From Statistics to Machine Learning

MLI LATAM Webinar

23.09.2020

From Statistics to Machine Learning

- Statistics and Econometrics:
  - Under given probabilistic assumptions estimate **parameters** of the model
  - Focus on significance of the parameters and the derivation of the distributions
- Supervised Machine Learning:
  - In supervised machine learning one focuses on the accuracy of the **predictions**
  - Probabilistic assumptions are not necessarily

Supervised machine learning problem

- **Machine learning** addresses a fundamental prediction problem: construct a nonlinear predictor, $\hat{Y}(X)$, of an output, Y , given a high dimensional input $(X_1, \ldots, X_P)$ with P variables
- Machine learning can be simply viewed as the study and construction of an input-output map of the form $Y = F(X)$ where $X = (X_1, \ldots, X_P)$
- The output variable, Y , can be continuous, discrete or mixed
- For example, in a classification problem, $F : X \rightarrow Y$ where $Y \in [1, \ldots, K]$ and K is the number of categories. When Y is a continuous vector and F is a semi-affine function, then we recover the linear model

$$Y = AX + b$$

- To find the parameters β of the model with training set $D = \{Y_i, X_i\}_{i=1}^{T}$, we need to solve the constrained **optimization**:

$$\hat{\beta} = \arg\min_{\beta} \frac{1}{T} \sum_{i=1}^{T} C(Y_i, \hat{Y}^{\beta}(X_i))$$

- Where $C(Y_i, \hat{Y}^{\beta}(X_i))$ is a **cost** or **loss** function. For example, in case of Y is continuous L_2 norm can be applied:

$$C(Y, \hat{Y}^{\beta}(X)) = ||Y - \hat{Y}^{\beta}(X)||_2^2 = (Y - \hat{Y}^{\beta}(X))^2$$

Train / test error and overfitting

- **Training error** - sample error of the trained function $\hat{Y}^{\beta}$ on the same **training** sample D which it was trained on:

$$\hat{err}\left(\hat{Y}^{\beta}, D\right) = \frac{1}{T}\sum_{i=1}^{T} C(Y_i, \hat{Y}^{\beta}(X_i))$$

- **Test error** - expected error over all possible inputs / outputs:

$$\bar{err}\left(\hat{Y}^{\beta}\right) = \mathbb{E}_{X,Y} C(Y, \hat{Y}^{\beta}(X))$$

- **Overfitting** - test error of the trained function $\hat{Y}^{\beta}$ is significantly larger then its training error:

$$\bar{err}\left(\hat{Y}^{\beta}\right) \gg \hat{err}\left(\hat{Y}^{\beta}, D\right)$$

- The following factors can increase the overfitting problem:
  - Higher dimension of inputs
  - Less data (with fixed model complexity)
  - Larger noise in data
  - More complex model (given the amount of input data is fixed)
- We can do the following to detect and avoid the overfitting problem:
  - Test the fitted prediction function on the test sample
  - Use **regularization** techniques in the training
  - Use simpler model whenever it is possible

Underfitting

- If we too simplify the model we can come across **underfitting**
- For example if we try to fit non-linear model by linear approximation
- **Minimal test error** is:

$$err^* = \min_{all \hat{Y}^\beta} \bar{err}\left(\hat{Y}^\beta\right)$$

  - The minimal test error can be thought of as **irreducible** error from noise in the data
  - The error arises from data-generating process and does not depend on the learning algorithm

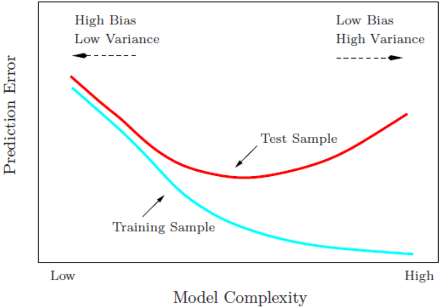
- **Underfitting** - training error of a learned prediction function is significantly larger than the minimal test error:

$$\hat{err}\left(\hat{Y}^\beta, D\right) \gg err^*$$

- **Underfitting** is normally not an issue in the supervised ML as there are many complex models and it is easy to overcomplicate the things
- The problem arises when we try to find the **optimal** complexity of the model to balance in between **Underfitting** and **Overrfitting**

