Evaluation metrics and model selection

Marta Arias

Dept. CS, UPC

Fall 2018

Quantifying the performance of a binary classifier, I

x_1 ... x_n	True class	Predicted class		
— ... —	0	0	correct	*true negative*
— ... —	0	1	mistake	*false positive*
— ... —	1	0	mistake	*false negative*
— ... —	1	1	correct	*true positive*

Confusion matrix

		Predicted class	
		positive	negative
True class	positive	*tp*	*fn*
	negative	*fp*	*tn*

- *tp*: true positives
- *tn*: true negatives
- *fp*: false positives (false alarms)
- *fn*: false negatives

Confusion matrix

From the scikit-learn documentation

Examples

```
>>> from sklearn.metrics import confusion_matrix
>>> y_true = [2, 0, 2, 2, 0, 1]
>>> y_pred = [0, 0, 2, 2, 0, 2]
>>> confusion_matrix(y_true, y_pred)
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])
```

```
>>> y_true = ["cat", "ant", "cat", "cat", "ant", "bird"]
>>> y_pred = ["ant", "ant", "cat", "cat", "ant", "cat"]
>>> confusion_matrix(y_true, y_pred, labels=["ant", "bird", "cat"])
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])
```

In the binary case, we can extract true positives, etc as follows:

```
>>> tn, fp, fn, tp = confusion_matrix([0, 1, 0, 1], [1, 1, 1, 0]).ravel()
>>> (tn, fp, fn, tp)
(0, 2, 1, 1)
```

Quantifying the performance of a binary classifier, II

Confusion matrix

		Predicted class	
		positive	negative
True class	positive	tp	fn
	negative	fp	tn

Accuracy, hit ratio

$$acc = \frac{tp + tn}{tp + tn + fp + fn}$$

Error rate

$$err = \frac{fp + fn}{tp + tn + fp + fn}$$

Alternative measures

Sometimes accuracy is insufficient

- Ability to detect positive examples:

  Sensitivity (recall in IR): ratio of true positives to all positively labeled cases;

$$recall = \frac{tp}{tp + fn}$$

- Precision: ratio of true positives to all positively predicted cases;

$$prec = \frac{tp}{tp + fp}$$

- Specificity: ratio of true negatives to all negatively labeled cases.

$$spec = \frac{tn}{tn + fn}$$

Why precision/recall is important sometimes

The unbalanced data case

If we have a vast majority of one (uninteresting) class, and a few rare cases we are interested in

- Fraud detection
- Diagnosis of a rare disease

Example

99.9% of examples are negative, 0.1% of examples are positive (e.g. fraudulent credit card purchases). Easy to get very good accuracy with "always predict negative" simple classifier.

What is precision and recall in this case?

Precision: from all purchases tagged as fraudulent, how many were in fact fraudulent?

Recall: from all fraudulent purchases, how many were detected?

