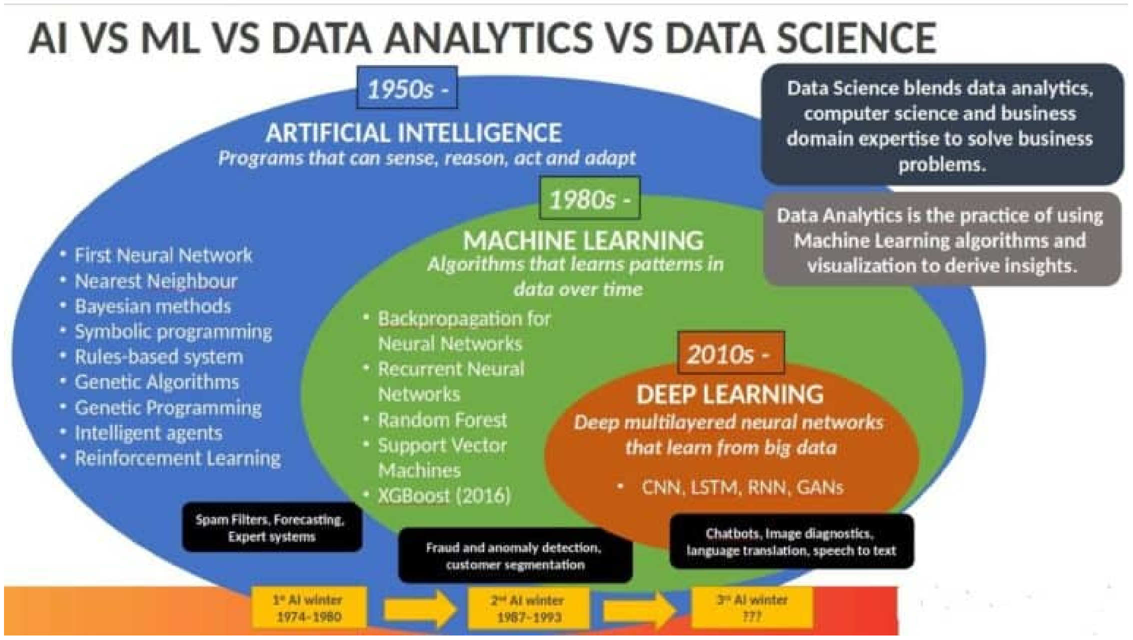
Day 1

May 3, 2020

1 Q1. What is the difference between AI, Data Science, ML, and DL?


AI VS ML VS DATA ANALYTICS VS DATA SCIENCE
1950s -
ARTIFICIAL INTELLIGENCE
Programs that can sense, reason, act and adapt
• First Neural Network
• Nearest Neighbour
• Bayesian methods
• Symbolic programming
• Rules-based system
• Genetic Algorithms
• Genetic Programming
• Intelligent agents
• Reinforcement Learning
1980s -
MACHINE LEARNING
Algorithms that learns patterns in data over time
• Backpropagation for Neural Networks
• Recurrent Neural Networks
• Random Forest
• Support Vector Machines
• XGBoost (2016)
2010s -
DEEP LEARNING
Deep multilayered neural networks that learn from big data
• CNN, LSTM, RNN, GANs
Data Science blends data analytics, computer science and business domain expertise to solve business problems.
Data Analytics is the practice of using Machine Learning algorithms and visualization to derive insights.
Spam Filters, Forecasting, Expert systems
Fraud and anomaly detection, customer segmentation
Chatbots, Image diagnostics, language translation, speech to text
1st AI winter 1974-1980
2nd AI winter 1987-1993
3rd AI winter ???

Artificial Intelligence: AI is purely math and scientific exercise, but when it became computational, it started to solve human problems formalized into a subset of computer science. Artificial intelligence has changed the original computational statistics paradigm to the modern idea that machines could mimic actual human capabilities, such as decision making and performing more "human" tasks. Modern AI into two categories 1. General AI - Planning, decision making, identifying objects, recognizing sounds, social & business transactions 2. Applied AI - driverless/ Autonomous car or machine smartly trade stocks

Machine Learning: Instead of engineers "teaching" or programming computers to have what they need to carry out tasks, that perhaps computers could teach themselves – learn something without being explicitly programmed to do so. ML is a form of AI where based on more data, and they can change actions and response, which will make more efficient, adaptable and scalable. e.g., navigation apps and recommendation engines. Classified into:- 1. Supervised 2. Unsupervised 3. Reinforcement learning

Data Science: Data science has many tools, techniques, and algorithms called from these fields, plus others –to handle big data

The goal of data science, somewhat similar to machine learning, is to make accurate predictions and to automate and perform transactions in real-time, such as purchasing internet traffic or automatically generating content. on math and coding and more on data and building new systems to process the data. Relying on the fields of data integration, distributed architecture, automated machine learning, data visualization, data engineering, and automated data-driven decisions, data science can cover an entire spectrum of data processing, not only the algorithms or statistics related to data.

Deep Learning: It is a technique for implementing ML.

ML provides the desired output from a given input, but DL reads the input and applies it to another data. In ML, we can easily classify the flower based upon the features. Suppose you want a machine to look at an image and determine what it represents to the human eye, whether a face, flower, landscape, truck, building, etc.

Machine learning is not sufficient for this task because machine learning can only produce an output from a data set – whether according to a known algorithm or based on the inherent structure of the data. You might be able to use machine learning to determine whether an image was of an "X" – a flower, say – and it would learn and get more accurate. But that output is binary (yes/no) and is dependent on the algorithm, not the data. In the image recognition case, the outcome is not binary and not dependent on the algorithm.

The neural network performs MICRO calculations with computational on many layers. Neural networks also support weighting data for 'confidence. These results in a probabilistic system, vs. deterministic, and can handle tasks that we think of as requiring more 'human-like' judgment.

2 Q2. What is the difference between supervised learning, unsupervised learning and Reinforcement learning?

Ans 2: Machine learning is the scientific study of algorithms and statistical models that computer systems use to effectively perform a specific task without using explicit instructions, relying on patterns and inference instead.

Building a model by learning the patterns of historical data with some relationship between data to make a data-driven prediction.

Types of Machine Learning:

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning
- Supervised learning

In a supervised learning model, the algorithm learns on a labeled dataset, to generate reasonable predictions for the response to new data. (Forecasting outcome of new data)

- Regression
- Classification
- Unsupervised learning

An unsupervised model, in contrast, provides unlabelled data that the algorithm tries to make sense of by extracting features, co-occurrence and underlying patterns on its own. We use unsupervised learning for

- Clustering
- Anomaly detection
- Association
- Autoencoders
- Reinforcement Learning

2.1 Reinforcement learning is less supervised and depends on the learning agent in determining the output solutions by arriving at different possible ways to achieve the best possible solution.

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.

In general, any machine learning problem can be assigned to one of two broad classifications:

Supervised learning and Unsupervised learning.

Supervised Learning In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output.

Supervised learning problems are categorized into “regression” and “classification” problems. In a regression problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function. In a classification problem, we are instead trying to predict results in a discrete output. In other words, we are trying to map input variables into discrete categories.

Example 1:

Given data about the size of houses on the real estate market, try to predict their price. Price as a function of size is a continuous output, so this is a regression problem.

We could turn this example into a classification problem by instead making our output about whether the house “sells for more or less than the asking price.” Here we are classifying the houses based on price into two discrete categories.

Example 2:

(a) Regression - Given a picture of a person, we have to predict their age on the basis of the given picture

(b) Classification - Given a patient with a tumor, we have to predict whether the tumor is malignant or benign.

Unsupervised Learning Unsupervised learning allows us to approach problems with little or no idea what our results should look like. We can derive structure from data where we don't necessarily know the effect of the variables.