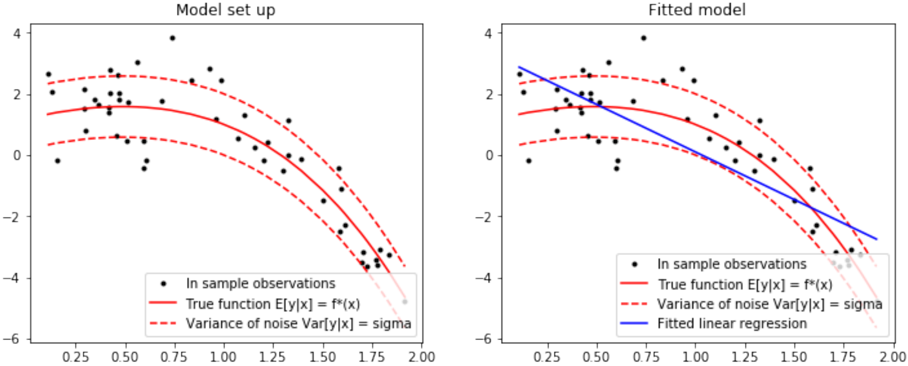
Overfitting vs Underfitting

- The **training error** always goes down as the model $\hat{f}(X)$ becomes more complex as it has more parameters and its able to explain more relationship between inputs and outputs
- The more flexibility you have in fitting the model, the closer you should be able to come to a perfect fit. For example, in extreme case $n = p$ for least squares estimation
- On the **test sample**, the error goes down up to some point as the model complexity increases
- After that point the error starts to increase because of **overfitting** the training sample
- **Overfitting** - when a model is fit to a dataset and that has sufficient ”complexity” to start fitting random features in the data



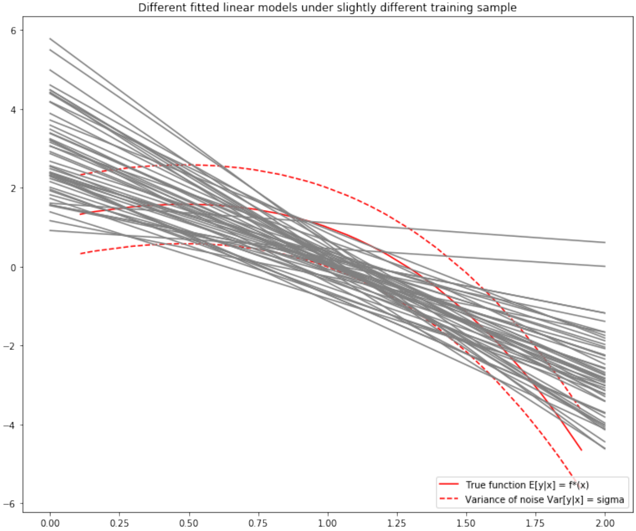
Bias and variance

- Consider the following data, true function and fitted linear model:



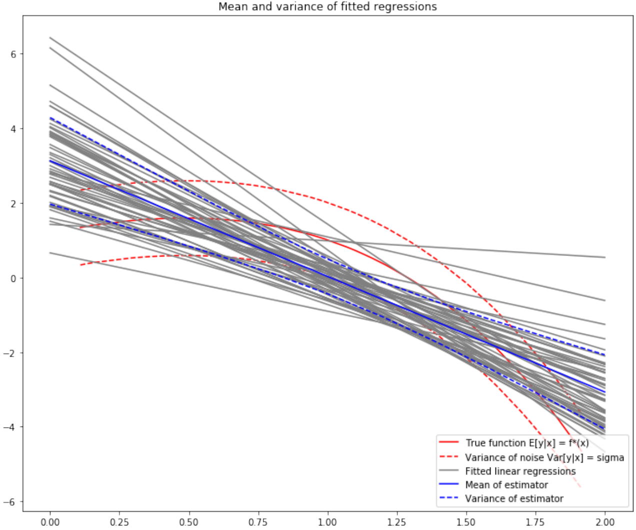
Bias and variance

- Consider many different realized training samples that are generated from same initial distribution
- Let us fit each of the sample by linear model



