introduction-to-machine-learning

May 15, 2021

Run in Google Colab

1 Learning Tools

1.1 Books Reference

These introductory lessons assume a basic level of statistical and mathematical knowledge. No previous knowledge of Machine Learning is assumed. For this reason I have decided to use two basic texts for the preparation of these lessons that you can consult for further details on the topics we are going to deal with.

- John C. Hull, **Machine Learning in Business, An Introduction to the World of Data Science**, Amazon (2019)
- Paul Wilmott}, **Machine Learning, An Applied Mathematics Introduction**, Panda Ohana Publishing (2019)

1.2 A Few Words on Google Colab

Although it is not essential to work in a colab environment (all the course notebooks are in fact designed to be able to run without problems locally on your pc), it is useful to know some basic elements of the interaction with colab. In particular, in the cells below you will find two examples for the use of external files. In the first case it is shown how to load a text file from your local PC into the google virtual machine. The second example relates to the opposite operation: let's create a simple pandas dataframe into the colab environment and export it in csv format to the local machine.

1.2.1 How Upload a File on Google Colab

[65]:
```
if 'google.colab' in str(get_ipython()):
    from google.colab import files
    uploaded = files.upload()
    path = ''
else:
    path = './data/'
```

[66]:
```
with open(path + "carroll-alice.txt", "r") as f:
    alice = f.read()
```

```
alice[:392]
```

[66]:
```
"[Alice's Adventures in Wonderland by Lewis Carroll 1865]\n\nCHAPTER I. Down the
Rabbit-Hole\n\nAlice was beginning to get very tired of sitting by her sister on
the\nbank, and of having nothing to do: once or twice she had peeped into
the\nbook her sister was reading, but it had no pictures or conversations
in\nit, 'and what is the use of a book,' thought Alice 'without pictures
or\nconversation?'"
```

1.2.2 How Download a File on Google Colab

[67]:
```
import pandas as pd

cars = {'Brand': ['Honda Civic','Toyota Corolla','Ford Focus','Audi A4'],
        'Price': [22000,25000,27000,35000]
        }

df = pd.DataFrame(cars, columns= ['Brand', 'Price'])
```

[68]:
```
if 'google.colab' in str(get_ipython()):
    # if we run in google environment first we save in virtual machine...
    df.to_csv ('export_dataframe.csv', index = False, header=True)
    # ...then we download to local machine
    from google.colab import files
    files.download("export_dataframe.csv")
else:
    # if we are working in local we save directly with the usual method
    df.to_csv ('./data/export_dataframe.csv', index = False, header=True)
```

2 General Ideas

2.1 What is Machine Learning?

Two definitions of Machine Learning are offered. Arthur Samuel described it as: "the field of study that gives computers the ability to learn without being explicitly programmed." This is an older, informal definition.

Tom Mitchell provides a more modern definition: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Example: playing checkers.

E = the experience of playing many games of checkers

T = the task of playing checkers.

P = the probability that the program will win the next game.

To use machine learning effectively **you have to understand how the underlying algorithms work**. It is tempting to learn a language such as Python or R and apply various packages to your data without really understanding what the packages are doing or even how the results should be interpreted. This would be a bit like a finance specialist using the Black and Scholes model to value options without understanding where it comes from or its limitations.

2.1.1 Type of Machine Learning Models

There are four main categories of machine learning models:

- Supervised Learning
- Unsupervised Learning
- Semi-Supervised Learning
- Reinforcement Learning

Supervised learning is concerned with using data to make predictions. We can distinguish between supervised learning models that are used to predict a variable and models that are used for classification.

Unsupervised learning is concerned with recognizing patterns in data. The main object is not to forecast a particular variable, rather it is to understand the data environment better.

2.1.2 Jargon

The data for supervised learning contains whare are referred to as **features** and **labels**. The **labels** are the values of the target that is to be predicted. The **features** are the variables from which the predictions are to be made. For example when predicting the price of a house the **features** could be the swuare meters of living space, the number of bedrooms, the number of bathrooms, the size of the garage and so on. The **label** would be the house price.

The data for unsupervised learning consists of features but no labels because the model is being used to identify patterns not to forecast something.

2.1.3 Type of Data

There are two types of data:

- Numerical
- Categorical

Numerical data consists of numbers. Categorical data is data which can fall into a number of different categories, for example data to predict a house price might categorize driveways as asphalt, concrete, grass, etc. Categorical data must be converted to numbers for the purposes of analysis.

The standard way of dealing with categorical features is to create a dummy variable for each category. The value of this variable is 1 if the feature is in the category and 0 otherwise. For example in the situation in which individuals are categorized as male or female, we could create two dummy variables. For man the first dummy variable would be 1 and the second would be 0. The opposite for women. This procedure is appropriate when there is no natural ordering between the feature values.

When there is a natural ordering, we can reflect this in the numbers assigned. For example if the size of an order is classified as small, medium or large, we can replace the feature by a numerical variable where *small = 1*, *medium = 2* and *large = 3*.

2.2 Feature Normalization

The success of a machine learning algorithm highly depends on the quality of the data fed into the model. Real-world data is often dirty containing outliers, missing values, wrong data types, irrelevant features, or non-standardized data. The presence of any of these will prevent the machine learning model to properly learn. For this reason, transforming raw data into a useful format is an essential stage in the machine learning process. One technique you will come across multiple times when pre-processing data is feature normalization.

Data Normalization is a common practice in machine learning which consists of transforming numeric columns to a common scale. In machine learning, some feature values differ from others multiple times. The features with higher values will dominate the leaning process. However, it does not mean those variables are more important to predict the outcome of the model. Data normalization transforms multiscaled data to the same scale. After normalization, all variables have a similar influence on the model, improving the stability and performance of the learning algorithm.

There are multiple normalization techniques in statistics. In this notebook, we will cover the most important ones:

- The maximum absolute scaling
- The min-max feature scaling
- The z-score method

2.2.1 The maximum absolute scaling

The maximum absolute scaling rescales each feature between -1 and 1 by dividing every observation by its maximum absolute value.

$$x_{new} = \frac{x_{old}}{\max |x_{old}|}$$

2.2.2 The min-max feature scaling

The min-max approach (often called normalization) rescales the feature to a fixed range of [0,1] by subtracting the minimum value of the feature and then dividing by the range:

$$x_{new} = \frac{x_{old} - x_{min}}{x_{max} - x_{min}}$$

2.2.3 Z-Score

The z-score method (often called standardization) transforms the data into a distribution with a mean of 0 and a standard deviation of 1. Each standardized value is computed by subtracting the mean of the corresponding feature and then dividing by the standard deviation.

$$x_{new} = \frac{x_{old} - \mu}{\sigma}$$

Unlike min-max scaling, the z-score does not rescale the feature to a fixed range. The z-score typically ranges from -3.00 to 3.00 (more than 99% of the data) if the input is normally distributed.

It is important to bear in mind that z-scores are not necessarily normally distributed. They just scale the data and follow the same distribution as the original input. This transformed distribution has a mean of 0 and a standard deviation of 1 and is going to be the standard normal distribution only if the input feature follows a normal distribution.

2.3 Cost Functions

2.3.1 Linear Cost Function

In Machine Learning a cost function or loss function is used to represent how far away a mathematical model is from the real data. One adjusts the mathematical model, usually by varying parameters within the model, so as to minimize the cost function.

Let's take for example the simple case of a linear fitting. We want to find a relationship of the form

$$y = \theta_0 + \theta_1 x \tag{1}$$

where the θs are the parameters that we want to find to give us the best fit to the data. We call this linear function $h_\theta(x)$ to emphasize the dependence on both the variable x and the two parameters θ_0 and θ_1.

We want to measure how far away the data, the $y^{(n)}$s, are from the function $h_\theta(x)$. A common way to do this is via the quadratic *cost function*

$$J(\theta) = \frac{1}{2N} \sum_{n=1}^{N} \left[h_\theta \left(x^{(n)} \right) - y^{(n)} \right]^2 \tag{2}$$

This is called *Ordinary Least Squares.*

In this case, the minimum is easily find analitically, differentiate (2) with respect to both θs and set the result to zero:

$$\begin{aligned} \tfrac{\partial J}{\partial \theta_0} &= \sum_{n=1}^{N} \left(\theta_0 + \theta_1 x^{(n)} - y^{(n)} \right) = 0 \\ \tfrac{\partial J}{\partial \theta_1} &= \sum_{n=1}^{N} x^{(n)} \left(\theta_0 + \theta_1 x^{(n)} - y^{(n)} \right) = 0 \end{aligned} \tag{3}$$

The solution is trivially obtained for both θs

$$\begin{aligned} \theta_0 &= \tfrac{(\sum y)\left(\sum x^2\right) - (\sum x)(\sum xy)}{N(\sum x^2)(\sum x)^2} \\ \theta_1 &= \tfrac{N(\sum xy) - (\sum y)(\sum x)}{N(\sum x^2)(\sum x)^2} \end{aligned} \tag{4}$$

2.4 Gradient Descent

The scheme works as follow: start with an initial guess for each parameter θ_k. Then move θ_k in the direction of the slope:

$$\theta_k^{new} = \theta_k^{old} + \beta \frac{\partial J}{\partial \theta_k} \tag{5}$$

Update all θ_k simultaneously and repet until convergence. Here β is a *learning factor* that governs how far you move. if β is too small it will take a long time to converge, if too large it will overshoot and might not converge at all.

The loss function J is a function of all of the data points. In the above description of gradient descent we have used all of the data points simultaneously. This is called *batch gradient* descent. But rather than use all of the data in the parameter updating we can use a technique called *stochastic gradient descent*. This is like batch gradient descent except that you only update using *one* of the data points each time. And that data point is chosen randomly.

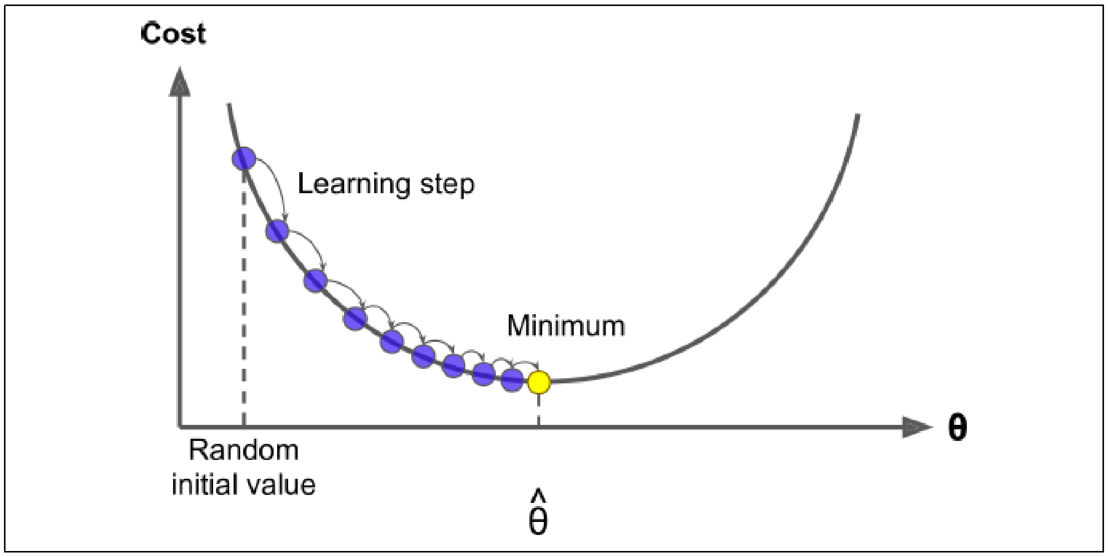
$$J(\theta) = \sum_{n=1}^{N} J_n(\theta) \tag{6}$$

Stochastic gradient descent means pick an n at random and then update according to

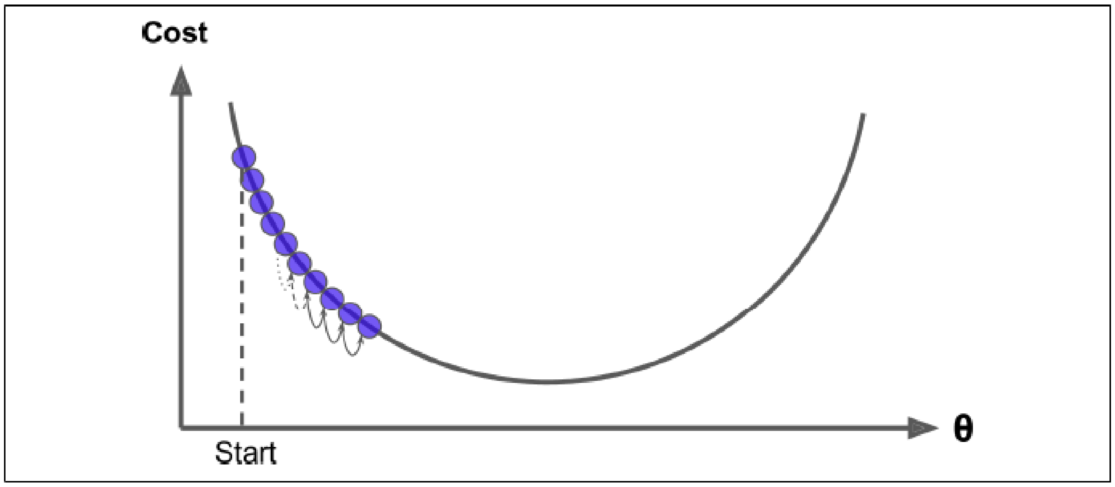
$$\theta_k^{new} = \theta_k^{old} + \beta \frac{\partial J_n}{\partial \theta_k} \tag{7}$$

Repeat, picking another data point at random, etc.

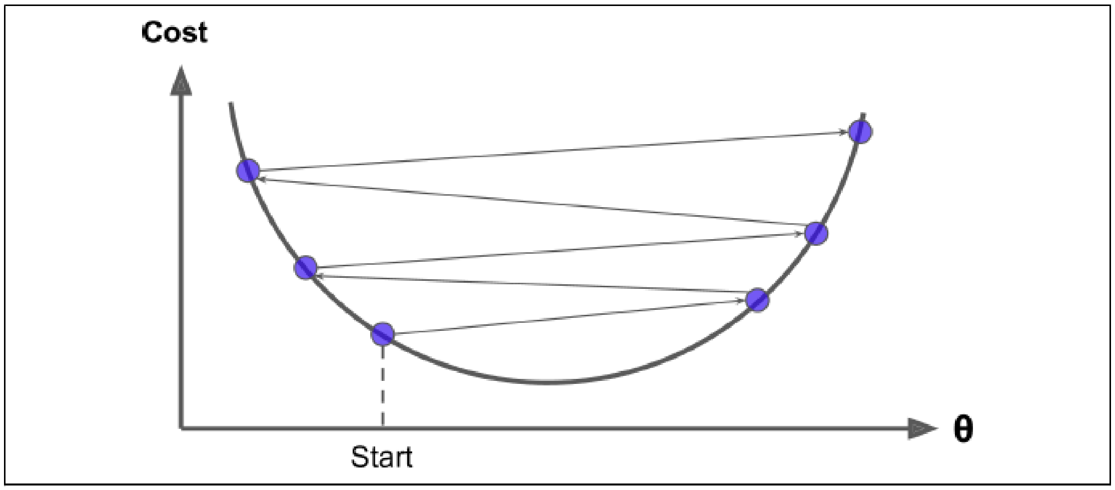
An important parameter in Gradient Descent is the size of the steps, determined by the learning rate hyperparameter.



