02-Models-Performance

March 1, 2021

1 Model's Performance

We need to evaluate of our model's performance after training procedure. In machine learning, a common task is the study and construction of algorithms that can learn from and make predictions on data. The data used to build final model usually comes from multiple datasets. In particular, three datasets are commonly used in different stages of the creation of the model.

We can perform this task by dividing our dataset into 3 parts.

1.0.1 Training Set

The model is initially fit on a training set. That is a set of examples used to fit the parameters of the model. We can denote the training dataset as

$$(X, y) = (x^1, y^1), (x^2, y^2), \cdots, (x^m, y^m)$$

1.0.2 Validation Set

The validation dataset provides an unbiased evaluation of a model fit on the training dataset while tuning the model's hyperparameters. For example, a set of examples used to tune the parameters of a classifier, in the MLP case, we would use the validation set to find the "optimal" number of hidden units or determine a stopping point for the back-propagation algorithm

1.0.3 Test Set

The test set dataset is a dataset used to provide an unbiased evaluation of a final model fit on the training dataset. We can compute errors from this dataset.

Let's show training/validation/test split.

[5]:
```
import pandas as pd
df = pd.read_csv('./data/Credit.csv')
df.drop(columns=['Unnamed: 0'],inplace=True)
df.head()
```

[5]:
```
     Income  Limit  Rating  Cards  Age  Education  Gender Student Married  \
0   14.891   3606     283      2   34         11    Male      No     Yes
1  106.025   6645     483      3   82         15  Female     Yes     Yes
2  104.593   7075     514      4   71         11    Male      No      No
3  148.924   9504     681      3   36         11  Female      No      No
4   55.882   4897     357      2   68         16    Male      No     Yes

   Ethnicity  Balance
0  Caucasian      333
1      Asian      903
2      Asian      580
3      Asian      964
4  Caucasian      331
```

[6]:
```python
print('Number of the samples: {0}'.format(len(df)))
```

```
Number of the samples: 400
```

[7]:
```python
def trainTestSplit(data, ratio = 0.8):
    if isinstance(data, pd.DataFrame):
        data = data.sample(frac=1).reset_index(drop=True)
        train_pct_index =  int(ratio * len(data))
        train = data.iloc[:train_pct_index,:]
        test = data.iloc[train_pct_index:,:]
        test.reset_index(inplace=True, drop = True)
        return train, test
    elif isinstance(data,np.array):
        X_train, X_test = data[:train_pct_index,0], data[train_pct_index:,0]
        Y_train, Y_test = data[:train_pct_index,1:], data[train_pct_index:,1:]
        return X_train, X_test, Y_train, Y_test
```

[8]:
```python
train, test = trainTestSplit(df,0.6)
```

[9]:
```python
train.head()
```

[9]:
```
   Income  Limit  Rating  Cards  Age  Education  Gender Student Married  \
0  53.598   3714     286      3   73         17  Female      No     Yes
1  26.427   5533     433      5   50         15  Female     Yes     Yes
2  30.111   4336     339      1   81         18    Male      No     Yes
3  39.609   2539     188      1   40         14    Male      No     Yes
4  24.543   3206     243      2   62         12  Female      No     Yes

          Ethnicity  Balance
0  African American        0
1             Asian     1404
2         Caucasian      347
3             Asian        0
```

```
4         Caucasian       95
```

[10]:
```python
test.head()
```

[10]:
```
   Income  Limit  Rating  Cards  Age  Education  Gender Student Married  \
0  59.879   6906     527      6   78         15  Female      No      No
1  39.055   5565     410      4   48         18  Female      No     Yes
2  36.362   5183     376      3   49         15    Male      No     Yes
3  49.927   6396     485      3   75         17  Female      No     Yes
4  62.413   6457     455      2   71         11  Female      No     Yes

          Ethnicity  Balance
0         Caucasian     1032
1         Caucasian      772
2  African American      654
3         Caucasian      890
4         Caucasian      762
```

[11]:
```python
print('Number of samples in training set: {0} \nNumber of samples in test set:␣
↪{1}'.format(len(train),len(test)))
```

```
Number of samples in training set: 240
Number of samples in test set: 160
```

Now let's split the test set as validation/test.

[12]:
```python
test, val = trainTestSplit(test,0.5)
```

[13]:
```python
test.head()
```

[13]:
```
   Income  Limit  Rating  Cards  Age  Education  Gender Student Married  \
0  26.400   5640     398      3   58         15  Female      No      No
1  12.000   4160     320      4   28         14  Female      No     Yes
2  16.482   3326     268      4   41         15    Male      No      No
3  16.711   5274     387      3   42         16  Female      No     Yes
4  29.638   5833     433      3   29         15  Female      No     Yes

   Ethnicity  Balance
0      Asian      905
1  Caucasian      602
2  Caucasian      271
3      Asian      863
4      Asian      942
```

[14]:
```python
val.head()
```

[14]:
```
    Income  Limit  Rating  Cards  Age  Education  Gender Student Married  \
0   10.503   2923     232      3   25         18  Female      No     Yes
```

```
1  115.123   7760      538       3   83          14  Female       No        No
2  128.669   9824      685       3   67          16    Male       No       Yes
3   39.055   5565      410       4   48          18  Female       No       Yes
4   61.620   5140      374       1   71           9    Male       No       Yes

          Ethnicity  Balance
0  African American      191
1  African American      661
2             Asian     1243
3         Caucasian      772
4         Caucasian      302
```

[15]:
```
print('Number of samples in test set: {0} \nNumber of samples in validation set:
 ↪ {1}'.format(len(test),len(val)))
```

```
Number of samples in test set: 80
Number of samples in validation set: 80
```

1.1 Evaluating Regression Models

1.1.1 Evaluating Linear Regression

After we built a Linear Regression model, we need to see evaluation metrics based on this model due to validity of our model.

The L^2 loss, mean squared error (MSE), is one of the tools for evaluation. We defined last week as:

$$L(\hat{y}, y) = \frac{1}{m}\sum_i [y^i - (\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n]^2$$

$$= \frac{1}{m}\sum_i (y^i - \hat{y}^i)^2$$

MSE or Mean Squared Error is one of the most preferred metrics for regression tasks. It is simply the average of the squared difference between the target value and the value predicted by the regression model. As it squares the differences, it penalizes even a small error which leads to over-estimation of how bad the model is.

If MSE is relatively big, that mean our model is not suitable for the data or vice-versa.