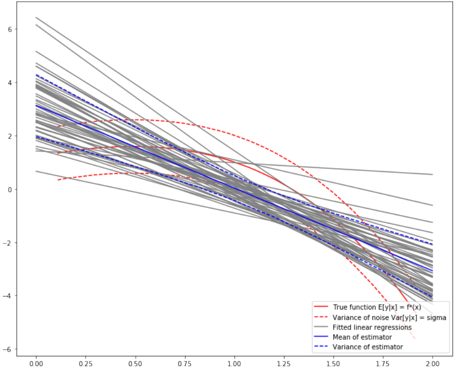
Bias and variance

- Now we are able to calculate mean and variance of the estimators for any fixed input x:
  - Gap between true mean and prediction mean - ”bias”
  - Variance of the predictions - ”variance”

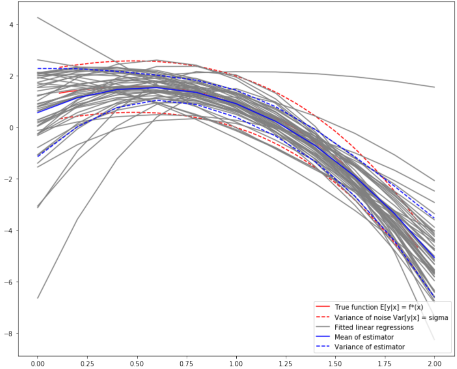


Bias and variance

- Let us now do the same for more complex model - polynomial regression of the power 3:
  - For linear model - **higher bias and lower variance**
  - For polynomial regression - **lower bias but higher variance**



Mean and variance of fitted regressions



Mean and Variance of Fitted polynomial regressions with power = 3

Bias-variance tradeoff

- Consider general case: assume we want to predict Y using X. Assume there is relationship: $Y = f(X) + \epsilon$, where $\epsilon \sim (0, \sigma_\epsilon^2)$
- We may estimate our model $f(X)$ as $\hat{f}(X)$ and want to understand how well $\hat{f}(\cdot)$ fits some future random observation: (x_0, y_0)?
- If $\hat{f}(X)$ is a good model, then $\hat{f}(x_0)$ should be close to y_0 - this is a notion of **prediction error**
- Prediction error is estimated as

$$PE(x_0) = E[(Y - \hat{f}(x_0))^2]$$

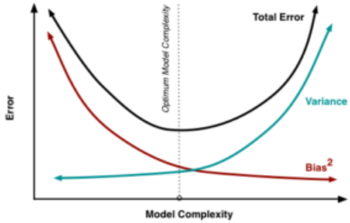
- When we deal with supervised machine learning it is important to understand the errors of predictions, which can be decomposed into the two main components:
  - Error due to ”Bias”: The error due to bias is taken as the difference between the expected (or average) prediction of our model and the correct value which we are trying to predict. Bias measures how far off in general these models' predictions are from the correct value. High bias can cause an algorithm to miss the relevant relations between features and target outputs ⇒ **Underfitting** issues
  - Error due to ”Variance”: The error from sensitivity to small fluctuations in the training set. High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs ⇒ **Overfitting** issues

Bias-Variance trade-off: derivation

- Let us prove the Bias-Variance trade-off:

$$PE(x_0) = E[(Y - \hat{f}(x_0))^2] =$$
$$E[Y^2 + \hat{f}(x_0)^2 - 2Y\hat{f}(x_0)] = E[Y^2] + E[\hat{f}(x_0)^2] - E[2Y\hat{f}(x_0)] =$$
$$\mathrm{Var}\,[Y] + (E[Y])^2 + \mathrm{Var}\left[\hat{f}(x_0)\right] + (E[\hat{f}(x_0)])^2 - 2f(x_0)E[\hat{f}(x_0)] =$$
$$\mathrm{Var}\,[Y] + \mathrm{Var}\left[\hat{f}(x_0)\right] + (f(x_o) - E[\hat{f}(x_0)])^2 = \sigma_\epsilon^2 + Variance + Bias^2$$