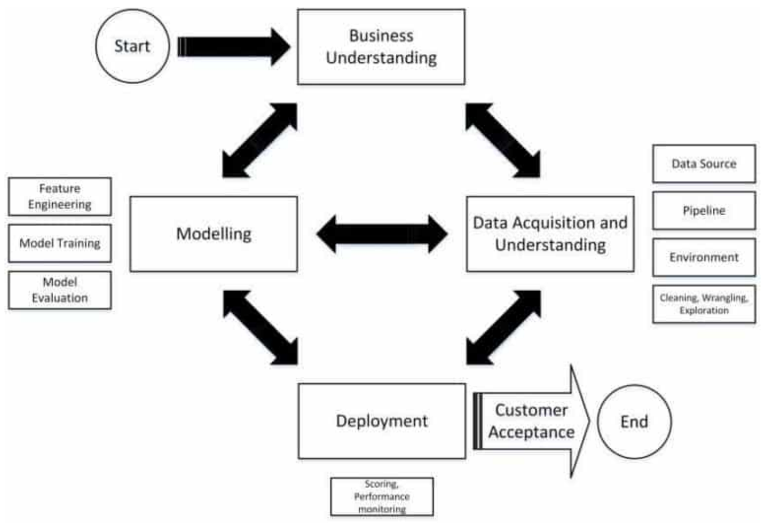
We can derive this structure by clustering the data based on relationships among the variables in the data.

2.2 With unsupervised learning there is no feedback based on the prediction results.

[0]:
```
#!/usr/bin/env python
# coding: utf-8

# # Supervised Learning Algorithms: Naive Bayes classifiers

# *In this template, only **data input** and **input/target variables** need to␣
 ↪be specified (see "Data Input & Variables" section for further instructions).␣
 ↪None of the other sections needs to be adjusted. As a data input example, .csv␣
 ↪file from IBM Box web repository is used.*

# ## 1. Libraries

# *Run to import the required libraries.*

# In[1]:


get_ipython().run_line_magic('matplotlib', 'notebook')
import pandas as pd
from sklearn.model_selection import train_test_split


# ## 2. Data Input and Variables

# *Define the data input as well as the input (X) and target (y) variables and␣
 ↪run the code. Do not change the data & variable names **['df', 'X', 'y']** as␣
 ↪they are used in further sections.*

# In[2]:


### Data Input
# df =

### Defining Variables
# X =
# y =
### Data Input Example
df = pd.read_csv('Iris.csv' , error_bad_lines=False)
X = df[['SepalLengthCm', 'SepalWidthCm' , 'PetalLengthCm' , 'PetalWidthCm']]
y = df['Species']


# ## 3. The Model

# *Run to build the model.*

# In[3]:


from sklearn.naive_bayes import GaussianNB

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)

nbclf = GaussianNB().fit(X_train, y_train)
print('Breast cancer dataset')
print('Accuracy of GaussianNB classifier on training set: {:.2f}'
     .format(nbclf.score(X_train, y_train)))
print('Accuracy of GaussianNB classifier on test set: {:.2f}'
     .format(nbclf.score(X_test, y_test)))
```

```
Breast cancer dataset
Accuracy of GaussianNB classifier on training set: 0.95
Accuracy of GaussianNB classifier on test set: 1.00
```

3 Q3. Describe the general architecture of Machine learning.



Business understanding: Understand the give use case, and also, it's good to know more about the domain for which the use cases are built.

Data Acquisition and Understanding: Data gathering from different sources and understanding the data. Cleaning the data, handling the missing data if any, data wrangling, and EDA(Exploratory data analysis).

Modeling: Feature engineering – scaling the data, feature selection – not all features are important. We use the backward elimination method, correlation factors, PCA and domain knowledge to select the features.

Model Training based on trial and error method or by experience, we select the algorithm and train with the selected features.

Model evaluation Accuracy of the model, confusion matrix and cross-validation.

If accuracy is not high, to achieve higher accuracy, we tune the model...either by changing the algorithm used or by feature selection or by gathering more data, etc.

Deployment: – Once the model has good accuracy, we deploy the model either in the cloud or Rasberry py or any other place. Once we deploy, we monitor the performance of the model.if its good...we go live with the model or reiterate the all process until our model performance is good.

It's not done yet!!!

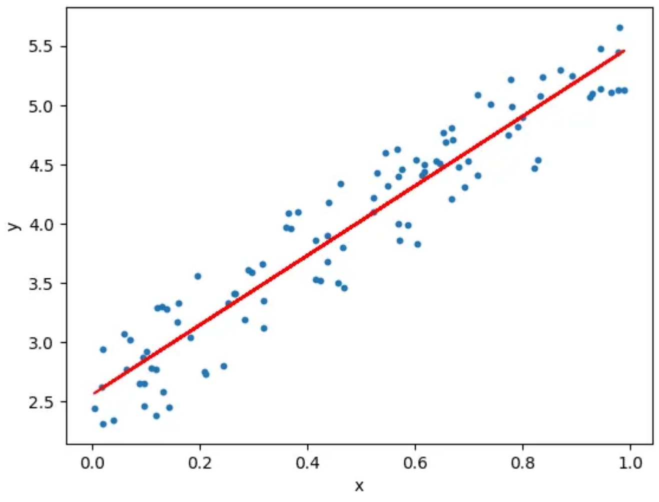
What if, after a few days, our model performs badly because of new data. In that case, we do all the process again by collecting new data and redeploy the model.

4 Q4. What is Linear Regression?

Ans 4:

Linear Regression tends to establish a relationship between a dependent variable(Y) and one or more independent variable(X) by finding the best fit of the straight line.

The equation for the Linear model is Y = mX+c, where m is the slope and c is the intercept



In the above diagram, the blue dots we see are the distribution of 'y' w.r.t 'x.' There is no straight line that runs through all the data points. So, the objective here is to fit the best fit of a straight line that will try to minimize the error between the expected and actual value.

Learn more youtobe

[0]:
```
%matplotlib notebook
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
### Data Input Example
df = pd.read_csv('Iris.csv')
df.describe()

X = df[['SepalLengthCm',
 'SepalWidthCm',
 'PetalLengthCm',
 'PetalWidthCm']]
y = df['Species']
from sklearn.linear_model import LinearRegression

X_train, X_test, y_train, y_test= train_test_split(X, y, test_size=0.2␣
 ↪,random_state = 0)


# Feature Normalization
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)

linreg = LinearRegression()
linreg = linreg.fit(X_train_scaled, y_train)
# *To Predict our test data (unseen data)*
linreg.predict(X_test) #
```

```
[0]: array([468.22107454, 356.6539504 , 213.95472476, 475.26063587,
            223.91469255, 504.2915459 , 220.28935519, 400.90573247,
            399.89411947, 368.84483841, 439.18688523, 395.27396683,
            392.70116657, 399.14243727, 401.94505285, 220.62350376,
            399.39232957, 386.20662626, 227.03148078, 220.70688883,
            443.02892225, 402.8223182 , 242.03820109, 226.09766656,
            424.12920234, 206.76950332, 248.06424517, 380.01407494,
            336.9227442 , 237.41128923])
```

```
[0]: # Dataset Description

print('linear model coeff (w): {}'.format(linreg.coef_) , " are the coefficient␣
 ↪for each column or A.")
print('linear model intercept (b): {:.3f}'.format(linreg.intercept_) , " is our␣
 ↪intercept or B.")
print(" X is the predictor")
```

```
linear model coeff (w): [-9.00520539 -1.72093528 40.80616866 47.07989908]  are
the coefficient for each column or A.
linear model intercept (b): 204.167  is our intercept or B.
 X is the predictor
```

```
[0]: print('R-squared score (training): {:.3f}' .format(linreg.score(X_train_scaled,␣
 ↪y_train)))
print('R-squared score (test): {:.3f}'.format(linreg.score(X_test_scaled,␣
 ↪y_test)))
```

```
R-squared score (training): 0.934
R-squared score (test): 0.906
```

What Is R-Squared?

