POLYTECHNIC UNIVERSITY
Department of Computer Science / Finance and Risk Engineering

Estimating and Reducing the Error of a Classifier or Predictor

K. Ming Leung

Abstract: Methods, such as holdout, random subsampling, k-fold cross-validation, and bootstrap, for making error estimation are discussed. Also considered are general techniques, such as bagging and boosting, for increasing model accuracy.

Directory

Table of Contents

1. Accuracy and Error Measures

After building a classifier or predictor, we would like an estimate of how accurately they are for making prediction for unseen samples. You may even have tried different methods to build more than one classifier (or predictor) and now wish to compare their accuracy. We want to define more precisely the meaning of accuracy, and consider ways in which the accuracy can be estimated.

1.1. Classifier Accuracy Measures

Using training data to derive a classifier or predictor, M, and then to estimate the accuracy of the resulting learned model can result in misleading over-optimistic estimate due to over-specialization of the learning algorithm to the data. Instead, accuracy is better measured on a test set consisting of class-labeled samples that were not used to train the model. The **accuracy** of a classifier, $acc(M)$, on a given test set is the percentage of test set samples that are correctly classified by the classifier.

We can also speak of the **error rate** or **misclassification rate** of a classifier, M, which is simply $1 - acc(M)$. If we were to use the training data to estimate the error rate, this quantity is known as the **resubstitution error**. This error estimate is optimistic of the true error rate.

The **confusion matrix** is a useful tool for analyzing how well your classifier can recognize samples of different classes. Such a matrix for the case of two classes is shown in the table.

		Predicted class	
		C_1	C_2
Actual class	C_1	true positive	false positive
	C_2	false negative	true negative

Given k classes, a confusion matrix is a table of at least size $m \times m$. An entry, $CM_{i,j}$ in the first m rows and m columns indicates the number of samples of class i that were labeled by the classifier as class j. For a classifier to have good accuracy, ideally most of the sample would be represented along the diagonal of the confusion matrix, with the rest of the entries being close to zero.

The table may have additional rows or columns to provide totals or recognition rate per class.

Given two classes, we can talk in terms of **positive samples**, *pos*, (samples of the main class of interest, e.g. $buy = yes$) versus **negative samples** (e.g. $buy = no$), *neg*. **True positive**, t_p, refer to the positive samples that were correctly labeled by the classifier, while **true negatives**, t_n, are the negative samples that were correctly labeled by the classifier. **False positives**, f_p, are the negative samples that were incorrectly labeled. Similarly, **false negatives**, f_n, are positive samples that were incorrectly labeled. One can see that the accuracy of a classifier, which defined as the percentage of test set samples that are correctly classified, is given by

$$Acc = \frac{t_p + t_n}{pos + neg}.$$

However there may be other more effective ways to define accuracy. Suppose you have trained a classifier to classify medical data samples as either "*cancer*" or "*not-cancer*". We would like to be able to know how well the classifier can recognize ”*cancer*” samples (the *pos* samples) and how well it recognizes ”*not-cancer*” samples (the *neg* samples). The **sensitivity**, *sen*, and **specificity**, *spec*, measures can be used, respectively, for this purpose. Sensitivity is also referred to as the *true positive recognition rate*, it is the fraction of positive samples that are correctly identified

$$sen = \frac{t_p}{pos}.$$

Specificity is the *true negative recognition rate*, it is the fraction of negative samples that are correctly identified

$$spec = \frac{t_n}{neg}.$$

In addition, we may use **precision**, *prec*, to denote the fraction of samples labeled as ”*cancer*” that actually are ”*cancer*” samples,

$$prec = \frac{t_p}{t_p + f_p}.$$

It can be shown that accuracy is a function of sensitivity and specificity:

$$acc = sen\ \frac{pos}{pos + neg} + spec\ \frac{neg}{pos + neg}.$$

It may also be important to assess the **costs** and **benefits** associated with a classification model. The cost associated with a false positive (such as, incorrectly predicting that a cancerous patient is not cancerous) is far greater than that of a false positive (incorrectly labeling a noncancerous patient as cancerous). In such cases, we can outweigh one type of error over another by assigning a different cost to each. Examples of such cost-benefit analysis are loan application decisions and target marketing mail-outs. The cost of giving out a loan to a defaulter greatly exceeds that of the lost business incurred by denying a loan to a non-defaulter. Similarly the cost of mail-outs to numerous households that do not respond may outweigh the cost of lost business from not mailing to households that would have responded.