- σ_ϵ^2 - the irreducible error, the noise term that cannot fundamentally be reduced by any model
- Given the true model and infinite data to calibrate it, we should be able to reduce both the bias and variance terms to 0. However, in a world with imperfect models and finite data, there is a tradeoff between minimizing the bias and minimizing the variance
    - As model becomes more complex (more terms included), local structure/curvature can be picked up
    - But coefficient estimates suffer from high variance as more terms are included in the model

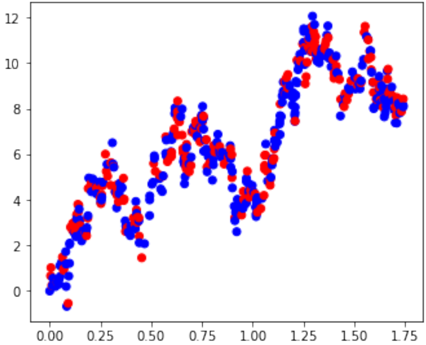


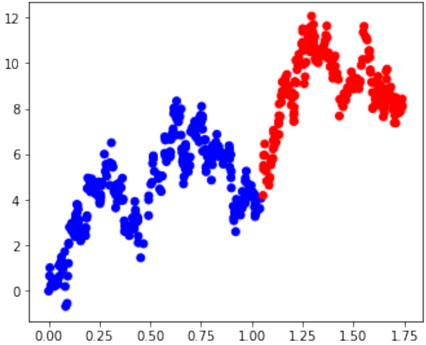
Cross-Validation

- In Statistics, method that splits the data into training and validation sets is defined as **cross-validation**
- Typical split for cross-validation could be 70% / 30% or 80% / 20%
- When working with data, given we have enough of it, we can split the data into three sets:
  - **Training sample** - is used to estimate or fit (relatively small) set of models. For example, we can use set of linear model with different number of features and estimate $\hat{\beta}^{OLS}$
  - **Validation sample** - is used to pick which model that is best based on how it predicting the Y-values for validation data, to determine the right level of complexity (number of regressors in linear model) or the structure (linear vs non-linear)
  - **Test sample** - once the model is selected from previous two samples one can use the training sample and validation sample to re-fit the data and do final check in test sample
- There are several ways to do cross-validation:
  - **Leave-N-out** cross-validation - involves using N observations as the validation set and the remaining observations as the training set. This is repeated on all ways to cut the original sample on a validation set of N observations and a training set (How many?)
  - **K-fold** cross-validation - the original sample is randomly partitioned into K equal sized subsamples. Of the K subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $K-1$ subsamples are used as training data. The cross-validation process is then repeated K times (the folds), with each of the K subsamples used exactly once as the validation data. The K results from the folds can then be averaged to produce a single **measure of fit**
- **Measure of fit** for cross-validation could be:
  - **RMSE** - root of mean squared error: $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2}$
  - **MAD** - median absolute deviation: $median|\hat{Y}_i - Y_i|$

Cross-validation for timeseries

- To split time series for cross validation it is better not to randomise otherwise test error is optimistic due to ”looking” in the future

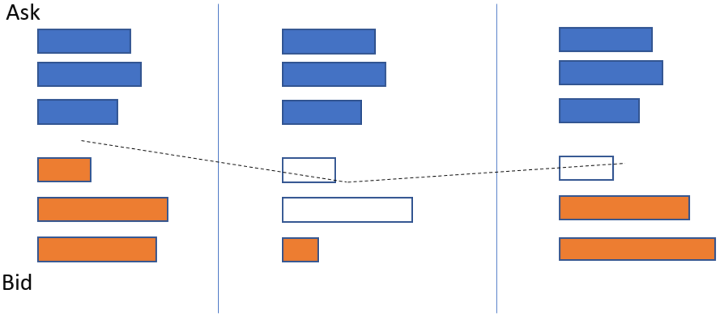

Incorrect way of splitting timeseries for cross validation


Correct way of splitting timeseries for cross validation

”Sliding window” for timeseries

- If we have total amount of data points in timeseries T than the algorithm becomes as follows
- For each t in t_0, ..., $T - h$
    - Train the model on the training sample: $t - window$, ..., t
    - Use the validation sample: $t + 1$, ..., $t + h$
    - Move forward to time $t + h$ and repeat the methodology
- Forecasting horizon *h* and *window* should be same as in your final model

