15-388/688 - Practical Data Science: Evaluating ML models

J. Zico Kolter
Carnegie Mellon University
Spring 2018

Outline

Evaluating machine learning algorithms

Classification metrics

A common strategy for evaluating algorithms

1. Divide data set into training and holdout sets

2. Train different algorithms (or a single algorithm with different hyperparameter settings) using the training set

3. Evaluate performance of all the algorithms on the holdout set, and report the best performance (e.g., lowest holdout error)

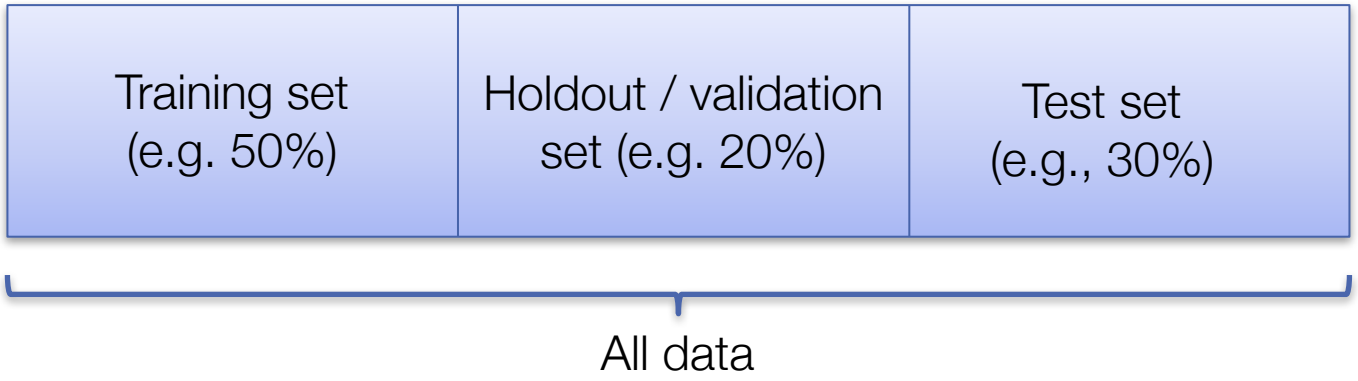
What is wrong with this?

Issues with the previous evaluation

Even though we used a training/holdout split to fit the *parameters*, we are still effectively fitting the *hyperparameters* to the holdout set

Imagine an algorithm that ignores the training set and makes random predictions; given a large enough hyperparameter search (e.g., over random seed), we could get perfect holdout performance

What to do instead



1. Divide data into training set, holdout set, and test set

2. Train algorithm on training set (i.e., to learn parameters), use holdout set to select hyperparameters

3. (Optional) retrain system on training + holdout

4. Evaluate performance on test set

In practice…

"Leakage" of test set performance into algorithm design decisions in almost always a reality when dealing with any fixed data set (in theory, as soon as you look at test set performance once, you have corrupted that data as a valid set set)

This is true in research as well as in data science practice

The best solutions: evaluate your system "in the wild" (where it will see truly novel examples) as often a possible; recollect data if you suspect overfitting to test set; look at test set performance sparingly

An interesting and very active area of research: adaptive data analysis (differential privacy to theoretically guarantee no overfitting)

Outline

Evaluating machine learning algorithms

Classification metrics

Classification metrics

So far, we have considered accuracy (0/1 loss) as the primary method for evaluating classifiers

However, sometimes the benefits for correctly classifying positive and negative examples are different, as are the costs for predictive a positive example to be negative, and vice versa

In cancer dataset, it is a very different thing (in terms of real-world effects) to predict that an actually malignant tumor is benign, versus predicting a benign tumor is malignant

Confusion matrix

A *confusion matrix* explicitly lists the number of examples for each actual class and each prediction

	Predicted Positive	Predicted Negative
Actual Positive	True Positive	False Negative
Actual Negative	False Positive	True negative

Can compute these (and all associated metrics) on training / holdout / testing sets, but we'll just show examples on training sets here

```
import sklearn.metrics
sklearn.metrics.confusion_matrix(y, clf.predict(X))
```