1.2. Predictor Error Measures

Let T be a test set of the form $(\mathbf{X}_1, y_1), (\mathbf{X}_2, y_2), \ldots, (\mathbf{X}_m, y_m)$, where $(\mathbf{X}_i$ are the n-dimensional test samples with associated known values, y_i, for a response variable, y, and m is the number of samples in T. Since predictors return a continuous values rather than a categorical label, it is difficult to say exactly whether the predicted value, y_i', for $\mathbf{X}_i$ is correct. Instead we need to look at how far off the predicted value is from the actual known value.

The **lost** functions measure the error between y_i and the predicted value, y_i'. The most common loss functions are

$$Absolute\ error = |y_i - y_i'|$$

$$Squared\ error = (y_i - y_i')^2.$$

The test error or generalization error, is the average loss over the test set. Thus, we have the following error rates (percentages)

$$Mean\ absolute\ error = \frac{1}{m} \sum_{i=1}^{m} |y_i - y_i'|$$

$$\textit{Mean squared error} = \frac{1}{m} \sum_{i=1}^{m} (y_i - y_i')^2.$$

Sometimes we take the square root of the mean squared error, the resulting error measure is called the **root mean squared error**. This is useful in that it allows the error measured to be of the same unit as the quantity being predicted.

Sometimes, we may want the error to be relative to what it would have been if we had just predicted $\bar{y}$, the mean value for y from T.That is, we normalize the total loss by dividing by the total loss incurred from always predicting the mean. Relative measure of error include:

$$\textit{Relative absolute error} = \frac{\sum_{i=1}^{m} |y_i - y_i'|}{\sum_{i=1}^{m} |y_i - \bar{y}|}$$

$$\textit{Relative squared error} = \frac{\sum_{i=1}^{m} (y_i - y_i')^2}{\sum_{i=1}^{m} (y_i - \bar{y})^2}.$$

We can take the square root of the relative squared error to obtain the **root relative squared error** so that the resulting error has the same unit as the quantity predicted.

2. Evaluating the Accuracy of a Classifier or Predictor

How can we use the above measures to obtain reliable estimate of classifier accuracy or predictor error? Holdout, random subsampling, cross-validation, and the bootstrap are common techniques for assessing accuracy based on randomly sampled partitions of the given data set. The use of such techniques to estimate accuracy or error increases the overall computation time, yet is useful for model selection.

2.1. Holdout Method and Random Subsampling

In the **holdout** method, the given data are randomly partitioned into two sets, a training set (typically consisting of two-thirds of the data) and a test set (typically consisting of the remaining one-third of the data). The training set is used to derive the model, whose accuracy is estimated with the test set.

Random subsampling is a variation of the holdout method in which the holdout method is repeated a number of times. The overall accuracy is taken as the average of the accuracies obtained from each run. For prediction, we can take the average of the prediction error rate.

2.2. Cross-Validation

In k**-fold cross-validation**, the initial data are randomly partitioned into k mutually exclusive subsets of "folds", D_1, D_2,...,D_k, each of approximately equal size. Training and testing is performed k times. In run i, subset D_i is used as the test set, and the remaining subsets are collectively used to train the model. Thus each sample is used $(k-1)$ times for training and once for testing.

For classification, the accuracy estimate is the overall number of correct classification from each run, divided by the number of sample in the initial data, m. For prediction, the error estimate can be computed as the total error from the k runs, divided by m.

Leave-one-out is a special case of k-fold cross-validation where $k = m$. That is,only one sample is "left out" at a time for the test set.

In **stratified cross-validation**, the folds are stratified so that the class distribution of the samples in each fold is approximately the same as in the initial data.

In general, stratified 10-fold cross-validation is recommended for estimating accuracy due to its relatively low bias and variance.

2.3. Bootstrap

The **bootstrap** method samples the given training data uniformly with replacement. Each time a sample is selected, it remains in the training set and is equally likely to be selected again.