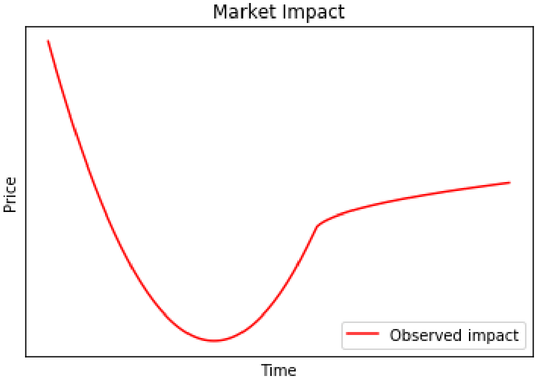
Microstructure



- *Market impact* of the transaction is the difference between price in absence of the transaction and the price in the transaction
- Order can swipe the liquidity on the corresponding side causing spreads to widen
- At next moment market participant can fill back some liquidity on the side but not at the original level

Market Impact

- *Market impact* can be decomposed into two effects: *temporary market impact* and *permanent market impact*
- **Temporary market impact** - price that is paid during short time, where the impact caused by limited amount of liquidity immediately available on the market
- **Permanent market impact** - price that is paid during longer period of time after market recovery



Market impact

- *Market impact model* - estimate the market impact of the particular market order over some period of time given security, order size, period of time
- Why we need the model:
  - Estimate the cost of the execution for some particular trade
  - Can be used in the development of market simulators
  - May feed into other algorithmic trading models

Factors and Inputs of the model

- *Order size* - the larger an order is the more impact it has
- *Trading volume of the stock* - less liquid stocks have more impact
- *Price volatility*
- *Period of the execution*
- Based on the factor above we define the following inputs of the model
  - $\frac{s}{V}$ - size of the order *s* divided by the daily traded volume *Q* - a measure of liquidity of the asset. Let us call it **"Liquidity"**
  - $\frac{s}{dv}$ - size of the order *s* divided by the traded volume *dv* for a given period of time - a measure of relative size of the order. We can call it **"Aggressiveness"** of the order
  - σ - price volatility in a given period of time

Output and proxies of the inputs

- Assume we want to estimate the market impact over period Δt
- We define market impact as $\frac{mid_0 - \bar{mid}}{mid_0}$
  - Where mid_0 - midprice at the beginning of the period Δt and $\bar{mid}$ - average of midprice over that period
- However what if we do not have enough history of the orders to calculate the inputs and outputs
- Artificial orders are created: if over some period of time Δt market buy-trades $>$ market sell-trades we consider that there is one market buy-order
- Assumed total trading volume for Δt is q:

$$q = |q^b| + |q^s|$$

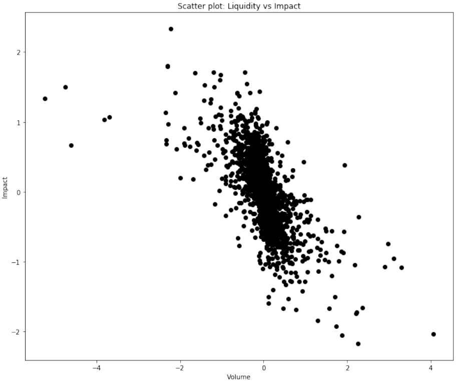
  - q^b - volume of buy initiated trades
  - q^s - volume of sell initiated trades
- Then we have an order of absolute size