R-squared is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression.

The definition of R-squared is fairly straight-forward; it is the percentage of the response variable variation that is explained by a linear model. Or:

`R-squared = Explained variation / Total variation`

R-squared is always between 0 and 100%:

```
0% indicates that the model explains none of the variability of the response data
around its mean.
```

```
100% indicates that the model explains all the variability of the response data
around its mean.
```

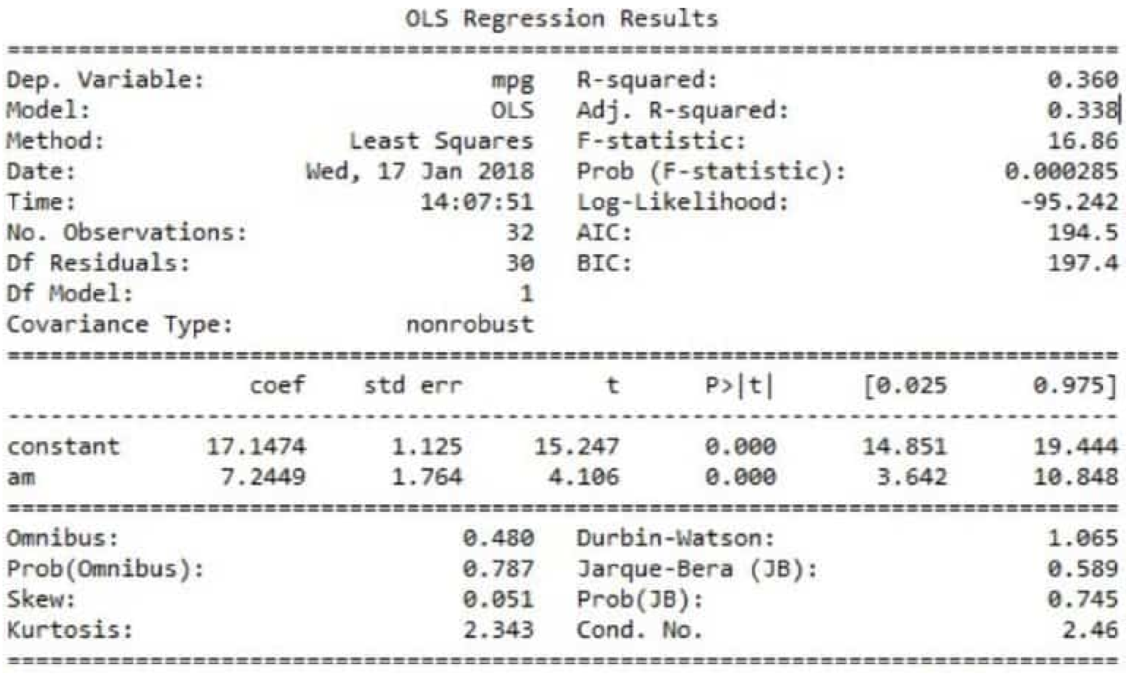
In general, the higher the R-squared, the better the model fits your data. However, there are important conditions for this guideline that I'll talk about both in this post and my next post.

5 Q5. OLS Stats Model (Ordinary Least Square)

Ans 5:

OLS is a stats model, which will help us in identifying the more significant features that can has an influence on the output. OLS model in python is executed as:

lm = smf.ols(formula = 'Sales ~ am+constant', data = data).fit() lm.conf_int() lm.summary() And we get the output as below,



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    mpg   R-squared:                       0.360
Model:                            OLS   Adj. R-squared:                  0.338
Method:                 Least Squares   F-statistic:                     16.86
Date:                Wed, 17 Jan 2018   Prob (F-statistic):           0.000285
Time:                        14:07:51   Log-Likelihood:                -95.242
No. Observations:                  32   AIC:                             194.5
Df Residuals:                      30   BIC:                             197.4
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
constant      17.1474      1.125     15.247      0.000      14.851      19.444
am             7.2449      1.764      4.106      0.000       3.642      10.848
==============================================================================
Omnibus:                        0.480   Durbin-Watson:                   1.065
Prob(Omnibus):                  0.787   Jarque-Bera (JB):                0.589
Skew:                           0.051   Prob(JB):                        0.745
Kurtosis:                       2.343   Cond. No.                         2.46
==============================================================================
```

The higher the t-value for the feature, the more significant the feature is to the output variable. And also, the p-value plays a rule in rejecting the Null hypothesis(Null hypothesis stating the features has zero significance on the target variable.). If the p-value is less than 0.05(95% confidence interval) for a feature, then we can consider the feature to be significant.

[0]:
```
%matplotlib inline

import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
```

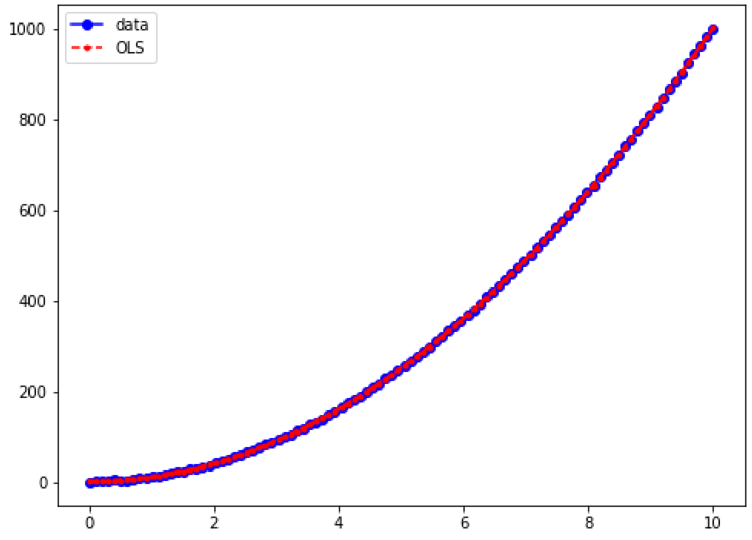
```python
from statsmodels.sandbox.regression.predstd import wls_prediction_std

np.random.seed(9876789)

nsample = 100
x = np.linspace(0, 10, 100)
X = np.column_stack((x, x**2))
beta = np.array([1, 0.1, 10])
e = np.random.normal(size=nsample)

# Our model needs an intercept so we add a column of 1s:


X = sm.add_constant(X)
y = np.dot(X, beta) + e


model = sm.OLS(y, X)
results = model.fit()
print(results.summary())
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in the
public API at pandas.testing instead.
  import pandas.util.testing as tm

                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                 4.020e+06
Date:                Sun, 26 Apr 2020   Prob (F-statistic):          2.83e-239
Time:                        05:45:55   Log-Likelihood:                -146.51
No. Observations:                 100   AIC:                             299.0
Df Residuals:                      97   BIC:                             306.8
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.3423      0.313      4.292      0.000       0.722       1.963
x1            -0.0402      0.145     -0.278      0.781      -0.327       0.247
x2            10.0103      0.014    715.745      0.000       9.982      10.038
==============================================================================
Omnibus:                        2.042   Durbin-Watson:                   2.274
Prob(Omnibus):                  0.360   Jarque-Bera (JB):                1.875
Skew:                           0.234   Prob(JB):                        0.392
Kurtosis:                       2.519   Cond. No.                         144.
```

```
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[0]:
```python
print('Parameters: ', results.params)
print('R2: ', results.rsquared)

prstd, iv_l, iv_u = wls_prediction_std(results)

fig, ax = plt.subplots(figsize=(8,6))

ax.plot(x, y, 'b-o', label="data")
# ax.plot(x, y, 'b-', label="True")
ax.plot(x, results.fittedvalues, 'r--.', label="OLS")
ax.plot(x, iv_u, 'r--')
ax.plot(x, iv_l, 'r--')
ax.legend(loc='best');
```

```
Parameters:  [ 1.34233516 -0.04024948 10.01025357]
R2:  0.9999879365025871
```



Ans 6:

The main objective of creating a model(training data) is making sure it fits the data properly and reduce the loss. Sometimes the model that is trained which will fit the data but it may fail and give a poor performance during analyzing of data (test data). This leads to overfitting. Regularization came to overcome overfitting.

