DIGITAL TWIN AI and Machine Learning: The Bias-Variance Tradeoff and Model Selection

Prof. Andrew D. Bagdanov
`andrew.bagdanov AT unifi.it`

Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Firenze

13/19 November 2020

Outline

Introduction

Overview

- Today will bring to a close the first part of this course.
- We will first look at a very important concept: the Bias-Variance Decomposition.
- This gives us a conceptual model of estimator performance in terms of generalization error.
- With this in hand, we will then look at some concrete tools from we can use to manage the trade-off between model bias and variance:
    - Cross-validation: the basis for understanding bias and variance.
    - Learning curves: to understand how model variance depends on training split size.
    - Validation curves: to understand how hyperparameters affect model performance.
    - Model selection: to systematically explore hyperparameter setting through grid search.

The Bias-Variance Decomposition

The Math

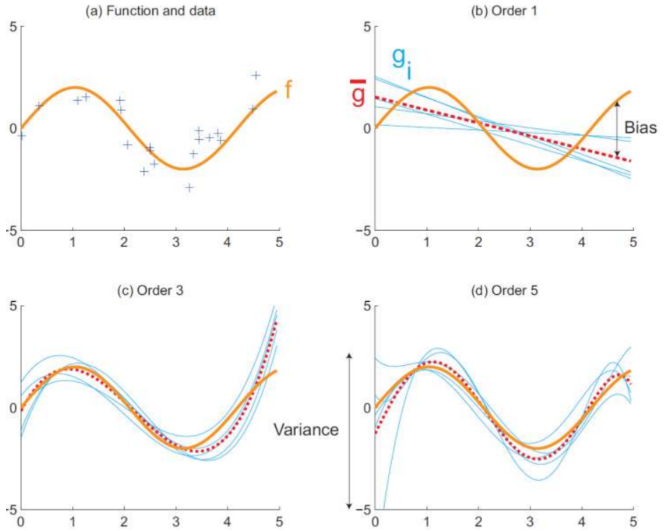
- The biasvariance decomposition is a way of analyzing a model's expected generalization error: the bias, the variance, and the irreducible error resulting from noise in the problem itself.
- Say we estimate a true function $f(x)$ by $y = \hat{f}(x) + \varepsilon$, where ε is the noise with zero-mean and variance σ^2.
- It can be shown that the expected error is equal to:

$$\begin{aligned} E[f(x) - \hat{f}(x) + \varepsilon] &= (\mathbf{Bias}[\hat{f}(x)])^2 + \mathbf{Var}[\hat{f}(x)] + \sigma^2, \text{ where} \\ \mathbf{Bias}[\hat{f}(x)]) &= E[\hat{f}(x)] - E[f(x] \\ \mathbf{Var}[\hat{f}(x)] &= E[\hat{f}(x)^2] - E[\hat{f}(x)]^2 \end{aligned}$$

The Main Point

- As we increase model complexity:
  - Bias decreases: a better fit to data.
  - Variance increases: fit model varies more with data.
  - Imagine the hierachy of polynomial models:
    - $f(x) = c$
    - $f(x) = ax + c$
    - $f(x) = ax^2 + bx + c$
    - ...
  - As we go up in this hierarchy, model complexity increases and bias decreases.
  - But, the model parameters estimated from data will wildly fluctuate with changing data – *even if drawn from the same distribution*.

