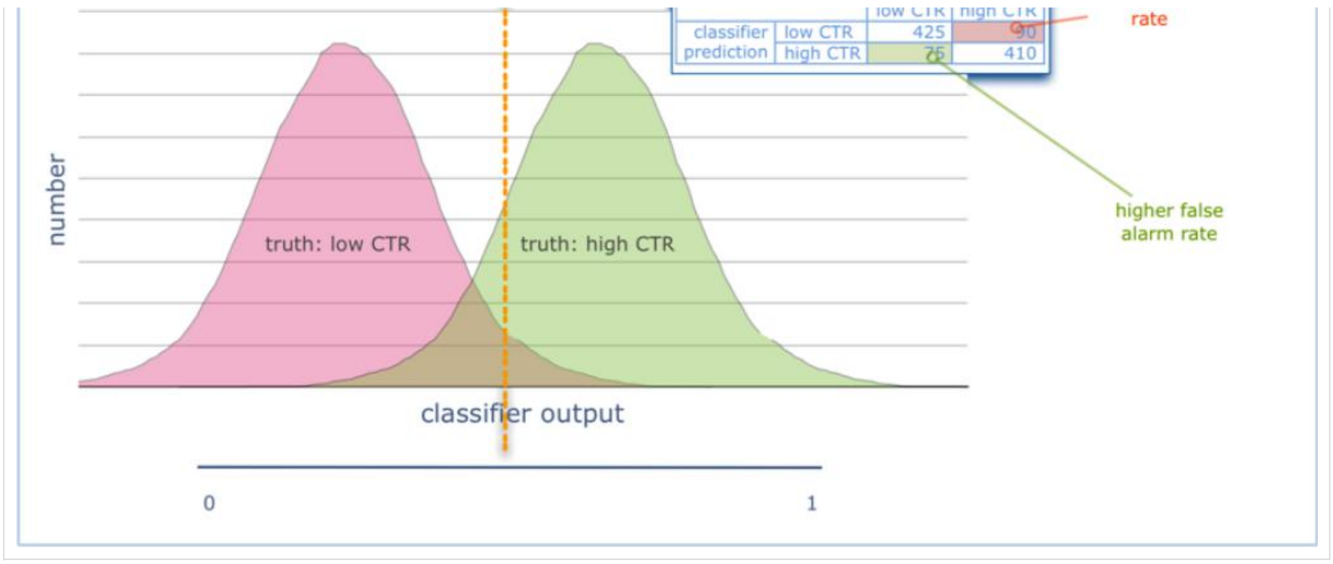
The classifier isn’t perfect, so sometimes it will make mistakes. Sometimes the classifier will incorrectly predict that a high CTR item is low (a “miss”), and sometimes the classifier will incorrectly predict that a low CTR item is high (a “false alarm”).

The following figures show a frequency distribution for the classifier output for both low and high CTR items. High CTR items, in green, tend to result in a classifier output towards 1. Low CTR items, in red, tend to result in classifier output towards 0. Note that in all three figures the distance between the two curves remains unchanged. All that changes is the threshold.

