1 Evaluating a Classifier

When we use a classifier, such as Naive Bayes, it’s important to evaluate it effectiveness. How do we know it is working? How do we characterize the kinds of mistakes it makes?

In order to make an estimate of the quality of a classifier, we want to evaluate its performance on a test set, which should be data the classifier did not use during its training process. After running the data through the classifier, there are a number of methods of analyzing the results.

One of the most common methods of representing classifier results is as a confusion matrix. A confusion matrix indicates not only how many instances of each class $C_i$ the classifier correctly labeled, but also how many instances of class $C_i$ were labeled as each of the other classes $C_j$. For a 2-class problem, the confusion matrix is a 2x2 matrix. For an $N$ class problem, the confusion matrix is $N \times N$.

The rows represent the true class values and the columns represent the predicted class values. For example, consider the case in which we are classifying dogs as Jack Russell Terriers or Germans Shepherds. We could imagine a confusion matrix that looks like this:

<table>
<thead>
<tr>
<th>Label</th>
<th>Predict Jack Russell</th>
<th>Predict German Shepherd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jack Russell Terrier</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>German Shepherd</td>
<td>30</td>
<td>70</td>
</tr>
</tbody>
</table>

In this case the classifier has classified 200 dogs. It correctly identifies all 100 Jack Russell Terriers as such. But with German Shepherds, it doesn’t do so well (it mis-categorizes 30 of them as Jack Russells). Perhaps it is because we are accidentally considering puppies?

Next, consider the iris data set, in which we have 3 classes (one for each species of iris):

<table>
<thead>
<tr>
<th>Label</th>
<th>Iris-setosa (predicted)</th>
<th>Iris-versicolor</th>
<th>Iris-virginica (predicted)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris-setosa (actual)</td>
<td>50</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Iris-versicolor</td>
<td>0</td>
<td>47</td>
<td>3</td>
</tr>
<tr>
<td>Iris-virginica (actual)</td>
<td>0</td>
<td>3</td>
<td>47</td>
</tr>
</tbody>
</table>

This classifier has done a pretty good job. However, there is some confusion. Which two iris species get confused?
1.1 Binary classification

When there are two classes, (i.e. it is a binary classification problem), then we use a special set of terminology—we speak of true and false positives and negatives. There are several common metrics compute to assess performance. It often useful to look at them per class value (e.g. in our dog data set to look at the “Jack Russell” class). To be general, we will call the class $a$.

- True Positives (TP): the number of data points in class $a$ that are classified as class $a$
- True Negatives (TN): the number of data points not in class $a$ that are classified as some class other than $a$
- False Positives (FP): the number of data points not in class $a$ that are classified as class $a$
- False Negatives (FN): the number of data points in class $a$ that are classified as some class other than $a$
- TP rate: $TP/(TP+FN)$
- FP rate: $FP/(FP+TN)$
- precision: $TP/(TP+FP)$
- recall: same as TP rate
- F-Measure (used in information retrieval tasks):
  \[
  \frac{2 \cdot \text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}} = \frac{2TP}{2TP + FP + FN}
  \]

(Note that I have defined these terms in such a way that it is possible to use them for classification with more than 2 classes. But it is more common to use them in binary classification. So I have put these terms here.)

Let’s examine a confusion matrix for a binary classifier. For example, on a network inference task the confusion matrix may look like any of the three cases below.

In the case (a), the system finds only 60% of the true edges, but it mostly avoids incorrectly detecting an edge when none exists. In case (b), the false negatives and false positives are balanced: the system makes false positive and false negative errors at about the same rate. In the case (c), the system finds almost all of the faces in the data set, but it also classifies many non-face examples as faces.

In my research, I vary the threshold for determining an edge is present. The lower the threshold, the more edges are detected. To see how all the method performs without having to fix the threshold, I plot a Receiver Operating Characteristic (ROC) curve. For each threshold, I compute the false positive rate and the true positive rate. I associate the x-axis with the false positive rate and the y-axis with the true positive rate. I plot points for all the thresholds I test and find a curve that is slightly above the diagonal. This means, my method is better (but not by much) than random guessing.
Almost all machine learning methods have some type of parameter that lets the developer tune the performance of the system to encourage or discourage false positives or negatives. This means there are many possible operating points. The purpose of the system defines which operating point is most appropriate. (Note that for Naive Bayes, we don’t have much to control. We will have more to control in future methods, though.)

### 1.2 Testing data

It is important to reserve some data for testing. Suppose we use all of our data to construct a tree, and that we have all pure leaves in the tree. Then we test our tree on the training data. We compute the error and find that all data points are correctly classified. Does that tell us anything about how well the tree will classify new data points? No.

There are two basic strategies:

1. Use separate data for validation and testing.

2. Use cross-validation. If you don’t have a large data set, then you probably don’t want to leave any data out of your training set. So, the solution is to separate your data into partitions (folds) and let each fold take a turn at being a testing set.

   - Separate your data into $D$ folds
   - For each fold
     - Use the remaining data to train a classifier (e.g. build a tree)
     - Test the classifier using the data in the fold. This involves classifying the data point according to this tree.

Note that this means that each data point is used as a test point exactly once. So you have a predicted class for each data point. You also have the actual values of the class for each data point. So you can compute a confusion matrix and all of the other performance metrics listed above.
Regardless of which method you choose, you will want to be smart about how you partition the data. It is important to **stratify** the data – keep the proportion of data points from each class consistent with its proportion in the entire data set. So if 50% of your dogs are Jack Russells, then each partition should be 50% Jack Russell.

These notes were adapted from those of Bruce Maxwell.