Also the Root Mean Squared Error (RMSE) can be used for evaluation. RMSE is the most widely used metric for regression tasks and is the square root of the averaged squared difference between the target value and the value predicted by the model. It is preferred more in some cases because the errors are first squared before averaging which poses a high penalty on large errors. This implies that RMSE is useful when large errors are undesired.

$$L(\hat{y}, y) = \sqrt{\frac{\sum_i (y^i - \hat{y}^i)^2}{m}}$$

Another example for evaluation is Mean Absolute Error (MAE). MAE is the absolute difference between the target value and the value predicted by the model. The MAE is more robust to outliers and does not penalize the errors as extremely as mse. MAE is a linear score which means all the individual differences are weighted equally. It is not suitable for applications where you want to pay more attention to the outliers.

$$L(\hat{y}, y) = \frac{1}{m} \sum_i |y^i - \hat{y}^i|$$

Another powerfull example for evaluation is R^2 error (also known as the Coefficient of Determination). The MSE provides an absolute measure of the lack of fit of the model to the data. But since it is measured in the units of y, it is not always clear what constitutes a good MSE. The R^2 statistics provides an alternative measure of fit. It takes the form of proportion and so it always takes on a value between 0 and 1, and it is independent of the scale of Y.

To calculate R^2, we use the formula

$$\mathrm{R}^2 = \frac{\sum_i (y^i - \bar{y})^2 - \sum_i (y^i - \hat{y}^i)^2}{\sum_i (y^i - \bar{y})^2}$$

$$= 1 - \frac{\sum_i (y^i - \hat{y}^i)^2}{\sum_i (y^i - \bar{y})^2}$$

where,

$$\bar{y} = \frac{1}{m} \sum_i y^i$$

An R^2 statistic, if feature X can predict the target, then the proportion is high and the R^2 value will be close to 1. If opposite is true, the R^2 value is then closer to 0.

[1]:

```python
import numpy as np # array (dizi)
import os # file system
import matplotlib.pyplot as plt # data visualization
import pandas as pd # dataframe

# scikitlearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error


pd.options.mode.chained_assignment = None

df = pd.read_csv('./data/slr.csv')
df.head()
```

```
# PCA -> principal component analysis
```

[1]:

	SAT	GPA
0	1714	2.40
1	1664	2.52
2	1760	2.54
3	1685	2.74
4	1693	2.83

[2]:
```
df.describe()
```

[2]:

	SAT	GPA
count	84.000000	84.000000
mean	1845.273810	3.330238
std	104.530661	0.271617
min	1634.000000	2.400000
25%	1772.000000	3.190000
50%	1846.000000	3.380000
75%	1934.000000	3.502500
max	2050.000000	3.810000

[3]:
```
def trainTestSplit(data, ratio = 0.8):
    if isinstance(data, pd.DataFrame):
        data = data.sample(frac=1).reset_index(drop=True)
        train_pct_index =  int(ratio * len(data))
        train = data.iloc[:train_pct_index,:]
        test = data.iloc[train_pct_index:,:]
        test.reset_index(inplace=True, drop = True)
        return train, test
    elif isinstance(data,np.array):
        X_train, X_test = data[:train_pct_index,0], data[train_pct_index:,0]
        Y_train, Y_test = data[:train_pct_index,1:], data[train_pct_index:,1:]
        return X_train, X_test, Y_train, Y_test
```

[4]:
```
df_train, df_test = trainTestSplit(df, ratio = 0.9)
```

[5]:
```
print('Number of samples in data: {0} \nNumber of samples in training set: {1}␣
 ↪\nNumber of samples in test set: {2}'
      .format(len(df),len(df_train),len(df_test)))
```

```
Number of samples in data: 84
Number of samples in training set: 75
Number of samples in test set: 9
```

[6]:
```
X_train = df_train.iloc[:,0]
y_train = df_train.iloc[:,1]
```

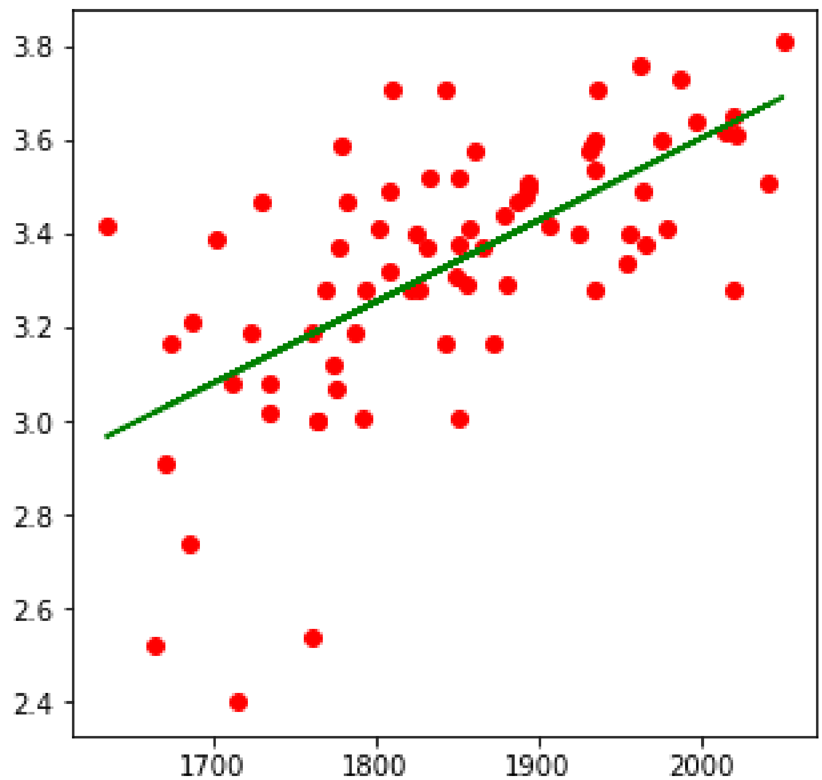
[7]:
```python
X_train_arr = np.array(X_train) #
X_train_arr.shape = (X_train_arr.shape[0],1)
y_train_arr = np.array(y_train)
y_train_arr.shape = (y_train_arr.shape[0],1)



model = LinearRegression();
model.fit(X_train_arr, y_train_arr)

y_pred = model.predict(X_train_arr);
```

[10]:
```python
from sklearn.metrics import r2_score
mse = mean_squared_error(y_train_arr,y_pred)
r2 = r2_score(y_train_arr,y_pred)
print(mse,r2)
```

```
0.045316419096650816 0.4120723365154918
```

[11]:
```python
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(5,5))
plt.scatter(X_train, y_train, color = "red")
plt.plot(X_train, y_pred, color = "green")
```

[11]:
```
[<matplotlib.lines.Line2D at 0x7f5884c0dac8>]
```



1.1.2 Evaluating Logistic Regression

Misclassification Error Before evaluating Logistic Regression, let's do a probabilistic evaluation of errors on decision making.

