Lecture Notes on Evaluating Classifiers and Training/Test/Validation Sets

Chengkai Li
Department of Computer Science and Engineering
The University of Texas at Arlington

March 8th, 2015

After we have developed a classifier, why do we need to evaluate it? "Evaluation" means estimating the error rate of a classifier. The estimate will give us an idea about how well the classifier may perform on future unseen cases. After all, its performance on unseen data, instead of known data, is what really matters. Having an accurate estimate of the error rate is apparently important in comparing and choosing among alternative classifiers. It is also critical for simply knowing what to expect from our choice of classifier—An estimate will allow others to decide whether to adopt our classifier; it will allow decide what actions to take under different classification outcomes. For example, if your email service provider tells you that their spam detector has an error rate of 25%, I bet you would want to turn off the spam detector.

What measures to use for evaluation? Error rate (#incorrectly_classified_instances / # all_classified_instances) is simple and intuitive. It is perhaps what Kaggle uses for the problem you are working on in Programming Assignment 2 (P2). So using that as the measure should be fine for your P2. Later we will know that there are measures that may make more sense in a variety of scenarios. (Note that I used "error rate" instead of "accuracy" (i.e., 1 – error_rate), since we will also talk about accuracy of the estimate. It can be confusing when one says "accuracy of the estimate of a classifier's accuracy".)

So let's focus on how to evaluate, i.e., how to obtain an accurate estimate of our classifier's error rate. Suppose we are given known/seen data D—the set of all records of which we know the class labels.

(1) Training error/resubstitution error:

We can train a classifier C using the whole known data D and apply the classifier on the known data itself. We can then use the error rate of C on D to estimate its error rate on future unseen cases. This is an optimistic estimate since a classifier with excellent error rate may suffer from over-fitting. Hence, optimizing for this optimistic estimate, i.e., choosing a classifier with good error rate under this optimistic estimate, can be a bad idea.

A general rule is that test data should not participate in training. The above scheme makes training and test data identical and thus largely violates the rule.

(2) Splitting known data into training data and test data:

Randomly split D into test data (D0) and training data (D1). Train a classifier C on D1 and test it (i.e., apply it) on D0. Compared with the aforementioned training error, C's error rate on D0 is a more reliable estimate of its error rate on future unseen data. Typically people would use random 2/3 of D as training data D1 and the remaining 1/3 as test data D0.

After we get an estimate of the error rate on D0, we can re-train C (same classification algorithm/method and parameter settings) using the whole known data D. In principle, the more training data, the better classifier. Note that this doesn't violate the aforementioned rule. When estimating the error rate, the training data (D1) and test data (D0) do not overlap. The obtained error rate

is still an estimate of C's error rate after we re-train it using D, since D0, D1 are both supposed to be representative of D, and all of them are supposed to be representative of future data.

(3) Cross-validation:

As you may have noticed, approach (2) allows you to use only 2/3 of known data to train a classifier and only 1/3 to estimate its error rate. Typically the more training data, the better error rate; the more test data, the more accurate estimate of the error rate. When D is large, 2/3 and 1/3 can be large enough, respectively, for low error rate and accurate estimate of the error rate. However, when D is small, training and test data present a tradeoff. When we increase the size of D1 and thus gets less error rate, D0 gets smaller and we will have a less accurate estimate of the error rate. On the contrary, a smaller D1 leads to more errors by our classifier although the estimate of that error rate gets more accurate.

We can use k-fold cross-validation to address this issue. Randomly split the known data D into k equi-size partitions D_1, ..., D_k. Do training/test k times. At the i-th iteration, use D_i as test data and other k-1 partitions as training data. Average the error rates obtained from the k iterations. This way, every record in D has been used exactly once as test record.

Using k-fold cross-validation, we would have produced k classifiers, one for each iteration. Each classifier uses the same model and the same parameters, but different training data. Then we produce a single classifier using the whole known data D. The estimate of its error rate is the average error rate of the k iterations, as mentioned above.

In practice, people often use stratified 10-fold cross-validation (i.e., k=10). "Stratified" means evenly distributing records from multiple classes into D_1, ..., D_k. "Evenly" can only be done roughly. If there are n records (j*k less than n less than (j+1)*k) in class X in the known data, you cannot possibly make each D_i contain the same number of class-X records.

(4) Validation set:

Often times we would want to compare multiple classifiers and choose the best one. These classifiers may be obtained by tuning parameters of a classification algorithm and/or using multiple algorithms.

To choose from the competing classifiers, one may decide to train multiple classifiers on the same training data D1, apply them on the same test data D0, and choose the classifier with the best error rate on D0. Say, C is the chosen classifier—a naive Bayes classifier using laplace smoothing, without applying stemming. The drawback with this scheme is: since C is chosen because it has the best error rate on D0, that error rate is an optimistic estimate of the error rate on future unseen data.

In order to have a more realistic estimate of the error rate, we can instead split D into three partitions— training data (D1), validation data (D2), and test data (D0). A partitioning scheme can be 50%, 30%, 20%, for D1, D2, and D0, respectively. Then we train multiple classifiers on the same D1, apply them on the same D2, choose the one with the best error rate on D2, and then apply the chosen classifier on D0 to obtain an estimate of its error rate on future data.

After choosing C based on D2, we can train C again using D1+D2 as the training data. Remember, generally speaking, the more training data, the better classifier. Then we can apply the re-trained C on D0 to obtain an estimate of its error rate on future data. After getting the estimated error rate on D0, we can even further re-train the chosen C using D1+D2+D0, i.e., the whole known data D.

(5) The idea of having a separate validation set in addition to training/test sets can be combined with cross-validation: Hold out a test set (D0), then evenly split the rest of known data into D1, ..., D10. Do training/test k times. At the i-th iteration, use D_i as validation set and other k-1

partitions as training set. Average the error rates obtained from the k iterations. The classifier C with the best average error rate is chosen. It is trained again using D1, ..., D10 altogether as training data. Then test it on D0 to obtain an estimate of its error rate. Again, we can further retrain it using the whole D.

(6) Consider P2. The test data Test.csv is already hold out, by Kaggle. You will use it to evaluate your classifier, by uploading your classifier's predicted lables on Test.csv into Kaggle. In a more typical setting in your future career, you will have access to a single known data set D. It is up to you to determine how to split it, using the various schemes discussed above. (Based on this reason, the naming of the files Train.csv and Test.csv by Kaggle is a bit confusing. However, it is correct. Earlier I may have given you the impression that it is incorrect.)

Train.csv is provided to you by Kaggle.

If you choose scheme (2), then you simply implement a classification method using train.csv as the training data.

If you choose scheme (4), then you will split train.csv into D1 and D2.

If you choose scheme (5), then you will split train.csv into k partitions.