Lasso Regression (Least Absolute Shrinkage and Selection Operator) adds “Absolute value of magnitude” of coefficient, as penalty term to the loss function.

Lasso shrinks the less important feature’s coefficient to zero; thus, removing some feature altogether. So, this works well for feature selection in case we have a huge number of features.

Methods like Cross-validation, Stepwise Regression are there to handle overfitting and perform feature selection work well with a small set of features. These techniques are good when we are dealing with a large set of features.

Along with shrinking coefficients, the lasso performs feature selection, as well. (Remember the 'selection' in the lasso full-form?) Because some of the coefficients become exactly zero, which is equivalent to the particular feature being excluded from the model.

6 Q7. L2 Regularization(L2 = Ridge Regression)

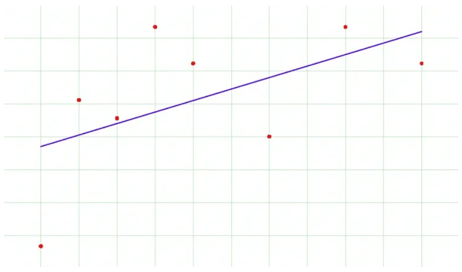
Ans 7:

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|^2$$

Overfitting happens when the model learns signal as well as noise in the training data and wouldn't perform well on new/unseen data on which model wasn't trained on.

To avoid overfitting your model on training data like cross-validation sampling, reducing the number of features, pruning, regularization, etc.

So to avoid overfitting, we perform Regularization.



The Regression model that uses L2 regularization is called Ridge Regression.

The formula for Ridge Regression:

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \boxed{\lambda \sum_{j=1}^{n} \theta_j^2}\right]$$

$$\min_\theta J(\theta)$$

Regularization adds the penalty as model complexity increases. The regularization parameter (lambda) penalizes all the parameters except intercept so that the model generalizes the data and won't overfit.

Ridge regression adds “squared magnitude of the coefficient” as penalty term to the loss function. Here the box part in the above image represents the L2 regularization element/term.

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Lambda is a hyper-parameter.

If lambda is zero, then it is equivalent to OLS. But if lambda is very large, then it will add too much weight, and it will lead to under-fitting.

Ridge regularization forces the weights to be small but does not make them zero and does not give the sparse solution.

Ridge is not robust to outliers as square terms blow up the error differences of the outliers, and the regularization term tries to fix it by penalizing the weights

Ridge regression performs better when all the input features influence the output, and all with weights are of roughly equal size.

L2 regularization can learn complex data patterns.

Similarity: L1 , L2 regularization does prevent the model from overfitting.

Differance: Lasso Regularization as reducing the quantity of features: By reducing the sum of absolute values of the coefficients, what Lasso Regularization (L1 Norm) does is to reduce the number of features in the model altogether to predict the target variable.

Ridge Regularization as reducing the quality of features: by reducing the sum of square of coefficients, Ridge Regularization (L2 Norm) doesn’t necessarily reduce the number of features per se, but rather reduces the magnitude/impact that each features has on the model by reducing the coefficient value.

Combination of Lasso and Ridge Regularization: ElasticNet would be the ideal type of regularization to perform on a model.**

"""t81_558_class_05_1_reg_ridge_lasso.ipynb

Automatically generated by Colaboratory.

Original file is located at

Link for reg ridge lasso

7 T81-558: Applications of Deep Neural Networks

Module 5: Regularization and Dropout * Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis * For more information visit the class website.

8 Module 5 Material

- **Part 5.1: Part 5.1: Introduction to Regularization: Ridge and Lasso** [Video] [Notebook]
- Part 5.2: Using K-Fold Cross Validation with Keras [Video] [Notebook]
- Part 5.3: Using L1 and L2 Regularization with Keras to Decrease Overfitting [Video] [Notebook]
- Part 5.4: Drop Out for Keras to Decrease Overfitting [Video] [Notebook]
- Part 5.5: Benchmarking Keras Deep Learning Regularization Techniques [Video] [Notebook]

9 Google CoLab Instructions

The following code ensures that Google CoLab is running the correct version of TensorFlow.

10 Part 5.1: Introduction to Regularization: Ridge and Lasso

Regularization is a technique that reduces overfitting, which occurs when neural networks attempt to memorize training data, rather than learn from it. Humans are capable of overfitting as well. Before we examine the ways that a machine accidentally overfits, we will first explore how humans can suffer from it.

Human programmers often take certification exams to show their competence in a given programming language. To help prepare for these exams, the test makers often make practice exams available. Consider a programmer who enters a loop of taking the practice exam, studying more, and then taking the practice exam again. At some point, the programmer has memorized much of the practice exam, rather than learning the techniques necessary to figure out the individual questions. The programmer has now overfit to the practice exam. When this programmer takes the real exam, his actual score will likely be lower than what he earned on the practice exam.

A computer can overfit as well. Although a neural network received a high score on its training data, this result does not mean that the same neural network will score high on data that was not inside the training set. Regularization is one of the techniques that can prevent overfitting. A number of different regularization techniques exist. Most work by analyzing and potentially modifying the weights of a neural network as it trains.

10.0.1 L1 and L2 Regularization

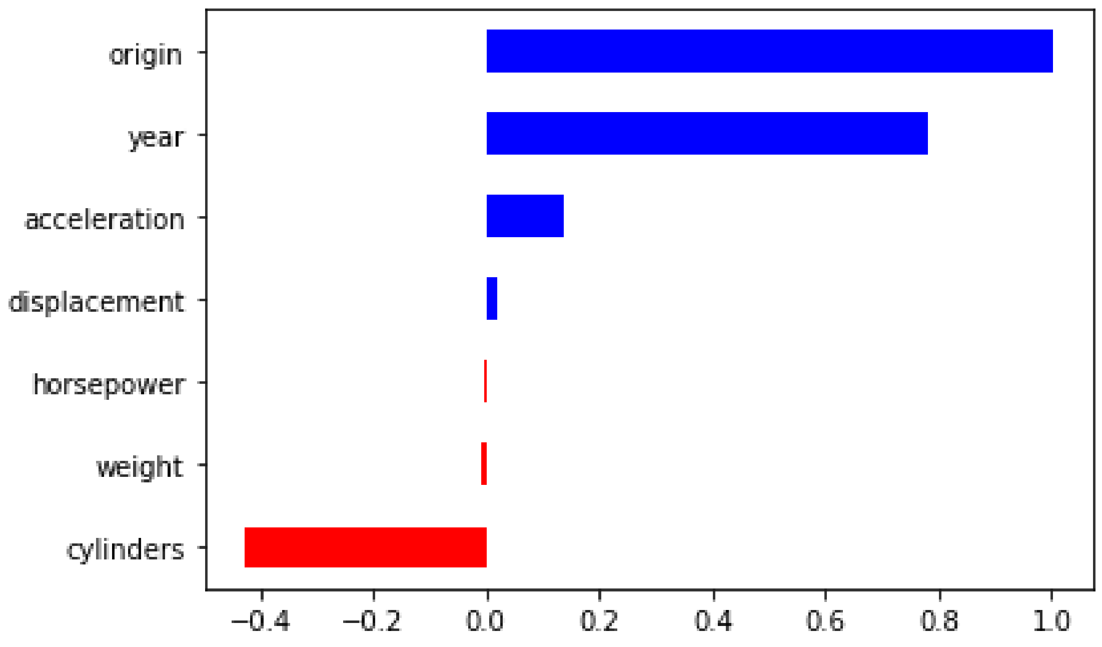
L1 and L2 regularization are two common regularization techniques that can reduce the effects of overfitting (Ng, 2004). Both of these algorithms can either work with an objective function or as a part of the backpropagation algorithm. In both cases the regularization algorithm is attached to the training algorithm by adding an additional objective.