Suppose that we have two classes $C = \{C_k : k \in \{1,2\}\}$ and we have random samples from a Gaussian Distribution, x. This samples are generated from two Gaussian Distribution. Let's denote this two Gussian Distribution as $C_1 \sim \mathcal{N}(\mu_1, \boldsymbol{\sigma}_1)$ and $C_2 \sim \mathcal{N}(\mu_1, \boldsymbol{\sigma}_1)$. Some samples are generated from first distribution that are correspond to class C_1 and others are generated from second distribution that are correspond to class C_2. As you can see; that implies, this is a binary classification task. In real life, our generated samples x are not real data. But it is clear to see the concepts.

Now let's define our distributions. First, let's choose distribution of C_1 bimodal. In other words, it is concatenation of two Gaussian Distribution. Choose parameters for C_1, $\mu_{11} = -2$, $\mu_{12} = 25$, $\sigma_{11} = 5$, $\sigma_{12} = 7$. So, the distribution of C_1 is,

$$p(x, C_{1_1}) = \frac{1}{5\sqrt{2\pi}} \exp\left(\frac{-(x-(-2))^2}{2 \cdot 25}\right)$$

$$p(x, C_{1_1}) = \frac{1}{7\sqrt{2\pi}} \exp\left(\frac{-(x-25)^2}{2 \cdot 49}\right)$$

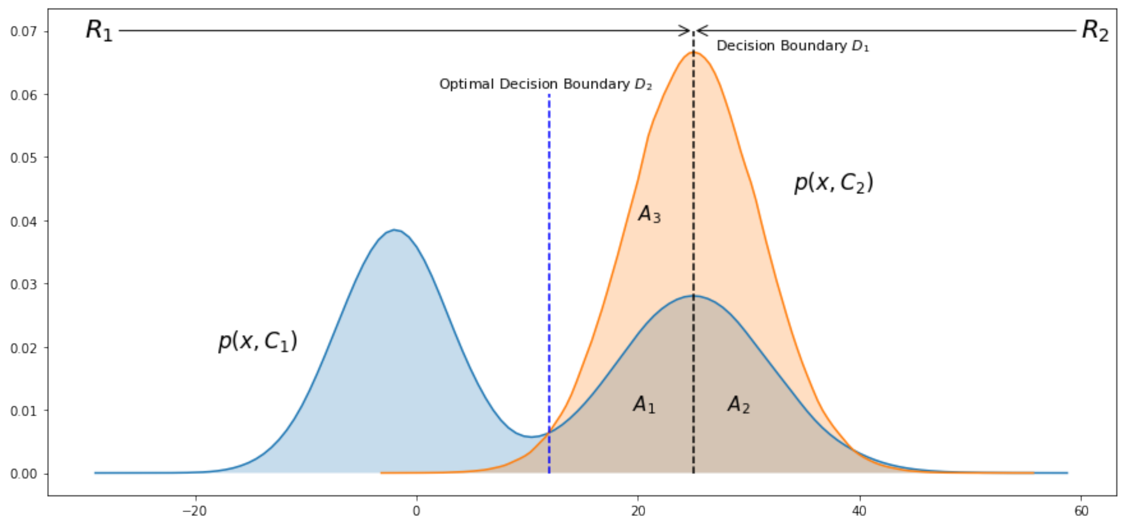
$$p(x, C_1) = \left[\frac{1}{5\sqrt{2\pi}} \exp\left(\frac{-(x-(-2))^2}{2 \cdot 25}\right); \frac{1}{7\sqrt{2\pi}} \exp\left(\frac{-(x-25)^2}{2 \cdot 49}\right)\right]$$

And the distribution of C_2 as follows,

$$p(x, C_2) = \frac{1}{6\sqrt{2\pi}} \exp\left(\frac{-(x-25)^2}{2 \cdot 36}\right)$$

[29]:
```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

mean2, std2 = 25, 6

mean11, std11 = -2, 5
mean12, std12 = 25, 7
X1 = np.random.normal(mean11, std11, 100000)
X2 = np.random.normal(mean12, std12, 100000)
dist1 = np.concatenate([X1, X2])

dist2 = np.random.normal(mean2,std2,750000)

fig = plt.figure(figsize = (15,7))
sns.kdeplot(dist1, shade=True)
sns.kdeplot(dist2,shade=True)
x1, y1 = [25, 25], [0, 0.07]
x2, y2 = [12,12], [0, 0.06]
plt.plot(x2,y2,linestyle='dashed',color='blue')
plt.plot(x1,y1,linestyle='dashed',color='black')
plt.text(19.5, 0.01, '$A_1$',fontsize=15)
plt.text(28, 0.01, '$A_2$',fontsize=15)
plt.text(20, 0.04, '$A_3$',fontsize=15)
plt.annotate('$R_1$', xy=(25,0.07), xytext=(-30,0.07),
             arrowprops={'arrowstyle': '->'}, va='center',fontsize=20)
plt.annotate('$R_2$', xy=(25,0.07), xytext=(60,0.07),
             arrowprops={'arrowstyle': '->'}, va='center',fontsize=20)
plt.text(27, 0.067, 'Decision Boundary $D_1$',fontsize=11)
plt.text(2, 0.061, 'Optimal Decision Boundary $D_2$',fontsize=11)
plt.text(-18, 0.02, '$p(x,C_1)$',fontsize=17)
plt.text(34, 0.045, '$p(x,C_2)$',fontsize=17)
plt.plot()
```

[29]: []



We need a rule that assigns each value of x to one of the available classes. Such a rule will divide the input space into regions R_k called *decision regions*, one for each class, such that all points in R_k are assigned into class C_k. The boundaries between decision regions are called *decision boundaries*.

A mistake occurs when an input vector belonging to class C_1 is assigned to class C_2 or vice versa. The probability of this occurrence is given by:

$$p(mistake) = p(x \in R_1, C_2) + p(x \in R_2, C_1)$$

$$= \int_{R_1} p(x, C_2)dx + \int_{R_2} p(x, C_1)dx$$

Clearly to minimize $p(mistake)$, we should arrange that each x is assigned to whichever class has smaller value of in the integrals. Thus, if $p(x, C_1) > p(x, C_2)$ for a given value of x, then we should assign that x to class C_1. From the product rule of probability, we have $p(x, C_k) = p(C_k|x)p(x)$. As you can see, both sides are divided by factor $p(x)$ which is common. So we can rewrite the inequality as $p(C_1|x) > p(C_2|x)$. This means that the minimum probability of making a mistake is obtained if each value of x is assigned to the class for which the posterior probability $p(C_k|x)$ is largest.

For the general case of multi-class tasks, we can easily say that maximizing the correctnes is slightly easier that minimizing the mistake

$$p(correct) = \sum_{k=1}^{K} p(x \in R_k, C_k)$$

$$= \sum_{k=1}^{K} \int_{R_k} p(x, C_k)dx$$

To explain the above figure, values of $x > D_1$ are classified as class C_2 and hence belong to decision region R_2. Whereas points $x < D_1$ classified as C_1 and belong to R_1. The mistake/error arise from regions A_1, A_2 and A_3. So that; for $x < D_1$, the errors are due to points from class C_2 being misclassified as C_1 (computed as sum $A_1 + A_3$). And; for $x > D_1$, the errors are due to points from class C_1 being misclassified as C_2 (area of A_2).