If the learning rate is too small, then the algorithm will have to go through many iterations to converge, which will take a long time…



… on the other hand, if the learning rate is too high, you might jump across the valley. This might make the algorithm diverge failing to find a good solution.



2.5 Validation and Testing

When data is used for forecasting there is a danger that the machine learning model will work very well for data, but will not generalize well to other data. An obvious point is that it is important that the data used in a machine learning model be representative of the situations to which the model is to be applied. It is also important to test a model out-of-sample, by this we mean that the model should be tested on data that is different from the sample data used to determine the parameters of the model.

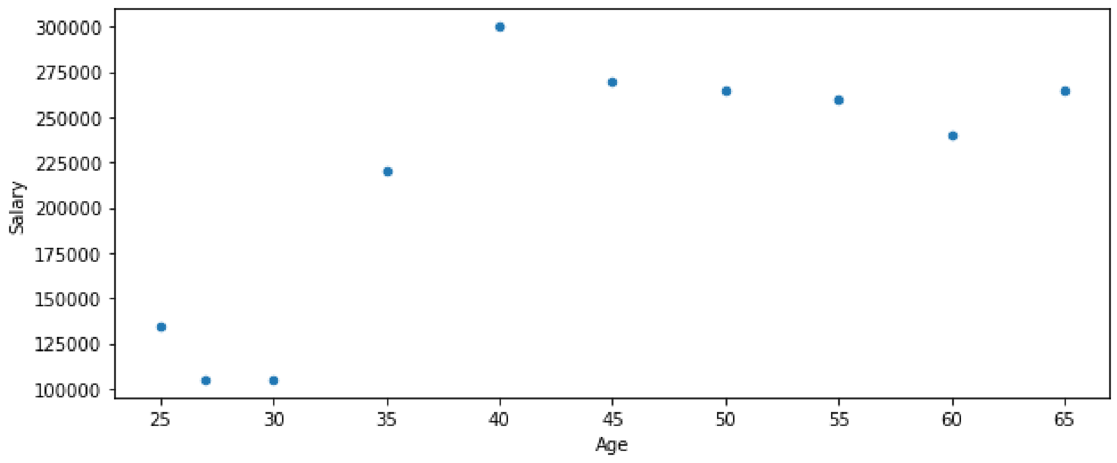
Data scientist refer to the sample data as the **training set** and the data used to determine the accuracy of the model as the **test set**, often a **validation set** is used as well as we explain later;

[186]:
```
if 'google.colab' in str(get_ipython()):
    from google.colab import files
    uploaded = files.upload()
    path = ''
else:
    path = './data/'
```

[187]:
```
# Load the Pandas libraries with alias 'pd'
import pandas as pd
# Read data from file 'salary_vs_age_1.csv'
# (in the same directory that your python process is based)
# Control delimiters, with read_table
df1 = pd.read_table(path + "salary_vs_age_1.csv", sep=";")
# Preview the first 5 lines of the loaded data
print(df1.head())
```

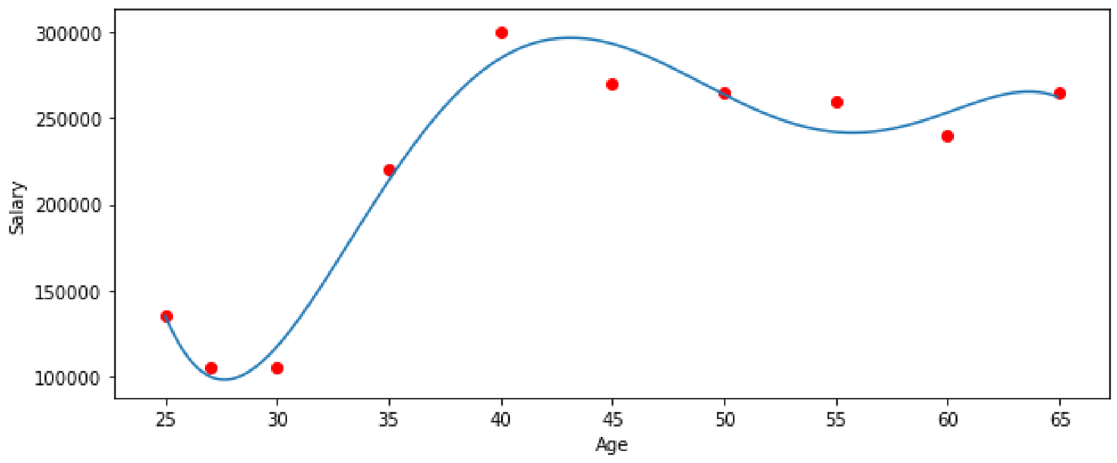
```
   Age  Salary
0   25  135000
1   27  105000
2   30  105000
3   35  220000
4   40  300000
```

[188]:
```
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = [10, 4]
ax=plt.gca()

df1.plot(x ='Age', y='Salary', kind = 'scatter', ax=ax)
plt.show()
```



polynomial fitting with pandas

[189]:
```
import numpy as np

x1 = df1['Age']
y1 = df1['Salary']

n = len(x1)

degree = 5

weights = np.polyfit(x1, y1, degree)
model   = np.poly1d(weights)

xx1 = np.arange(x1[0], x1[n-1], 0.1)
plt.plot(xx1, model(xx1))
plt.xlabel("Age")
plt.ylabel("Salary")
plt.scatter(x1,y1, color='red')
plt.show()
```



[190]:
```
y1  = np.array(y1)
yy1 = np.array(model(x1))

rmse = np.sqrt(np.sum((y1-yy1)**2)/(n-1))

print('Root Mean Square Error:')
print(rmse)
```

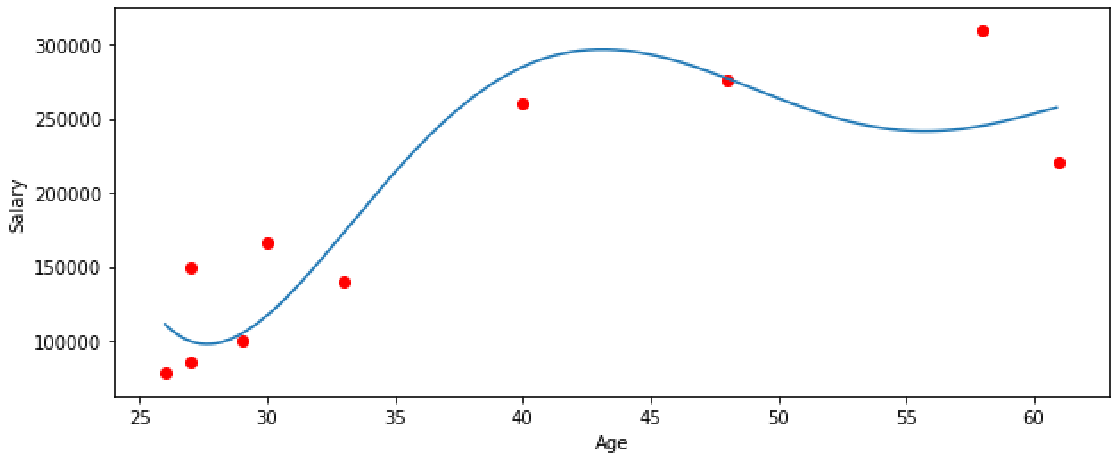
```
Root Mean Square Error:
12902.203044371492
```

```
[191]: if 'google.colab' in str(get_ipython()):
           from google.colab import files
           uploaded = files.upload()
           path = ''
       else:
           path = './data/'
```

```
[192]: df2 = pd.read_table(path + "salary_vs_age_2.csv", sep=";")
       x2 = df2['Age']
       y2 = df2['Salary']
       n  = len(x2)

       y2  = np.array(y2)
       yy2 = np.array(model(x2))

       rmse = np.sqrt(np.sum((y2-yy2)**2)/(n-1))

       print('Root Mean Square Error:')
       print(rmse)
```

```
Root Mean Square Error:
38825.220509174826
```

```
[193]: xx2 = np.arange(x2[0], x2[n-1], 0.1)
       plt.plot(xx2, model(xx2))
       plt.xlabel("Age")
       plt.ylabel("Salary")
       plt.scatter(x2,y2, color='red')
       plt.show()
```



- The root mean squared error (rmse) for the training data set is $12,902
- The rmse for the test data set is $38,794

We conclude that the model overfits the data. The complexity of the model should be increased only until out-of-sample tests indicate that it does not generalize well.

2.6 Bias and Variance

Suppose there is a relationship between an independent variable x and a dependent variable y:

$$y = f(x) + \epsilon \tag{8}$$

Where ϵ is an error term with mean zero and variance σ^2. The error term captures either genuine randomness in the data or noise due to measurement error.

Suppose we find a deterministic model for this relationship:

$$y = \hat{f}(x) \tag{9}$$

Now it comes a new data point x' not in the training set and we want to predict the corresponding y'. The error we will observe in our model at point x' is going to be

$$\hat{f}(x') - f(x') - \epsilon \tag{10}$$

There are two different sources of error in this equation. The first one is included in the factor ϵ, the second one, more interesting, is due to what is in our training set. A robust model should give us the same prediction whatever data we used for training out model. Let's look at the average error:

$$E\left[\hat{f}(x')\right] - f(x') \tag{11}$$

where the expectation is taken over random samples of training data (having the same distributio as the training data).

This is the definition of the **bias**

$$\text{Bias}\left[\hat{f}(x')\right] = E\left[\hat{f}(x')\right] - f(x') \tag{12}$$

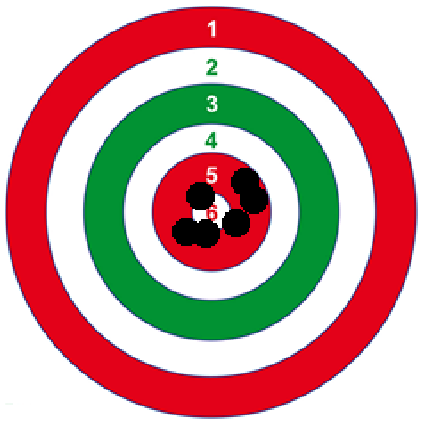
We can also look at the mean square error

$$E\left[\left(\hat{f}(x') - f(x') - \epsilon\right)^2\right] = \left[\text{Bias}\left(\hat{f}(x')\right)\right]^2 + \text{Var}\left[\hat{f}(x')\right] + \sigma^2 \tag{13}$$

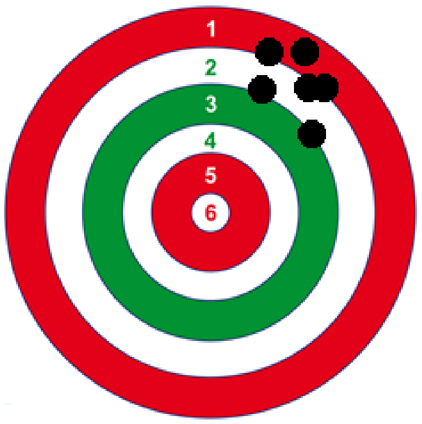
Where we remember that $\hat{f}(x')$ and ϵ are independent.

This show us that there are two important quantities, the **bias** and the **variance** that will affect our results and that we can control to some extent.

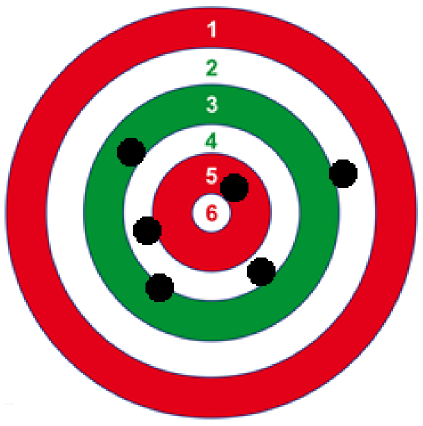
FIGURE 1.1 - A good model should have low bias and low variance



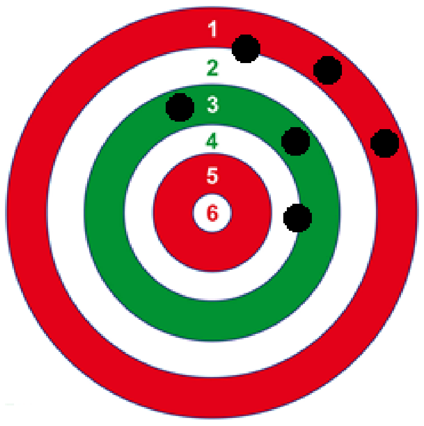
Low Bias and Low Variance



High Bias and Low Variance



Low Bias and High Variance

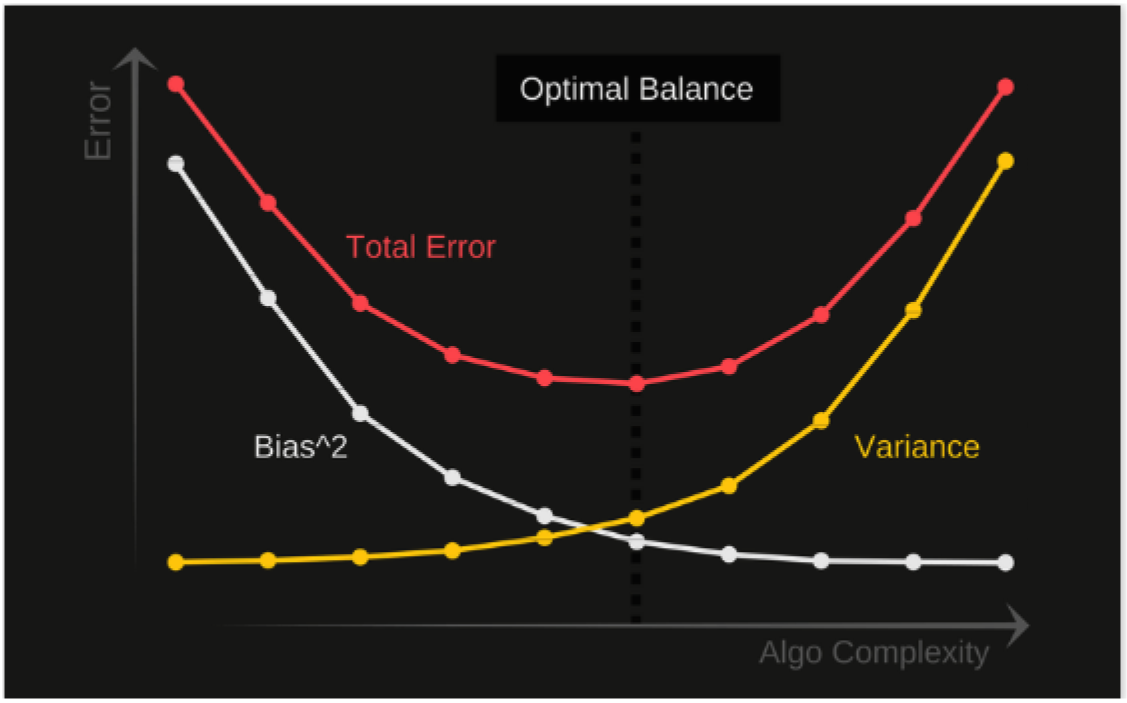


High Bias and High Variance

Bias is how far away the trained model is from the correct result on average. Where *on average* means over many goes at training the model using different data. And **Variance is a measure of the magnitude of that error**.