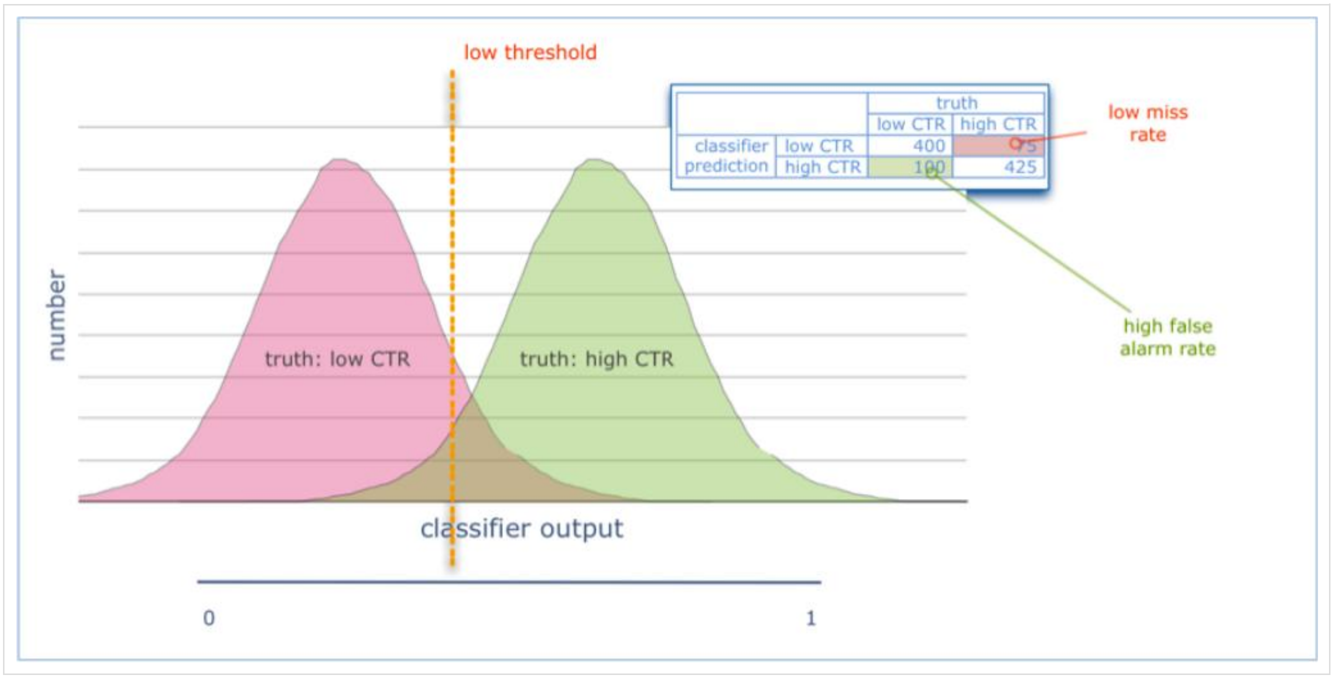


Assume we have a total of 1,000 data points, evenly split between low and high CTR. In the above figure, our threshold is rather high (say, 0.85). This results in a relatively low false alarm rate (only 50 items in total are false alarms), and a rather high miss rate (100 items that are in truth high CTR are missed).

In the next figure, below, moving the threshold to the left (say, to 0.75) means a higher false alarm rate (75) and a lower miss rate (90).



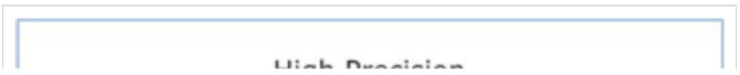
In the figure below, at the lowest threshold (say, 0.40), our false alarm rate goes up to 100, and our miss rate goes down to 75. So as we lower our threshold, our false alarm rate goes up from 50 to 75 to 100, while our miss rate goes down from 100 to 90 to 75.

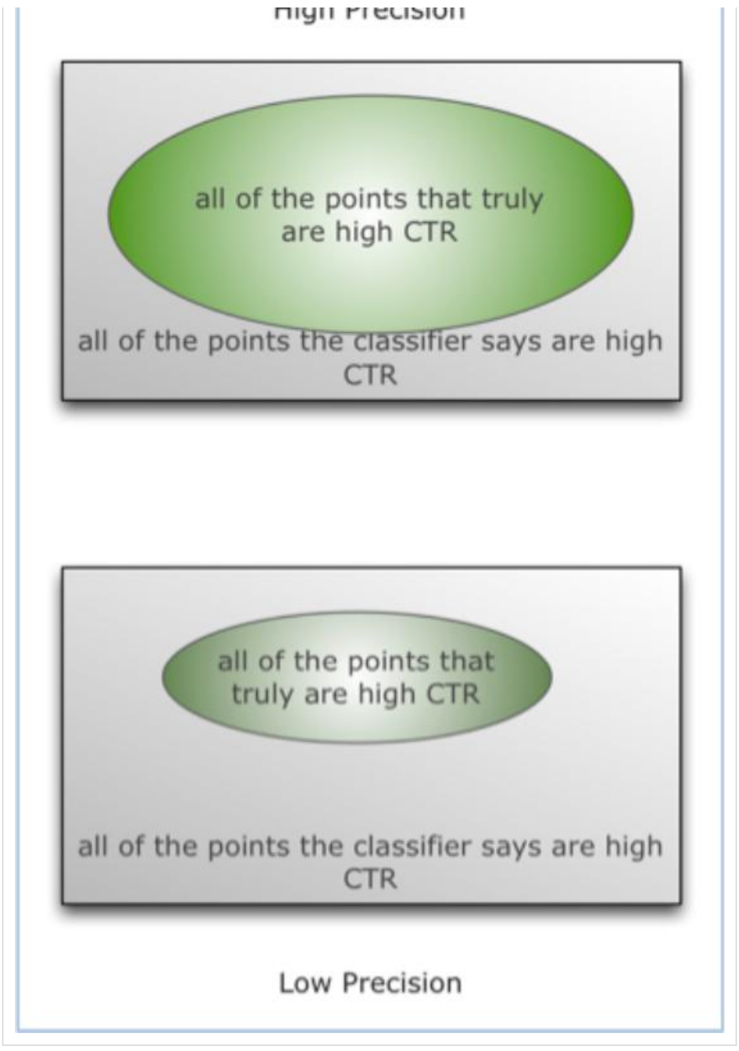


We draw the threshold based upon which type of error we want to avoid more, misses or false alarms. A higher threshold means more misses and fewer false alarms. A lower threshold means the opposite. Changing the threshold does not change the underlying accuracy of the classifier, it just moves the error around.

