Empirical Evaluation of Management Practices I

Predictions and Machine Learning

Prof. Dr. Dirk Sliwka   Jesper Armouti-Hansen

17/11/20

Content

1. Introduction to Machine Learning

General definition

[Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed.

- (Arthur Samuel, 1959)

More specific definition

A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

- (Tom Mitchell, 1997)

Types of Machine Learning

Supervised Learning:

The task of learning a function that maps the input(s) X to an output y based on example input-output pairs.

- Regression: The output is continuous or discrete and ordered.
  - Example: Predicting house prices based on house characteristics.
- Classification: The output is a discrete and unordered set.
  - Example: Classifying an email as spam or ham based on the use of certain words.

This is a "mini-course" on supervised learning with Python.

Unsupervised Learning:

We observe inputs but no output. We can seek to understand the relationship between the variables or the observations.

- For example, we might observe multiple characteristics for potential customers.
  - We can then try to cluster potential customers into groups based on these characteristics
- We might also try to project our inputs into a lower dimensional space.
  - This can be a beneficial pre-processing for supervised learning when dealing with high-dimensional data.

We will not deal with unsupervised learning in this course.

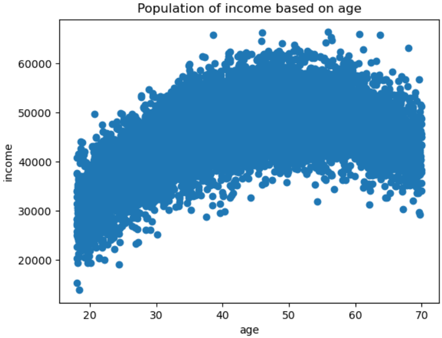
Terminology

The terms used in the ML literature differs slightly from that used in Econometrics:

- Supervised learning → *Regression, classification, predicting* y_i *given* x_i.
- Features → x_i, *independent variables, explanatory variables, regressors, predictors.*
- Target → y_i, *dependent variable.*
- Training → *Estimating a model.*
- Testing → *Evaluating a model.*
- Training data → *The sample we use to train our model.*
- Test data → *The sample we use to test our model.*

Let us start with an example:

- Suppose our task is to predict income based on a person's age and that we had data on the whole population:



- What would be a "good" prediction here?

More formally

- Let X_i be a random vector (i.e. the vector of features).
- Let Y_i be a real variable (i.e. the response).
- We are interested in a function $f(X_i)$ which makes "good" prediction about Y_i.
- To know what a "good" prediction is, we require a loss function: $L(Y_i, f(X_i))$ which penalizes bad predictions
  - Common choice: Squared error – penalizes the quadratic distance:

$$L(Y_i, f(X_i)) = (Y_i - f(X_i))^2 \tag{1}$$

Recall from the lecture on Part 2:

CEF Prediction Property

Let $f(X_i)$ be any function of X_i. The conditional expectation function (CEF) solves:

$$E[Y_i|X_i] = \arg\min_{f(X_i)} E[(Y_i - f(X_i))^2] \tag{2}$$

- Thus, in terms of prediction, we can do no better than the CEF
- Also, recall:

CEF Decomposition Property

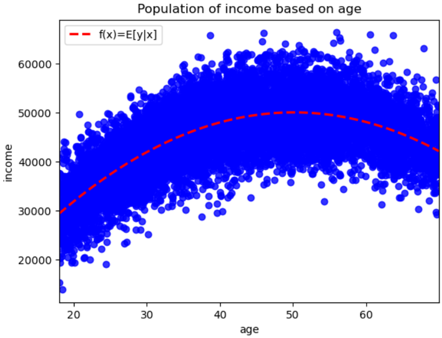
We can decompose Y_i such that

$$Y_i = E[Y_i|X_i] + \epsilon_i \tag{3}$$

Where:

1. ϵ_i is mean independent of X_i: $E[\epsilon_i|X_i] = 0$.
2. ϵ_i is uncorrelated with any function of X_i.

Thus, if we knew the population, calculating the CEF is usually not a difficult task:



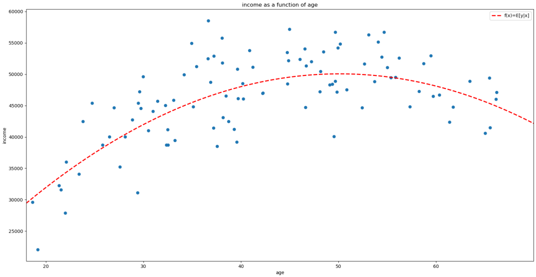
We just predict the average income for people of the same age.

- In this case, we are dealing with simulated data with the following conditional expectation function:

$$f(income) = 2000 * age - 20 * age^2 \tag{4}$$

Usually, we do not know the whole population. Rather, we are working with a sample.