Unfortunately, we often find that there is a trade-off between bias and variance. As one is reduced, the other is increased. This is the matter of over- and under-fitting.

Overfitting is when we train our algorithm too well on training data, perhaps having too many parameters for fitting.



2.7 Regularization

2.7.1 Ridge Regression

Ridge regression is a regularization technique where we change the function that is to be minimize. Reduce magnitude of regression coefficients by choosing a parameter λ and minimizing

$$\frac{1}{2N}\sum_{n=1}^{N}\left[h_\theta\left(x^{(n)}\right)-y^{(n)}\right]^2+\lambda\sum_{n=1}^{N}b_i^2$$

This change has the effect of encouraging the model to keep the weights b_j as small as possibile. The Ridge regression should only be used for determining model parameters using the training set. Once the model parameters have been determined the penalty term should be removed for prediction.

[178]:
```
columns_titles = ["Salary","Age"]
df2=df1.reindex(columns=columns_titles)
df2
```

[178]:
```
   Salary  Age
0  135000   25
1  105000   27
2  105000   30
3  220000   35
4  300000   40
5  270000   45
6  265000   50
```

```
7  260000   55
8  240000   60
9  265000   65
```

[179]:
```
df2['Salary'] = df2['Salary']/1000
df2['Age2']=df2['Age']**2
df2['Age3']=df2['Age']**3
df2['Age4']=df2['Age']**4
df2['Age5']=df2['Age']**5
df2
```

[179]:
```
   Salary  Age  Age2    Age3      Age4        Age5
0   135.0   25   625   15625    390625     9765625
1   105.0   27   729   19683    531441    14348907
2   105.0   30   900   27000    810000    24300000
3   220.0   35  1225   42875   1500625    52521875
4   300.0   40  1600   64000   2560000   102400000
5   270.0   45  2025   91125   4100625   184528125
6   265.0   50  2500  125000   6250000   312500000
7   260.0   55  3025  166375   9150625   503284375
8   240.0   60  3600  216000  12960000   777600000
9   265.0   65  4225  274625  17850625  1160290625
```

We can compute the z-score in Pandas using the .mean() and std() methods.

[180]:
```
# apply the z-score method in Pandas using the .mean() and .std() methods
def z_score(df):
    # copy the dataframe
    df_std = df.copy()
    # apply the z-score method
    for column in df_std.columns:
        df_std[column] = (df_std[column] - df_std[column].mean()) /␣
 ↪df_std[column].std()

    return df_std

# call the z_score function
df2_standard = z_score(df2)
df2_standard['Salary'] = df2['Salary']
df2_standard
```

[180]:
```
   Salary       Age      Age2      Age3      Age4      Age5
0   135.0 -1.289948 -1.128109 -0.988322 -0.873562 -0.782128
1   105.0 -1.148195 -1.045510 -0.943059 -0.849996 -0.770351
2   105.0 -0.935566 -0.909699 -0.861444 -0.803378 -0.744782
3   220.0 -0.581185 -0.651577 -0.684372 -0.687799 -0.672266
4   300.0 -0.226804 -0.353745 -0.448740 -0.510508 -0.544103
```

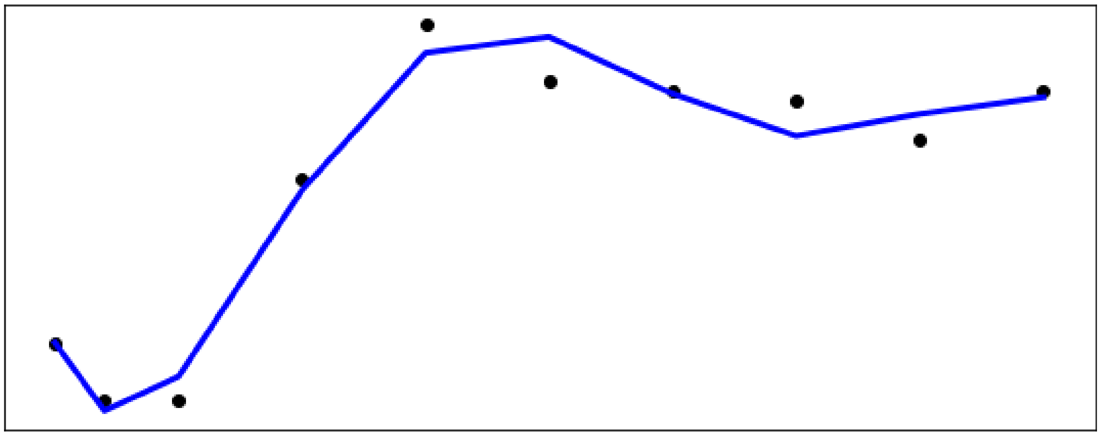
```
5   270.0  0.127577 -0.016202 -0.146184 -0.252677 -0.333075
6   265.0  0.481958  0.361052  0.231663  0.107030 -0.004250
7   260.0  0.836340  0.778017  0.693166  0.592463  0.485972
8   240.0  1.190721  1.234693  1.246690  1.229979  1.190828
9   265.0  1.545102  1.731080  1.900602  2.048447  2.174155
```

[181]:
```python
y = df2_standard['Salary']
X = df2_standard.drop('Salary',axis=1)
```

[182]:
```python
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score

lr = LinearRegression()
lr.fit(X, y)

y_pred = lr.predict(X)

# The coefficients
print('Coefficients: \n', lr.coef_)
# The mean squared error
print('Mean squared error: %.2f'
      % mean_squared_error(y, y_pred))
```

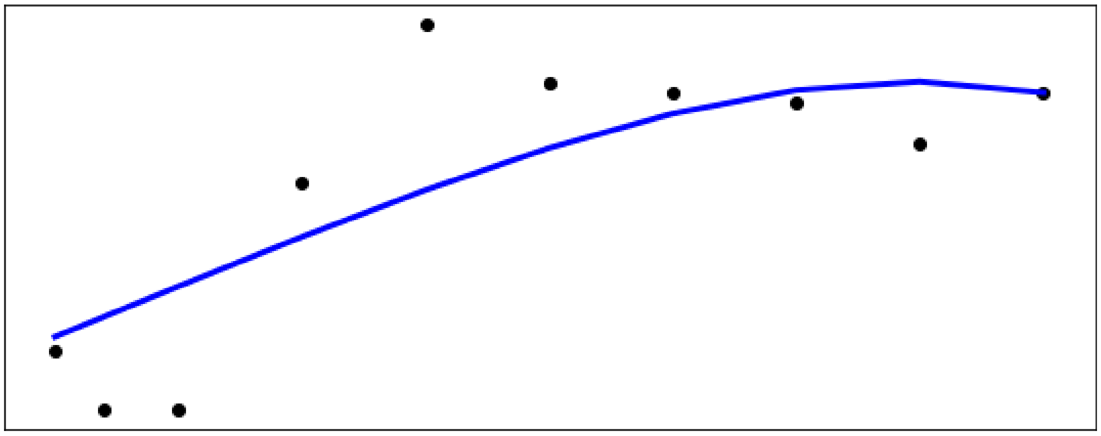
```
Coefficients:
 [ -32622.57240727  135402.73116519 -215493.11781297  155314.61367273
  -42558.76209732]
Mean squared error: 149.82
```

[183]:
```python
# Plot outputs
plt.scatter(X['Age'], y,  color='black')
plt.plot(X['Age'], y_pred, color='blue', linewidth=3)

plt.xticks(())
plt.yticks(())

plt.show()
```



[184]:
```
rr = Ridge(alpha=0.1, normalize=True)
# higher the alpha value, more restriction on the coefficients; low alpha >␣
↪more generalization,
# in this case linear and ridge regression resembles
rr.fit(X, y)

y_pred_r = rr.predict(X)
```

[185]:
```
# Plot outputs
plt.scatter(X['Age'], y,  color='black')
plt.plot(X['Age'], y_pred_r, color='blue', linewidth=3)

plt.xticks(())
plt.yticks(())

plt.show()
```



[85]:
```
# The coefficients
print('Coefficients: \n', rr.coef_)
# The mean squared error
print('Mean squared error: %.2f'
      % mean_squared_error(y, y_pred_r))
```

```
Coefficients:
 [ 59.22629315  29.03476514   3.29697763 -16.42544961 -30.17329953]
Mean squared error: 1615.07
```

2.7.2 Lasso Regression

Lasso is short for *Least Absolute Shrinkage and Selection Operator*. It is similar to ridge regression except we minimize

$$\frac{1}{2N}\sum_{n=1}^{N}\left[h_\theta\left(x^{(n)}\right)-y^{(n)}\right]^2+\lambda\sum_{n=1}^{N}|b_n|$$

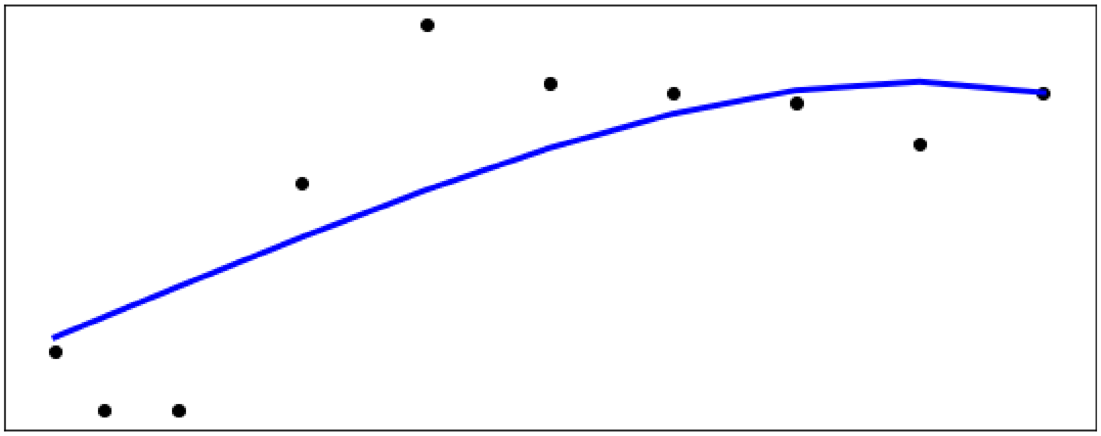
This function cannot be minimized analytically and so a variation on the gradient descent algorithm must be used. Lasso regression also has the effect of simplifying the model. It does this by setting the weights of unimportant features to zero. When there are a large number of features, Lasso can identify a relatively small subset of the features that form a good predictive model.

[86]:
```
from sklearn.linear_model import Lasso

lsr = Lasso(alpha=.02, normalize=True, max_iter=1000000)
# higher the alpha value, more restriction on the coefficients; low alpha >␣
 ↪more generalization,
# in this case linear and ridge regression resembles
lsr.fit(X, y)

y_pred_lsr = rr.predict(X)
```

[87]:
```
# The coefficients
print('Coefficients: \n', lsr.coef_)
# The mean squared error
print('Mean squared error: %.2f'
      % mean_squared_error(y, y_pred_lsr))
```

```
Coefficients:
 [ 344.99709034   -0.         -471.80600937   -0.          183.42041303]
Mean squared error: 1615.07
```

[88]:
```
# Plot outputs
plt.scatter(X['Age'], y,  color='black')
plt.plot(X['Age'], y_pred_lsr, color='blue', linewidth=3)
plt.xticks(())
plt.yticks(())

plt.show()
```



2.7.3 Elastic Net Regression

Middle ground between Ridge and Lasso. Minimize

$$\frac{1}{2N}\sum_{n=1}^{N}\left[h_\theta\left(x^{(n)}\right)-y^{(n)}\right]^2+\lambda_1\sum_{n=1}^{N}b_n^2+\lambda_2\sum_{n=1}^{N}|b_n|$$

In Lasso some weights are reduced to zero but others may be quite large. In Ridge, weights are small in magnitude but they are not reduced to zero. The idea underlying Elastic Net is that we may be able to get the best of both by making some weights zero while reducing the magnitude of the others.

[89]:
```
from sklearn.linear_model import ElasticNet

# define model
model = ElasticNet(alpha=1.0, l1_ratio=0.5)
```

3 What is Unsupervised Learning

Unsupervised learning uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention. Its ability to discover similarities and differences in information make it the ideal solution for exploratory data analysis, cross-selling strategies, customer segmentation, and image recognition.

3.1 *k*-Means Clustering

k-means clustering is one of the simplest and popular unsupervised machine learning algorithms. Typically, unsupervised algorithms make inferences from datasets using only input vectors without referring to known, or labelled, outcomes. The objective of K-means is simple: group similar data points together and discover underlying patterns. To achieve this objective, K-means looks for a fixed number (k) of clusters in a dataset.

A cluster refers to a collection of data points aggregated together because of certain similarities. You'll define a target number k, which refers to the number of centroids you need in the dataset. A centroid is the imaginary or real location representing the center of the cluster.

Every data point is allocated to each of the clusters through reducing the in-cluster sum of squares. In other words, the K-means algorithm identifies k number of centroids, and then allocates every data point to the **nearest** cluster, while keeping the centroids as small as possible.