The main objective

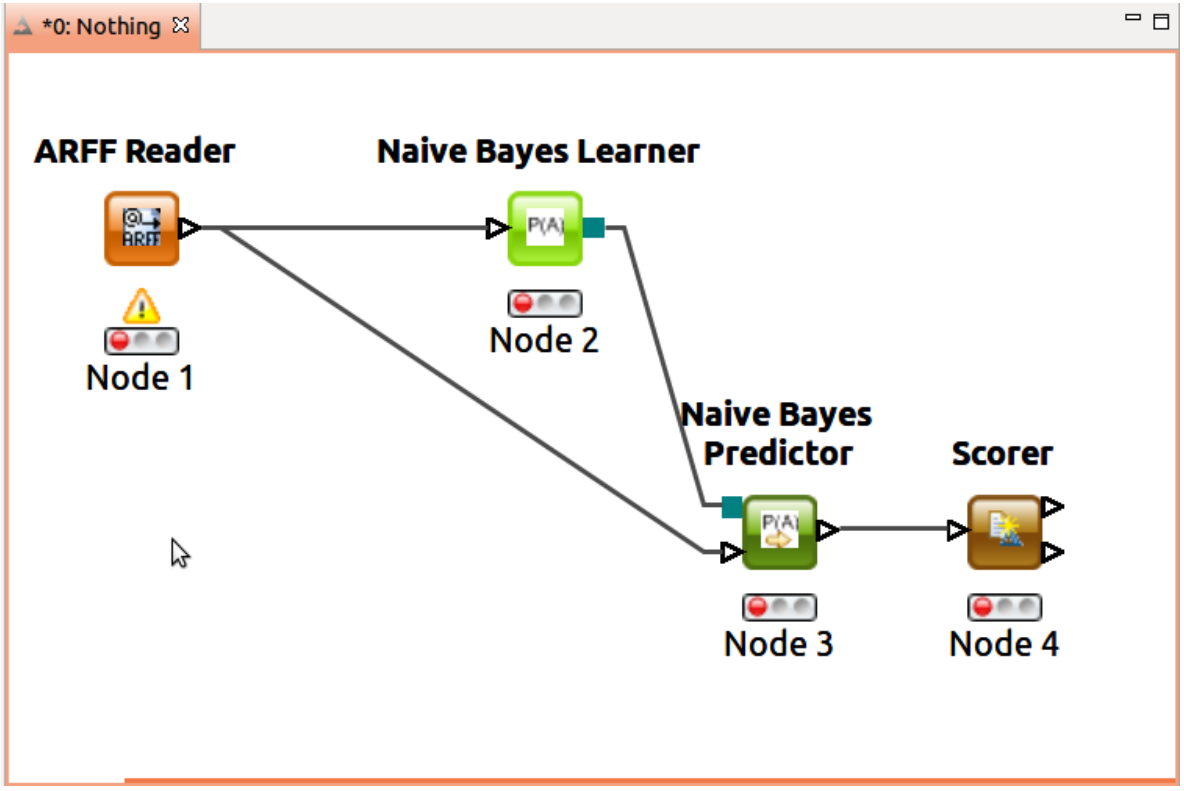
Learning a good classifier

A good classifier is one that has good generalization ability, i.e. is able to predict the label of unseen examples correctly

How to Test a Predictor, I

On the original data?