Both of these algorithms work by adding a weight penalty to the neural network training. This penalty encourages the neural network to keep the weights to small values. Both L1 and L2 calculate this penalty differently. For gradient-descent-based algorithms, such as backpropagation, you can add this penalty calculation to the calculated gradients. For objective-function-based training, such as simulated annealing, the penalty is negatively combined with the objective score.

We are going to look at linear regression to see how L1 and L2 regularization work. The following code sets up the auto-mpg data for this purpose. """

[0]:
```
# -*- coding: utf-8 -*-


# Commented out IPython magic to ensure Python compatibility.
try:
#     %tensorflow_version 2.x
    COLAB = True
    print("Note: using Google CoLab")
except:
    print("Note: not using Google CoLab")
    COLAB = False




from sklearn.linear_model import LassoCV
import pandas as pd
import os
import numpy as np
from sklearn import metrics
from scipy.stats import zscore
from sklearn.model_selection import train_test_split

df = pd.read_csv(
    "https://data.heatonresearch.com/data/t81-558/auto-mpg.csv",
    na_values=['NA', '?'])

# Handle missing value
df['horsepower'] = df['horsepower'].fillna(df['horsepower'].median())

# Pandas to Numpy
names = ['cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'year', 'origin']
x = df[names].values
y = df['mpg'].values # regression

# Split into train/test
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.25, random_state=45)

# Commented out IPython magic to ensure Python compatibility.
# Simple function to evaluate the coefficients of a regression
# %matplotlib inline
from IPython.display import display, HTML

def report_coef(names,coef,intercept):
    r = pd.DataFrame( { 'coef': coef, 'positive': coef>=0  }, index = names )
    r = r.sort_values(by=['coef'])
    display(r)
    print(f"Intercept: {intercept}")
    r['coef'].plot(kind='barh', color=r['positive'].map({True: 'b', False: 'r'}))

"""# Linear Regression

To understand L1/L2 regularization, it is good to start with linear regression. ␣
↪L1/L2 were first introduced for [linear regression](https://en.wikipedia.org/
↪wiki/Linear_regression).  They can also be used for neural networks.  To fully␣
↪understand L1/L2 we will begin with how they are used with linear regression.

The following code uses linear regression to fit the auto-mpg data set.  The␣
↪RMSE reported will not be as good as a neural network.
"""

import sklearn

# Create linear regression
regressor = sklearn.linear_model.LinearRegression()

# Fit/train linear regression
regressor.fit(x_train,y_train)
# Predict
pred = regressor.predict(x_test)

# Measure RMSE error.  RMSE is common for regression.
score = np.sqrt(metrics.mean_squared_error(pred,y_test))
print(f"Final score (RMSE): {score}")

report_coef(
  names,
  regressor.coef_,
  regressor.intercept_)
```

```
Note: using Google CoLab
Final score (RMSE): 3.0019345985860784

                  coef  positive
cylinders    -0.427721     False
weight       -0.007255     False
horsepower   -0.005491     False
displacement  0.020166      True
acceleration  0.138575      True
year          0.783047      True
origin        1.003762      True


Intercept: -19.101231042200084
```



"""# L1 (Lasso) Regularization

L1 Regularization, also called LASSO (Least Absolute Shrinkage and Selection Operator) is should be used to create sparsity in the neural network. In other words, the L1 algorithm will push many weight connections to near 0. When a weight is near 0, the program drops it from the network. Dropping weighted connections will create a sparse neural network.

Feature selection is a useful byproduct of sparse neural networks. Features are the values that the training set provides to the input neurons. Once all the weights of an input neuron reach 0, the neural network training determines that the feature is unnecessary. If your data set has a large number of input features that may not be needed, L1 regularization can help the neural network detect and ignore unnecessary features.

L1 is implemented by adding the following error to the objective to minimize:

$ E_1 = $\alpha \sum_w\{|w|\}$$

You should use L1 regularization to create sparsity in the neural network. In other words, the L1 algorithm will push many weight connections to near 0. When a weight is near 0, the program drops it from the network. Dropping weighted connections will create a sparse neural network. Feature selection is a useful byproduct of sparse neural networks. Features are the values that the training set provides to the input neurons. Once all the weights of an input neuron reach 0, the neural network training determines that the feature is unnecessary. If your data set has a large number of input features that may not be needed, L1 regularization can help the neural network detect and ignore unnecessary features.

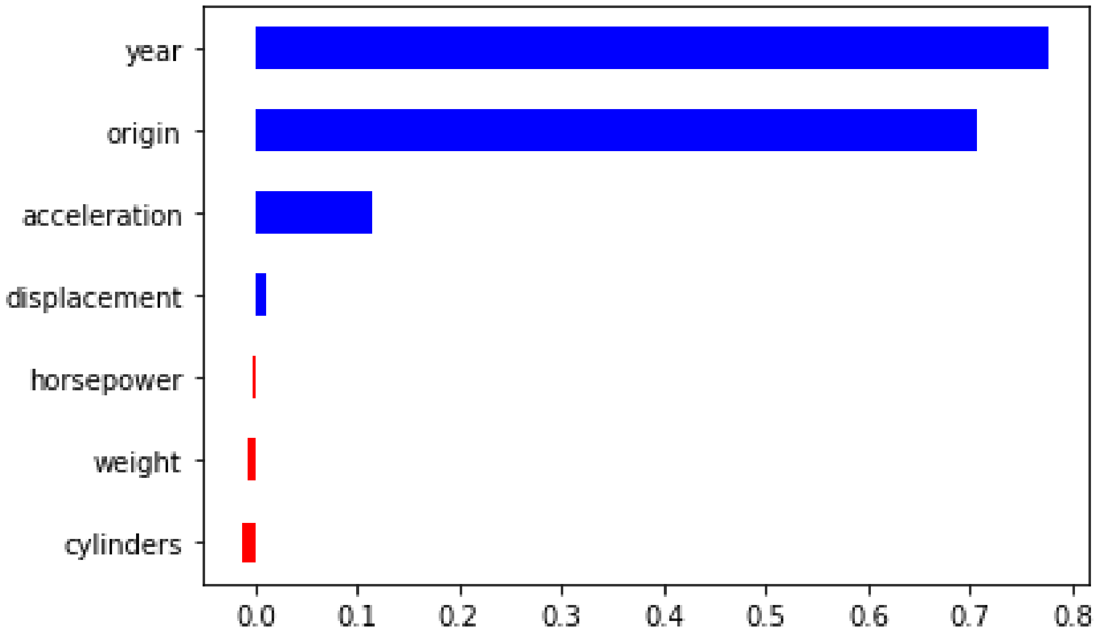
The following code demonstrates lasso regression. Notice the effect of the coefficients compared to the previous section that used linear regression. """

