Evaluating and using ML classifiers

Chapter 2

Evaluating classifiers

The process of fitting a model (linear in our case) is called training. The data that was used to come up with the model is called the training data.

After we train a classifier we would like to measure its accuracy. There are many ways to measure accuracy, the simplest one being the percentage of examples that are correctly classified.

Is the accuracy of the classifier on the training data representative of what we would expect to obtain on unseen data?

No!

Why not?

Using classifiers

At this point we have learned two classification algorithms:

- The closest centroid classifier
- The perceptron algorithm

How do we measure how well they are doing on my dataset?

\[
\begin{align*}
\text{Input:} & \text{ labeled data } D \text{ in homogeneous coordinates} \\
\text{Output:} & \text{ weight vector } w \\
& w = 0 \\
& \text{ converged } = \text{ false} \\
& \text{ while } \text{ not converged} : \\
& \text{ converged } = \text{ true} \\
& \text{ for } i = 1, \ldots, |D| : \\
& \text{ if } x_i \text{ is misclassified update } w \text{ and set } \text{ converged } = \text{ false}
\end{align*}
\]
Evaluating classifier performance

Is the accuracy of the classifier on the training data representative of what we would expect to obtain on unseen data?

No! For several reasons:

- We have a finite sample that may not fully represent the domain that we are trying to model.
- Data is often noisy:
  - The inputs $x_i$ might be noisy
  - The labels may be incorrectly assigned
- A consequence of noise: not a good idea to perfectly fit the training data – overfitting the noise.

The simplest classifier evaluation protocol:

- Divide your labeled data into a training set and test set.
- Train a classifier on the training set
- Classify the examples in the test set, and measure accuracy

We will consider other protocols

Go to PyML demo

Measuring classifier performance

Accuracy or success rate of a classifier is defined as:

$$acc = \frac{1}{|Te|} \sum_{x \in Te} I[\hat{c}(x) = c(x)]$$

Where:

- $Te$ is the test set used for evaluating performance
- $I[\cdot]$ is the indicator function that evaluates to 1 if its argument is true and 0 otherwise
- $c(x)$ is the true class of $x$
- $\hat{c}(x)$ is the predicted class of $x$

The error rate of a classifier is $1 - acc$
Measures of classifier performance

Classifier performance can be summarized by a table known as the confusion matrix or contingency table:

<table>
<thead>
<tr>
<th>Predicted Labels:</th>
<th>-1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>True Labels</td>
<td>-1 1439 61</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 62 1438</td>
<td></td>
</tr>
</tbody>
</table>

Let's take a closer look at the contingency table:

How do we compute accuracy from the contingency table?

Sum of the diagonal elements of the matrix divided by the number of examples

For binary classification problems it is customary to express the contingency table as:

<table>
<thead>
<tr>
<th>Predicted Labels:</th>
<th>-1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>True Labels</td>
<td>-1 TN FP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 FN TP</td>
<td></td>
</tr>
</tbody>
</table>

TP - number of true positives
TN - number of true negatives
FP - number of false positives
FN - number of false negatives
Measures of classifier performance

For binary classification problems it is customary to express the contingency table as:

<table>
<thead>
<tr>
<th></th>
<th>-1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>TN</td>
<td>FP</td>
</tr>
<tr>
<td>1</td>
<td>FN</td>
<td>TP</td>
</tr>
</tbody>
</table>

Neg = TN + FP
Pos = TP + FN

True positive rate/sensitivity/recall: $\text{TP} / \text{Pos}$
True negative rate/specificity: $\text{TN} / \text{Neg}$
False positive rate: $\text{FP} / \text{Neg}$
Precision: $\text{TP} / (\text{TP + FP})$

Balanced success rate

Accuracy for a binary classification problem can be expressed as:

$$\text{acc} = \frac{1}{\text{Pos}} \text{tpr} + \frac{1}{\text{Neg}} \text{tnr}$$

Balanced success rate replaces this with

$$\frac{1}{2} \text{tpr} + \frac{1}{2} \text{tpr}$$

The choice of classification threshold

All the classifiers we will study provide a scoring function $f(x)$ whose magnitude indicates how sure we are it belongs to a given class.

The choice of the threshold for choosing positive examples is somewhat arbitrary, and in a given application we may prefer to ignore positive predictions that are associated with small values of $f(x)$.

To have a view of classifier performance that is independent of the choice of threshold we consider the ROC curve.

Measures of classifier performance

Suppose you have a dataset with very few positive examples compared to negative examples (unbalanced data)

A classifier that classifies every example as negative would still attain high accuracy (this is called the majority class classifier).

Need an alternative measure of accuracy!
The ROC curve is a plot of the true positive rate as a function of false positive rate as you vary the classification threshold.

How does the ROC curve of a perfect classifier look like?

For a random classifier?

ROC curves and ranking

An ROC curve is often summarized by the area under the curve (AUC).

$$AUC = \frac{1}{\text{Pos} \cdot \text{Neg}} \sum_{x \in T^0, x' \in T^\oplus} I[f(x) > f(x')]$$

This is also a nice way of comparing classifiers:

Cross-validation

Sometimes you have a small amount of data, so the accuracy estimated using a single train/test split may not be dependable.

Solution: Cross-validation

Generate such a plot using PyML's roc module (the plotROCs function):

```python
>>> from pyml.evaluators import roc
>>> roc.plotROCs(results_list)
```
Cross-validation

Cross validation:
- Randomly partition the data into k parts ("folds").
- Set one fold aside for testing and train a model on the remaining k-1 folds and evaluate it on the test fold.
- Repeat until each fold has been used in testing

5-fold cross-validation

The reported accuracy is the average over the accuracies for each fold (not trivial for ROC curves!)
Cross validation:
- Randomly partition the data into k parts ("folds").
- Set one fold aside for testing and train a model on the remaining k-1 folds and evaluate it on the test fold.
- Repeat until each fold has been used in testing.

The extreme case where k = number of examples is called leave-one-out cross-validation.