Training error



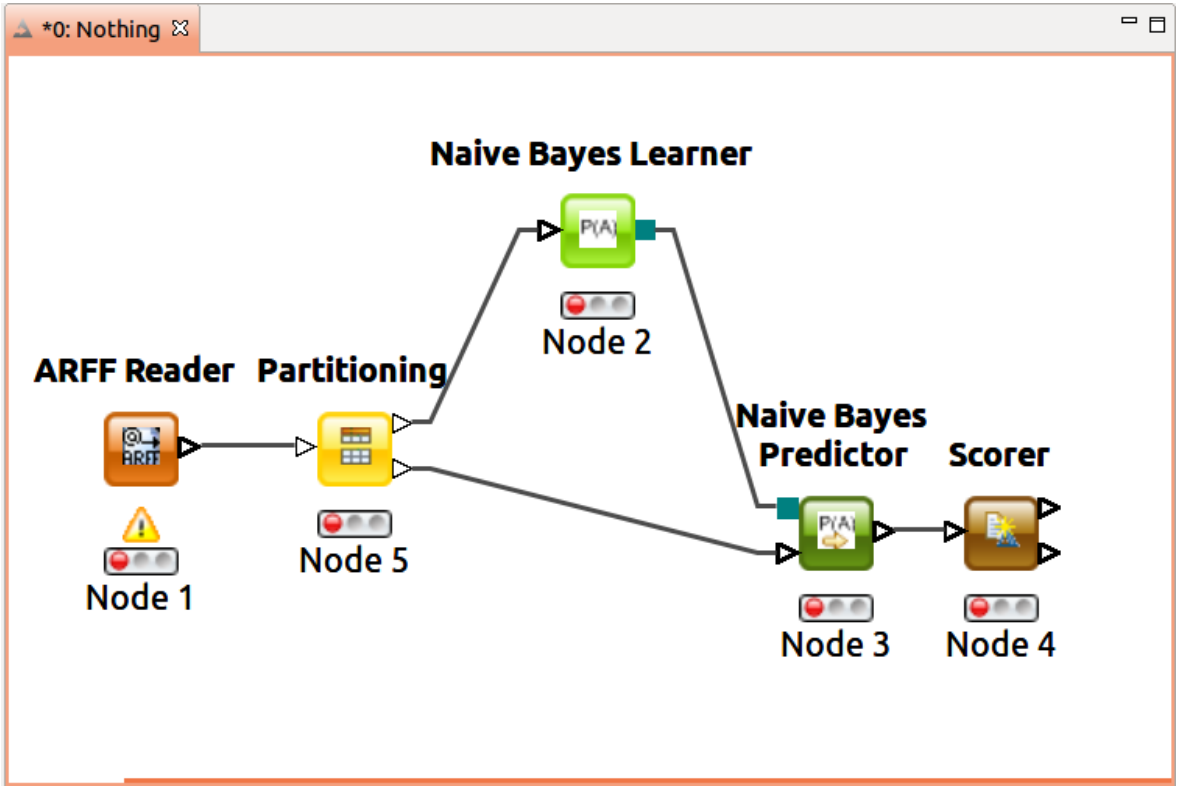
Far too optimistic!

How to Test a Predictor, II

On holdout data?

Test error

after training on a different subset.



How to Test a Predictor, III

Advantages and disadvantages

Training error

- Employs data to the maximum.
- However, it cannot detect overfitting:
  - A predictor overfits when it adjusts very closely to peculiarities of the specific instances used for training.
  - Overfitting may hinder predictions on unseen instances.

Holdout data

- Requires us to balance scarce instances into two tasks: training and test.
- Usual: train with 2/3 of the instances — but, which ones?
- It does not sound fully right that some available data instances are never seen for training.
- It sounds even worse that some are never used for testing.