The 'means' in the K-means refers to averaging of the data; that is, finding the centroid.

By Chire - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=59409335

3.1.1 How the K-means algorithm works

To process the learning data, the K-means algorithm in data mining starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids. It halts creating and optimizing clusters when either:

- The centroids have stabilized — there is no change in their values because the clustering has been successful.
- The defined number of iterations has been achieved.

3.1.2 A Distance Measure

For clustering we need a distance measure. The simplest distance measure is the Euclidean Distance measure:

$$Distance = \sqrt{(x_B - x_B)^2 + (y_B - y_A)^2}$$

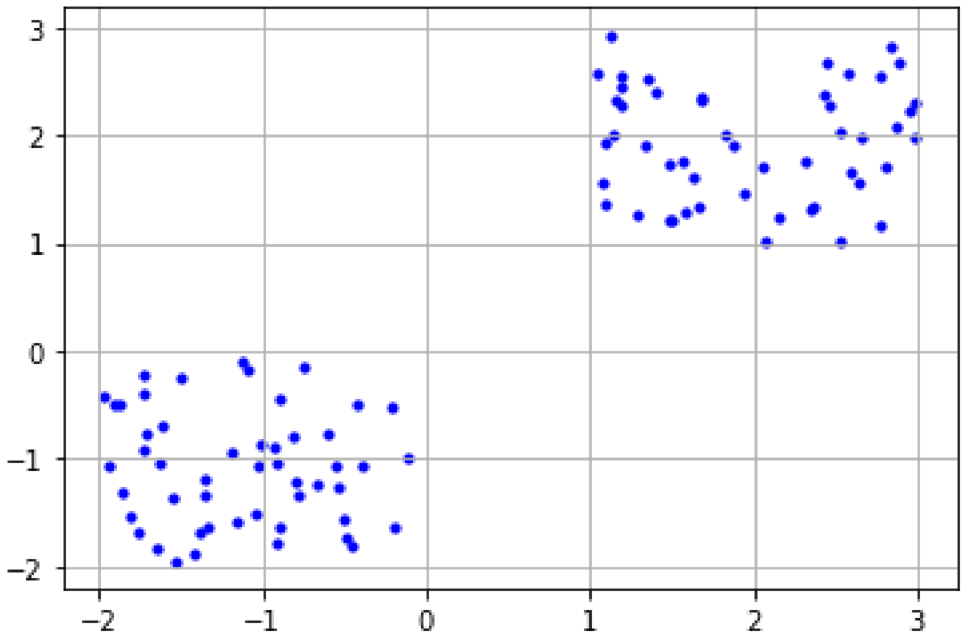
3.1.3 K-means algorithm example problem

Let's see the steps on how the K-means machine learning algorithm works using the Python programming language. We'll use the Scikit-learn library and some random data to illustrate a K-means clustering simple explanation.

[90]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
%matplotlib inline
```

Here is the code for generating some random data in a two-dimensional space:

[91]:
```
X  =    -2 * np.random.rand(100,2)
X1 = 1 + 2 * np.random.rand(50,2)
X[50:100, :] = X1
plt.scatter(X[ : , 0], X[ :, 1], s = 10, c = 'b')
plt.grid()
plt.show()
```



This give us two sets approximately centered about (-1,-1) and (2, 2). We'll use some of the available functions in the Scikit-learn library to process the randomly generated data.

[92]:
```
from sklearn.cluster import KMeans
Kmean = KMeans(n_clusters=2)
Kmean.fit(X)
```

[92]:
```
KMeans(n_clusters=2)
```

In this case, we arbitrarily gave k (n_clusters) an arbitrary value of two. Here is the output of the K-means parameters we get if we run the code:

[93]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
 n_clusters=2, n_init=10, n_jobs=1, precompute_distances='auto',
 random_state=None, tol=0.0001, verbose=0)
```

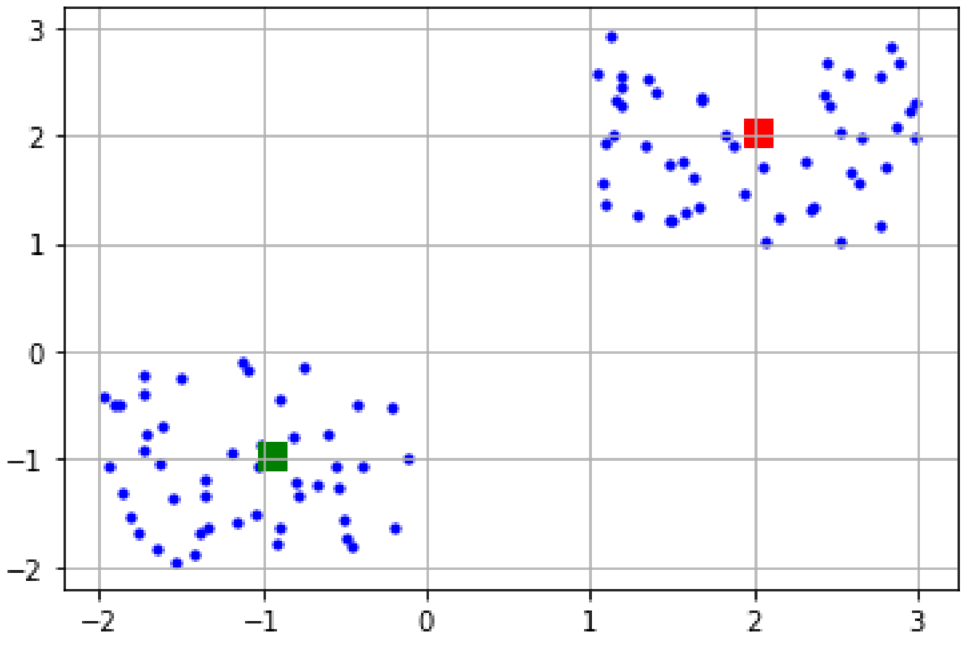
[93]:
```
KMeans(n_clusters=2, n_jobs=1, precompute_distances='auto')
```

[94]:
```
Kmean.cluster_centers_
```

```
[94]: array([[ 1.9733796 ,  1.92595225],
             [-1.13860893, -1.07919834]])
```

Let's display the cluster centroids

```
[95]: plt.scatter(X[ : , 0], X[ : , 1], s =10, c='b')
      plt.scatter(-0.94665068, -0.97138368, s=100, c='g', marker='s')
      plt.scatter( 2.01559419,  2.02597093, s=100, c='r', marker='s')
      plt.grid()
      plt.show()
```



Here is the code for getting the labels property of the K-means clustering example dataset; that is, how the data points are categorized into the two clusters.

```
[96]: Kmean.labels_
```

```
[96]: array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
             1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

As you can see above, 50 data points belong to the 0 cluster while the rest belong to the 1 cluster.

For example, let's use the code below for predicting the cluster of a data point:

```
[97]: sample_test=np.array([-3.0,-3.0])
      second_test=sample_test.reshape(1, -1)
      Kmean.predict(second_test)
```

```
[97]: array([1])
```

3.2 A Country Risk Example

```
[98]: # loading packages

      import os

      import pandas as pd
      import numpy as np

      # plotting packages
      %matplotlib inline
      from mpl_toolkits.mplot3d import Axes3D
      import matplotlib.pyplot as plt
      import matplotlib.cm as cm
      import matplotlib.colors as clrs

      # Kmeans algorithm from scikit-learn
      from sklearn.cluster import KMeans
      from sklearn.metrics import silhouette_samples, silhouette_score
```

3.2.1 The Country Risk Dataset (J. C. Hull, 2019, Chapter 2)

Consider the problem of understanding the risk of countries for foreign investment. Among the features that can be used for this are:

- GDP growth rate (IMF)
- Corruption index (Transparency international)
- Peace index (Institute for Economics and Peace)
- Legal Risk Index (Property Rights Association)

Values for each of the features for 122 countries are found in the `countryriskdata.csv` (available here)

```
[99]: # load raw data
      DATA_FOLDER = './data'
      raw = pd.read_csv(os.path.join(DATA_FOLDER, 'countryriskdata.csv'))

      # check the raw data
      print("Size of the dataset (row, col): ", raw.shape)
      print("\nFirst 5 rows\n", raw.head(n=5))
```

```
Size of the dataset (row, col):  (122, 6)
```

```
First 5 rows
      Country Abbrev  Corruption  Peace  Legal  GDP Growth
0     Albania     AL          39  1.867  3.822       3.403
1     Algeria     DZ          34  2.213  4.160       4.202
2   Argentina     AR          36  1.957  4.568      -2.298
3     Armenia     AM          33  2.218  4.126       0.208
4   Australia     AU          79  1.465  8.244       2.471
```

The GDP growth rate (%) is typically a positive or negative number with a magnitude less than 10. The corruption index is on a scale from 0 (highly corrupt) to 100 (no corruption). The peace index is on a scale from 1 (very peaceful) to 5 (not at all peaceful). The legal risk index runs from 0 to 10 (with high values being favorable).

3.2.2 Simple exploratory analysis

Print summary statistics

Note that all features have quite different variances, and Corruption and Legal are highly correlated.

```
[100]: # print summary statistics
       print("\nSummary statistics\n", raw.describe())
       print("\nCorrelation matrix\n", raw.corr())
```

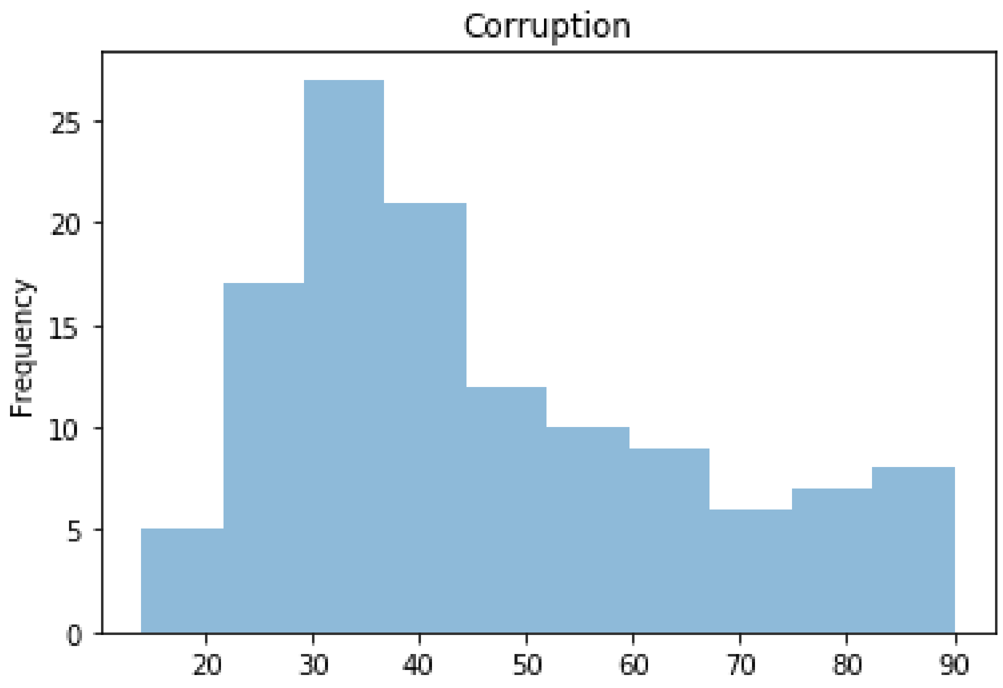
```
Summary statistics
        Corruption       Peace       Legal  GDP Growth
count  122.000000  122.000000  122.000000  122.000000
mean    46.237705    2.003730    5.598861    2.372566
std     19.126397    0.447826    1.487328    3.241424
min     14.000000    1.192000    2.728000  -18.000000
25%     31.250000    1.684750    4.571750    1.432250
50%     40.000000    1.969000    5.274000    2.496000
75%     58.750000    2.280500    6.476750    4.080000
max     90.000000    3.399000    8.633000    7.958000