Visualizing Bias and Variance



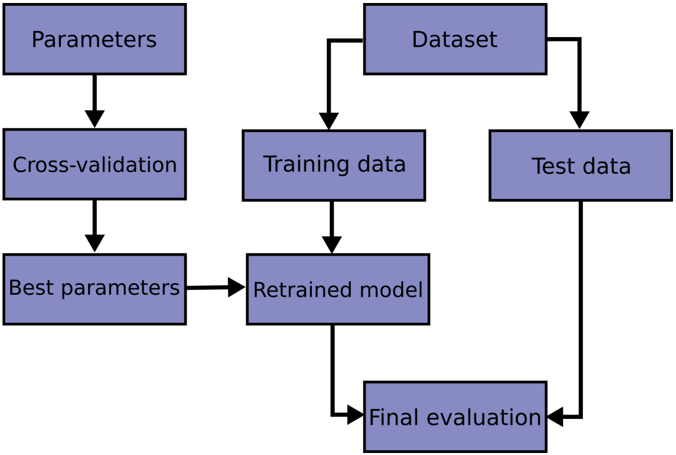
Cross-validation

Cross-validation Overview

- Learning the parameters of any model and testing it on the same data is a methodological mistake.
- A model that just memorizes the labels of the training samples would have a perfect score.
- But, of course it would fail miserably to classify any samples not yet seen.
- This is an extreme example of what is called overfitting.
- To avoid it, it is common practice when performing supervised machine learning to hold out part of the available data as a test set (as we have done since the beginning).

A Useful Flowchart

- Here is a flowchart of the cross-validation workflow for training:



Validation Set

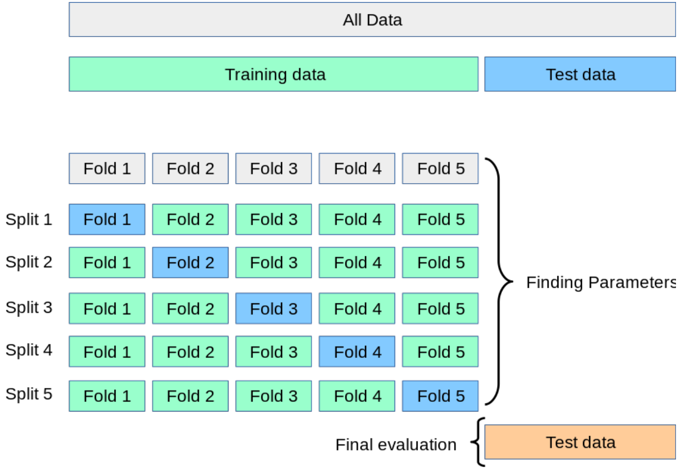
- When evaluating different hyperparameter settings, there is still a risk of overfitting on the test set.
- If we tweak parameters until estimator is optimal, knowledge about the test set can "leak" into the model and evaluation metrics no longer reflect generalization performance.
- To solve this problem, usually another part of the dataset can be held out as a validation set: we train on training set, then evaluate on the validation set, and when the model seems to work well the *final evaluation is done on the test set*.
- However, by partitioning the available data into three sets, we drastically reduce the number of samples used for learning.
- Moreover, the results can depend on a particular random choice for train and validation sets.

Enter, Cross-validation

- A solution to this problem is a procedure called cross-validation.
- A test set should still be held out for final evaluation, but the validation set is no longer needed.
- The basic approach is called k-fold cross-validation: the training set is split into k equally-sized, smaller sets, and the following procedure is followed for each of the k folds:
  1. A model is trained using $k - 1$ of the folds as training data;
  2. The resulting model is validated on the remaining part of the data by computing a performance measure such as accuracy on it.
- Important: the average performance over the k folds gives us a lower bound on the generalization of the model to unseen data.

Cross-validation (continued)

- Here is a diagram explaining the k-fold process:



Cross-validation (continued)

- In sklearn we can easily do k-fold cross-validation using the `sklearn.model_selection.cross_val_score` function:

```
from sklearn.model_selection import cross_val_score
from sklearn.svm import LinearSVC
model = LinearSVC(C=100, verbose=3)
scores = cross_val_score(model, X_tr, y_tr, cv=3,
                         verbose=3, n_jobs=4)
```

- Some parameters to pay attention to:
  - `cv`: number of folds to use.
  - `verbose`: logging level – useful to have feedback for long runs.
  - `n_jobs`: number of parallel jobs to use.
  - `scoring`: function to use for scoring (defaults to `model.score()`.

