W19 Predictive Analysis

What are the difference between fit line? Purposes?

Accurately predict sales price → optimization, analysis

What account for the Change in MSE between test and validation set?

Sampling

`DataFrame.sample( n=None , frac=None , replace=False)`

- n= #of items from axis to return
- replace allow sampling from same row more than once
- ignore_index = True : resulting index 0,1,2...n-1

Question Framing

- **relationship** of the variable (being predicted) with other variables

```
#DATA CLEANING
sales=pd.read_csv("NYC_Sales_Samples.csv").replace("-",np.nan).dropna()

sales.iloc[:,2::]=sales.iloc[:,2::].astype("float")
sales[sales.columns[2:]]=sales[sales.columns[2:]].astype("float")

sales=sales.assign(log_GSF=np.log(sales.GROSS_SQUARE_FEET), log_SP=np.log(sales.SALE_PRICE))
#RANDOM SAMPLING
sales=sales.sample(n=sales.shape[0],replace=False,ignore_index=True)
  # n= #of items from axis to return
  # replace allow sampling from same row more than once
  # ignore_index = True : resulting index 0,1,2...n-1
#manipulate existing columns and assign them to table
plt.scatter(sales.log_GSF, sales.log_SP)
plt.xlabel("log GROSS_SQUARE_FEET")
plt.ylabel("log SALE_PRICE")

# fit line

def fit_line(intercept,slope):
    x=np.array([np.min(sales.log_GSF),np.max(sales.log_GSF)])
    y=intercept+slope*x
    plt.plot(x,y, color="red")
fit_line(4,1)
fit_line(5,1)

x=np.linspace(5,13,100)
beta0_g1,beta1_g1=1,2
y=beta0_g1+beta1_g1*x
plt.plot(x,y,label="line1")
beta0_g2,beta1_g2=6,1
y=beta0_g2+beta1_g2*x
plt.plot(x,y,label="line2")
```

1. **Nearest Neighbor** :
2. **Linear regression** : fit a straight line for the dots to be as close as possible
    a. *Good when outcome variable is continuous*

value to be estimated

unobserved factor affect y

$$y = \beta_0 + \beta_1 \underbrace{x_1 + \beta_2 x_2}_{\text{var. that explain variation. (PREDICTORS)}} + \epsilon$$

dependent variable

assume $\epsilon = 0$

Variable = predicator + predictor

$$log_S P = \beta 0 + \beta 1 log_G SP + \epsilon$$

- Find out the best estimate for β0 and β1 → estimate y → predicting

Ordinary Linear Square(OLS)

Find out parameter values(β0, β1, β2) to **minimize the mean squared errors (MSE)**

$$\underset{\beta_0, \beta_1}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \epsilon_i^2 = \frac{1}{N} \sum_{i=1}^{N} (y_i - (\beta_0 + \beta_1 x_{1i}))^2$$

estimation of parameters

$$\hat{y} = \widehat{\beta_0} + \widehat{\beta_1} x_1 + \widehat{\beta_2} x_2$$

estimated mean of y

▼ **This can be done using :**

`sklearn.linear_model.LinearRegression`

`model=LinearRegression(fit_intercept=True)`

- fit interception = True : calculate β0

1. **model fitting**

   `model.fit (X,y)` :

      - X is 2D structure. Muti col: `Xsup[:,[0,3]]`
      - y is 1D, `Xsup[:,i]` i is **dependent**

   - estimation of the non-intercept βi: `model.coef_`
     - βi estimate = change in y when I change 1 unit, other constant
   - estimate of intercept β0: `model.intercept_`
     - expected value of y when all other variable = 0

   ```
   model.fit(Sales[["log_GSG"]], Sale["log_SP"])
   ```

2. **Prediction**

   - `model.predict_(X)`
     - X = variable which the change in outcome depend on
     - `y = model.intercept_ + model.coef_[0] * X1`

```
prediction=model.predict(sales[["log_GSF"]])
sales=sales.assign(predicted_value=prediction)

plt.plot(sales.log_GSF,sales.predicted_value,label="line1")
```

3. Error estimation (MSE)

`mean_squared_error(y, y_predicted)`

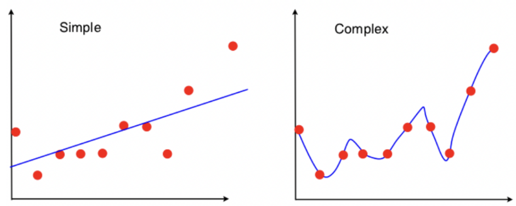
```
prediction=model.predict(sales[["log_GSF"]])
mean_squared_error(sales.log_SP, prediction)
```

Model Selection and Variance & Bias trade off

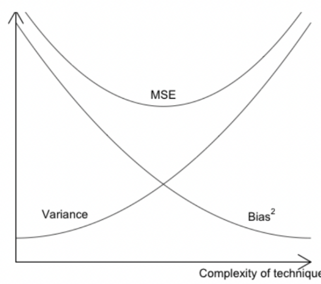
- Choosing between different models : CROSS VALIDATION (balance between simple and complex)
  - **Underfitting** : too much “Bias”, missing relationship between feature and target
    - simple: high bias, low variance