- In this example, we will be working with a sample of 100 data points.
- Our goal is to construct a model $\hat{f}(age)$ which makes “good” predictions about income



Estimating f - K Nearest Neighbor (KNN) Regression

The CEF (f) is almost always unknown, so how can we estimate it?

- For a given observation x_i, we could approximate the CEF by predicting the average of Y_i accross observations with $X_i = x_i$.
- Problem?
  - We might have very few or no other observations with $X_i = x_i$
- Instead, we can settle with predicting the average of Y_i of the K nearest known neighbors of x_i:

$$\hat{f}(x_i) = \frac{1}{K} \sum_{j \in N_K} y_j \tag{5}$$

  - Where N_K is a neighborhood containing the indices of the K closest x's.

Estimating f - Linear Regression

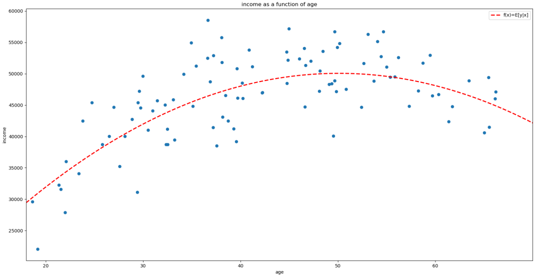
Another way to estimate the CEF is to assume that it is approximately linear in its arguments:

$$\hat{f}(x_i) = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \hat{\beta}_2 x_{i2} + \cdots + \hat{\beta}_k x_{ik} \tag{6}$$

- Thus, estimating the CEF amounts to finding the β's that minimize the squared error in the sample.
- *We now know of two machine learning models: KNN and Linear Regression!*

Example

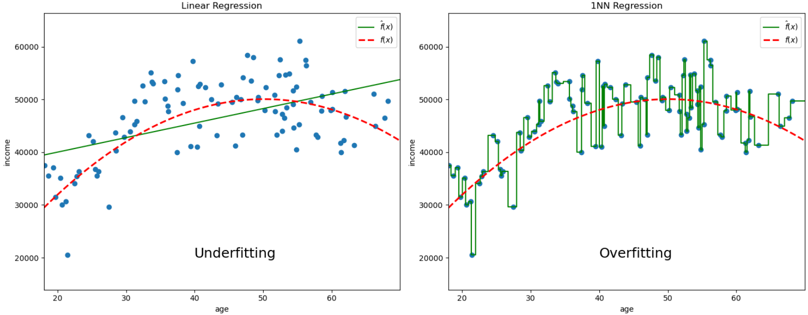
Let us return to our example:



In order to make predictions based on our sample, suppose we train two models: A simple linear regression:

1. $\hat{f}(age_i) = \hat{\beta}_0 + \hat{\beta}_1 age_i$
2. A KNN regression with $K = 1$

Underfitting vs. Overfitting



Clearly, the linear regression is too inflexible and the 1NN regression is too flexible to properly approximate the CEF.

The simple linear regression makes bad predictions because it underfits our training data:

- It has high bias and low variance.

The 1NN regression makes bad predictions because it overfits our training data:

- It has low bias and high variance.

Thus, when dealing with predictions, we face a trade-off:

- The Bias-Variance trade-off

Essentially, we want to find a model that has low bias and low variance, if possible.

The Bias-Variance trade-off

More formally, given a realized point x_0, the expected squared error is given by:

$$E[(y_0 - \hat{f}(x_0))^2] = Var[\hat{f}(x_0)] + [f(x_0) - E[\hat{f}(x_0)]]^2 + \sigma_\epsilon^2 \quad (7)$$

The first term is the variance of our model at x_0, the second the squared bias at x_0, and the third is the irreducible error.

Introduction to Sci-kit Learn (sklearn)

Linear Regression in sklearn

- To fit a linear regression model on the data X and y, we first import the module:
  `from sklearn.linear_model import LinearRegression`
- Then we can perform a regression with the following code:
  `reg = LinearRegression().fit(X,y)`
- We can access one of its attributes to get the coefficients:
  `reg.coef_`
- Suppose the regression is given by $\hat{f}(x_i) = \hat{\beta}_0 + \hat{\beta}_1 * age + \hat{\beta}_2 * age^2$ and that I would like to get the prediction for a 20 year old person:
  `reg.predict([[20, 20**2]])[0]`

KNN Regression in sklearn

- To fit a KNN regression model on the data X and y, we first import the module:

```
from sklearn.neighbors import KNeighborsRegressor
```

- Then we can perform a regression with the following code:

```
knn = KneighborsRegressor().fit(X,y)
```

- Again, suppose we want the prediction of a 20 year old person:

```
knn.predict([[20]])[0]
```

Drawing a random sample with pandas

- Suppose we would like a random sample of 50 observations from a pandas dataframe `df`:

```
df_sample = df.sample(n=50, random_state=181)
```