Chapter 6

Model Evaluation and Hyperparameter Tuning

July 13, 2016

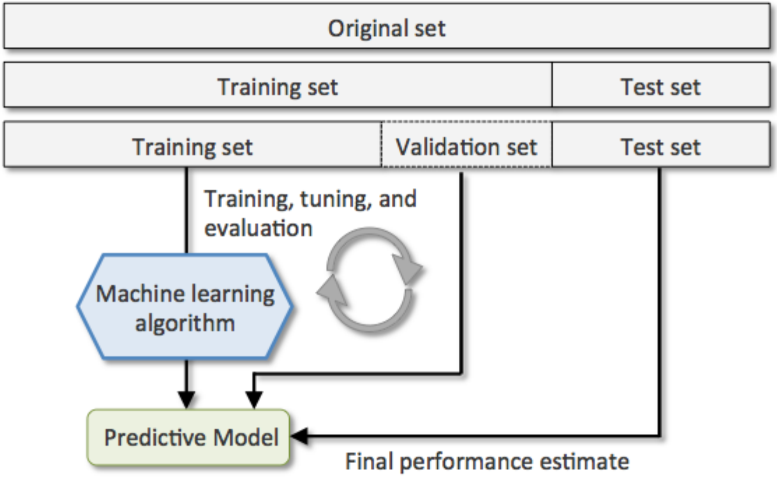
How do we know the model is working?

- Model evaluation
- How do we obtain an unbiased estimate of model's performance?
- Key concept: estimate model performance on **unseen** data
- Hyperparameters vs. model parameters (e.g. weights)
- Model fine-tuning
- Performance metrics

The holdout method

- Split data into training and test datasets
- However, typically we cannot test immediately after training
- Need to tune the model to further improve the performance
- Select optimial values of hyperparameters
- This step is known as *model selection*
- A better approach: training set + validation set + test set
- Validation set is used for model selection

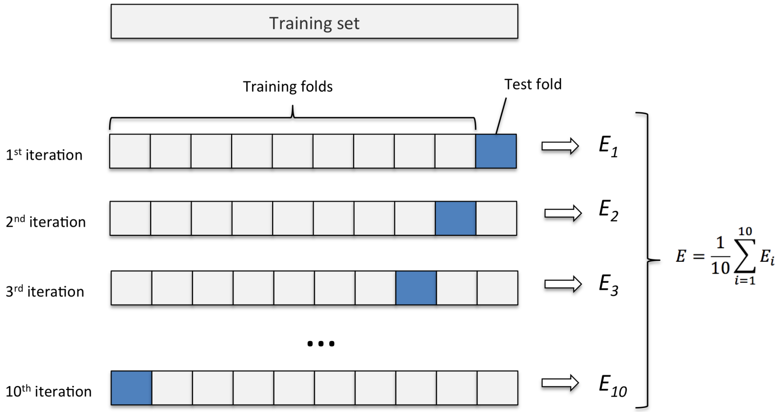
The holdout method


Original set
Training set
Test set
Training set
Validation set
Test set
Training, tuning, and evaluation
Machine learning algorithm
Predictive Model
Final performance estimate

K-fold cross-validation

- Disadvantage of the holdout method: sensitive to partitioning
- Randomly split the training dataset into k folds
- Of these, $k - 1$ folds are used for training and one for testing
- Repeat this procedure k times and average across k folds
- Each sample will be part of train and test sets
- Lower-variance estimate of the model performance (than holdout)

K-fold cross-validation



How do we pick the number of folds?

- The standard value is $k = 10$
- For small datasets, increase the number of folds
- Which will increase the amount of training data
- For larger datasets, we can decrease the number of folds
- E.g. $k = 5$ is a reasonable choice

Variations on the theme of k-fold cross-validation

- **Leave-one-out cross-validation**
  - Set the number of folds equal to the number of training samples
  - Only a single training sample used for testing during each iteration
  - Recommended approach for very small datasets
- **Stratified k-fold cross-validation**
  - Class proportions preserved in each fold
  - I.e. each fold is representative of the training set
  - Better performance estimates for imbalanced data

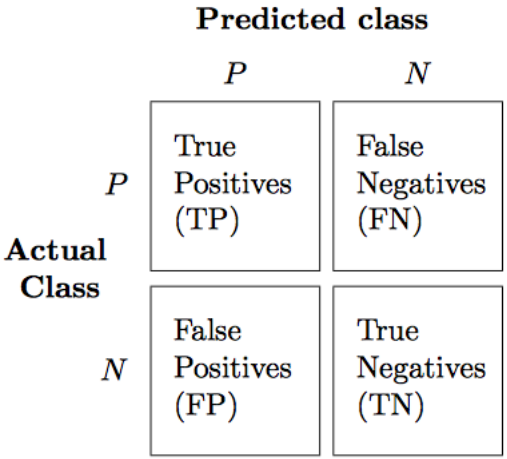
Grid search

- Many ML algorithms offer a number of hyperparameters
- Link to libsvm command-line arguments
- Find the optimal combination of hyperparameter values
- Brute-force exhaustive search of hyperparameter space
- Obviously, this can be computationally very expensive

Performance evaluation metrics

- We've been using accuracy
- Accuracy can be misleading for imbalanced datasets
- Need ways to compute the performance for a specific class
- Confusion matrix helps visualize different types of errors a classifier can make by reporting the counts of these errors
- I.e. true positive (TP), true negative (TN), false positive (FP), false negagive (FN) predictions

Confusion matrix



		Predicted class	
		P	N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

Deducing performance metrics from a confusion matrix

The error can be understood as the sum of all false predictions divided by the number of total predictions, and the accuracy is calculated as the sum of correct predictions divided by the total number of predictions, respectively:

$$Error = \frac{FP + FN}{FP + FN + TP + TN}$$

The prediction accuracy can then be calculated directly from the error:

$$Accuracy = \frac{TP + TN}{FP + FN + TP + TN} = 1 - Error$$

Precision, Recall, F1

Precision:

$$P = \frac{TP}{TP + FP}$$

Recall:

$$R = TPR = \frac{TP}{P} = \frac{TP}{FN + TP}$$

F1 score:

$$\text{F1} = 2 \times \frac{P \times R}{P + R}$$

●