Derived quantities

Several common metrics are associated with entries of the confusion matrix (TP = true positive, FP = false positive, TN = true negative, TP = true negative)

$$\text{TP Rate (also called Recall)} = \frac{TP}{TP + FN}$$

$$\text{FP Rate} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Different metrics (and man others) are standard for different domains

Changing the prediction threshold

Classifiers are implicitly trained around a “threshold” of zero (positive hypothesis means predict positive, negative means predict negative)

But there is no reason to use only this threshold when we want to make predictions (may want to “overpredict” one class or the other)

Key idea: by *sorting* the hypothesis function outputs, and adjusting the threshold at which we call something positive or negative, we can sweep out *all* possible classifications that a classifier can produce

Example thresholds

Sorted hypothesis function outputs (assume 10 total examples, 5 total positive examples):

$$\text{sorted}(h_\theta(x^{(i)})) = \begin{bmatrix} 10 \\ 9 \\ 8.5 \\ \vdots \end{bmatrix}, y = \begin{bmatrix} +1 \\ -1 \\ +1 \\ \vdots \end{bmatrix}$$

TP Rate = 0.0, FP Rate = 0.0

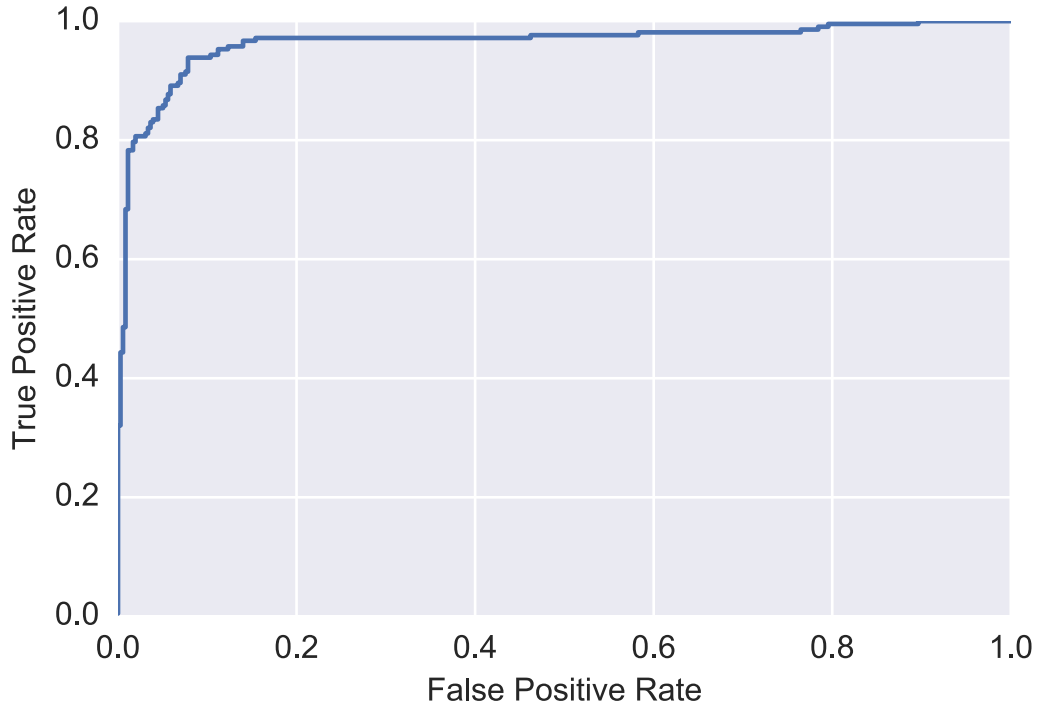
TP Rate = 0.2, FP Rate = 0.0

TP Rate = 0.2, FP Rate = 0.2

TP Rate = 0.4, FP Rate = 0.2

ROC Curve

If we plot the true positive rate versus the false positive rate for this procedure, we get a figure known as an ROC (receiver operating characteristic) curve



Precision recall curves

We can perform similar operations for other metrics, to for e.g. a precision-recall curve (plot of recall vs. precision as threshold varies)