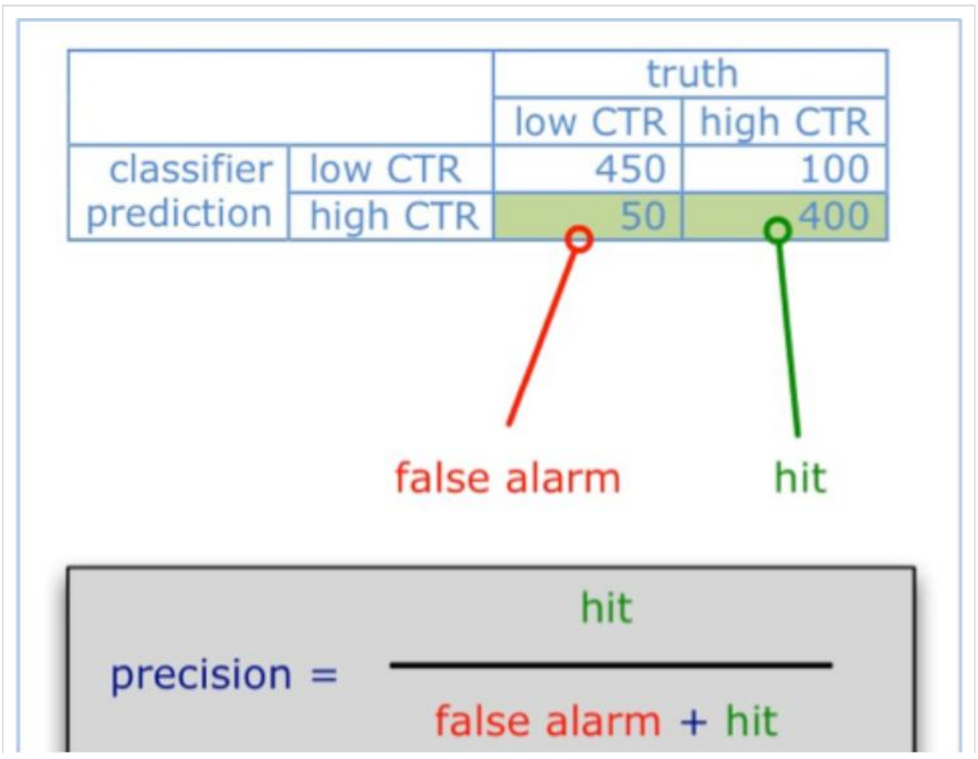
MEAN PRECISION

The effectiveness of the classifier is the distance between the two means, which does not vary as threshold changes. One way of measuring the effectiveness of the classifier is the "precision". Precision is the number of truly correct items ("hits") divided by the number of items that the classifier says are correct (hits + false alarms).




High Precision
all of the points that truly are high CTR
all of the points the classifier says are high CTR
all of the points that truly are high CTR
all of the points the classifier says are high CTR
Low Precision

In our example above, we have 1,000 data points, of which 500 are high CTR. Our classifier correctly catches 400 high CTRs. The classifier indicates an additional 50 points are high CTRs, when in fact they are low CTRs. This means we have 50 false alarms. These results in a precision of 400/450 = 88.88% items that the classifier says are correct (hits + false alarms).



		truth	
		low CTR	high CTR
classifier prediction	low CTR	450	100
	high CTR	50	400

false alarm
hit

$$\text{precision} = \frac{\text{hit}}{\text{false alarm} + \text{hit}}$$

$$\frac{400}{50 + 400} = 88.88\%$$

“Mean precision” takes into account the issues with choosing a threshold, noted above, by performing this calculation at a range of thresholds and taking the mean.

IMPROVEMENT ABOVE CHANCE

Unfortunately, in reality it is not as easy as this: projects vary greatly in the number of high CTR keywords. Some projects truly contain only 15% high CTR, while others are as high as 80% or 90%. The finance sector, as an example, will always have a lower average CTR than a popular Facebook game. We need more than mean precision to measure classifier performance.

Imagine our classifier has a mean precision of 95%. Although this sounds good, if the project is 95% high CTR, it’s less impressive. In this case, if we always guessed that something was high CTR, we would do as well as the classifier! Thus, we must report the mean precision after subtracting out the true percentage of high CTR items for that project. We call this “mean precision above chance”. In our reports, a mean precision of 10% means that the classifier has a 10-point higher precision than guessing alone.

Results

In our test set, projects that had a true high CTR value of 1 (all high CTR), or of less than 0.05 (less than 5% high CTR), were removed, resulting in 72 projects for further analysis. Remember, each project may contain dozens of targeting combinations, and each targeting combination may contain hundreds of keywords.

For the neural network, the mean improvement for the classifier was 0.15 (see table below). A paired t-test on the results showed $t(71) = 7.72$, $p < 0.0001$ (a p-value of less than 0.05 is traditionally considered to be statistically significant, meaning that the chance of this effect being due to chance is 1 in 20 or less). This means that the neural network classifier showed a statistically significant improvement in detecting high CTR items as compared to chance.

For the random forest, the mean improvement for the classifier was 0.06 (see table below). A paired t-test on the results showed $t(71) = 3.17$, $p < 0.003$. This means that the random forest classifier showed a statistically significant improvement in detecting high CTR items as compared to chance.