Code for train-test split

From the scikit-learn documentation

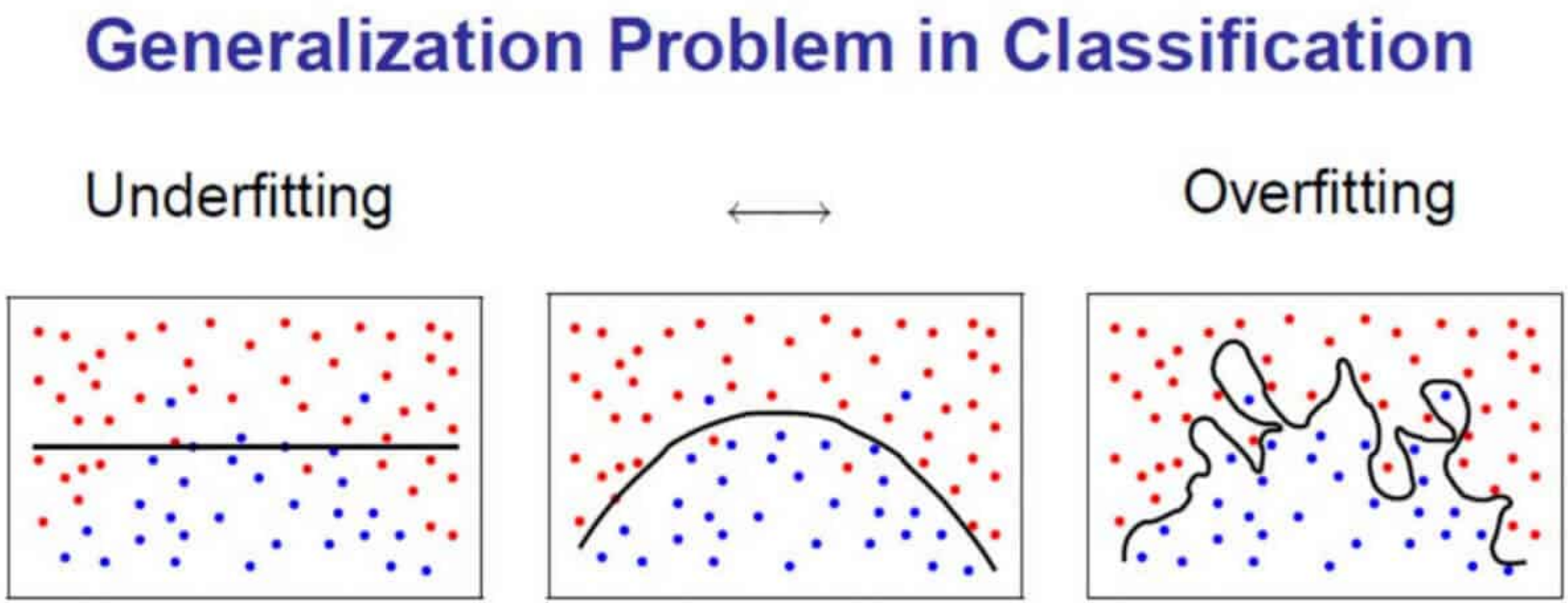
Examples

```
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split
>>> X, y = np.arange(10).reshape((5, 2)), range(5)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
>>> list(y)
[0, 1, 2, 3, 4]
```

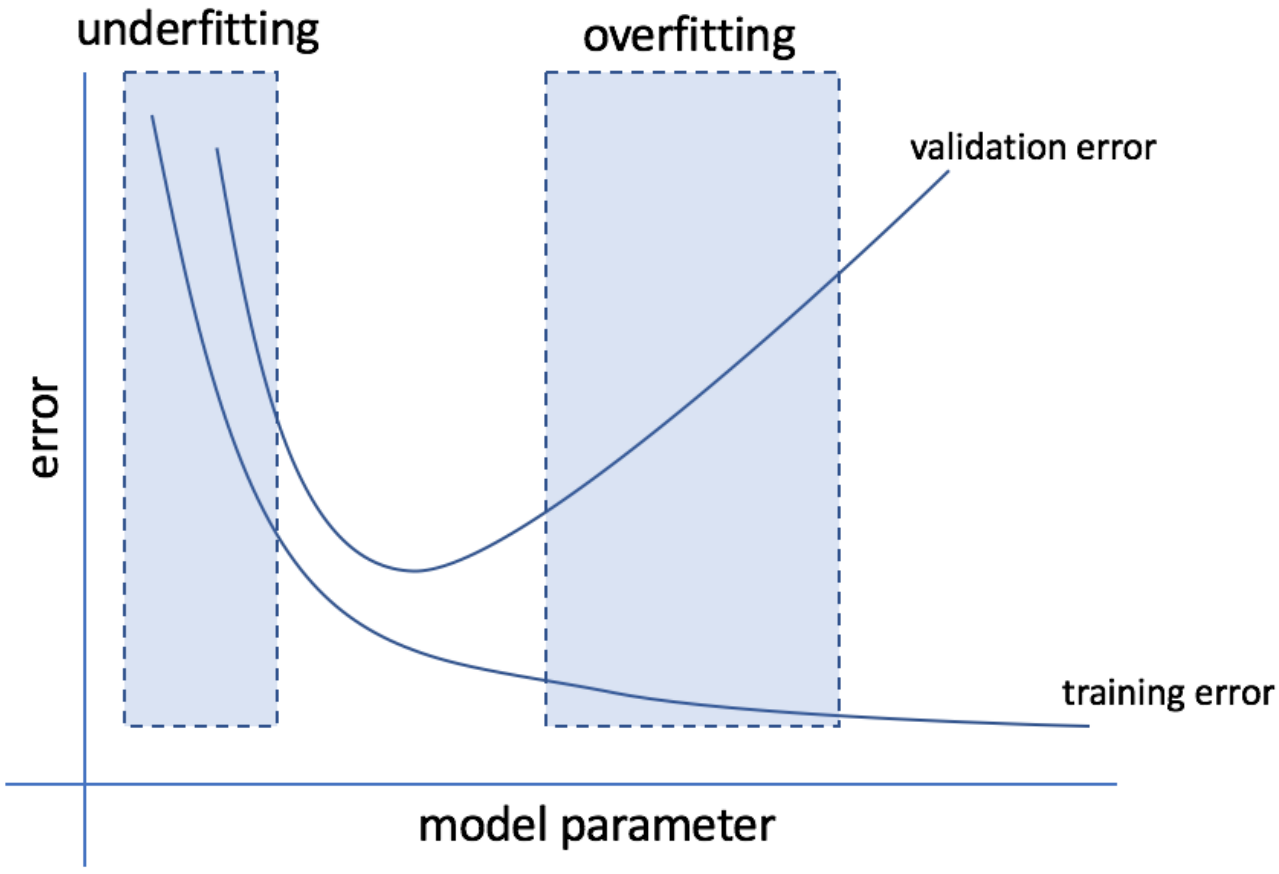
```
>>> X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.33, random_state=42)
...
>>> X_train
array([[4, 5],
       [0, 1],
       [6, 7]])
>>> y_train
[2, 0, 3]
>>> X_test
array([[2, 3],
       [8, 9]])
>>> y_test
[1, 4]
```

```
>>> train_test_split(y, shuffle=False)
[[0, 1, 2], [3, 4]]
```