Actually, as you can see, the optimal choice for decision boundary is line D_2. Because in this case, region A_3 disappears. This is equivalent to the minimum misclassification rate decision rule, which assigns each value of x to the class having the higher posterior probability $p(C_k|x)$.

Now let's apply what we learn above to evaluating Logistic regression.

Confusion Matrix As in the Linear Regression, the binary cross entropy loss can be used for model evaluation.

$$L(\hat{y}, y) = -\sum_i^m y^i \log(p^i) + (1 - y^i)\log(1 - p^i) = -\sum_i^m y^i \log(\hat{y}^i) + (1 - y^i)\log(1 - \hat{y}^i)$$

But as in linear regression, loss is a relative metric. It cannot say anything about our model's validty or *accuracy*.

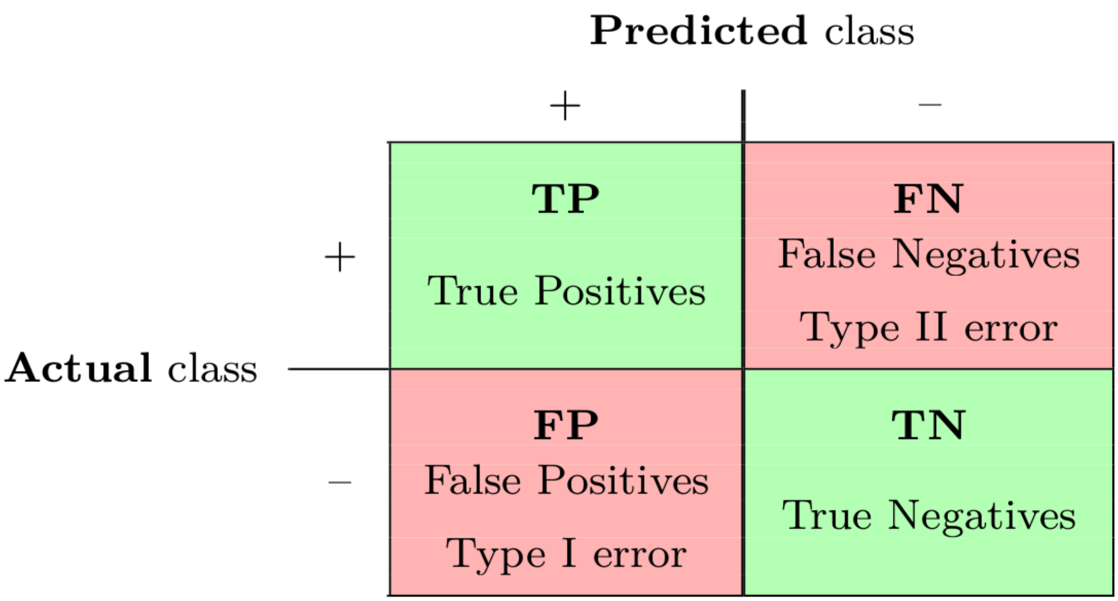
The confusion matrix is used for this evaluation:

In the field of machine learning and specifically the problem of statistical classification, a confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a matching matrix). Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa). The name stems from the fact that it makes it easy to see if the system is confusing two classes (i.e. commonly mislabeling one as another).

[2]:

```python
from IPython.display import Image
from IPython.core.display import HTML
Image(filename= "./img/cm.png",width=500, height=500)
```

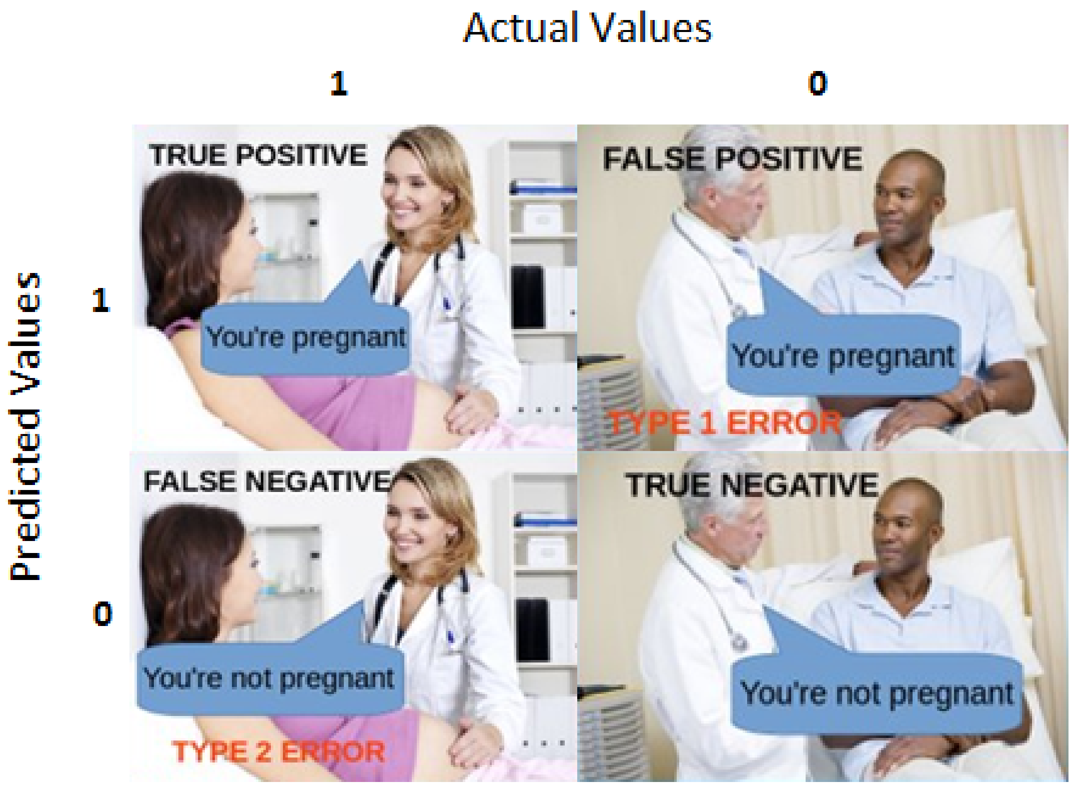
[2]:



- **True Positive (TP)**: For a given classes $C = \{C_0 = 0, C_1 = 1\}$, the input is predictes as $C_1 = 1$ and actual class of input is $C_1 = 1$.
- **True Negative (TN)**: For a given classes $C = \{C_0 = 0, C_1 = 1\}$, the input is predictes as $C_0 = 0$ and actual class of input is $C_0 = 0$.
- **False Positive (FP, Type 1 Error)**: For a given classes $C = \{C_0 = 0, C_1 = 1\}$, the input is predictes as $C_1 = 1$ but actual class of input is $C_0 = 0$.
- **False Negative (FN, Type 2 Error)**: For a given classes $C = \{C_0 = 0, C_1 = 1\}$, the input is predictes as $C_0 = 0$ but actual class of input is $C_1 = 1$.