Correlation matrix
             Corruption     Peace     Legal  GDP Growth
Corruption    1.000000 -0.700477  0.923589    0.102513
Peace        -0.700477  1.000000 -0.651961   -0.199855
Legal         0.923589 -0.651961  1.000000    0.123440
GDP Growth    0.102513 -0.199855  0.123440    1.000000
```

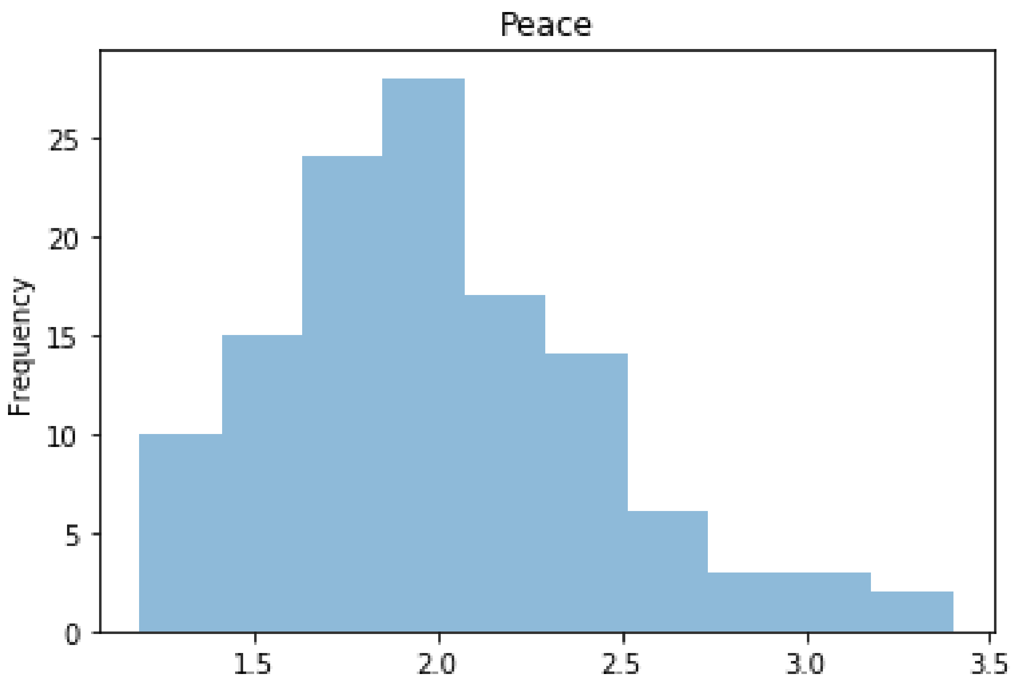
Plot histogram

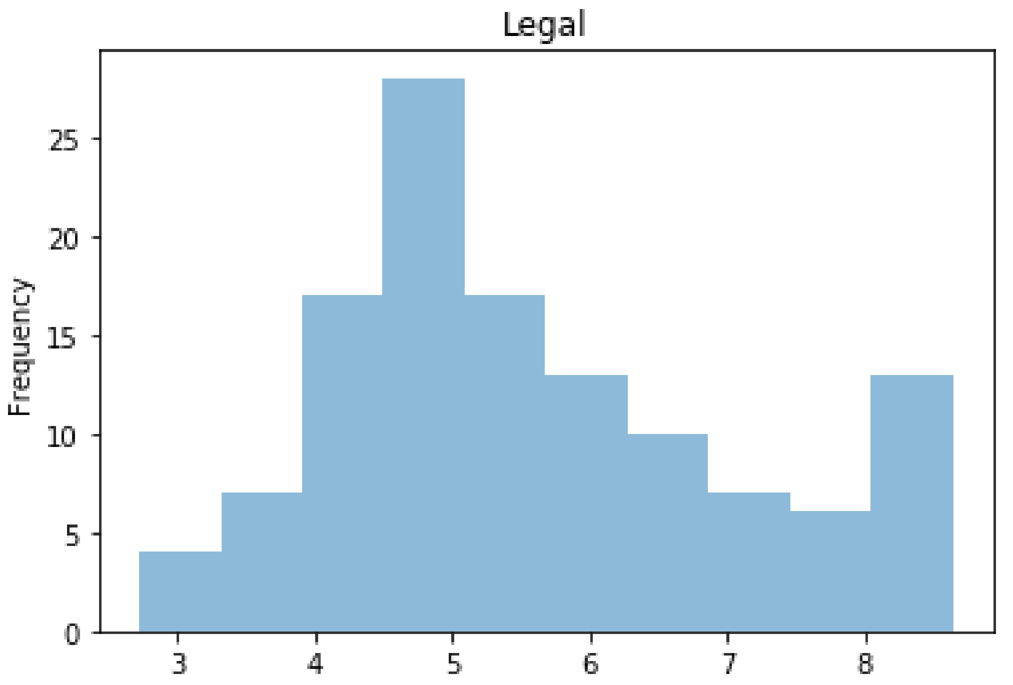
Note that distributions for GDP Growth is quite skewed.

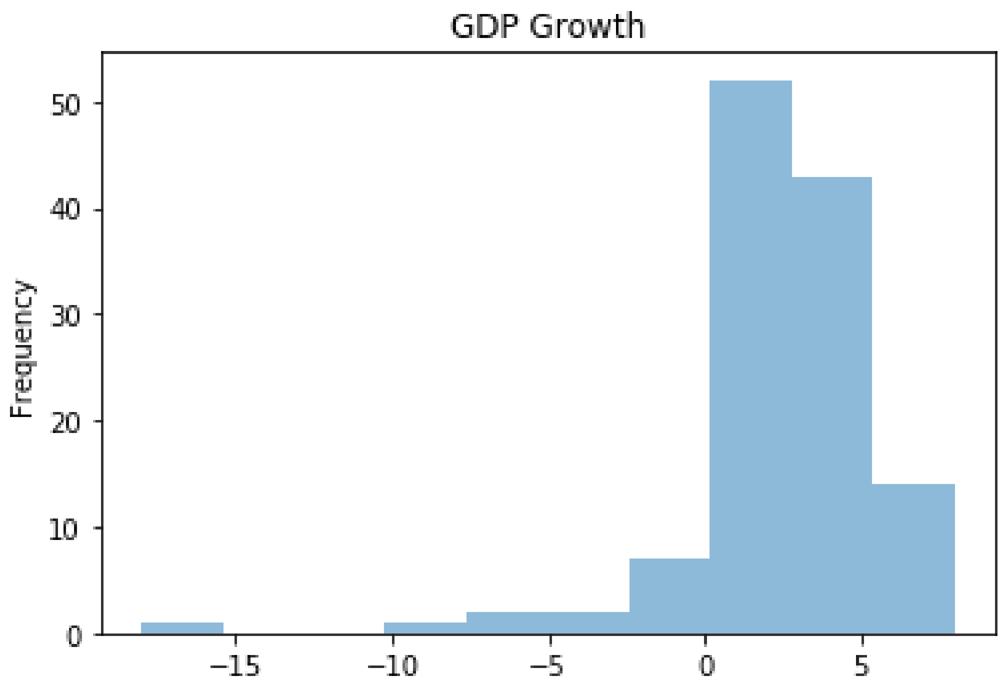
```
[101]: # plot histograms
       plt.figure(1)
       raw['Corruption'].plot(kind = 'hist', title = 'Corruption', alpha = 0.5)
```

```
plt.figure(2)
raw['Peace'].plot(kind = 'hist', title = 'Peace', alpha = 0.5)

plt.figure(3)
raw['Legal'].plot(kind = 'hist', title = 'Legal', alpha = 0.5)

plt.figure(4)
raw['GDP Growth'].plot(kind = 'hist', title = 'GDP Growth', alpha = 0.5)

plt.show()
```




Peace
Frequency
0
5
10
15
20
25
1.5
2.0
2.5
3.0
3.5


Legal
Frequency
0
5
10
15
20
25
3
4
5
6
7
8



3.2.3 K means cluster

Pick features & normalization

Since Corruption and Legal are highly correlated, we drop the Corruption variable, i.e., we pick three features for this analysis, Peace, Legal and GDP Grwoth. Let's normalize all the features, effectively making them equally weighted.

Ref. Feature normalization.

[102]:
```
X = raw[['Peace', 'Legal', 'GDP Growth']]
X = (X - X.mean()) / X.std()
print(X.head(5))
```

```
      Peace     Legal  GDP Growth
0 -0.305319 -1.194666    0.317896
1  0.467304 -0.967413    0.564392
2 -0.104348 -0.693096   -1.440899
3  0.478469 -0.990273   -0.667782
4 -1.202990  1.778450    0.030368
```

3.2.4 Perform elbow method

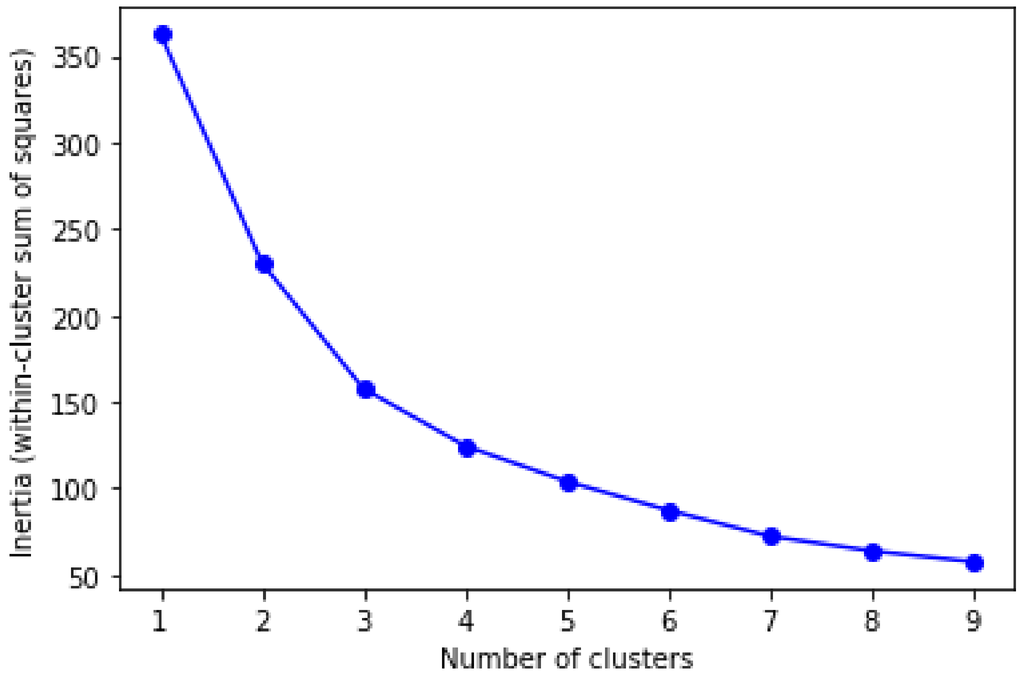
In cluster analysis, the elbow method is a heuristic used in determining the number of clusters in a data set. The method consists of plotting the explained variation as a function of the number of clusters, and picking the elbow of the curve as the number of clusters to use. The same method can be used to choose the number of parameters in other data-driven models, such as the number of principal components to describe a data set.

For example in the following picture k=4 is suggested

In our case, the marginal gain of adding one cluster dropped quite a bit from k=3 to k=4. We will choose k=3 (not a clear cut though).

Ref. Determining the number of clusters in a dataset.

[103]:
```
# https://stackoverflow.com/questions/41540751/
 ↪sklearn-kmeans-equivalent-of-elbow-method

Ks = range(1, 10)
inertia = [KMeans(i).fit(X).inertia_ for i in Ks]

fig = plt.figure()
plt.plot(Ks, inertia, '-bo')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia (within-cluster sum of squares)')
plt.show()
```



k-means with k=3

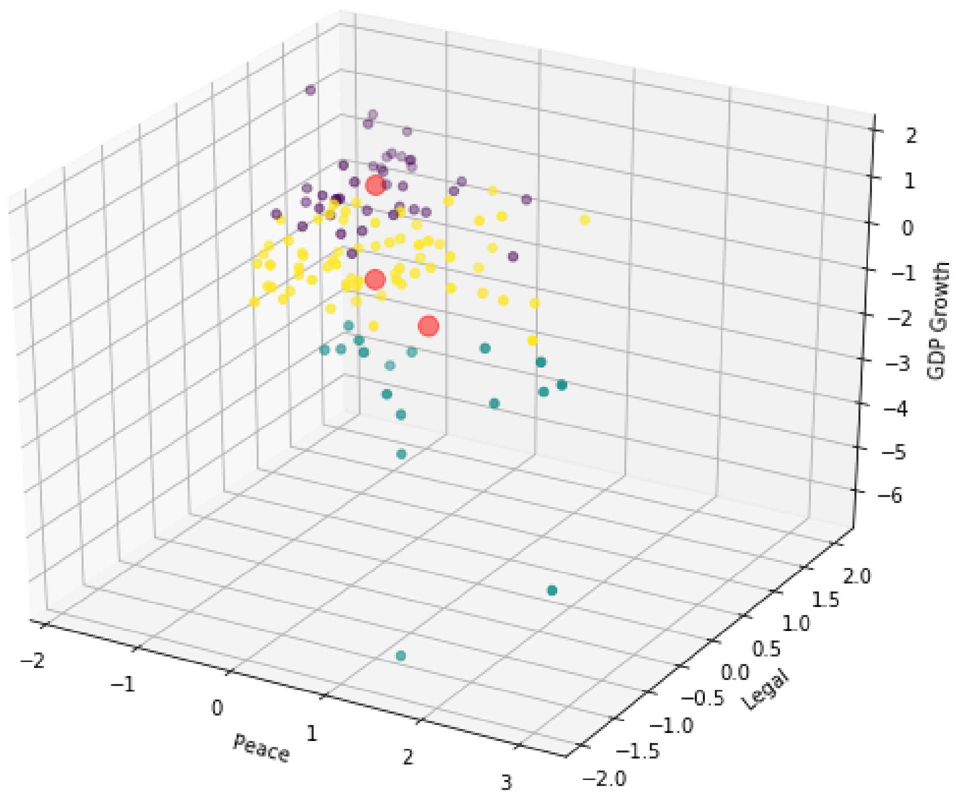
[104]:
```
k = 3
kmeans = KMeans(n_clusters=k, random_state=0)
kmeans.fit(X)
```

```
# print inertia & cluster center
print("inertia for k=2 is", kmeans.inertia_)
print("cluster centers: ", kmeans.cluster_centers_)