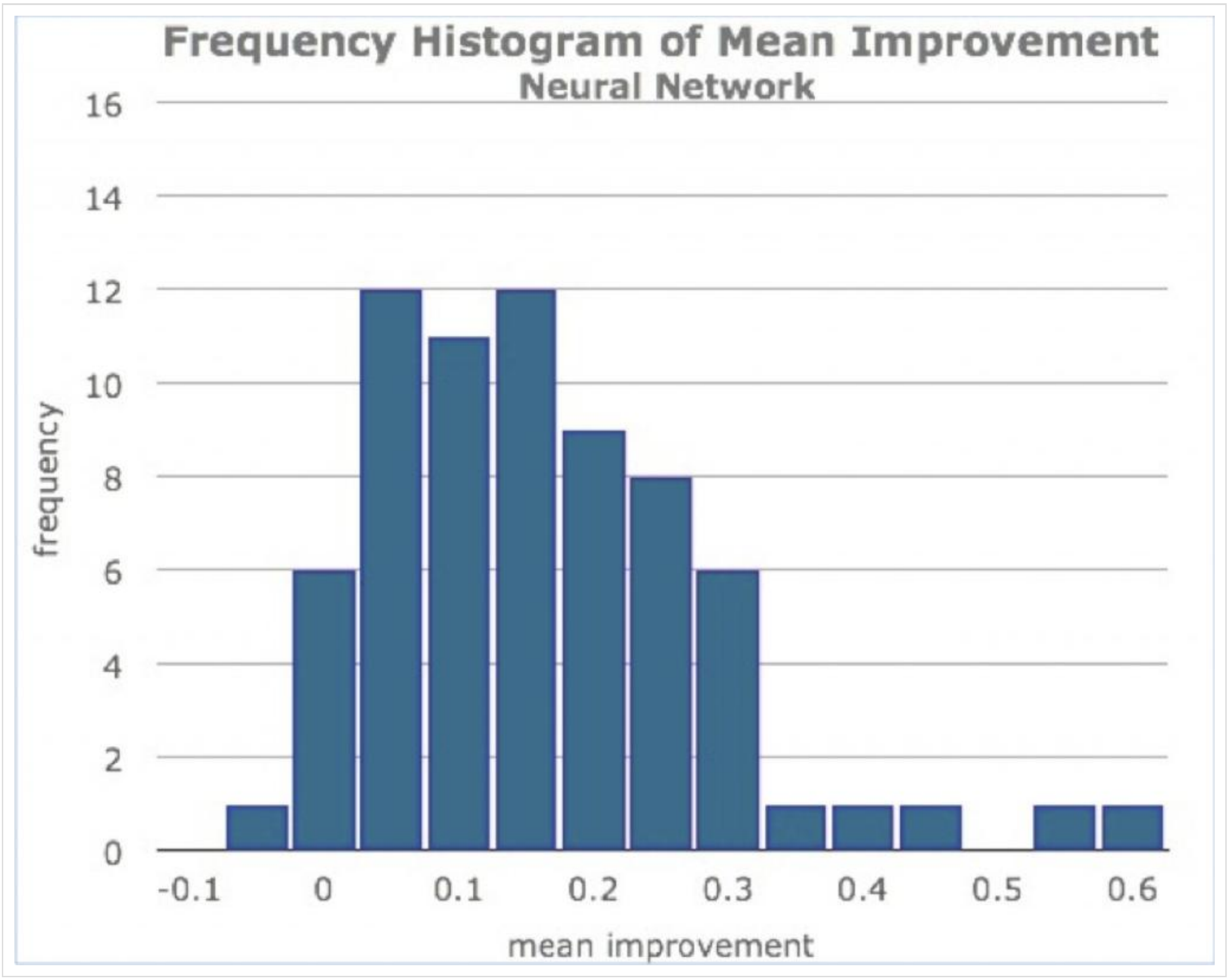
In comparing the mean improvement between the two classifiers, the difference was 0.09, in favor of the neural network. A paired t-test on the results showed t(71) = 3.63, p < 0.0006. This means that the neural network classifier showed a statistically significant improvement in detecting high CTR items as compared to the random forest classifier.

Both classifiers work. The neural network is significantly better than the random forest by 0.09.