Cross-validation: Analysis

- Cross-validation is a powerful tool for understanding how models (might) generalize.
- As we will see next, it is the basis for hyperparameter evaluation and selection.
- Problem: cross-validation is expensive as multiple models must be fit to multiple splits of data.

Hyperparameter Selection

Hyperparameter Selection

- Up to now we have used cross-validation only to obtain a more reliable estimate of the performance of our estimator.
- By training multiple times on random train/validation splits we make the most of available data.
- But this still leaves open the question of how to effectively select the hyperparameters of our model.
- Up to now we have used models that have relatively few hyperparameters.
- When we look at deep models based on neural networks, however, there will be significantly more.
- Fortunately, cross-validation also gives us a tool for robustly estimating performance over a grid of hyperparameters.

Hyperparameter Selection: Validation Curves

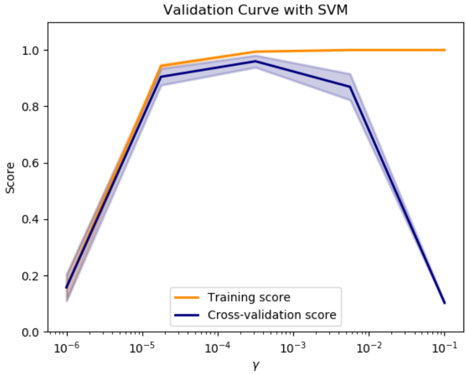
- An excellent way to get an overview of the sensitivity of a model to one hyperparameter is to plot a validation curve.

```
from sklearn.model_selection import validation_curve
(train_scores, val_scores) = validation_curve(
                              LinearSVC(), X_tr, y_tr,
                              "C", [0.1, 1.0, 10, 100, 1000],
                              cv=3)
val_scores

array([[0.85090745, 0.85135743, 0.82478248],
       [0.85180741, 0.84535773, 0.85238524],
       [0.84910754, 0.85300735, 0.83408341],
       [0.83620819, 0.84640768, 0.85358536],
       [0.85525724, 0.86170691, 0.84983498]])
```

Hyperparameter Selection: Validation Curves

- Useful: `validation_curve` returns the cross-validated scores for all folds for all parameters.
- This allows us to make useful plots:



See: *https://scikit-learn.org/stable/auto_examples/model_selection/plot_validation_curve.html*

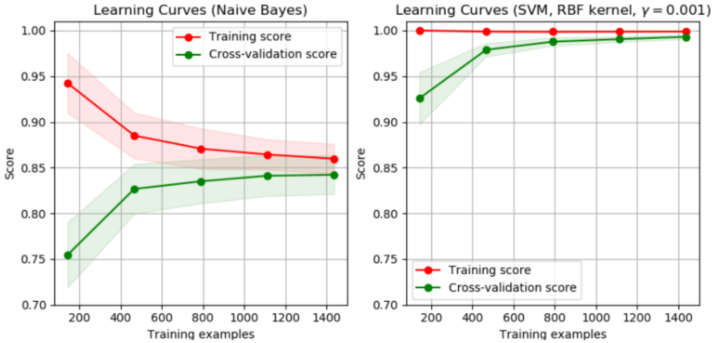
Hyperparameter Selection: Learning Curves

- An important factor in the variance of any model is the size of the training split.
- According to Geoffrey Hinton: "More labeled data is the best possible model regularizer. . . "
- Using `sklearn.model_selection.learning_curve()` we can evaluate model performance as a function of test split size:

```
from sklearn.model_selection import learning_curve
train_sizes, train_scores, test_scores = learning_curve(
                                             LinearSVC(),
                                             X_tr, y_tr, cv=3)
```