# take a quick look at the result
y = kmeans.labels_
print("cluster labels: ", y)
```

```
inertia for k=2 is 157.551489241025
cluster centers:  [[-0.92810589  1.16641329 -0.01445833]
 [ 1.21562552 -1.01677118 -1.61496953]
 [ 0.25320926 -0.45186802  0.43127408]]
cluster labels:  [2 2 1 1 0 0 1 2 2 0 2 2 2 0 1 2 1 2 0 1 0 2 2 0 2 2 0 1 0 2 1
2 2 0 2 0 0
 2 2 0 2 2 2 2 0 0 2 2 2 0 2 0 2 0 2 2 2 0 2 2 1 1 0 2 2 0 2 2 0 2 2 2 2 2
 2 0 0 2 1 0 2 2 2 2 2 2 0 0 0 2 1 2 2 2 2 2 0 0 0 0 0 2 0 0 0 2 2 2 1 2 2
 2 1 0 0 0 0 1 2 1 2 1]
```

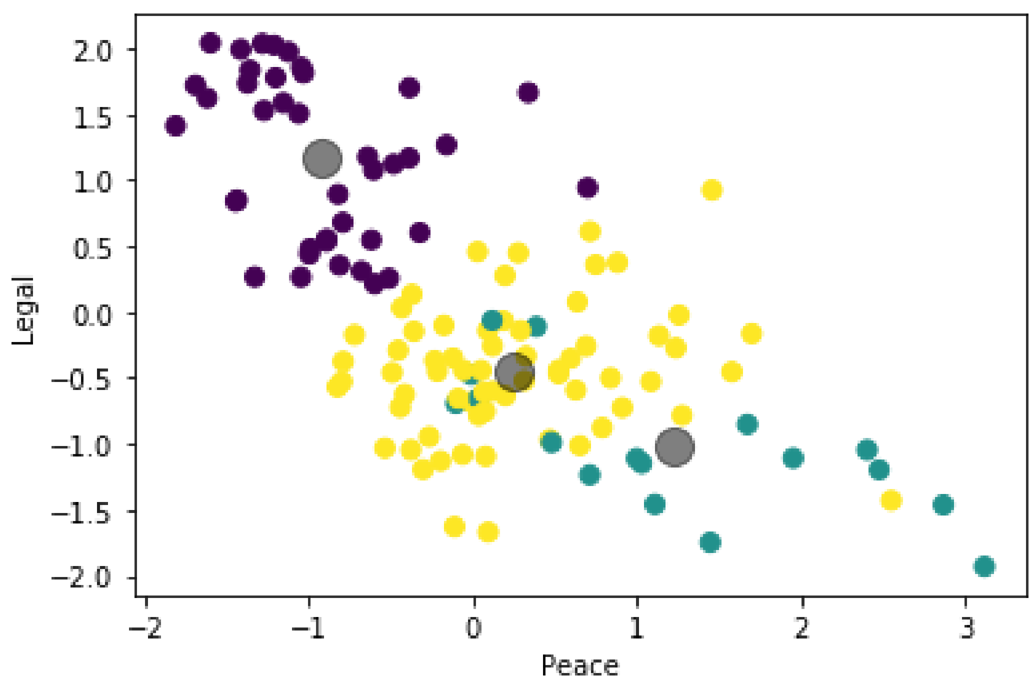
Visualize the result (3D plot)

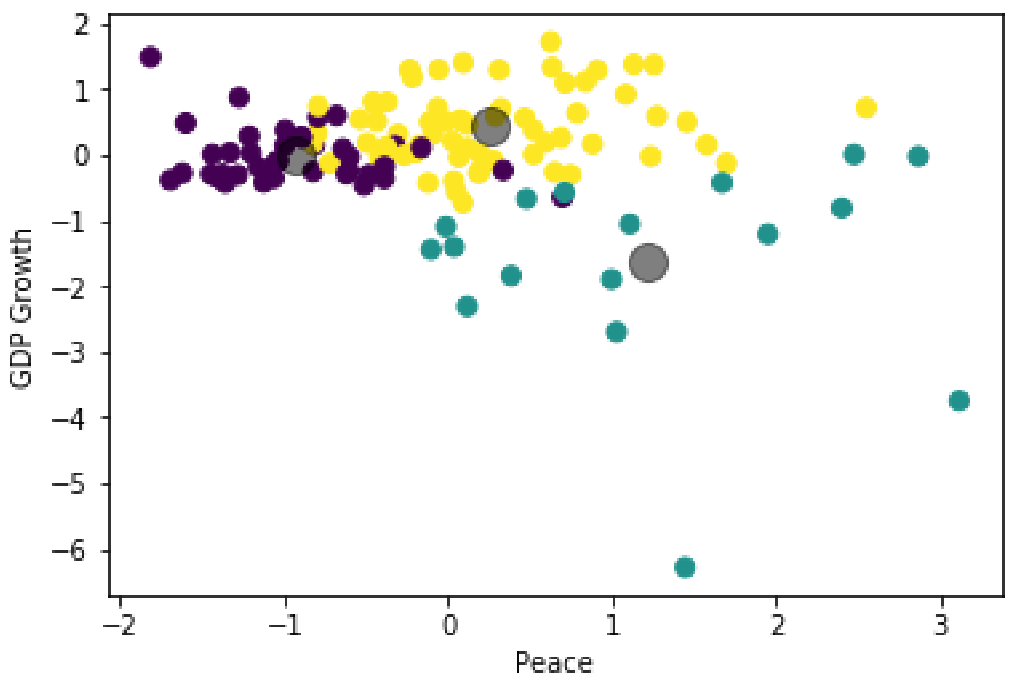
[105]:
```
# set up the color
norm = clrs.Normalize(vmin=0.,vmax=y.max())
cmap = cm.viridis

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(X.iloc[:,0], X.iloc[:,1], X.iloc[:,2], c=cmap(norm(y)), marker='o')

centers = kmeans.cluster_centers_
ax.scatter(centers[:, 0], centers[:, 1], c='red', s=100, alpha=0.5, marker='o')

ax.set_xlabel('Peace')
ax.set_ylabel('Legal')
ax.set_zlabel('GDP Growth')

plt.show()
```

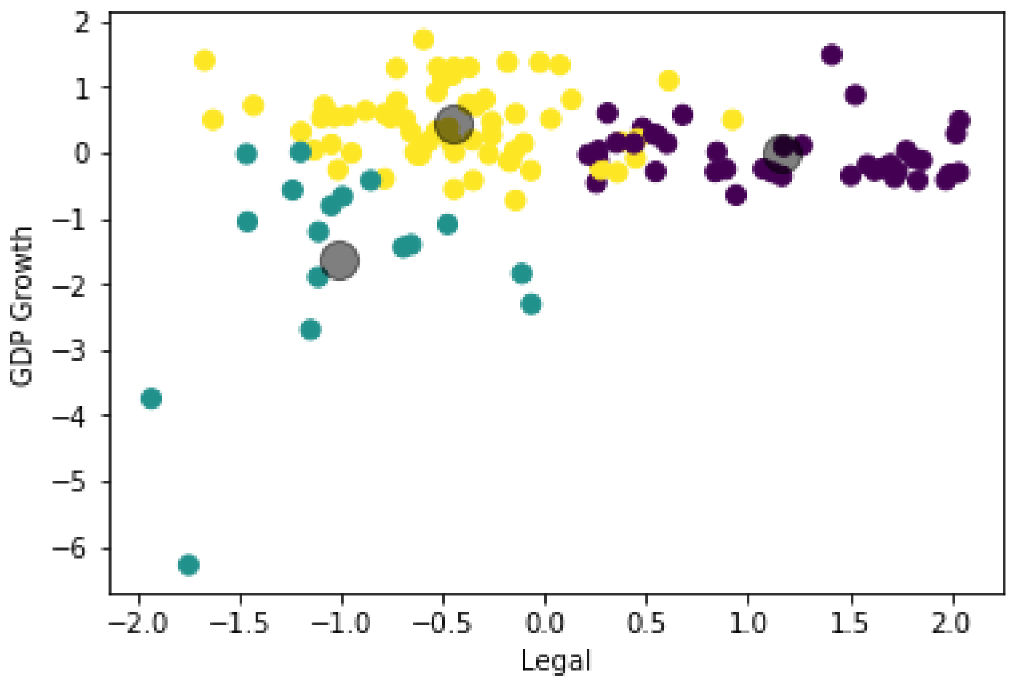


Visualize the result (3 2D plots)

[106]:
```
%matplotlib inline
import matplotlib.pyplot as plt

figs = [(0, 1), (0, 2), (1, 2)]
labels = ['Peace', 'Legal', 'GDP Growth']

for i in range(3):
    fig = plt.figure(i)
    plt.scatter(X.iloc[:,figs[i][0]], X.iloc[:,figs[i][1]], c=cmap(norm(y)),␣
 ↪s=50)
    plt.scatter(centers[:, figs[i][0]], centers[:, figs[i][1]], c='black',␣
 ↪s=200, alpha=0.5)
    plt.xlabel(labels[figs[i][0]])
    plt.ylabel(labels[figs[i][1]])

plt.show()
```


Legal
Peace
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2
−1
0
1
2
3

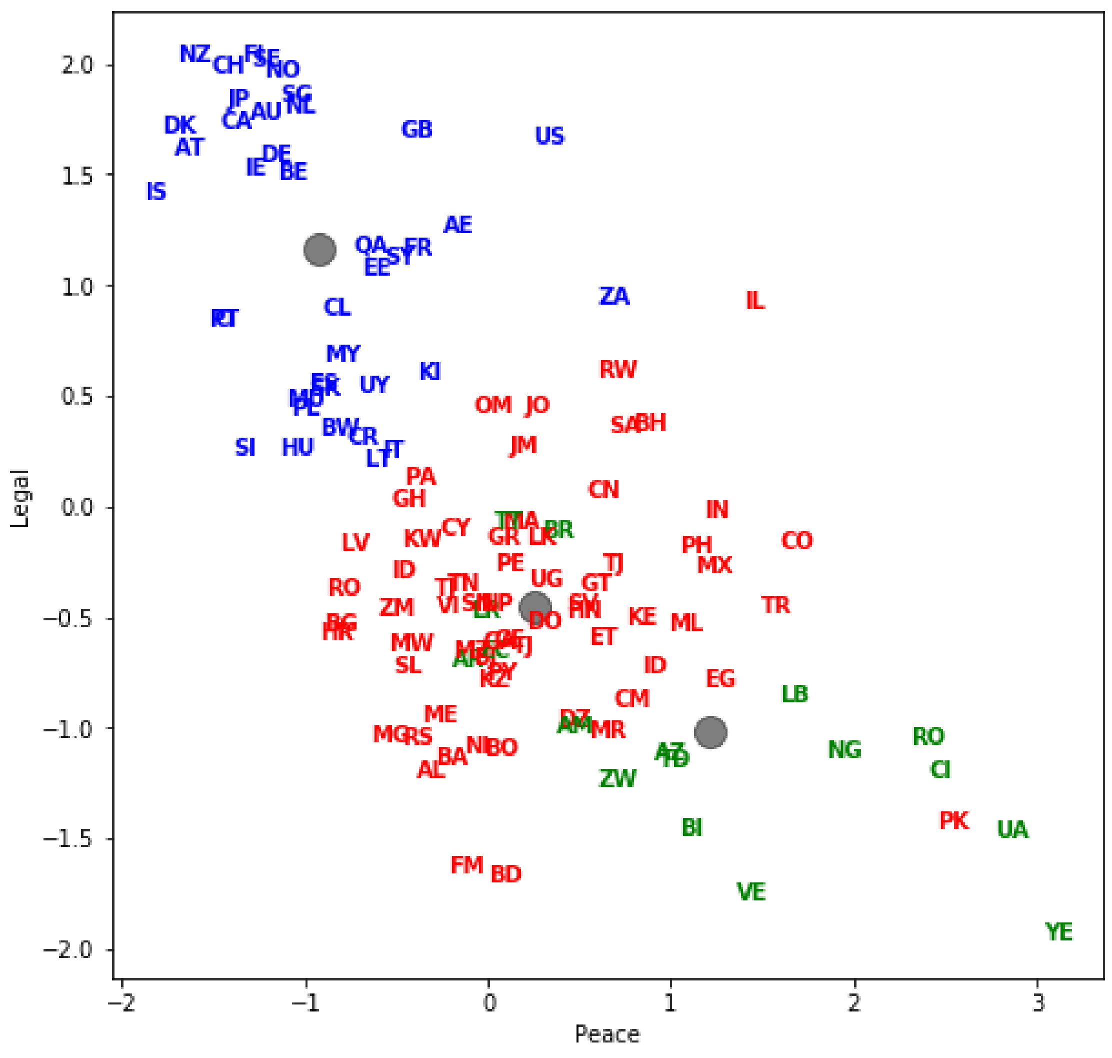

GDP Growth
Peace
2
1
0
−1
−2
−3
−4
−5
−6
−2
−1
0
1
2
3

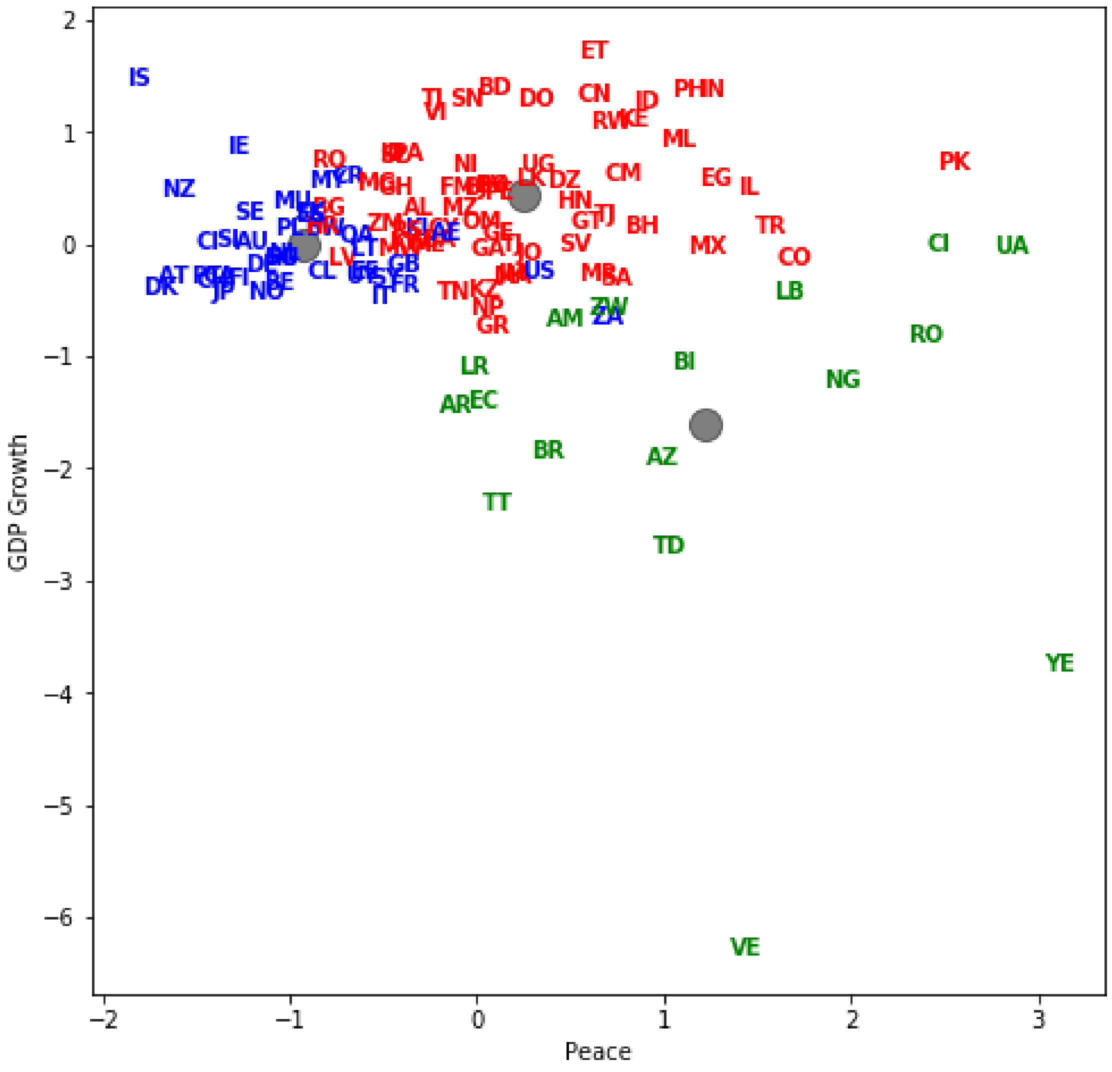


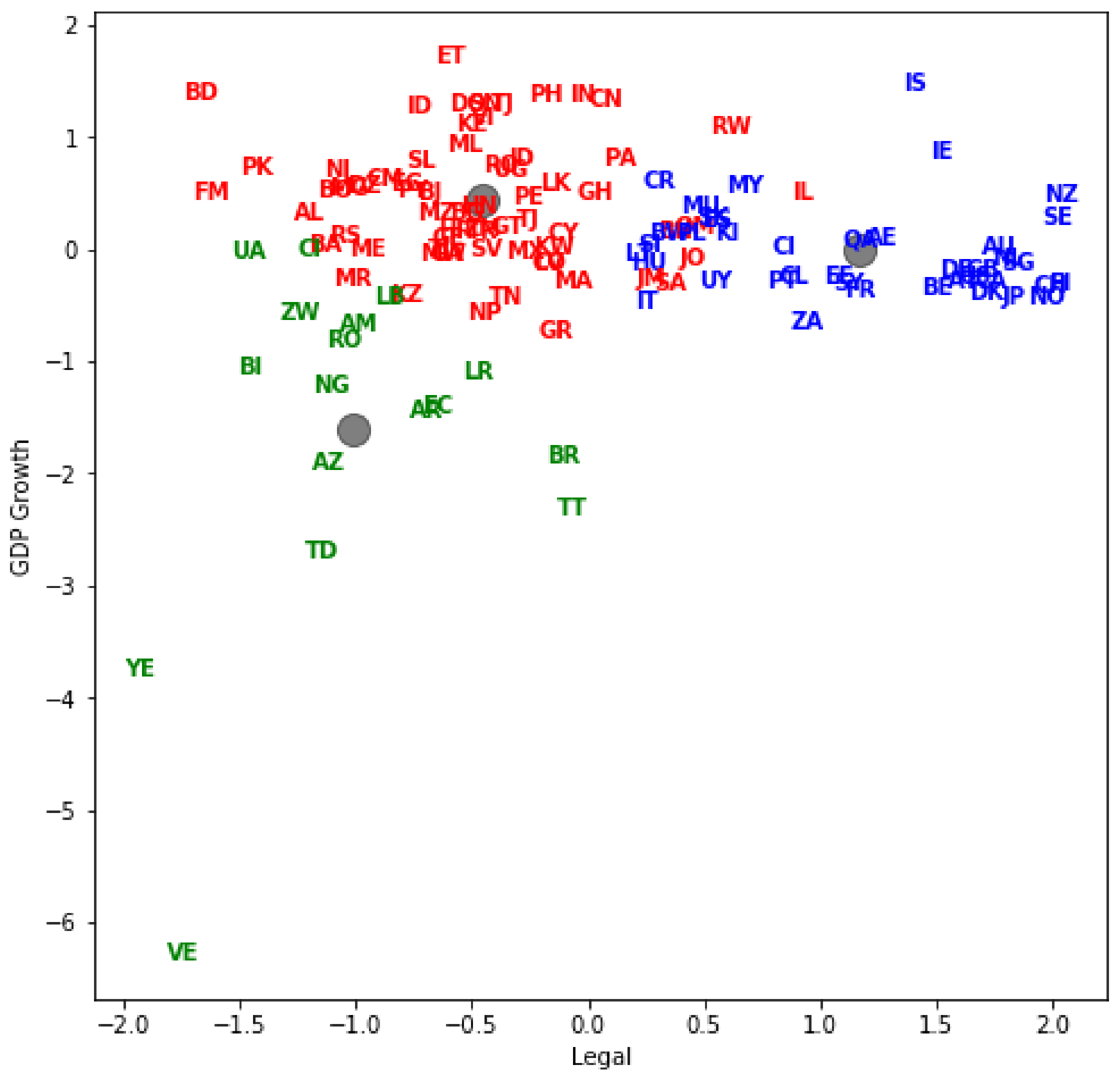
Visualize the result (3 2D plots)

plot country abbreviations instead of dots.

[107]:
```
%matplotlib inline
import matplotlib.pyplot as plt