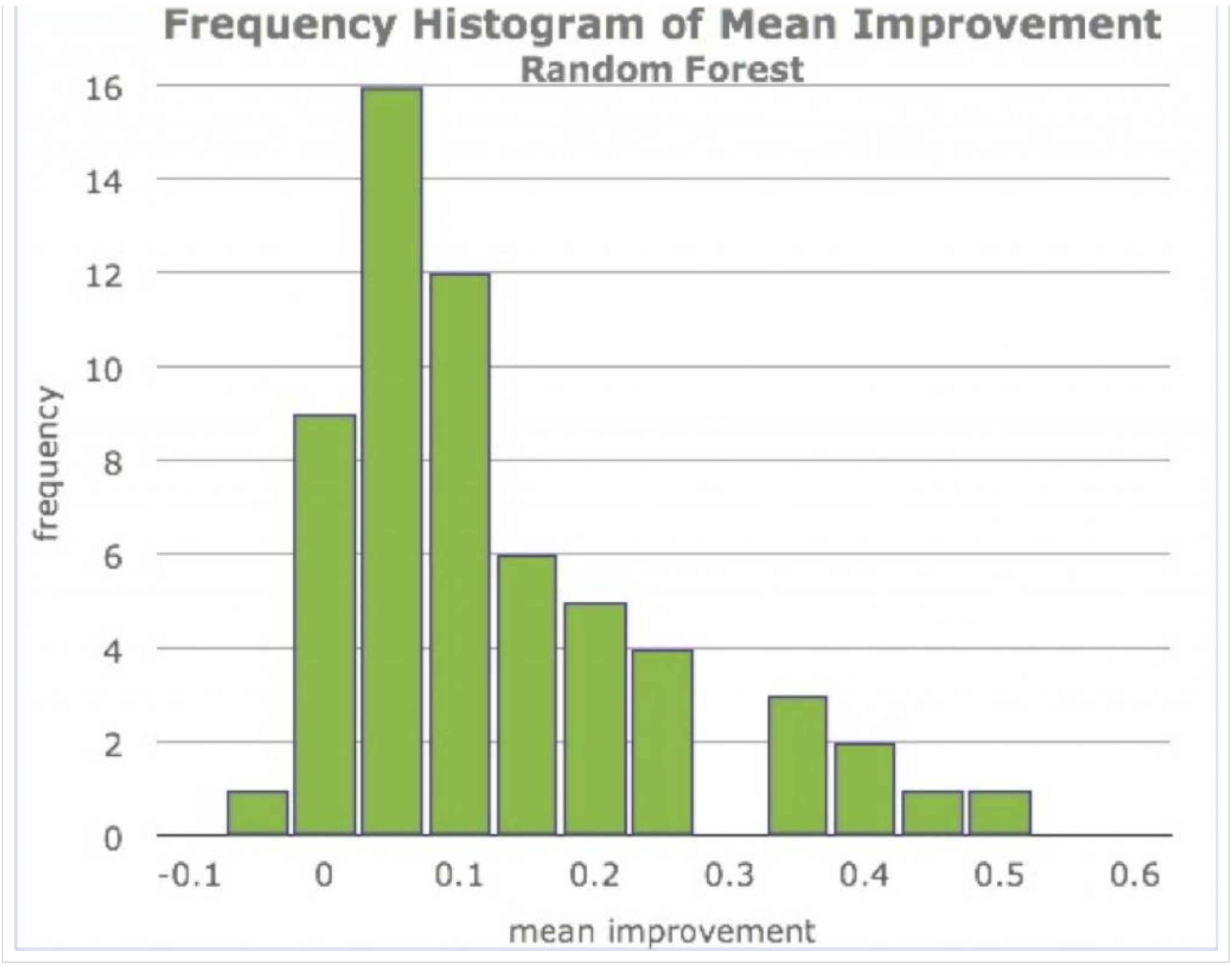
	Mean high CTR	Mean precision of classifier	Mean improvement
neural network	0.44	0.59	0.15
random forest	0.44	0.50	0.06

FREQUENCY HISTOGRAMS

This figure shows a frequency histogram of the mean precision improvement over chance for the 72 projects for the neural network:

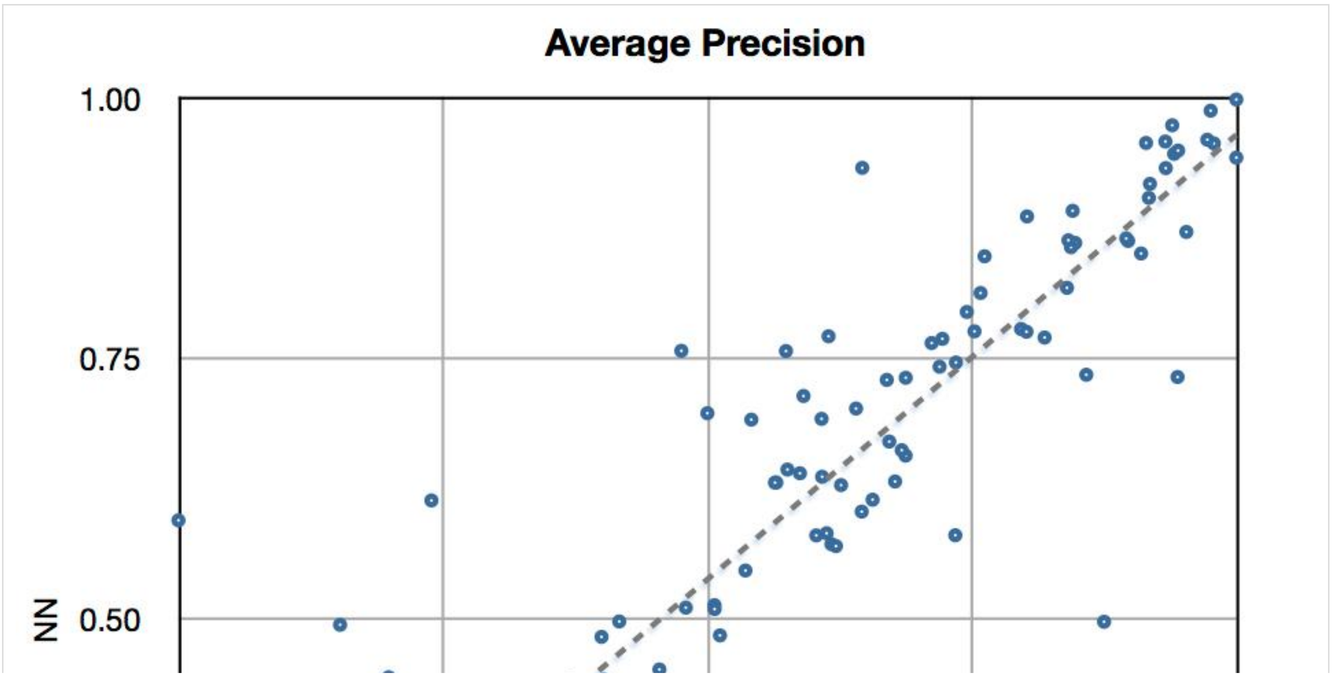


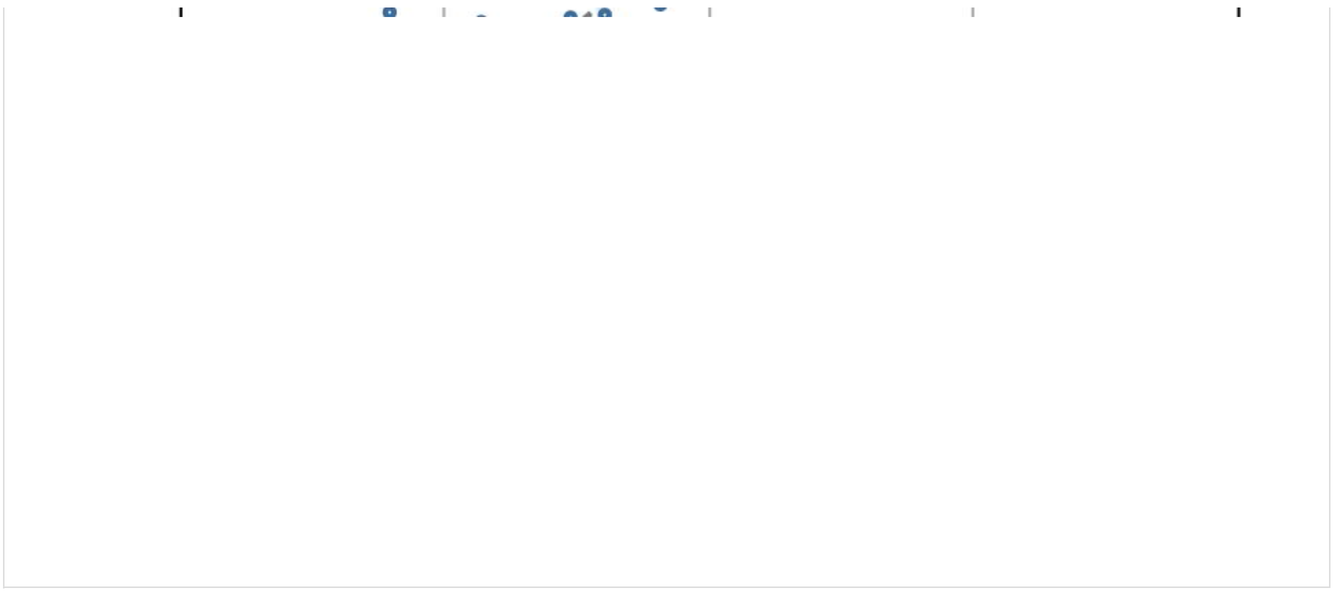
This figure shows a frequency histogram of the mean precision improvement over chance for the 72 projects for the random forest:



Neural Network vs Random Forest

