Notes on and around ISLP

Calvin Khor

Last compiled: October 26, 2023

1 Resources

1. The website for the book:
2. Solutions to ISLR:
3. Videos accompanying ISLR:
4. My errata: https://docs.oracle.com/javase/8/docs/api/java/lang/
5. These notes: https://dotty.epfl.ch/3.0.0/api

2 Statistical Learning

2.1 Definitions and Notation

$X_1, \ldots, X_p$ denotes training data, X_0 is test (out-sample) data. When its capital letters its either a random variable or an abstract model? The p denotes the number of predictors (i.e. independent variables/features/variables). The model is abstractly

$$Y = f(X) + \epsilon \tag{2.1}$$

Y is the response/dependent variable. ϵ is noise inherent to the model: WLOG $\mathbb{E}\,\epsilon = 0$, $\operatorname{Var}\epsilon = 1$, and independent from the predictors (i.e. we cannot predict the error)

We denote the number of observations by n, so that the observations of the predictors are $x_j = (x_{1j}, x_{2j}, \ldots x_{nj})$, for $j = 1, \ldots, p$.

Parametric and non-parametric models:

Prediction Accuracy:

Model Interpretability:

Test MSE:

Variance: "the amount by which $\hat{f}$ would change if we estimated it using a different training data set." - i.e. variance by treating the training data as random variables

Bias: the error that is introduced by approximating a real-life problem, which may be extremely complicated, by a much simpler model.

$$E|y_0 - \hat{f}(x_0)|^2 = \operatorname{Var}\hat{f}(x_0) + (\operatorname{Bias}\hat{f}(x_0))^2 + \operatorname{Var}\epsilon \tag{2.7}$$

—

Important quote from book:

> Here the notation $\mathbb{E}\,|y_0 - \hat{f}(x_0)|^2$ defines the expected test MSE at x_0, expected test MSE and refers to the average test MSE that we would obtain if we repeatedly estimated f using a large number of training sets, and tested each at x_0. The overall expected test MSE can be computed by averaging $\mathbb{E}\,|y_0 - \hat{f}(x_0)|^2$ over all possible values of x_0 in the test set.

Derivation of (2.7):

$$\begin{aligned}
\mathrm{LHS}(2.7) &= \mathbb{E}\,|f(x_0) - \hat{f}(x_0) + \epsilon|^2 \\
&= \mathbb{E}\,|f(x_0) - \mathbb{E}\,\hat{f}(x_0) + (\mathbb{E}\,\hat{f}(x_0) - \hat{f}(x_0)) + \epsilon|^2 \\
&= \mathbb{E}\,|f(x_0) - \mathbb{E}\,\hat{f}(x_0)|^2 + \mathbb{E}\,|\epsilon|^2 + \mathbb{E}\,|\hat{f}(x_0) - \mathbb{E}\,\hat{f}(x_0)|^2 + 2\,\mathbb{E}\,\epsilon(f(x_0) - \hat{f}(x_0)) \\
&= |f(x_0) - \mathbb{E}\,\hat{f}(x_0)|^2 + \mathbb{E}\,|\epsilon|^2 + \mathbb{E}\,|\hat{f}(x_0) - \mathbb{E}\,\hat{f}(x_0)|^2 + 2\,\mathbb{E}\,\epsilon(f(x_0) - \hat{f}(x_0)) \\
&=: \mathrm{Bias}^2 + \mathrm{Var}\,\epsilon + \mathrm{Var}\,\hat{f}(x_0) + 0 \text{ (since } \epsilon \text{ is independent)} = \mathrm{RHS}(2.7).
\end{aligned}$$

2.2 Bias-Variance Trade-off

3 Regression Models

4

5

6

7

8 Tree-based Methods

9

10

11 Python

11.1 Classes

https://stackoverflow.com/questions/100003/what-are-metaclasses-in-python

11.2 Conventions and patterns

- Subscript at the end of a variable name means we are looking at a coefficient or other computed quantity of an estimator.

- Saving a model using Joblib:

```
import joblib
# saving; the filetype extension is only for the user/reader
joblib.dump(pipe, 'filename.joblib')

# loading
loaded = joblib.load('filename.joblib')
```

  Make sure you use the same environment (module version etc.) or the model may change.

- Pipelines are DataFrame-friendly[1], if the component transformers are as well. If they are not, we can wrap the transformer so that it returns a DataFrame. For instance, `StandardScaler` which normally returns a `np.array`:

```
class DFStandardScaler(TransformerMixin):
    def __init__(self):
        self.ss = None
    def fit(self, X, y=None):
        self.ss = StandardScaler().fit(X)
        return self
    def transform(self, X):
        Xss = self.ss.transform (X)
        Xscaled = pd. DataFrame(Xss, index=X.index, columns=X.columns)
        return Xscaled
```

- Pipelines compose, e.g.:

```
pipeline = Pipeline([
    ('features', DFFeatureUnion([
        ('categoricals', Pipeline([
            ('extract', ColumnExtractor(CAT_FEATS)),
            ('dummy', DummyTransformer())
        ])),
        ('numerics', Pipeline([
            ('extract', ColumnExtractor(NUM_FEATS)),
            ('zero_fill', ZeroFillTransformer()),
            ('log', Log1pTransformer())
        ]))
    ])),
```

[1] This info is from https://www.youtube.com/watch?v=BFaadIqWlAg&list=PLzERW_Obpmv_t55kNFRet-E0h1nKeswWF&index=26, with Github repo at https://github.com/jem1031/pandas-pipelines-custom-transformers.

```
    ('scale', DFStandardScaler())
])
```

11.3 Custom Scikit-learn classes

From https://www.youtube.com/watch?v=WGirN6zBJ4s&list=PLzERW_Obpmv_t55kNFRet-E0h1nKeswWF&index=1. There are `Estimator`, `Predictor`, `Transformer`, and `Model` classes. An `Estimator` must implement

- `.fit(X,y)`, fitting the estimator to `X` and `y`
- `.get_params()` return the parameters of the estimator
- `.set_params(**params)` change the parameters of the estimator (e.g. for copying)

Sklearn `Predictor` needs

- `.predict(X)`

Sklearn `Transformer` needs

- `.transform(X, y=None)`

Sklearn `Model` needs

- `.score(X,y)`

11.3.1 Inheritance and Mixins

- You can implement `.get_params()` and `.set_params()` by inheriting from `BaseEstimator`
- There is also `base.TransformerMixin`, `base.RegressorMixin`, `base.ClassifierMixin`, `base.ClusterMixin`, `feature_selection.SelectorMixin`,...

11.3.2 Example 1: simple scaler

```
import numpy as np
from sklearn.base import BaseEstimator, TransformerMixin

class Standardizer(BaseEstimator, TransformerMixin):

def __init__(self,mean_after_transform = 0):
    self.mean after transform = mean after transform

def fit(self, X, y=None):
    self.mean_ = np.mean(X, axis=0) # columwise mean
    self.std_ = np.std(X, axis=0) # columwise std
    return self

def transform(self, X):
    return (X-self.mean_) / self.std_ + self.mean_after_transform
```

11.3.3 Example 2: Regression

Basic regressor that just predicts using the mean or median, using `RegressorMixin` (A regressor is a type of `Model`):

```python
import numpy as np
class MyDummyRegression(BaseEstimator):

def __init__(self, use_median=False):
    self.use_median = use_median

def fit(self, X, y):
    if self.use median:
        self.value_ = np.median(y)
    else:
        self.value_ = np.mean(y)
    return self

def predict(self, X):
    out = np.empty (len(X))
    out.fill(self.value_)
    return out
```

The mixin gives us a `.score` for free.