figs = [(0, 1), (0, 2), (1, 2)]
labels = ['Peace', 'Legal', 'GDP Growth']
colors = ['blue','green', 'red']

for i in range(3):
    fig = plt.figure(i, figsize=(8, 8))
    x_1 = figs[i][0]
    x_2 = figs[i][1]
    plt.scatter(X.iloc[:, x_1], X.iloc[:, x_2], c=y, s=0, alpha=0)
    plt.scatter(centers[:, x_1], centers[:, x_2], c='black', s=200, alpha=0.5)
    for j in range(X.shape[0]):
        plt.text(X.iloc[j, x_1], X.iloc[j, x_2], raw['Abbrev'].iloc[j],
                 color=colors[y[j]], weight='semibold', horizontalalignment =␣
↪'center', verticalalignment = 'center')
    plt.xlabel(labels[x_1])
    plt.ylabel(labels[x_2])

plt.show()
```


Legal
Peace
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2
−1
0
1
2
3
NZ
CH
FI
SE
NO
JP
SG
NL
DK
CA
AU
AT
GB
US
DE
IE
BE
IS
AE
QA
SY
FR
EE
ZA
IL
CL
MY
KI
RW
UY
OM
JO
SA
BH
BW
CR
SI
HU
LT
IT
JM
PA
CN
GH
IN
LV
KW
CY
GR
LK
BR
PH
CO
MX
ID
PE
TJ
RO
TN
UG
GT
ZM
VI
KE
ML
TR
DO
ET
MW
SL
ID
EG
LB
CM
ME
MR
MG
RS
NI
BO
BA
AL
NG
RO
CI
ZW
BI
PK
UA
FM
BD
VE
YE


GDP Growth
Peace
IS
IE
NZ
SE
ET
PK
CI
UA
YE
VE
TD
TT
BR
AZ
NG
RO
LB
MX
TR
CO
US



3.2.5 List the result

```
[108]: result = pd.DataFrame({'Country':raw['Country'], 'Abbrev':raw['Abbrev'],␣
        ↪'Label':y})
       with pd.option_context('display.max_rows', None, 'display.max_columns', 3):
           print(result.sort_values('Label'))
```

```
                         Country Abbrev  Label
23                    Costa Rica     CR      0
88                         Qatar     QA      0
26                Czech Republic     CI      0
87                      Portugal     PT      0
28                       Denmark     DK      0
86                        Poland     PL      0
```

79	Norway	NO	0
76	New Zealand	NZ	0
33	Estonia	EE	0
75	Netherlands	NL	0
35	Finland	FI	0
36	France	FR	0
68	Mauritius	MU	0
65	Malaysia	MY	0
39	Germany	DE	0
62	Lithuania	LT	0
57	Korea (South)	KI	0
53	Japan	JP	0
44	Hungary	HU	0
45	Iceland	IS	0
51	Italy	IT	0
96	Singapore	SG	0
49	Ireland	IE	0
20	Chile	CL	0
115	United States	US	0
114	United Kingdom	GB	0
113	United Arab Emirates	AE	0
9	Belgium	BE	0
97	Slovakia	SK	0
116	Uruguay	UY	0
104	Taiwan	SY	0
5	Austria	AT	0
13	Botswana	BW	0
103	Switzerland	CH	0
102	Sweden	SE	0
100	Spain	ES	0
99	South Africa	ZA	0
18	Canada	CA	0
98	Slovenia	SI	0
4	Australia	AU	0
119	Yemen	YE	1
117	Venezuela	VE	1
112	Ukraine	UA	1
108	Trinidad and Tobago	TT	1
78	Nigeria	NG	1
90	Russia	RO	1
61	Liberia	LR	1
60	Lebanon	LB	1
121	Zimbabwe	ZW	1
14	Brazil	BR	1
19	Chad	TD	1
27	Democratic Republic of Congo	CI	1
30	Ecuador	EC	1
6	Azerbaijan	AZ	1

16	Burundi	BI	1
2	Argentina	AR	1
3	Armenia	AM	1
24	Croatia	HR	2
91	Rwanda	RW	2
89	Romania	RO	2
93	Senegal	SN	2
94	Serbia	RS	2
95	Sierra Leone	SL	2
22	Colombia	CO	2
21	China	CN	2
25	Cyprus	CY	2
17	Cameroon	CM	2
92	Saudi Arabia	SA	2
47	Indonesia	ID	2
15	Bulgaria	BG	2
12	Bosnia and Herzegovina	BA	2
105	Tanzania	TJ	2
106	Thailand	TJ	2
107	The FYR of Macedonia	TJ	2
11	Bolivia	BO	2
109	Tunisia	TN	2
110	Turkey	TR	2
111	Uganda	UG	2
10	Benin	BJ	2
8	Bangladesh	BD	2
7	Bahrain	BH	2
118	Vietnam	VI	2
1	Algeria	DZ	2
101	Sri Lanka	LK	2
29	Dominican Republic	DO	2
85	Philippines	PH	2
84	Peru	PE	2
63	Madagascar	MG	2
40	Ghana	GH	2
41	Greece	GR	2
120	Zambia	ZM	2
59	Latvia	LV	2
58	Kuwait	KW	2
64	Malawi	MW	2
42	Guatemala	GT	2
55	Kazakhstan	KZ	2
54	Jordan	JO	2
43	Honduras	HN	2
52	Jamaica	JM	2
46	India	IN	2
50	Israel	IL	2
56	Kenya	KE	2

```
38                     Georgia    GE      2
66                        Mali    ML      2
67                  Mauritania    MR      2
83                    Paraguay    PY      2
82                      Panama    PA      2
81                    Pakistan    PK      2
80                        Oman    OM      2
31                       Egypt    EG      2
77                   Nicaragua    NI      2
32                 El Salvador    SV      2
34                    Ethiopia    ET      2
74                       Nepal    NP      2
73                  Mozambique    MZ      2
72                     Morocco    MA      2
71                  Montenegro    ME      2
70                     Moldova    FM      2
69                      Mexico    MX      2
37                       Gabon    GA      2
48                        Iran    ID      2
0                      Albania    AL      2
```

3.2.6 Silhouette Analysis

For each observation i calculate $a(i)$, the average distance from other observations in its cluster, and $b(i)$, the average distance from observations in the closest other cluster. The silhouette score for observation i, $s(i)$, is defined as

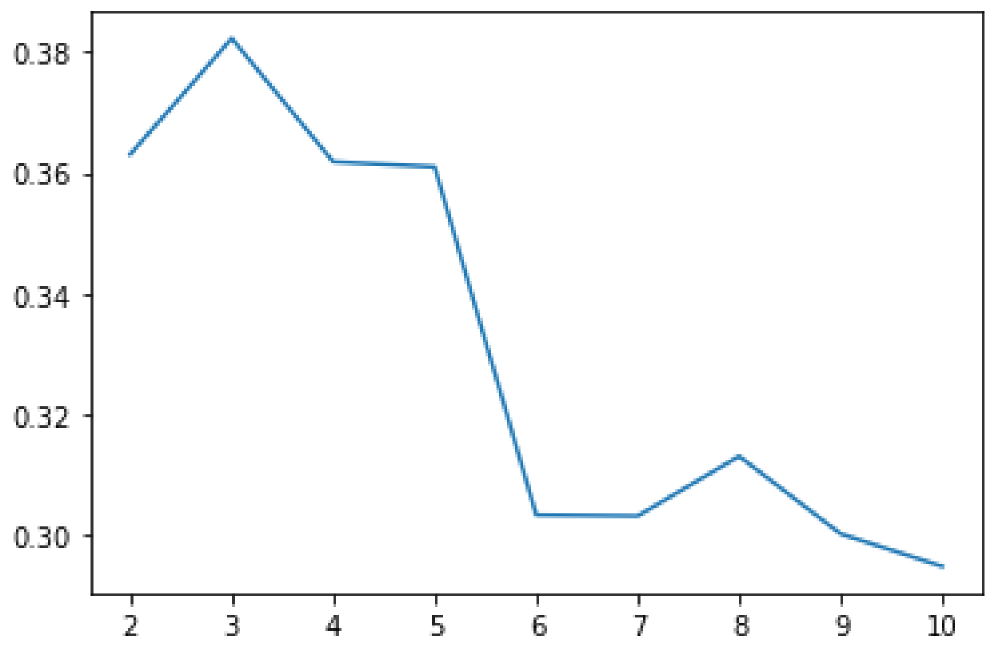
$$s(i) = \frac{b(i) - a(i)}{\max[a(i), b(i)]} \tag{14}$$

Choose the number of clusters that maximizes the average silhouette score across all observations

[109]:
```
# Silhouette Analysis
range_n_clusters = [2,3,4,5,6,7,8,9,10]
silhouette       = []
for n_clusters in range_n_clusters:
    clusterer=KMeans(n_clusters=n_clusters, random_state=0)
    cluster_labels=clusterer.fit_predict(X)
    silhouette_avg=silhouette_score(X,cluster_labels)
    silhouette.append(silhouette_avg)
    #print("For n_clusters=", n_clusters,
    #       "The average silhouette_score is :", silhouette_avg)
```

[110]:
```
plt.plot(range_n_clusters, silhouette)
```

[110]:
```
[<matplotlib.lines.Line2D at 0x22d12b2ca48>]
```



4 Introduction to Supervised Learning

4.1 Linear Regression

A linear model makes a prediction by simply computing a weighted sum of the input features, plus a constant called the *bias* term (also called the *intercept* term):

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \tag{15}$$

where:

- $\hat{y}$ is the predicted value;
- n is the number of features;
- x_i is the i^{th} feature value;
- θ_j is the j^{th} model parameter (including the bias term θ_0 and the feature weights $\theta_1, \theta_2, \ldots, \theta_n$

Training a model means setting its parameters so that the model best fits the training set. For this purpose, we first need a measure of how well (or poorly) the model fits the training data. The most common performance measure of a regression model is the Root Mean Square Error (RMSE), therefore, to train a Linear Regression model, you need to find the value of that minimizes the RMSE. In practice, it is simpler to minimize the Mean Square Error (MSE) than the RMSE, and it leads to the same result.

4.2 A Practical Example from Kaggle Competition

4.2.1 The Problem

The objective is to predict the prices of house in Iowa from features. We have 800 observations in training set, 600 in validation set, and 508 in test set

4.2.2 Categorical Features

Categorical features are features where there are a number of non-numerical alternatives. We can define a dummy variable for each alternative. The variable equals 1 if the alternative is true and zero otherwise. This is known as **one-hot encoding**. But sometimes we do not have to do this because there is a natural ordering of variables. For example in this problem one of the categorical features is concerned with the basement quality as indicated by the ceiling height. The categories are:

- *Excellent (< 100 inches)*
- *Good (90-99 inches)*
- *Typical (80-89 inches)*
- *Fair (70-79 inches)*
- *Poor (< 70 inches)*
- *No Basement*

This is an example of a categorical variable where *there is* a natural ordering. We created a new variable that had a values of 5, 4, 3, 2, 1 and 0 for the above six categories respectively.

The other categorical features specifies the location of the house as in one of 25 neighborhoods. We introduce 25 dummy variables with a one-hot encoding. The dummy variable equals one for an observation if the neighborhood is that in which the house is located and zero otherwise.

4.2.3 Loading data (J. C. Hull, 2019, Chapter 3)

To illustrate the regression techniques discussed in this chapter we will use a total of 48 feature. 21 are numerical and two are categorical and to this we had, as discussed above, 25 categorical variables for the neighborhoods.

[111]:
```
# Both features and target have already been scaled: mean = 0; SD = 1
data = pd.read_csv('.\data\Houseprice_data_scaled.csv')
```

First of all check how many records we have

[112]:
```
print("Number of available data = "  + str(len(data.index)))
```

```
Number of available data = 2908
```

Before starting we emphasize the need to divide all available data into three parts: a **training set**, a **validation set** and a **test set**. The training set is used to determine parameters for trial models. The validation set is used to determine the extent to chich the models created from the training set generalize to new data. Finally, the test set is used as a final estimate of the accuracy of the chosen model.

We had 2908 observations. We split this as follows: 1800 in the training set, 600 in the validation set and 508 in the test set.