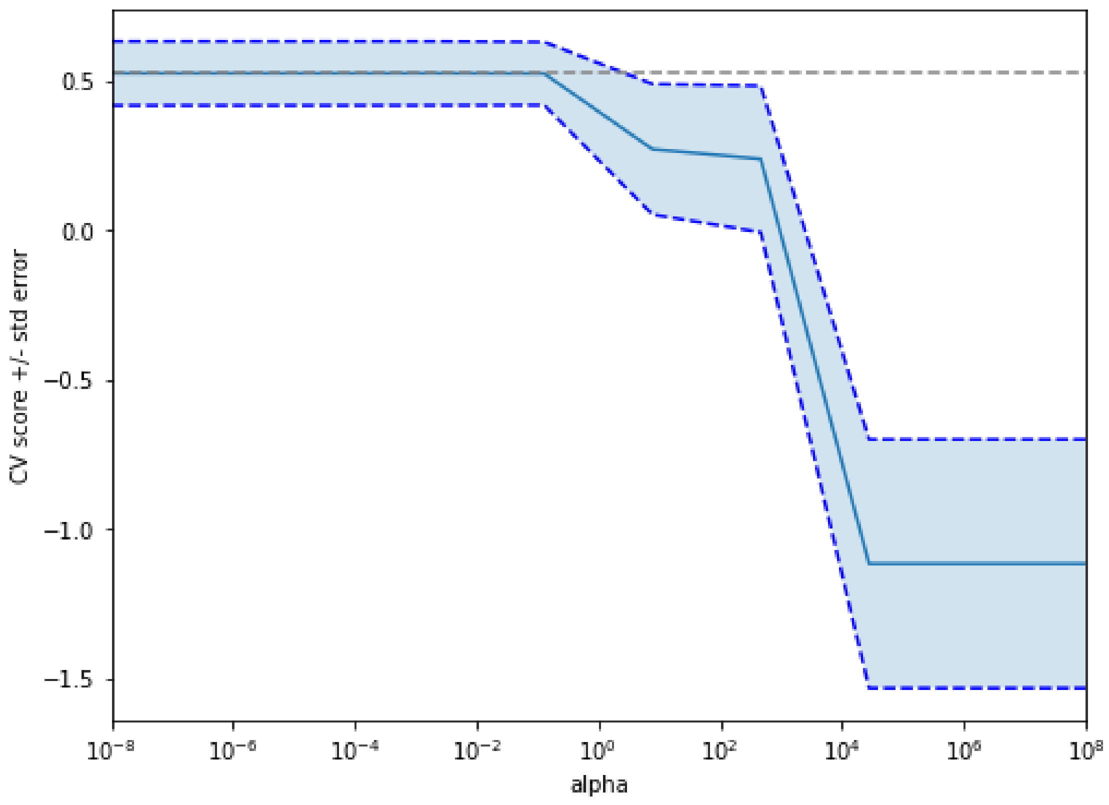
```
[0]: import sklearn
     from sklearn.linear_model import Lasso

     # Create linear regression
     regressor = Lasso(random_state=0,alpha=0.1)

     # Fit/train LASSO
     regressor.fit(x_train,y_train)
     # Predict
     pred = regressor.predict(x_test)

     # Measure RMSE error.  RMSE is common for regression.
     score = np.sqrt(metrics.mean_squared_error(pred,y_test))
     print(f"Final score (RMSE): {score}")

     report_coef(
       names,
       regressor.coef_,
       regressor.intercept_)

     import numpy as np
     import matplotlib.pyplot as plt

     from sklearn.linear_model import LassoCV
     from sklearn.linear_model import Lasso
     from sklearn.model_selection import KFold
     from sklearn.model_selection import cross_val_score

     lasso = Lasso(random_state=42)
     alphas = np.logspace(-8, 8, 10)

     scores = list()
     scores_std = list()

     n_folds = 3

     for alpha in alphas:
         lasso.alpha = alpha
         this_scores = cross_val_score(lasso, x, y, cv=n_folds, n_jobs=1)
         scores.append(np.mean(this_scores))
         scores_std.append(np.std(this_scores))

     scores, scores_std = np.array(scores), np.array(scores_std)

     plt.figure().set_size_inches(8, 6)
     plt.semilogx(alphas, scores)
```

```
# plot error lines showing +/- std. errors of the scores
std_error = scores_std / np.sqrt(n_folds)

plt.semilogx(alphas, scores + std_error, 'b--')
plt.semilogx(alphas, scores - std_error, 'b--')

# alpha=0.2 controls the translucency of the fill color
plt.fill_between(alphas, scores + std_error, scores - std_error, alpha=0.2)

plt.ylabel('CV score +/- std error')
plt.xlabel('alpha')
plt.axhline(np.max(scores), linestyle='--', color='.5')
plt.xlim([alphas[0], alphas[-1]])
```

```
Final score (RMSE): 3.0604021904033303

                  coef  positive
cylinders    -0.012995     False
weight       -0.007328     False
horsepower   -0.002715     False
displacement  0.011601      True
acceleration  0.114391      True
origin        0.708222      True
year          0.777480      True


Intercept: -18.506677982383223
```

[0]: (1e-08, 100000000.0)





"""# L2 (Ridge) Regularization

You should use Tikhonov/Ridge/L2 regularization when you are less concerned about creating a space network and are more concerned about low weight values. The lower weight values will typically lead to less overfitting.

$E_2 = \alpha \sum_w{w^2}$

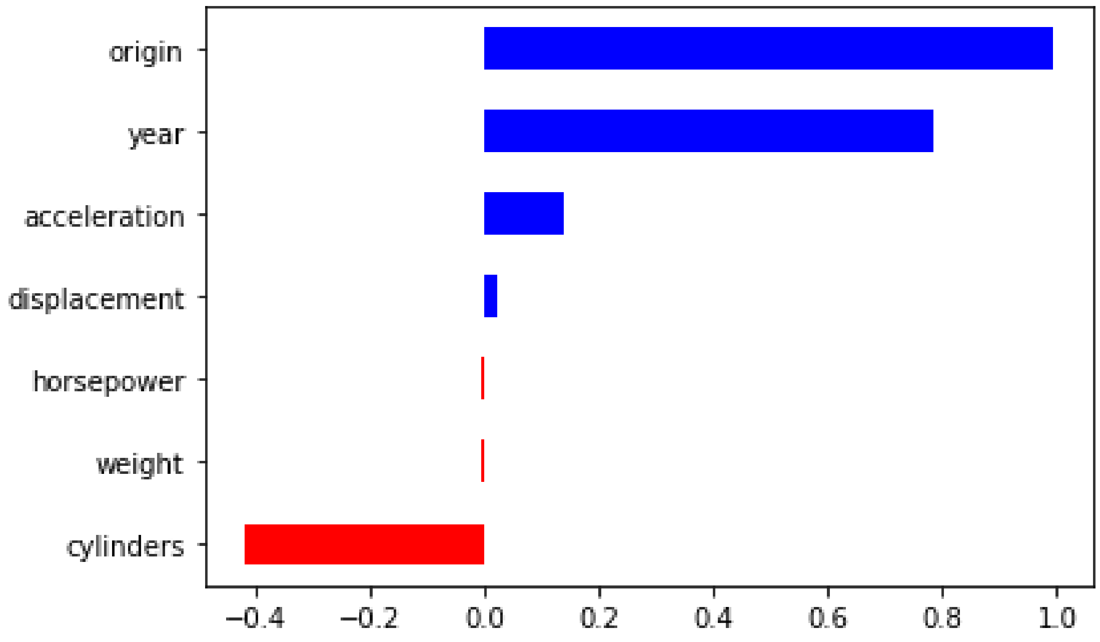
Like the L1 algorithm, the α value determines how important the L2 objective is compared to the neural network's error. Typical L2 values are below 0.1 (10%). The main calculation performed by L2 is the summing of the squares of all of the weights. The bias values are not summed.

You should use L2 regularization when you are less concerned about creating a space network and are more concerned about low weight values. The lower weight values will typically lead to less overfitting. Generally L2 regularization will produce better overall performance than L1. However, L1 might be useful in situations where there are a large number of inputs and some of the weaker inputs should be pruned.