[3]:
```
from IPython.display import Image
from IPython.core.display import HTML
Image(filename= "./img/cm_2.png",width=500, height=500)
```

[3]:



But TP, TN, FP, FN are not telling us informations about model's performance individually. Let's introduce metrics based on this values.

- **Accuracy:**

$$\frac{TP + TN}{TP + TN + FP + FN}$$

  is overall performance of model. Ratio between true predictions and true predictions plus false predictions.

- **Precision:**

$$\frac{TP}{TP + FP}$$

  it measures how accurate the positive predictions are.

Precision talks about how precise/accurate your model is out of those predicted positive, how many of them are actual positive. Precision is a good measure to determine, when the costs of False Positive is high. For instance, email spam detection. In email spam detection, a false positive means that an email that is non-spam (actual negative) has been identified as spam (predicted spam). The email user might lose important emails if the precision is not high for the spam detection model.

- **Recall:**

$$\frac{TP}{TP + FN}$$

it measures out of all the positive classes, how much we predicted correctly. It should be high as possible. Coverage of actual positive sample

Recall actually calculates how many of the Actual Positives our model capture through labeling it as Positive (True Positive). Applying the same understanding, we know that Recall shall be the model metric we use to select our best model when there is a high cost associated with False Negative.

For instance, in fraud detection or sick patient detection. If a fraudulent transaction (Actual Positive) is predicted as non-fraudulent (Predicted Negative), the consequence can be very bad for the bank.

- **F1 score:**

$$2\times\frac{precision \times recall}{precision + recall} = 2\times\frac{\frac{TP^2}{TP^2+TP\times FP+FN\times TP+FN\times FP}}{\frac{TP^2+FN\times TP+TP^2+FP\times TP}{TP^2+TP\times FP+FN\times TP+FN\times FP}} = 2\times\frac{TP^2}{2TP^2+FN\times TP+FP\times TP} = \frac{2T}{2TP+F}$$

It is difficult to compare two models with low precision and high recall or vice versa. So to make them comparable, we use F-Score. F-score helps to measure Recall and Precision at the same time. It uses Harmonic Mean in place of Arithmetic Mean by punishing the extreme values more. F1 Score might be a better measure to use if we need to seek a balance between Precision and Recall AND there is an uneven class distribution (large number of Actual Negatives).

Let's do an example.

[6]:
```python
from IPython.display import Image
from IPython.core.display import HTML
Image(filename= "./img/cm_example.jpeg",width=500, height=500)
```

[6]:

[12]:
```python
from sklearn.linear_model import LogisticRegression
import pandas as pd
import numpy as np
import warnings


warnings.filterwarnings("ignore", category=DeprecationWarning)

def trainTestSplit(data, ratio = 0.8):
    if isinstance(data, pd.DataFrame):
        data = data.sample(frac=1).reset_index(drop=True)
        train_pct_index =  int(ratio * len(data))
        train = data.iloc[:train_pct_index,:]
        test = data.iloc[train_pct_index:,:]
        test.reset_index(inplace=True, drop = True)
        return train, test
    elif isinstance(data,np.array):
        X_train, X_test = data[:train_pct_index,0], data[train_pct_index:,0]
        Y_train, Y_test = data[:train_pct_index,1:], data[train_pct_index:,1:]
        return X_train, X_test, Y_train, Y_test

df = pd.read_csv('./data/diabetes.csv')
df.head()
```

[12]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            2      138             62             35        0  33.6
1            0       84             82             31      125  38.2
2            0      145              0              0        0  44.2
3            0      135             68             42      250  42.3
4            1      139             62             41      480  40.7

   DiabetesPedigreeFunction  Age  Outcome
0                     0.127   47        1
1                     0.233   23        0
2                     0.630   31        1
3                     0.365   24        1
4                     0.536   21        0
```

[13]:
```python
df.describe()
```

[13]:
```
       Pregnancies      Glucose  BloodPressure  SkinThickness      Insulin  \
count  2000.000000  2000.000000    2000.000000    2000.000000  2000.000000
mean      3.703500   121.182500      69.145500      20.935000    80.254000
std       3.306063    32.068636      19.188315      16.103243   111.180534
min       0.000000     0.000000       0.000000       0.000000     0.000000
25%       1.000000    99.000000      63.500000       0.000000     0.000000
50%       3.000000   117.000000      72.000000      23.000000    40.000000
```

```
75%       6.000000   141.000000     80.000000     32.000000   130.000000
max      17.000000   199.000000    122.000000    110.000000   744.000000

              BMI  DiabetesPedigreeFunction          Age      Outcome
count  2000.000000               2000.000000  2000.000000  2000.000000
mean     32.193000                  0.470930    33.090500     0.342000
std       8.149901                  0.323553    11.786423     0.474498
min       0.000000                  0.078000    21.000000     0.000000
25%      27.375000                  0.244000    24.000000     0.000000
50%      32.300000                  0.376000    29.000000     0.000000
75%      36.800000                  0.624000    40.000000     1.000000
max      80.600000                  2.420000    81.000000     1.000000
```

[14]:
```
train, test = trainTestSplit(df,0.8)
print('Number of samples in data: {0} \nNumber of samples in training set: {1}␣
 ↪\nNumber of samples in test test: {2}'
      .format(len(df),len(train),len(test)))
```