```
[113]: # First 1800 data items are training set; the next 600 are the validation set
       train = data.iloc[:1800]
       val = data.iloc[1800:2400]
```

We now procede to create **labels** and **features**. As we have already said, the labels are the values of the target that is to be predicted, in this case the 'Sale Price', and we indicate that whit 'y':

```
[114]: y_train, y_val = train[['Sale Price']], val[['Sale Price']]
```

The features and dummy variables were scaled using the Z-score method. Also the target values (i.e. the house prices) have been scaled with the Z-score method. The features are the variables from which the predictions are to be made and, in this case, can be obtained simply dropping the column 'Sale Price' from our dataset:

```
[115]: X_train, X_val = train.drop('Sale Price', axis=1), val.drop('Sale Price',␣
        ↪axis=1)
```

```
[116]: X_train.columns
```

```
[116]: Index(['LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
              'BsmtFinSF1', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF',
              'GrLivArea', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'TotRmsAbvGrd',
              'Fireplaces', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
              'EnclosedPorch', 'Blmngtn', 'Blueste', 'BrDale', 'BrkSide', 'ClearCr',
              'CollgCr', 'Crawfor', 'Edwards', 'Gilbert', 'IDOTRR', 'MeadowV',
              'Mitchel', 'Names', 'NoRidge', 'NPkVill', 'NriddgHt', 'NWAmes',
              'OLDTown', 'SWISU', 'Sawyer', 'SawyerW', 'Somerst', 'StoneBr', 'Timber',
              'Veenker', 'Bsmt Qual'],
             dtype='object')
```

4.2.4 Linear Regression with sklearn

```
[117]: # Importing models
       from sklearn.linear_model import LinearRegression
       from sklearn.metrics import mean_squared_error as mse
```

```
[118]: lr = LinearRegression()
       lr.fit(X_train,y_train)
```

```
[118]: LinearRegression()
```

```
[119]: lr.intercept_
```

```
[119]: array([-3.06335941e-11])
```

```
[120]: # Create dataFrame with corresponding feature and its respective coefficients
coeffs = pd.DataFrame(
    [
        ['intercept'] + list(X_train.columns),
        list(lr.intercept_) + list(lr.coef_[0])
    ]
).transpose().set_index(0)
coeffs
```

```
[120]:                          1
0
intercept     -3.06336e-11
LotArea          0.0789996
OverallQual       0.214395
OverallCond      0.0964787
YearBuilt         0.160799
YearRemodAdd     0.0253524
BsmtFinSF1       0.0914664
BsmtUnfSF       -0.0330798
TotalBsmtSF       0.138199
1stFlrSF          0.152786
2ndFlrSF          0.132765
GrLivArea         0.161303
FullBath        -0.0208076
HalfBath         0.0171941
BedroomAbvGr    -0.0835202
TotRmsAbvGrd     0.0832203
Fireplaces       0.0282578
GarageCars       0.0379971
GarageArea       0.0518093
WoodDeckSF       0.0208337
OpenPorchSF      0.0340982
EnclosedPorch   0.00682223
Blmngtn         -0.0184305
Blueste         -0.0129214
BrDale          -0.0246262
BrkSide          0.0207618
ClearCr        -0.00737828
CollgCr        -0.00675362
Crawfor          0.0363235
Edwards       -0.000690065
Gilbert        -0.00834022
IDOTRR         -0.00153683
MeadowV          -0.016418
Mitchel         -0.0284821
Names           -0.0385057
NoRidge          0.0515626
```

```
NPkVill         -0.0219519
NriddgHt           0.12399
NWAmes          -0.0517591
OLDTown          -0.026499
SWISU          -0.00414298
Sawyer          -0.0181341
SawyerW         -0.0282754
Somerst          0.0275063
StoneBr          0.0630586
Timber         -0.00276173
Veenker         0.00240311
Bsmt Qual        0.0113115
```

[121]:
```
len(coeffs.index)
```

[121]:
```
48
```

[122]:
```
pred_t=lr.predict(X_train)
mse(y_train,pred_t)
```

[122]:
```
0.11401526431246334
```

[123]:
```
pred_v=lr.predict(X_val)
mse(y_val,pred_v)
```

[123]:
```
0.11702499460121653
```

For the data we are considering it turns out that this regression model generalizes well. The mean squared error for the validation set was only a little higher than that for the training set. However linear regression with no regularization leads to some strange results because of the correlation between features. For example it makes no sense that the weights for number of full bathrooms and number of bedrooms are negative!

[124]:
```
x1 = X_train['GrLivArea']
x2 = X_train['BedroomAbvGr']
x1.corr(x2)
```

[124]:
```
0.5347396038733943
```

Ridge Regression

[125]:
```
# Importing Ridge
from sklearn.linear_model import Ridge
```

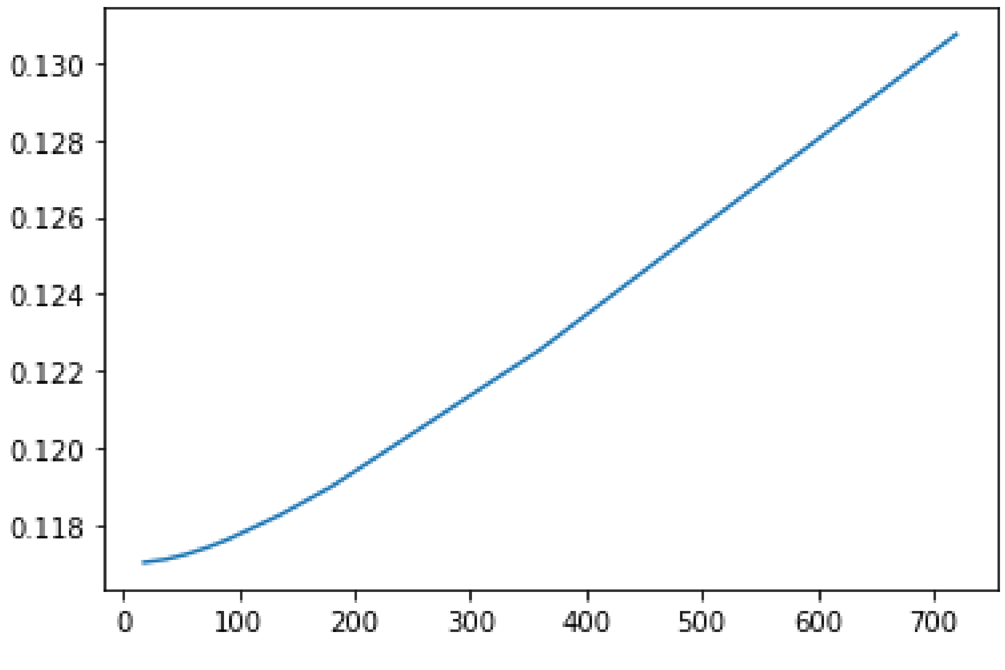
We try using Ridge regression with different values of the hyperparameter λ. The following code shows the effect of this parameter on the prediction error.

[126]:
```
# The alpha used by Python's ridge should be the lambda in Hull's book times␣
↪the number of observations
alphas=[0.01*1800, 0.02*1800, 0.03*1800, 0.04*1800, 0.05*1800, 0.075*1800,0.
↪1*1800,0.2*1800, 0.4*1800]
mses=[]
for alpha in alphas:
    ridge=Ridge(alpha=alpha)
    ridge.fit(X_train,y_train)
    pred=ridge.predict(X_val)
    mses.append(mse(y_val,pred))
    print(mse(y_val,pred))
```

```
0.11703284346091351
0.11710797319753011
0.11723952924901129
0.11741457158889518
0.11762384068711469
0.11825709631198025
0.11900057469147929
0.12254649996292954
0.1307359968074713
```

[127]:
```
plt.plot(alphas, mses)
```

[127]:
```
[<matplotlib.lines.Line2D at 0x22d13048048>]
```



As expected the prediction error increases as λ increases. Values of λ in the range 0 to 0.1 might be reasonably be considered because prediction errors increases only slightly when λ is in this range. However it turns out that the improvement in the model is quite small for these values of λ.

Lasso

```
[154]: # Import Lasso
       from sklearn.linear_model import Lasso
```

```
[155]: # Here we produce results for alpha=0.05 which corresponds to lambda=0.1 in␣
        ↪Hull's book
       lasso = Lasso(alpha=0.05)
       lasso.fit(X_train, y_train)
```

```
[155]: Lasso(alpha=0.05)
```

```
[156]: # DataFrame with corresponding feature and its respective coefficients
       coeffs = pd.DataFrame(
           [
               ['intercept'] + list(X_train.columns),
               list(lasso.intercept_) + list(lasso.coef_)
           ]
       ).transpose().set_index(0)
       coeffs
```

```
[156]:                           1
       0
       intercept     -1.25303e-11
       LotArea          0.0443042
       OverallQual       0.298079
       OverallCond              0
       YearBuilt        0.0520907
       YearRemodAdd     0.0644712
       BsmtFinSF1        0.115875
       BsmtUnfSF               -0
       TotalBsmtSF        0.10312
       1stFlrSF         0.0322946
       2ndFlrSF                 0
       GrLivArea         0.297065
       FullBath                 0
       HalfBath                 0
       BedroomAbvGr            -0
       TotRmsAbvGrd             0
       Fireplaces       0.0204043
       GarageCars        0.027512
       GarageArea       0.0664096
       WoodDeckSF      0.00102883
       OpenPorchSF     0.00215018
```

```
EnclosedPorch            -0
Blmngtn                  -0
Blueste                  -0
BrDale                   -0
BrkSide                   0
ClearCr                   0
CollgCr                  -0
Crawfor                   0
Edwards                  -0
Gilbert                   0
IDOTRR                   -0
MeadowV                  -0
Mitchel                  -0
Names                    -0
NoRidge            0.013209
NPkVill                  -0
NriddgHt          0.0842993
NWAmes                   -0
OLDTown                  -0
SWISU                    -0
Sawyer                   -0
SawyerW                  -0
Somerst                   0
StoneBr           0.0168153
Timber                    0
Veenker                   0
Bsmt Qual         0.0202754
```

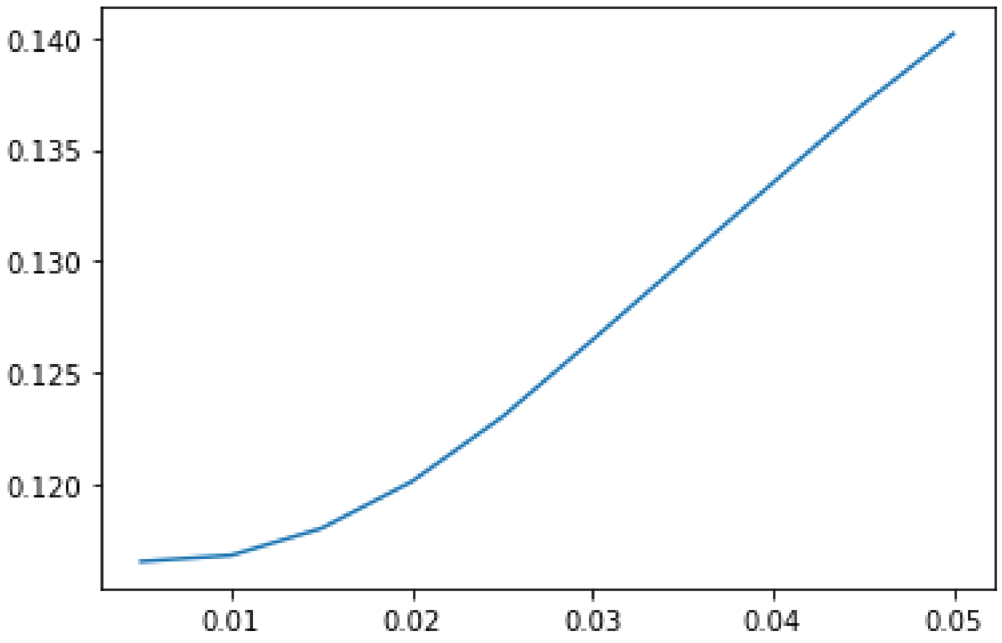
Lasso with different levels of alpha and its mse

[167]:
```
# We now consider different lambda values. The alphas are half the lambdas
alphas=[0.01/2, 0.02/2, 0.03/2, 0.04/2, 0.05/2, 0.06/2, 0.08/2, 0.09/2, 0.1/2]
mses=[]
for alpha in alphas:
    lasso=Lasso(alpha=alpha)
    lasso.fit(X_train,y_train)
    pred=lasso.predict(X_val)
    mses.append(mse(y_val,pred))
    print("lambda = " + '{:<05}'.format(alpha) + " - mse = " +␣
 ↪str(round(mse(y_val, pred),6)))
```

```
lambda = 0.005 - mse = 0.116548
lambda = 0.010 - mse = 0.116827
lambda = 0.015 - mse = 0.118033
lambda = 0.020 - mse = 0.120128
lambda = 0.025 - mse = 0.123015
lambda = 0.030 - mse = 0.126462
lambda = 0.040 - mse = 0.133492
lambda = 0.045 - mse = 0.137016
lambda = 0.050 - mse = 0.140172
```

[168]:
```
plt.plot(alphas, mses)
```

[168]:
```
[<matplotlib.lines.Line2D at 0x22d130aa188>]
```



Lasso regression leads to more interesting results. In the plot above you can see how the error in the validation set changes as tha value of the lasso λ increases. For small values of λ the error is actually less than when $\lambda = 0$ but as λ increases beyond about 0.03 the error starts to increase. A value of $\lambda = 0.04$ could be chosen.

5 References

John C. Hull, **Machine Learning in Business: An Introduction to the World of Data Science**, Amazon, 2019.

Paul Wilmott, **Machine Learning: An Applied Mathematics Introduction**, Panda Ohana Publishing, 2019.