The following code uses L2 with linear regression (Ridge regression): """

```
[0]: import sklearn
     from sklearn.linear_model import Ridge

     # Create linear regression
     regressor = Ridge(alpha=1)

     # Fit/train Ridge
     regressor.fit(x_train,y_train)
     # Predict
     pred = regressor.predict(x_test)

     # Measure RMSE error.  RMSE is common for regression.
     score = np.sqrt(metrics.mean_squared_error(pred,y_test))
     print("Final score (RMSE): {score}")

     report_coef(
       names,
       regressor.coef_,
       regressor.intercept_)

     """# ElasticNet Regularization

     The ElasticNet regression combines both L1 and L2.  Both penalties are applied. ␣
      ↪The amount of L1 and L2 are governed by the parameters alpha and beta.

     $ a * L1 + b * L2 $
     """

     import sklearn
     from sklearn.linear_model import ElasticNet

     # Create linear regression
     regressor = ElasticNet(alpha=0.1, l1_ratio=0.1)

     # Fit/train LASSO
     regressor.fit(x_train,y_train)
     # Predict
     pred = regressor.predict(x_test)

     # Measure RMSE error.  RMSE is common for regression.
     score = np.sqrt(metrics.mean_squared_error(pred,y_test))
     print(f"Final score (RMSE): {score}")

     report_coef(
       names,
       regressor.coef_,
       regressor.intercept_)
```

```
Final score (RMSE): {score}

                   coef  positive
cylinders    -0.421393     False
weight       -0.007257     False
horsepower   -0.005385     False
displacement  0.020006      True
acceleration  0.138470      True
year          0.782889      True
origin        0.994621      True


Intercept: -19.079800744254825
Final score (RMSE): 3.0450899960775026

                   coef  positive
cylinders    -0.274010     False
weight       -0.007303     False
horsepower   -0.003231     False
displacement  0.016194      True
acceleration  0.132348      True
year          0.777482      True
origin        0.782781      True


Intercept: -18.389355690429745
```



11 Q8. What is R square(where to use and where not)?

Ans 8.

R-squared is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression.

The definition of R-squared is the percentage of the response variable variation that is explained by a linear model.

R-squared = Explained variation / Total variation

R-squared is always between 0 and 100%.

0% indicates that the model explains none of the variability of the response data around its mean.

100% indicates that the model explains all the variability of the response data around its mean.

In general, the higher the R-squared, the better the model fits your data.

$$R^2 = 1 - \frac{\overset{\text{Sum Squared Regression Error}}{SS_{Regression}}}{\underset{\text{Sum Squared Total Error}}{SS_{Total}}}$$

$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

There is a problem with the R-Square. The problem arises when we ask this question to ourselves.** Is it good to help as many independent variables as possible?**

The answer is No because we understood that each independent variable should have a meaningful impact. But, even** if we add independent variables which are not meaningful**, will it improve R-Square value?

Yes, this is the basic problem with R-Square. How many junk independent variables or important independent variable or impactful independent variable you add to your model, the R-Squared value will always increase. It will never decrease with the addition of a newly independent variable, whether it could be an impactful, non-impactful, or bad variable, so we need another way to measure equivalent R-Square, which penalizes our model with any junk independent variable.

So, we calculate the Adjusted R-Square with a better adjustment in the formula of generic R-square.

$$R^2_{\text{adjusted}} = 1 - \frac{(1 - R^2)(N - 1)}{N - p - 1}$$

where

R^2 = sample R-square

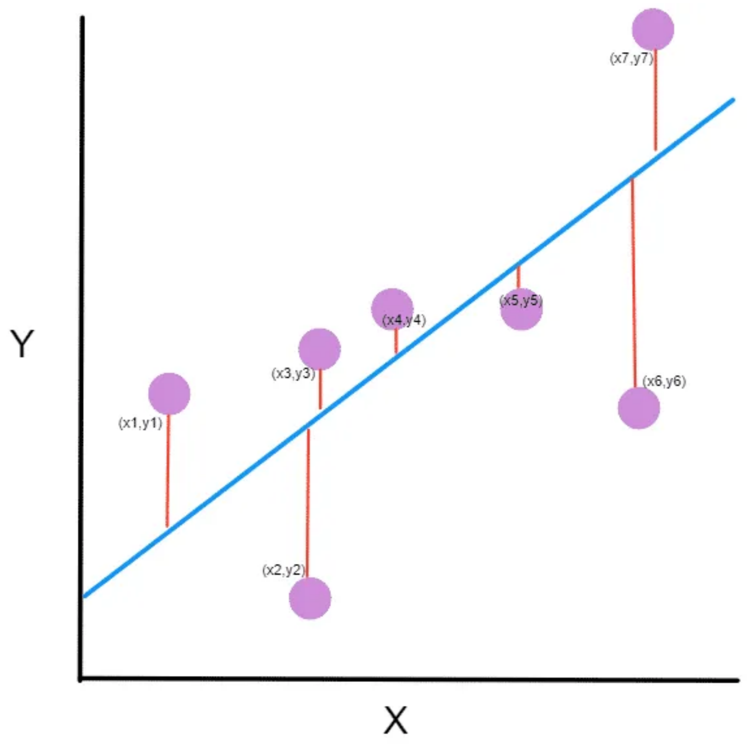
p = Number of predictors

N = Total sample size.

12 Q9. What is Mean Square Error?

The mean squared error tells you how close a regression line is to a set of points. It does this by taking the distances from the points to the regression line (these distances are the "errors") and squaring them.

Giving an intuition:



The line equation is y=Mx+B. We want to find M (slope) and B (y-intercept) that minimizes the squared error.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \tilde{y}_i)^2$$

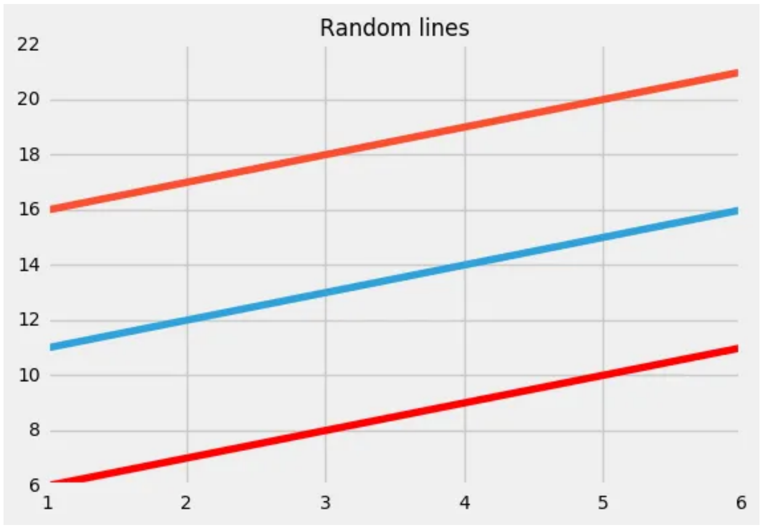
Q10. Why Support Vector Regression? Difference between SVR and a simple regression model?

Ans 10:

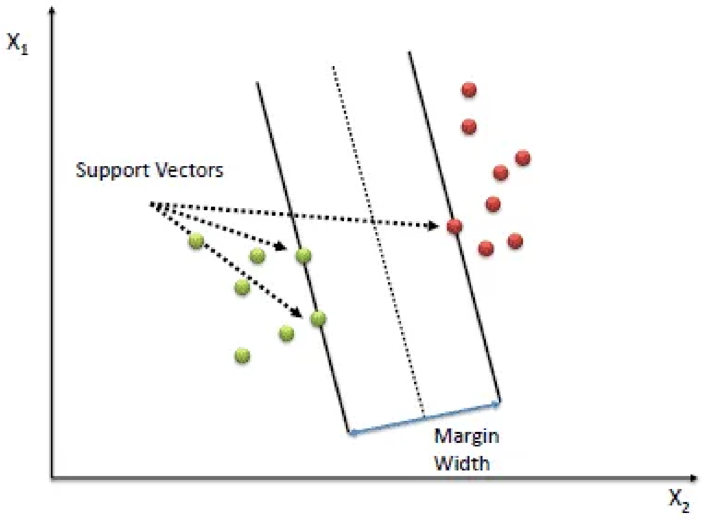
In simple linear regression, try to minimize the error rate. But in SVR, we try to fit the error within a certain threshold.