Hyperparameter Selection: Learning Curves

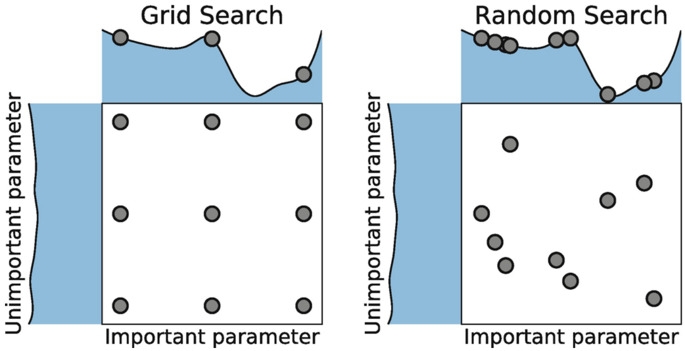
- Again, this returns all scores for all folds for all training set sizes.
- From these we can produce nice plots like:



See: https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html

Hyperparameter Selection: Grid Search

- Grid search is an unsophisticated, brute-force technique that works very well in practice.
- There are two main variations: Uniform and Random Grid Search



Hyperparameter Selection: Grid Search (continued)

- The first thing to do is understand which hyperparameters are of interest.
- This almost always requires a detailed perusal of the documentation.
- Consider a linear SVM with hinge loss: the model essentially has only one hyperparameter: the C used to weight model complexity versus empirical loss:

$$
\begin{aligned}
f(\mathbf{x}) &= \mathbf{w}^T\mathbf{x} + b \\
\mathcal{L}(D; \mathbf{w}, b) &= \min_{\mathrm{w}} ||\mathbf{w}||_2 + \sum_{(\mathrm{x},y)\in D} C \max(0, 1 - y f(\mathbf{x})) \\
\text{class}(\mathbf{x}) &= \begin{cases} -1 & \text{if } f(\mathbf{x}) \leq 0 \\ +1 & \text{if } f(\mathbf{x}) > 0 \end{cases}
\end{aligned}
$$

Hyperparameter Selection: Grid Search (continued)

- The key class in sklearn is `sklearn.model_selection.GridSearchCV`.
- What must provide to `GridSearchCV` is a grid of parameters to search:

```
from sklearn.model_selection import GridSearchCV
model = LinearSVC(max_iter=2000)
param_grid = {'C': [0.001, 0.1, 1.0, 10, 20, 50, 100, 1000]}
search = GridSearchCV(model, param_grid, cv=3, verbose=3, n_jobs=4)
search.fit(X_tr, y_tr)
test_score = accuracy_score(y_te, search.best_estimator_.predict(X_te))
print(f'Best parameters: {search.best_params_}')
print(f'Best cross-val score: {search.best_score_}')
print(f'Score on test set: {test_score}')
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
...
```

Hyperparameter Selection: Grid Search (continued)

- What if we have more hyperparameters?
- For example, in `LinearSVC` we can also choose the type of penalty (L1 or L2).
- Well, we can just add them to the grid:

```
...
param_grid = {'C': [0.001, 0.1, 1.0, 10, 20, 50, 100, 1000],
              'penalty': ['l1', 'l2']}
...
```

```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
```

Reflections

Model Selection

- Model selection is a fundamental fact of life when working with machine learning algorithms.
- Most of the models we have seen so far are low-variance models: they perform fairly stably over a range of hyperparameter settings.
- This is why these models are the tried-and-true techniques for supervised learning: they often just work.
- In the next part of the course we will start looking at neural network models.
- They can achieve significantly better performance. . .
- . . . at the cost, however, of significantly complicating the model selection process.

The Bias-Variance Decomposition

- Nutshell: The more complex the model $\hat{f}(x)$ is, the more data points it will capture, and the lower the bias will be; however, complexity will make the model "move" more to capture the data points, and hence its variance will be larger.
- Caveat: The Bias-Variance Decomposition is useful as a conceptual model – in practice the bias and variance of models is difficult to estimate.

Model Selection Lab

- The laboratory notebook for today:

  http://bit.ly/DTwin-ML5