- **Overfitting**: too much “variance”, model predict noise in training set
    - More complex model: low bias, high variance
    - model sensitive to change in X - lots of noise - hard to predict data never seen before → MSE high

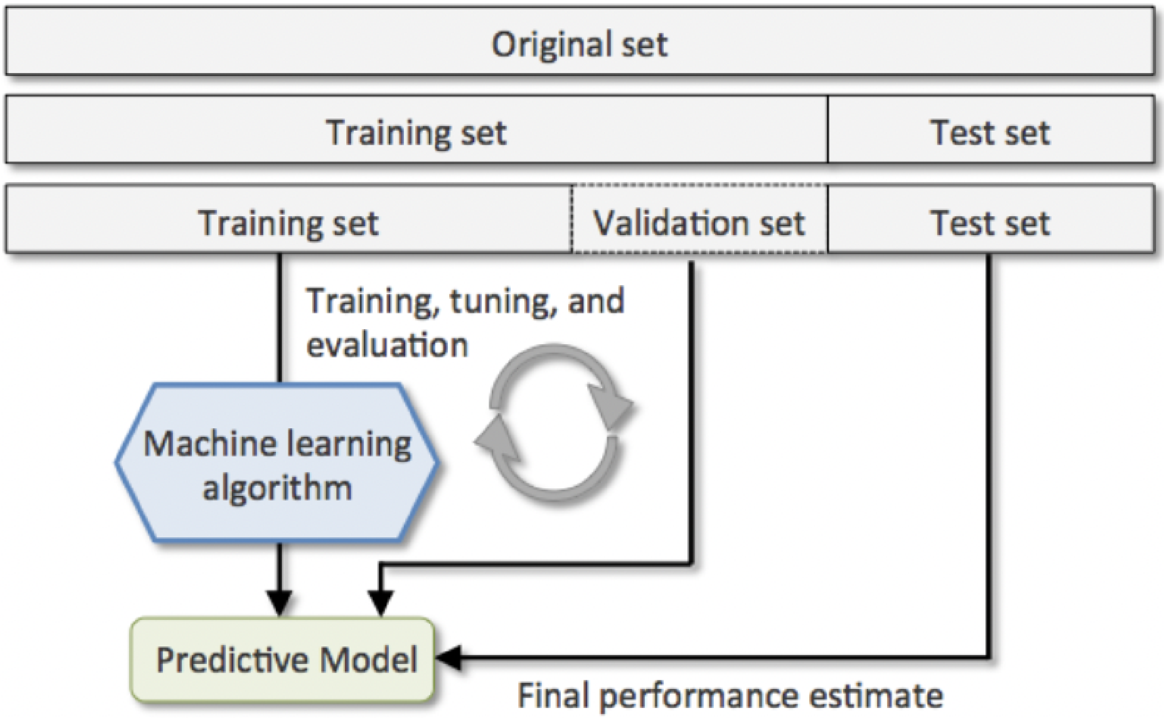
      Overfitting: use training data to test



- $y = \beta_0 + \beta_1 x_1 + \epsilon$
- $y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \epsilon$
- $y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_1^3 + \beta_4 x_1^4 + \beta_5 x_1^5 + \beta_6 x_1^6 + \epsilon$



▼ Cross Validation



Training set : loop through candidate mode, train with training set (recover ***beta*** β value)

Validation set : compute accuracy of each model → ***choose*** the best model

Test set : report ***accuracy*** of best model

- 1) know best model 2) know accuracy of each model

▼ OPTIONAL : **Combine Train Validation to retrain Best Model:**

```
Train_Valid = pd.concat([Train,Valid])
model.fit(Train_Valid[["log_GSF"]],Train_Valid["log_SP"])
predicted=model.predict(Test[["log_GSF"]])
mean_squared_error(predicted, Test["log_SP"])
```

- Use Test to measure accuracy

▼ **Error Measurement (MSE)**

$$MSE = \frac{\sum(y_i - \widehat{y_i})^2}{N}$$

`from sklearn.metrics import mean_squared_error`

`mean_squared_error(y_true, y_predict)`

Example:

- candidate model
  - Model 1: log(SALES_PRICE) on log(GROSS_SQUARE_FEET)
  - Model 2: log(SALES_PRICE) on log(GROSS_SQUARE_FEET) and total units

```
Train=sales[0:1000:]
Valid=sales[1000:1200:]
Test=sales[1200::]
sales.columns

#training
model1 = LinearRegression()
model1.fit(Train[["log_GSF", "TOTAL_UNITS"]], Train["log_SP"])

model2 = LinearRegression()
model2.fit(Train[["log_GSF"]], Train["log_SP"])

## model selection
model1_predict = model1.predict(Valid[["log_GSF", "TOTAL_UNITS"]])
model2_predict = model2.predict(Valid[["log_GSF"]])
mean_squared_error(Valid.log_SP,model1_predict), mean_squared_error(Valid.log_SP,model2_predict)
```