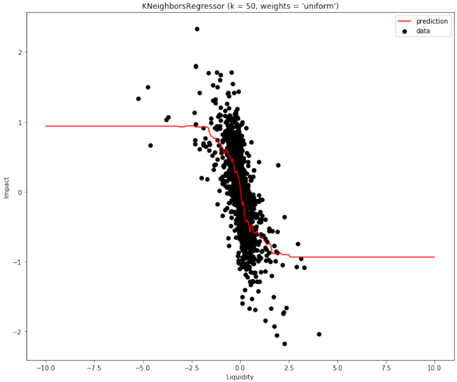
$$q = |q^b + q^s|$$

  - If $|q^b| > |q^s|$ - buy-order
  - $|q^s| > |q^b|$ - sell-order

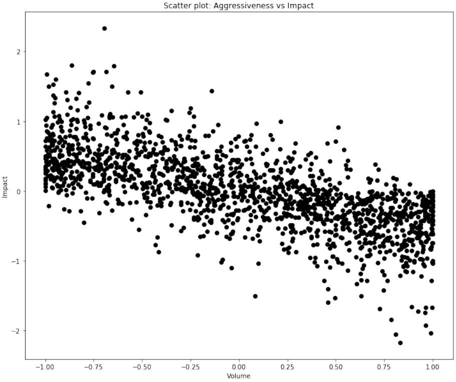
KNN method

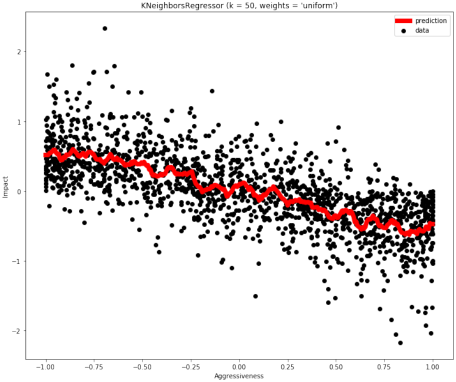
- We will use KNN (k - nearest neighbors) method to estimate market impact





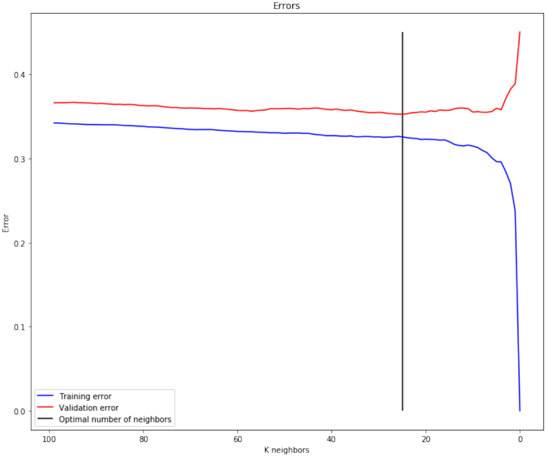
KNN method


Scatter plot: Aggressiveness vs Impact
Impact
2
1
0
−1
−2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
Volume

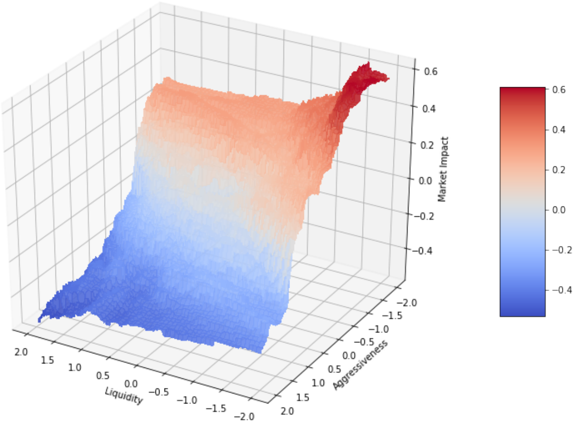

KNeighborsRegressor (k = 50, weights = 'uniform')
prediction
data
Impact
2
1
0
−1
−2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
Aggressiveness

Modelling steps

- Split for training, validation and test samples: 55/30/15
- Scale using mean and std from training sample
- For each hyperparameter k (number of the nearest neighbors) train on training sample and find the error validation sample
- The optimal k minimises validation error



Final model



- Optimal number of neighbors is 25
- R^2 on test sample is 63%, which is better as compared to benchmark linear model with $R^2 = 52\%$