Main Concepts:-

- Boundary
- Kernel
- Support Vector
- Hyper Plane

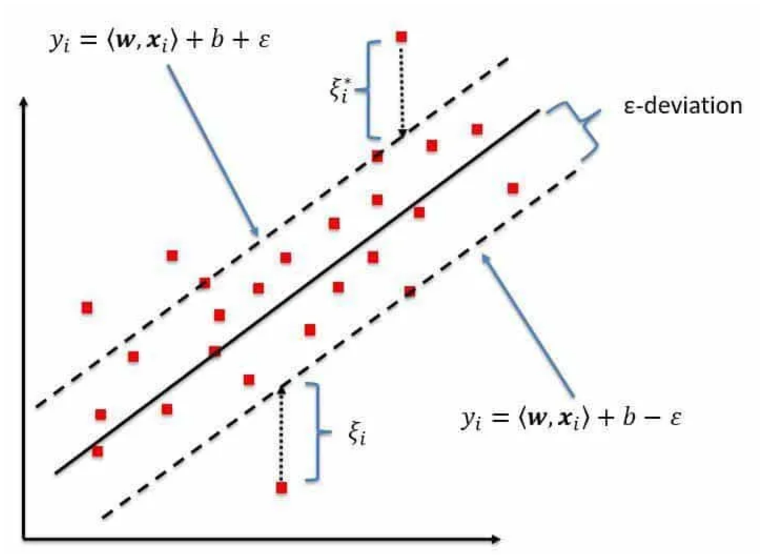


Blueline: Hyper Plane; Red Line: Boundary-Line



Our best fit line is the one where the hyperplane has the maximum number of points.

We are trying to do here is trying to decide a decision boundary at 'e' distance from the original hyperplane such that data points closest to the hyperplane or the support vectors are within that boundary line.

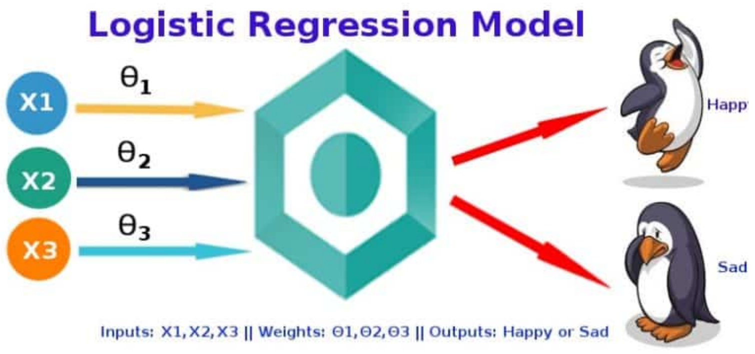

$y_i = \langle \mathbf{w}, \mathbf{x}_i \rangle + b + \varepsilon$
ξ_i^*
ε-deviation
ξ_i
$y_i = \langle \mathbf{w}, \mathbf{x}_i \rangle + b - \varepsilon$

Data science interview questions day2

13 Q11. What is Logistic Regression?

Answer:

The logistic regression technique involves the dependent variable, which can be represented in the binary (0 or 1, true or false, yes or no) values, which means that the outcome could only be in either one form of two. For example, it can be utilized when we need to find the probability of a successful or fail event.


Logistic Regression Model
X1
X2
X3
θ1
θ2
θ3
Happ
Sad
Inputs: X1,X2,X3 || Weights: θ1,θ2,θ3 || Outputs: Happy or Sad

Logistic Regression is used when the dependent variable (target) is categorical.

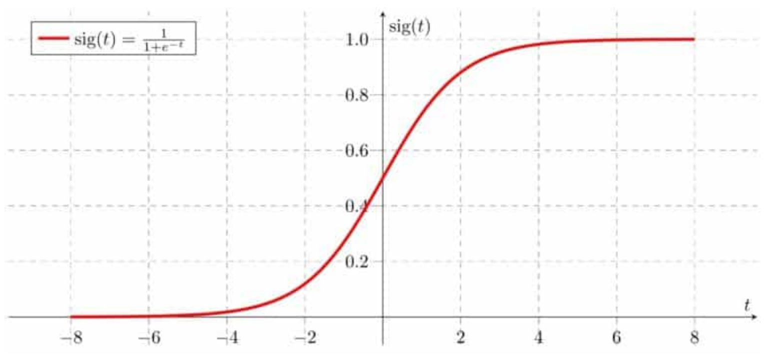
Model

Output = 0 or 1

Z=WX+B

h(x) = sigmoid (Z)

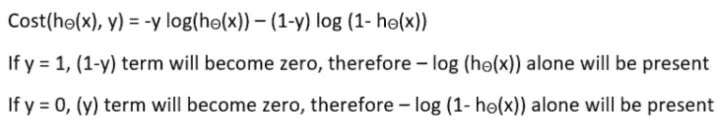
h(x) = log(P(X) / 1 – P(X)) = WX +B



If 'Z' goes to infinity, Y(predicted) will become 1, and if 'Z' goes to negative infinity, Y(predicted) will become 0.

The output from the hypothesis is the estimated probability. This is used to infer how confident can predicted value be actual value when given an input X.

Cost Function



Cost(hΘ(x), y) = -y log(hΘ(x)) – (1-y) log (1- hΘ(x))

If y = 1, (1-y) term will become zero, therefore – log (hΘ(x)) alone will be present

If y = 0, (y) term will become zero, therefore – log (1- hΘ(x)) alone will be present

Cost (h(x) , Y(Actual)) = -log (h(x)) if y=1 -log (1 – h(x)) if y=0

This implementation is for binary logistic regression. For data with more than 2 classes, softmax re gression has to be used.

[0]:
```
#!/usr/bin/env python
# coding: utf-8

# # Supervised Learning Algorithms: Logistic regression

# *In this template, only **data input** and **input/target variables** need to␣
 ↪be specified (see "Data Input & Variables" section for further instructions).␣
 ↪None of the other sections needs to be adjusted. As a data input example, .csv␣
 ↪file from IBM Box web repository is used.*

# ## 1. Libraries

# *Run to import the required libraries.*

get_ipython().run_line_magic('matplotlib', 'notebook')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
```

```python
from sklearn.model_selection import train_test_split


# ## 2. Data Input and Variables


### Data Input Example
df = pd.read_csv('Iris.csv' , error_bad_lines=False)
X = df[['SepalLengthCm', 'SepalWidthCm' , 'PetalLengthCm' , 'PetalWidthCm']]
y = df['Species']


# ## 3. The Model

# *Run to build the model.*

# In[3]:


from sklearn.linear_model import LogisticRegression

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    random_state = 0)

clf = LogisticRegression().fit(X_train, y_train)

print('Accuracy of Logistic regression classifier on training set: {:.2f}'
     .format(clf.score(X_train, y_train)))
print('Accuracy of Logistic regression classifier on test set: {:.2f}'
     .format(clf.score(X_test, y_test)))
```

```
Accuracy of Logistic regression classifier on training set: 0.98
Accuracy of Logistic regression classifier on test set: 0.97
```

[0]:
```python
for this_C in [0.1, 1, 100]:
    clf = LogisticRegression(C=this_C).fit(X_train, y_train)
    print('Logistic Regression with C = {}'.format(this_C))
    print('Accuracy of Logistic regression classifier on training set: {:.2f}'
         .format(clf.score(X_train, y_train)))
    print('Accuracy of Logistic regression classifier on test set: {:.2f}\n'
         .format(clf.score(X_test, y_test)))
```

```
Logistic Regression with C = 0.1
Accuracy of Logistic regression classifier on training set: 0.93
Accuracy of Logistic regression classifier on test set: 0.92

Logistic Regression with C = 1
Accuracy of Logistic regression classifier on training set: 0.98
Accuracy of Logistic regression classifier on test set: 0.97

Logistic Regression with C = 100
Accuracy of Logistic regression classifier on training set: 0.99
Accuracy of Logistic regression classifier on test set: 0.97


/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_logistic.py:940:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```