There are several bootstrap methods. A commonly used one is the **.632 bootstrap**, which works as follows. A given data set of m samples is sampled m times, with replacement, resulting in a bootstrap training set of m samples. In this training set, some of the original samples will occur more than once. But some samples did not make it into the training set. These are included in the test set. Since each sample has a probability of $1/m$ of being selected in any given selection, so the probability of not being chosen is $(1-1/m)$. We have to select like this m times, so the probability that a sample will not be chosen at all is $(1-1/m)^m$. If m is large, the probability approaches

$e^{-1} = 0.384$. Thus, 36.8% of samples will not be selected for training and thereby end up in the test set, and the remaining 63.2% will form the training set.

We can repeat the sampling procedure k times, where in each run, we use the current test set to obtain an accuracy estimate of the model obtained from the current bootstrap sample. The overall accuracy of the model is then estimated as

$$\mathrm{Acc}(M) = \sum_{i=1}^{k} \left(\left(1 - \frac{1}{e}\right) \times \mathrm{Acc}(M_i)_{\text{test\_set}} + \frac{1}{e} \times \mathrm{Acc}(M_i)_{\text{train\_set}} \right),$$

where $\mathrm{Acc}(M_i)_{\text{test\_set}}$ is the accuracy of the model obtained with bootstrap sample i when it is applied to test set i. $\mathrm{Acc}(M_i)_{\text{train\_set}}$ is the accuracy of the model obtained with bootstrap sample i when it is applied to the original set of data samples. The bootstrap method works well with small data sets.

3. Model Selection

Suppose that we have generated two models. M_1 and M_2 (for either classification or prediction) from our data. We have performed 10-fold cross-validation to obtain a mean error rate for each. How can we determine which model is best? We may select the model with the lowest error rate, however the mean error rates are just estimates of error on the try population of future data cases. There can be considerable variance between error rates within an given 10-fold cross-validation experiment. Although the mean error rates obtained for M_1 and M_2 may appear different, that difference may not be statistically significant. What if any difference between the two may just be attributed to chance?

3.1. Estimating Confidence Intervals

To determine if there is any "real" difference in the mean error rates of two models, we need to employ a test of statistical significance and obtain some confidence limits for our mean error rates. One common technique is to used the **t-test** or **Student's t-test**. You can consult the reference [1] for details.

3.2. ROC Curves

ROC Curves are a useful visual tool for comparing two classification models. ROC stands for Receiver Operating Characteristics. ROC Curves come from signal detection theory.

An ROC curve shows the trade-off between the true positive rate or sensitivity and the false-positive rate for a given model. That is, given a two-class problem, it allows us to visualize the trade-off between the rate at which the model can actually recognize 'yes' cases versus the rate at which it mistakenly identifies 'no' cases as 'yes' for different "portions" of the test set. Any increase in the true positive rate occurs at the cost of an increase in the false-positive rate. The area under the ROC curve is a measure of the accuracy of the model. [1]

4. Ensemble Methods – Increasing the Accuracy

Bagging and boosting are two general strategies for improving classifier and predictor accuracy. They are examples of **ensemble meth-**

ods, or methods that uses a combination of models. Each combines a series of k learned models, $M_1, M_2, \ldots, M_k$, with the aim of creating an improved composite model, M_*. Both bagging and boosting can be used for classification as well as prediction.

4.1. Bagging

Bagging stands for bootstrap aggregation. Given a set, D, of m samples, bagging works as follows in the case of classification.

Choose a number k of classification models to be used in the ensemble. For each $i = 1, 2, \ldots, , k$, a training set, D_i of m samples is sampled with replacement from the original set, D. A classifier model, M_i, is learned for each training set, D_i. To classify an unknown sample, $\mathbf{X}$, each classifier, M_i, returns its class prediction, which counts as one vote. The bagged classifier, M_*, counts the votes and assigns the class with the most votes to $\mathbf{X}$.

Bagging can be applied to the prediction of continuous values by taking the average value of each prediction for a given test sample.

The bagged classifier often has significantly greater accuracy than a single classifier derived from D, the original training data set. It will not be considerably worst and is more robust to the effects of noisy data. The increased accuracy occurs because the composite model reduces the variance of the individual classifiers. For prediction, it was theoretically proven that a bagged predictor will always have improved accuracy over a single predictor derived from D.