Overfitting vs. underfitting, I



Overfitting vs. underfitting, II



Splitting data into training and test sets

Usually, the split is done using 70% for training and 30% for testing, although this depends on many things e.g.: how much data we have, or how much data the learning algorithm needs (simpler hypotheses need less data than more complex ones).

The split should be done randomly.

For unbalanced datasets, stratified sampling is highly advisable

- Stratified sampling ensures that the proportion of positive to negative examples is kept the same in the train and test sets.

Estimating generalization ability

k-fold cross validation

We split the input data into k *folds*. Typical value for k is 10.

At each iteration, the blue folds are used for training, and red folds are used as validation

Each iteration produces a performance estimate, final estimate is computed as the average of iteration estimates.

Cross-validation vs. random split

Pros of cross-validation

- Estimates are more robust
- Better use of all available data

Cons of cross-validation

- Need to train multiple times

Cross-validation in scikit-learn

```
>>> from sklearn.model_selection import cross_val_score
>>> clf = svm.SVC(kernel='linear', C=1)
>>> scores = cross_val_score(clf, iris.data, iris.target, cv=5)
>>> scores
array([0.96..., 1.  ..., 0.96..., 0.96..., 1.        ])
```

The mean score and the 95% confidence interval of the score estimate are hence given by:

```
>>> print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
Accuracy: 0.98 (+/- 0.03)
```

On model selection

E.g. how to optimize k for nearest-neighbors

Suppose we want to optimize k to build a good nearest-neighbor classifier. We do the following: Compute the cross-validation error for each possible k, and select k that minimizes it.

Question: Is the cross-validation error of the best possible k a good estimate of the generalization ability of the chosen classifier?

Answer: No! Think why ...

On model selection

E.g. how to optimize k for nearest-neighbors

The "right way" of measuring generalization ability would be to get new data and test the chosen k-NN on that new data.

Alternatively:

1. Split data into train and test datasets
2. Use cross-validation to optimize k but using the training data only
3. Use the test data to estimate generalization ability of chosen k-NN