```
Number of samples in data: 2000
Number of samples in training set: 1600
Number of samples in test test: 400
```

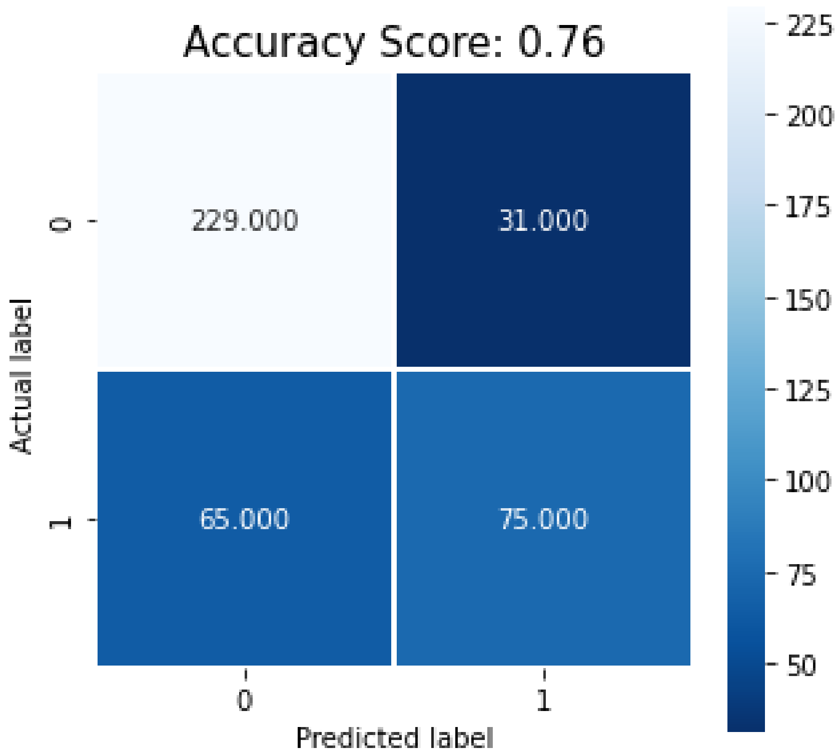
[15]:
```
X_train = np.array(train.iloc[:,:-1])
y_train = np.array(train.iloc[:,-1])
X_test = np.array(test.iloc[:,:-1])
y_test = np.array(test.iloc[:,-1])

with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    model = LogisticRegression();
    model.fit(X_train, y_train);
```

[16]:
```
score_train = model.score(X_train, y_train)
score_test = model.score(X_test, y_test)
print('Accuracy Train: {0} \nAccuracy Test: {1}'.format(score_train,score_test))
```

```
Accuracy Train: 0.78375
Accuracy Test: 0.76
```

[17]:
```
import matplotlib.pyplot as plt # vis
import seaborn as sns # vis / distribution

from sklearn import metrics

cm = metrics.confusion_matrix(y_test, model.predict(X_test))
```

[18]:
```
plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap =␣
 ↪'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(score_test)
plt.title(all_sample_title, size = 15);
```



[19]:
```
precision = cm[1,1] / (cm[1,1] + cm[0,1])
recall = cm[1,1] / (cm[1,1] + cm[1,0])
f1 = 2 * precision * recall / (precision + recall)
print('Precision: {0}, \nRecall: {1}, \nF1 Score: {2}'.
 ↪format(precision,recall,f1))
```