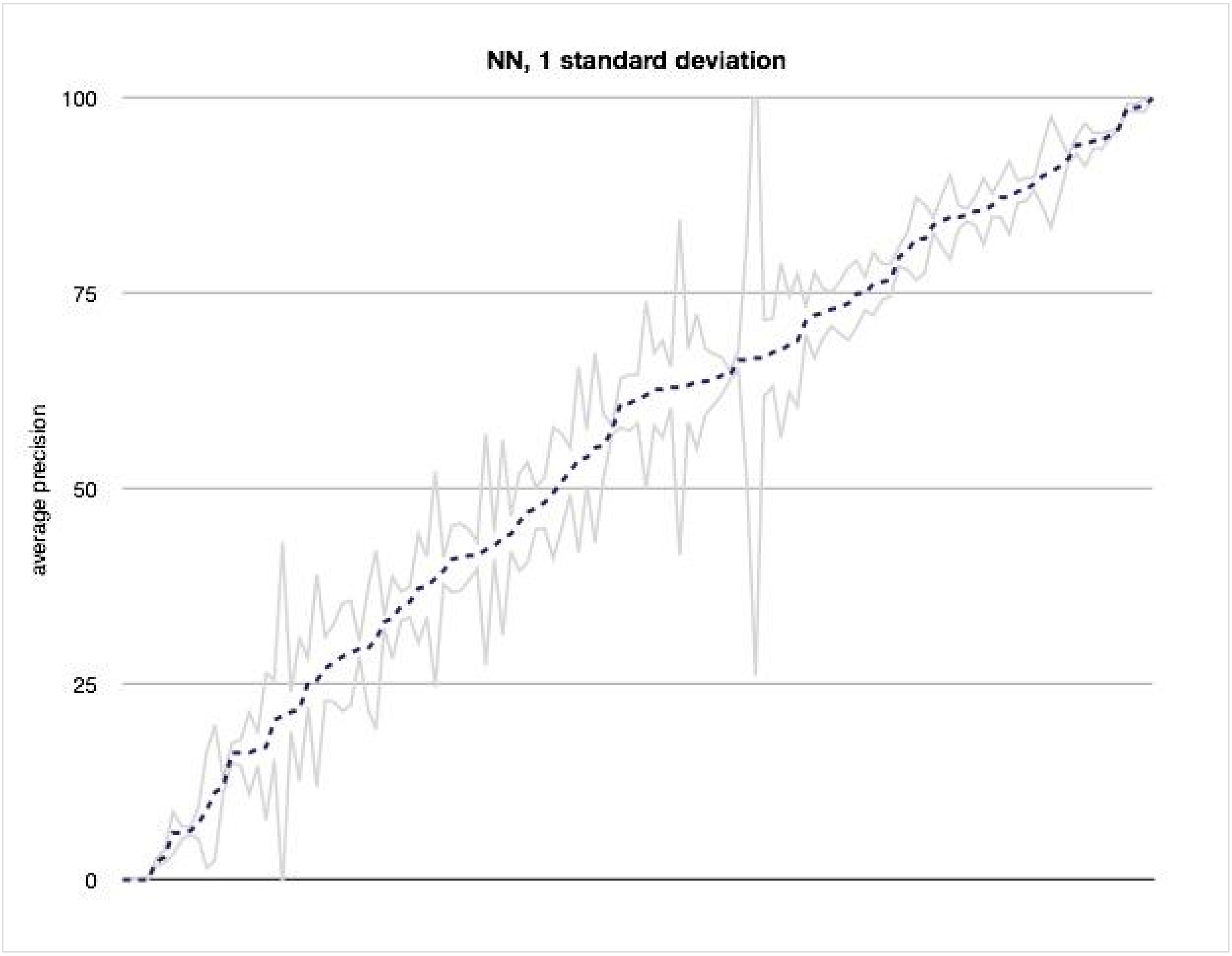
The graph below compares results of four neural networks with three random forests. It shows us that there's a great deal of variability in precision between projects and that each method tends to track the other, with a correlation of 0.8677.

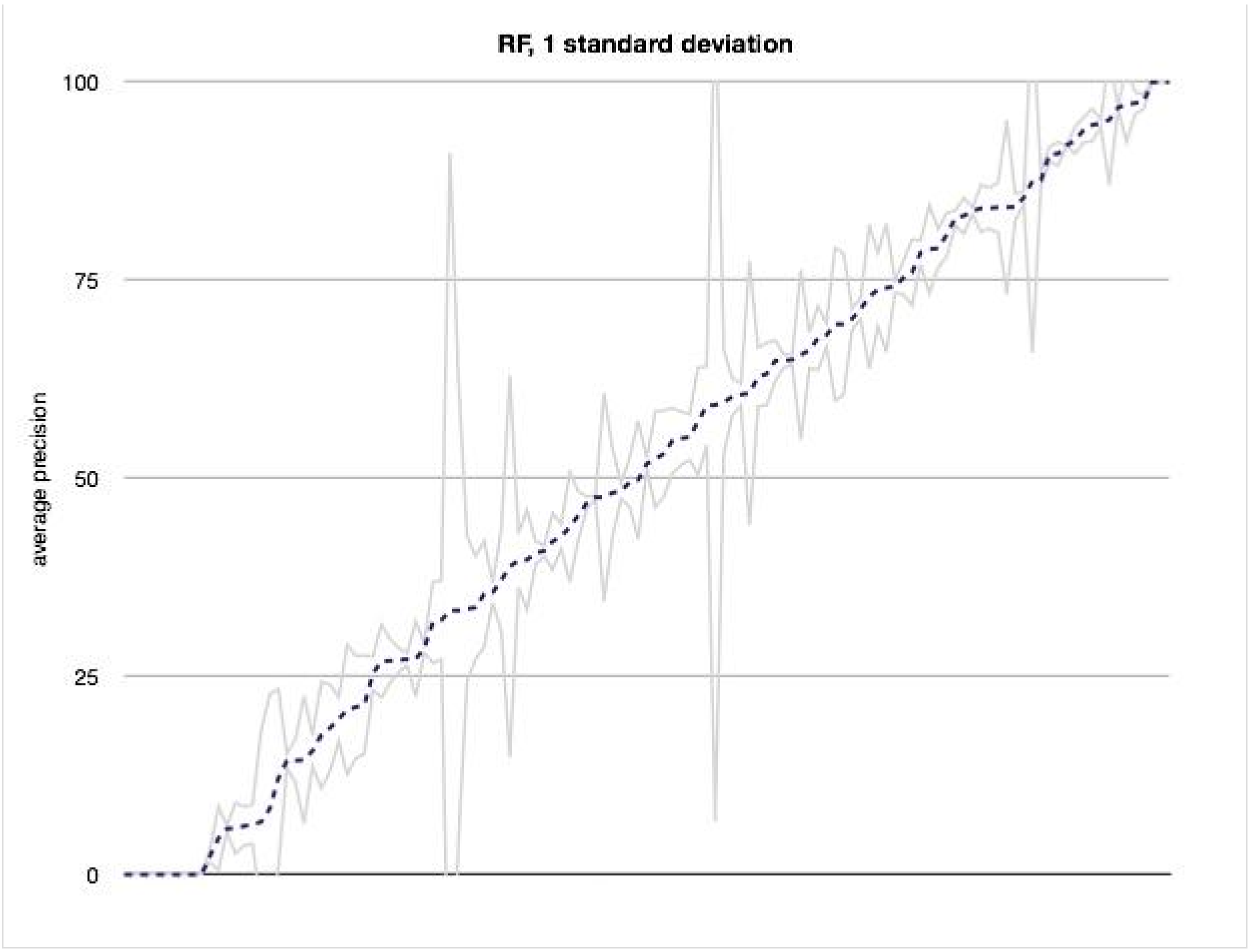




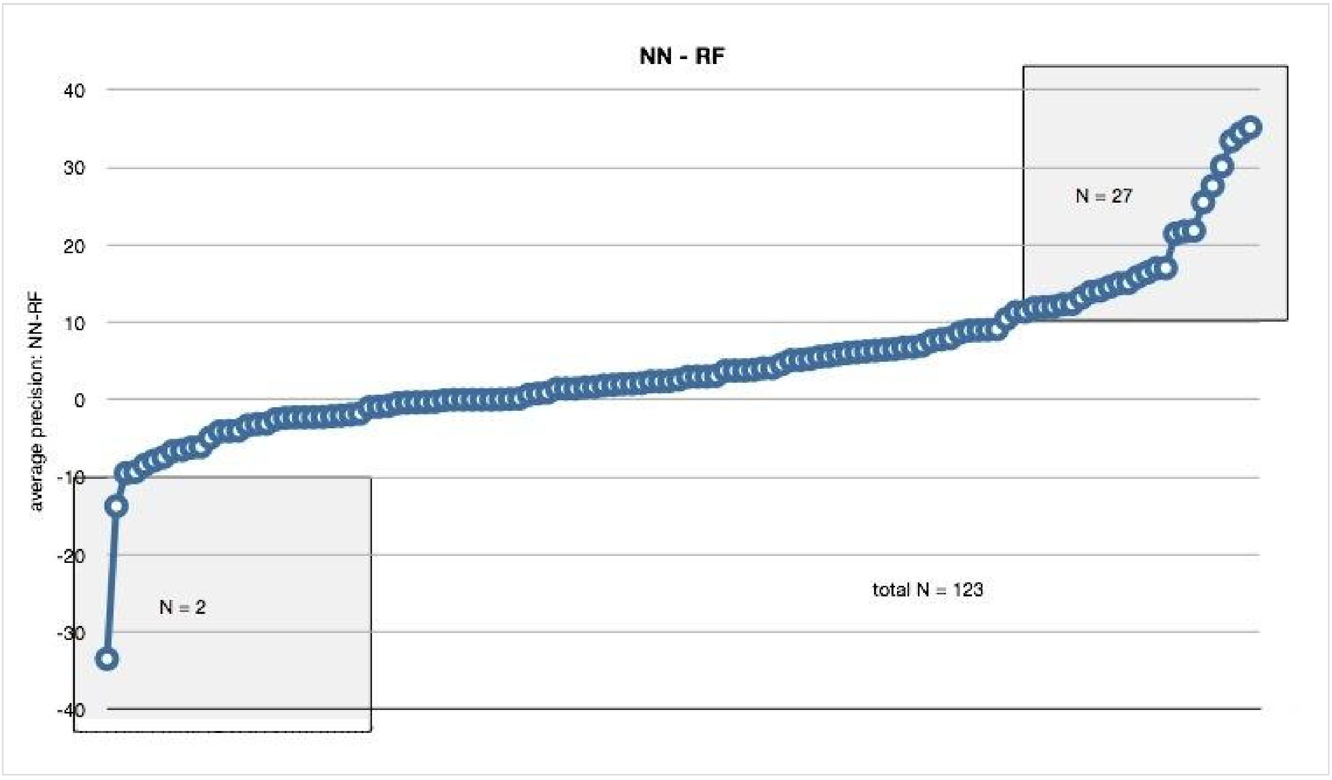
Neither method can be said to be better than the other in all cases. It remains to be seen if there is any systematicity as to why and where one method is better than another.

As the next two figures depict, both neural networks and random forests show low variability over our data in most but not all cases: that is, for most projects, rerunning the classifier several times usually results in about the same precision. In the next two figures, the various projects, in order of increasing precision in each case, are on the x-axis and precision is on the y-axis.





In the next graph, we have subtracted precision of neural network runs from precision of random forest runs, for the same project, out of a total of 123 projects. In this next figure, the various projects, in order of increasing precision, are on the x-axis and precision is on the y-axis.



What this picture tells us is that, in such a complicated space, there will be areas that will be difficult to model with just one learning method. We intuitively know that Neural Networks and Random Forests are sufficiently different algorithms, but this is proof that the projects on the left side of the S curve stand more to gain with the Random Forest method. Such ensemble approaches are sometimes the only way to obtain hard-to-reach gains (the winners of the Netflix Prize had a large example of one — they even named their team after it!)

These results are just preliminary, though, so stay tuned to see what further tweaking provides!

This entry was posted in Facebook Ad Optimization, Facebook Ad Targeting, Facebook Audience Optimization, Facebook Audience Prediction, Tech by Dan Benyamin. Bookmark the permalink [https://citizennet.com/blog/2012/11/10/random-forests-ensembles-and-performance-metrics/] .

ONE THOUGHT ON "A GENTLE INTRODUCTION TO RANDOM FORESTS, ENSEMBLES, AND PERFORMANCE METRICS IN A COMMERCIAL SYSTEM"

Pingback: Bootstrapping, Bagging, Boosting and Random Forest | My Visual Notes

Comments are closed.