4.2. Boosting

In **boosting**, weights are assigned to each training sample. A series of k classifiers is iteratively learned. After a classifier, M_i is learned, the weights are updated to allow the subsequent classifier, M_{i+1}, to "pay more attention" to the training samples that were misclassified by M_i.. The final boosted classifier, M_*, combines the votes of each individual classifier, where the weight of each classifier's vote is a function of its accuracy. The boosting algorithm can be extended for the prediction of continuous values.

Adaboost is a popular boosting algorithm. We are given D, a data set of m class-labeled samples, $(\mathbf{X}_1, y_1), (\mathbf{X}_2, y_2), \ldots, (\mathbf{X}_m, y_m)$, where y_j is the class label of sample $\mathbf{X}_j$. We need to choose a value for k to be the total number of classifiers to be created, and adopt a specific classification scheme.

- **Adaboost: Constructing Composite Model,** M_*

The Adaboost algorithm is:

```
 1. initialize the weight of each sample, $w_j$ in $D$ to $1/m$;
 2. for $i = 1$ to $k$ do
 3.     sample $D$ with replacement according to $w_j$ to obtain $D_i$;
 4.     use training set $D_i$ to derive a model, $M_i$;
 5.     compute $error(M_i)$, the error rate of $M_i$
 6.     if $error(M_i) > 0.5$ then
 7.         reinitialize the weights to $1/m$
 8.         go back to step 3;
 9.     endif
10.     for each sample in $D_i$ that was correctly classified do
```

```
11.            $w_j = w_j \times$ $error(M_i)$ /( 1 - $error(M_i)$ );
12.        normalize the weight of each sample
13.  endfor
```

To compute the error rate of model M_i in step 5, we sum the weights of each of the samples in D_i that M_i misclassified. That is

$$error(M_i) = \sum_{j=1}^{m} w_j \times err(\mathbf{X}_j),$$

where $err(\mathbf{X}_j)$ is the misclassification error of sample $\mathbf{X}_j$. If the sample was misclassified, then $err(\mathbf{X}_j)$ is 1, otherwise it is 0. If the performance of M_i is so poor that its error exceeds 0.5, then we abandon it in step 6. Instead, in steps 7-9, we try again by generating a new D_i training set, from which we derive a new M_i.

The error rate of M_i affects how the weights of the training samples are updated. If a sample in round i was correctly classified, it weight is multiplied by $error(M_i)$ /(1 - $error(M_i)$).

Once the the weight of all the correctly classified samples are updated (step 10), the weights for all samples (including the misclassified ones) are normalized so that their sum remains the same as before. To normalize a weight, we multiply it by the sum of the old weights, divided by the sum of the new weights. As a result, the weights of misclassified samples are increased and the weights of correctly classified samples are decreased.

• Using M_* to Classify An Unknown Sample

Once boosting is complete, the ensemble of classifiers is used to predict the class label of an unknown sample, **X**. Unlike bagging, where each classifier was assigned an equal vote, boosting assigns a weight to each classifier's vote, based on how well the classifier performed. The lower a classifier's error rate, the more accurate it is, and therefore, the higher its weight for voting should be. The weight of classifier M_i's vote is chosen to be

$$\log\left(\frac{1 - error(M_i)}{error(M_i)}\right).$$

For each class, c, we sum the weights of each classifier that assigned class c to **X**. The class with the highest sum is the "winner" and is returned as the class prediction for sample **X**.

The following summarizes the algorithm for using the composite model to classify an unknown sample, **X**:

```
1. initialize the weight of each class, $w_i$ to 0;
2. for $i = 1$ to $k$ do // for each classifier;
3.     $w_i = \log\left(\frac{1-error(M_i)}{error(M_i)}\right)$; // weight of classifier's vote
4.     $c = M_i(\mathbf{X})$; // get class prediction for X from $M_i$
5.     add $w_i$ to weight for class $c$
6. endfor
7. return the class with the largest weight
```

4.3. Remarks on Bagging and Boosting

Because of the way boosting focuses on the misclassified samples, it risks overfitting the resulting composite model to such data. Therefore, sometimes the resulting "boosted" model may be less accurate than a single model derived from the same data. Bagging is less susceptible to model overfitting. While both can significantly improve accuracy in comparison to a single model, boosting tends to achieve greater accuracy.

References

[1] Jiawei Han and Micheline Kamber, *Data Mining: Concepts and Techniques*, Elsevier 2006, ISBN 1558609016. This lecture notes is based on materials in chapter 6.

15