```
Precision: 0.7075471698113207,
Recall: 0.5357142857142857,
F1 Score: 0.6097560975609755
```

Others Look at other metrics: Adjusted R^2, Mallow's Cp, AIC, BIC etc.

1.2 The Problem of Overfitting

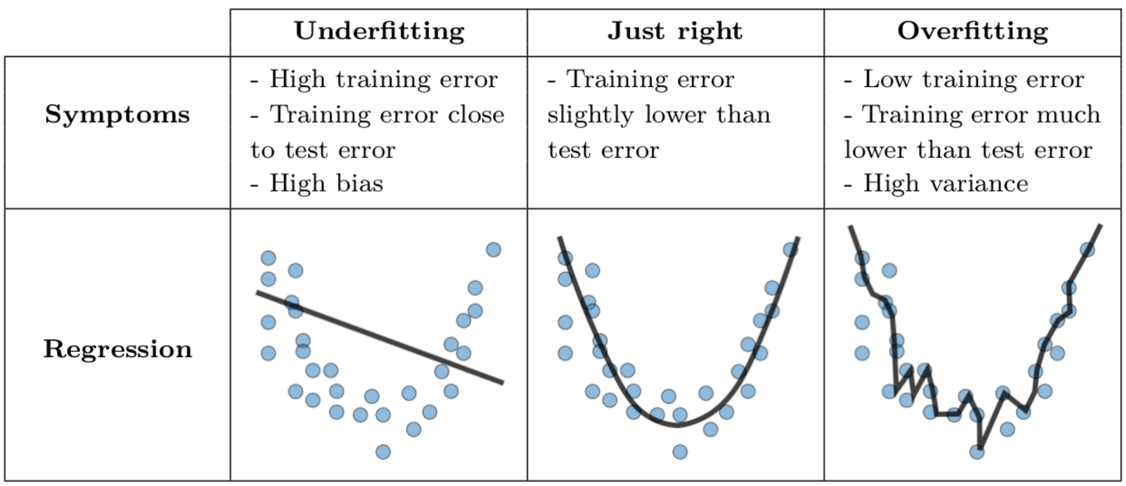
Simply put, overfitting arises when your model has fit the data too well. That can seem weird at first glance. The whole point of machine learning is to fit the data. How can it be that your model is too good at that? In machine learning, there are two really important measures you should be paying attention to at all times: the training error and the test error. We introduced them at the beginning of the notebook. Training error is a measure of how well your model performed in training, and test error is how well it performed in the wild.

When the model performs well in training data but it does not perform well in test data. We call it **overfitting**.

If we have too many features, the learned model may fit the training data set very well, but fail to generalize to new axamples (test set). That neans, if our model learns training set very well, we can't generalize these model for other subsets.

[8]:
```python
from IPython.display import Image
from IPython.core.display import HTML
Image(filename= "./img/of4.png",width=700, height=500)
```

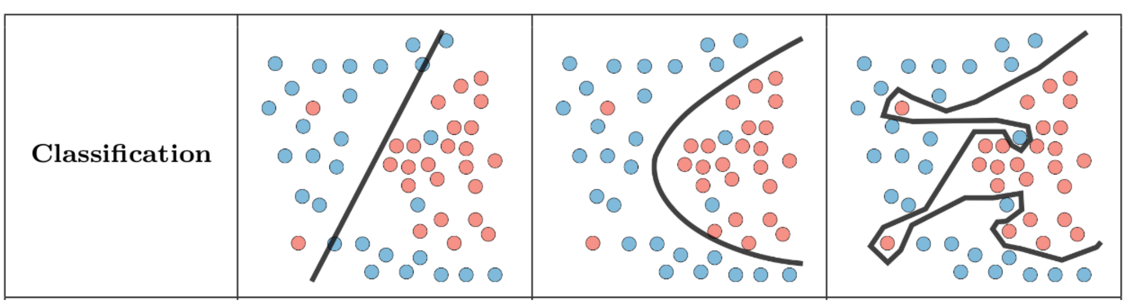
[8]:



	Underfitting	Just right	Overfitting
Symptoms	- High training error - Training error close to test error - High bias	- Training error slightly lower than test error	- Low training error - Training error much lower than test error - High variance
Regression			

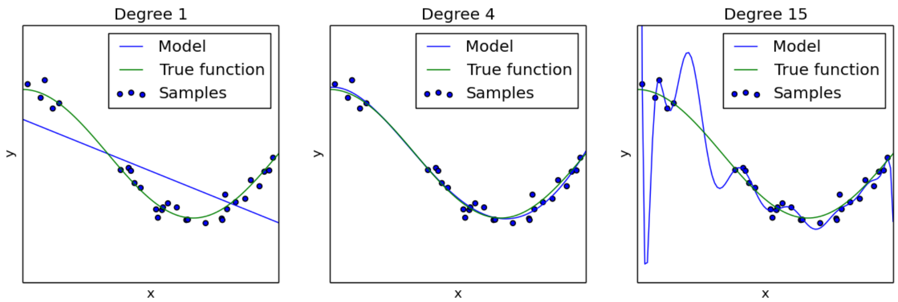
[9]:
```python
Image(filename= "./img/of5.png",width=700, height=500)
```

[9]:



[14]:
```python
Image(filename= "./img/of6.png",width=1000, height=500)
```

[14]:



Bias - Variance Trade-Off Let's visualize overfitting on linear regression.

High variance means overfitting, high bias means underfitting.

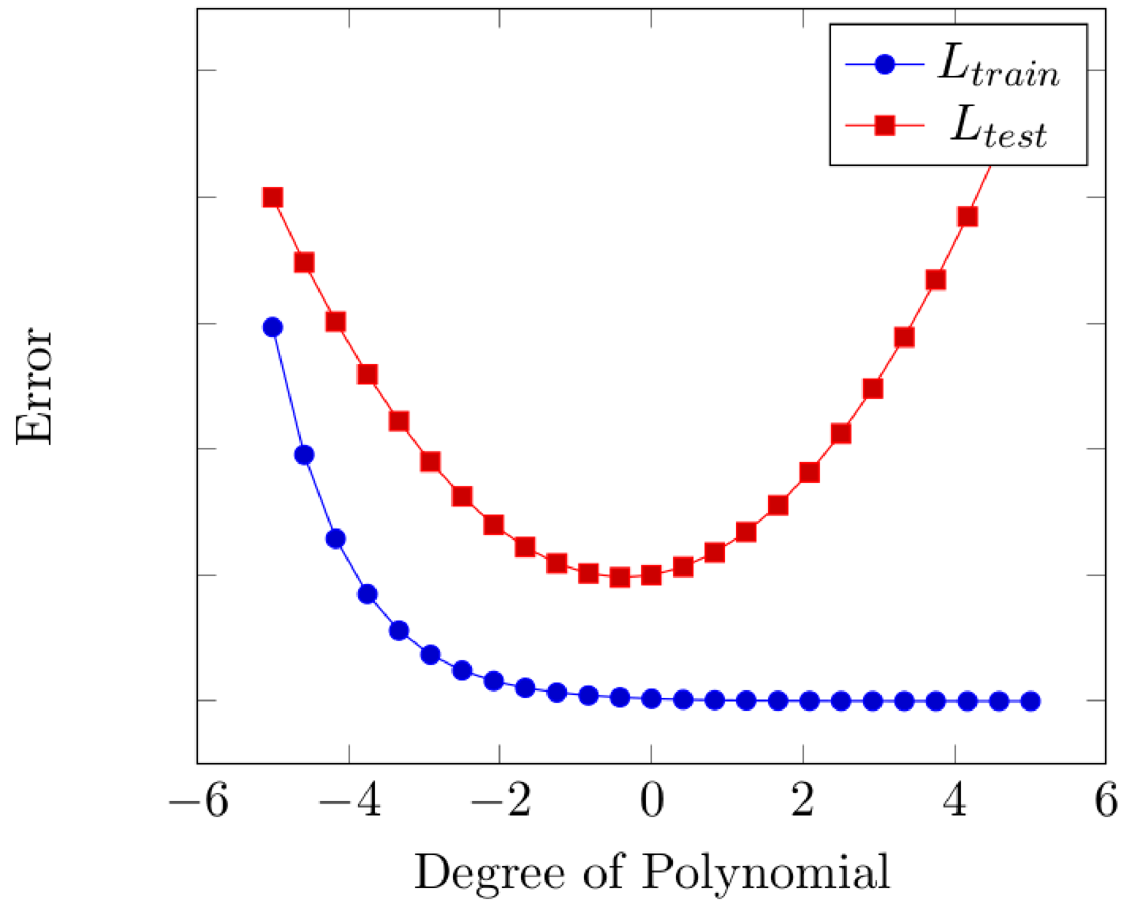
Defining training and test error:

$$L_{train} = \frac{1}{2m_{train}} \sum_i (y^i_{train} - \hat{y}^i_{train})^2$$

$$L_{test} = \frac{1}{2m_{test}} \sum_i (y^i_{test} - \hat{y}^i_{test})^2$$

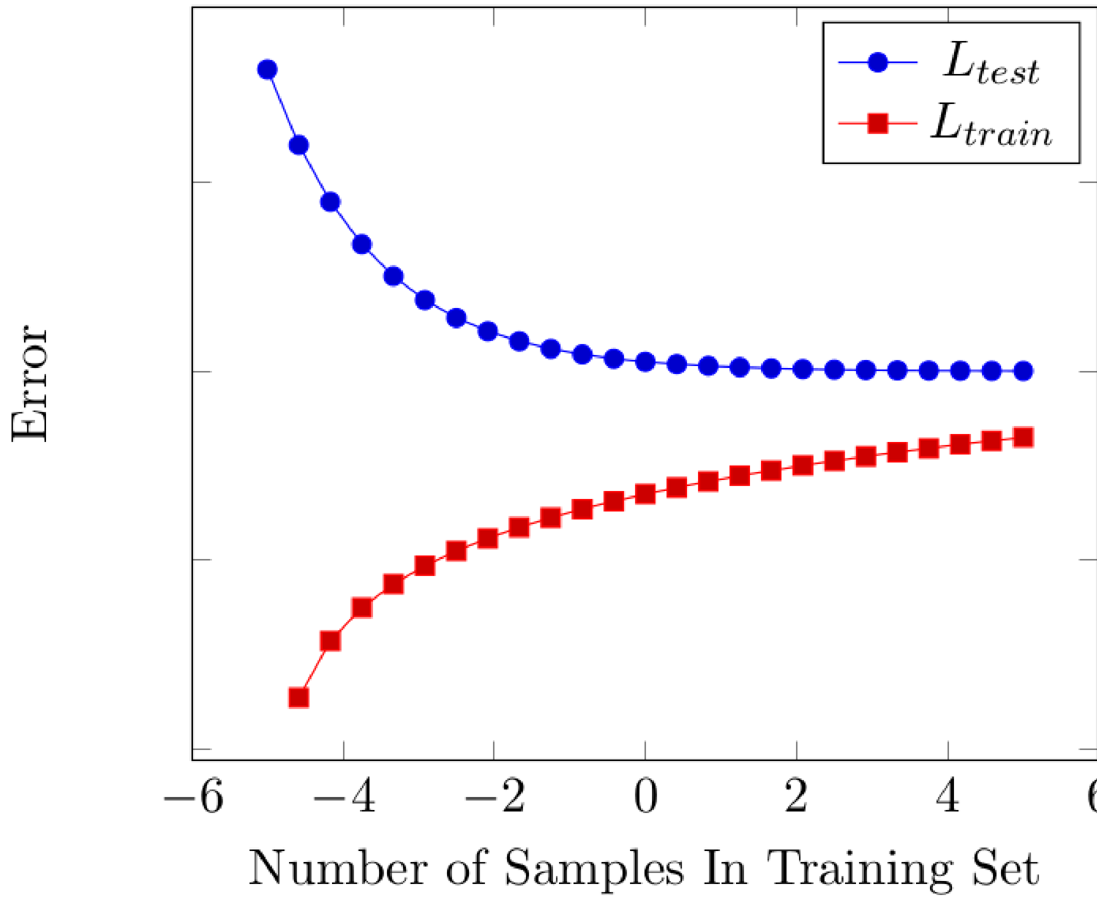
[19]:
```python
from IPython.display import Image
from IPython.core.display import HTML
Image(filename= "./img/biasvar1.png",width=400, height=40)
```

[19]:



[1]:
```python
from IPython.display import Image
from IPython.core.display import HTML
Image(filename= "./img/biasvar2.png",width=400, height=40)
```

[1]:



So, what we should do when overfitting happens?

- Get more training examples → fixes high variance.
- Try smaller sets of features (look at Principal Component Analysis) → fixes high variance.
- Try getting additional features → fixes high bias.
- Try adding polynomial features → fixes high bias.

So let's prove that the expected MSE, for a given observation x, can be always be decomposed into the sum of three fundamental quantities: the variance, the squared bias, the variance error ϵ.

Our objective is to, for a fixed point x, evaluate how closely the estimator can estimate the noisy observation Y corresponding to x.

Again, we can view D as the training data, and (x, y) as a test point — the test point x is probably not even in the training set D! Mathematically, we express our metric as the expected squared error between the estimator and the observation:

$$\mathbb{E}[(f(x;D) - y)^2] = \frac{1}{m}\sum_i (y^i - \hat{y}^i)^2$$

The error metric is difficult to interpret and work with, so let's try to decompose it into parts thatare easier to understand.

[5]:
```python
from IPython.display import Image
from IPython.core.display import HTML
Image(filename= "./img/biasvar_dec.png",width=800, height=800)
```

[5]:

derivation. At its core, it uses the technique that $\mathbb{E}[(Z-Y)^2] = \mathbb{E}[((Z-\mathbb{E}[Z]) + (\mathbb{E}[Z]-Y))^2]$ which decomposes to easily give us the variance of Z and other terms.

$$
\begin{aligned}
\varepsilon(\mathbf{x};h) &= \mathbb{E}[(h(\mathbf{x};\mathcal{D}) - Y)^2] \\
&= \mathbb{E}\left[\left(h(\mathbf{x};\mathcal{D}) - \mathbb{E}[h(\mathbf{x};\mathcal{D})] + \mathbb{E}[h(\mathbf{x};\mathcal{D})] - Y\right)^2\right] \\
&= \mathbb{E}\left[\left(h(\mathbf{x};\mathcal{D}) - \mathbb{E}[h(\mathbf{x};\mathcal{D})]\right)^2\right] + \mathbb{E}\left[\left(\mathbb{E}[h(\mathbf{x};\mathcal{D})] - Y\right)^2\right] + 2\mathbb{E}\left[\left(h(\mathbf{x};\mathcal{D}) - \mathbb{E}[h(\mathbf{x};\mathcal{D})]\right)\cdot\left(\mathbb{E}[h(\mathbf{x};\mathcal{D})] - Y\right)\right] \\
&= \mathbb{E}\left[\left(h(\mathbf{x};\mathcal{D}) - \mathbb{E}[h(\mathbf{x};\mathcal{D})]\right)^2\right] + \mathbb{E}\left[\left(\mathbb{E}[h(\mathbf{x};\mathcal{D})] - Y\right)^2\right] + 2\mathbb{E}[\cancel{h(\mathbf{x};\mathcal{D}) - \mathbb{E}[h(\mathbf{x};\mathcal{D})]}]\cdot\mathbb{E}[\mathbb{E}[h(\mathbf{x};\mathcal{D})] - Y] \\
&= \mathbb{E}\left[\left(h(\mathbf{x};\mathcal{D}) - \mathbb{E}[h(\mathbf{x};\mathcal{D})]\right)^2\right] + \mathbb{E}\left[\left(\mathbb{E}[h(\mathbf{x};\mathcal{D})] - Y\right)^2\right] \\
&= \mathrm{Var}((h(\mathbf{x};\mathcal{D})) + \mathbb{E}\left[\left(\mathbb{E}[h(\mathbf{x};\mathcal{D})] - Y\right)^2\right] \\
&= \mathrm{Var}((h(\mathbf{x};\mathcal{D})) + \mathbb{E}\left[\left(\mathbb{E}[h(\mathbf{x};\mathcal{D})] - \mathbb{E}[Y] + \mathbb{E}[Y] - Y\right)^2\right] \\
&= \mathrm{Var}((h(\mathbf{x};\mathcal{D})) + \mathbb{E}\left[\left(\mathbb{E}[h(\mathbf{x};\mathcal{D})] - \mathbb{E}[Y]\right)^2\right] + \mathbb{E}[(Y - \mathbb{E}[Y])^2] + 2\left(\mathbb{E}[h(\mathbf{x};\mathcal{D})] - \mathbb{E}[Y]\right)\cdot\cancel{\mathbb{E}[\mathbb{E}[Y] - Y]} \\
&= \mathrm{Var}((h(\mathbf{x};\mathcal{D})) + \mathbb{E}\left[\left(\mathbb{E}[h(\mathbf{x};\mathcal{D})] - \mathbb{E}[Y]\right)^2\right] + \mathbb{E}[(Y - \mathbb{E}[Y])^2] \\
&= \mathrm{Var}((h(\mathbf{x};\mathcal{D})) + \left(\mathbb{E}[h(\mathbf{x};\mathcal{D})] - \mathbb{E}[Y]\right)^2 + \mathrm{Var}(Y) \\
&= \mathrm{Var}((h(\mathbf{x};\mathcal{D})) + \left(\mathbb{E}[h(\mathbf{x};\mathcal{D})] - f(\mathbf{x})\right)^2 + \mathrm{Var}(Z) \\
&= \underbrace{\left(\mathbb{E}[h(\mathbf{x};\mathcal{D})] - f(\mathbf{x})\right)^2}_{bias^2\text{ of method}} + \underbrace{\mathrm{Var}(h(\mathbf{x};\mathcal{D}))}_{\text{variance of method}} + \underbrace{\mathrm{Var}(Z)}_{\text{irreducible error}}
\end{aligned}
$$

[ ]: