TensorFlow train and test own data images

Train and test our own images and classify them.

1 . go to git huh url: <https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/image_retraining>

And clone git hub repogitory to your local machine:

<https://github.com/tensorflow/tensorflow.git>

2. go to E:\Git1\tensorflow\tensorflow\examples\image\_retraining folder in mahine.

* Create road\_images and bottlenecks folder in image\_retraining
* And create sub folders in road\_images folder
* Sub folder are bridge, junction and straightroad
* Paste the respective training data into those sub folder , should minimum 20 images

3. refer this url: <https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/#4>

4. keep the test data images into image\_retraining folder

5. run this command for traing own images in the folder road\_images

python retrain.py --bottleneck\_dir=bottlenecks --how\_many\_training\_steps=500 --model\_dir=inception --summaries\_dir=training\_summaries/basic --output\_graph=retrained\_graph.pb --output\_labels=retrained\_labels.txt --image\_dir=road\_images

6. run this command for test your data in image\_retraining folder

python label\_image.py --image 19.jpg --graph retrained\_graph.pb --labels retrained\_labels.txt

TensorFlow Image Captioning

<https://github.com/tensorflow/models/tree/master/im2txt#architecture>

TensorFlow Object Detection

final command:

D:\For-BVP\NLP\object\_detection\models\research\object\_detection>python train.py

--logtostderr --train\_dir=training/ --pipeline\_config\_path=training/ssd\_mobilen

et\_v1\_pets.config

Keras Setup

1. Fallow the below link for Keras setup.

<https://stackoverflow.com/questions/34097988/how-do-i-install-keras-and-theano-in-anaconda-python-on-windows>

2. Install git and set git path to environmental variables:

<https://stackoverflow.com/questions/29957484/how-to-pip-install-from-github-powershell-gives-error-cannot-find-command-gi>

3. Setting the git path

http://www.chambaud.com/2013/07/08/adding-git-to-path-when-using-github-for-windows/

**NLP:**

Code github :

https://github.com/PythonProgramming/NLTK-3----Natural-Language-Processing-with-Python-series

Installing nltk via pip:

1. python -m pip install nltk
2. python -m pip install numpy
3. pip install opencv-python
4. python -m pip install scikit-image

solving errors from this link:

https://stackoverflow.com/questions/26693736/nltk-and-stopwords-fail-lookuperror

1. Method 1 for NLTK installation

] python

>>>> import nltk

>>> nltk.download()

1. Method for NLTK installation

>> pip install nltk

>> python -m nltk.downloader all

This downloads all the nltk data. You can also specify a directory with the option -d. For example

>> python -m nltk.downloader -d /usr/local/share/nltk\_data all

Well, if you are using Python 3, you can replace python in the above command to python3and pip to pip3.

**Project 1: Extracting the actionable sentences from Huge text file**

from nltk.tag import pos\_tag

from nltk.chunk import RegexpParser

import nltk

import random

from nltk.corpus import movie\_reviews

from nltk.tokenize import sent\_tokenize, word\_tokenize

from nltk.corpus import state\_union

from nltk.tokenize import PunktSentenceTokenizer

import shutil

import sys

import io

txt\_opn = open("action.txt",newline='',encoding='utf8')

text1 = txt\_opn.read()

text1 = text1.rstrip()

#print(text1)

words = text1.split()

sent = [w for w in words if not w == 'computer']

sentence = pos\_tag(sent)

sentences1 = sent\_tokenize(text1)

all\_words1 = []

query = [e1 for (e1, rel) in sentence if rel=='VB']

print("actions-start===================================================================================")

print(query)

print("actions-end ====================================================================================")

for w in sentences1:

for j in query:

if j in w:

all\_words1.append(w)

#print("\n")

#print(query)

all\_words1 = set(all\_words1)

#print(all\_words1)

j = 0

for word in all\_words1:

j = j + 1

print(str(j) +". "+ word)

**Project 2: Extracting the positive and negative words in the sentence**

#!/usr/bin/python3

# PYTHON EXAMPLE TO DO SENTIMENT ANALYSIS ON TWEETS

# Author: James Campbell

# Date: 2015-07-01

# Updated: 2015-11-16

# USE FOR PYTHON 3 only

import nltk

import sys

from sys import exit

import pickle

pos\_tweets = [('I love this car', 'positive'),

('This view is amazing', 'positive'),

('I feel great this morning', 'positive'),

('I am so excited about the concert', 'positive'),

('He is my best friend', 'positive'),

('Going well', 'positive'),

('Thank you', 'positive'),

('Hope you are doing well', 'positive'),

('I am very happy', 'positive'),

('Good for you', 'positive'),

('I am a good boy', 'positive'),

('It is all good. I know about it and I accept it.', 'positive'),

('This is really good!', 'positive'),

('Tomorrow is going to be fun.', 'positive'),

('Smiling all around.', 'positive'),

('These are great apples today.', 'positive'),

('How about them apples? Thomas is a happy boy.', 'positive'),

('Thomas is very zen. He is well-mannered.', 'positive')]

neg\_tweets = [('I do not like this car', 'negative'),

('This view is horrible', 'negative'),

('I feel tired this morning', 'negative'),

('I am not looking forward to the concert', 'negative'),

('He is my enemy', 'negative'),

('Escalated', 'negative'),

('I am a bad boy', 'negative'),

('This is not good', 'negative'),

('I am bothered by this', 'negative'),

('I am not connected with this', 'negative'),

('Sadistic creep you ass. Die.', 'negative'),

('All sorts of crazy and scary as hell.', 'negative'),

('Not his emails, no.', 'negative'),

('His father is dead. Returned obviously.', 'negative'),

('He has a bomb.', 'negative'),

('Too fast to be on foot. We cannot catch them.', 'negative')]

tweets = []

for (words, sentiment) in pos\_tweets + neg\_tweets:

words\_filtered = [e.lower() for e in words.split() if len(e) >= 3]

tweets.append((words\_filtered, sentiment))

def get\_words\_in\_tweets(tweets):

all\_words = []

for (words, sentiment) in tweets:

all\_words.extend(words)

return all\_words

def get\_word\_features(wordlist):

wordlist = nltk.FreqDist(wordlist)

word\_features = wordlist.keys()

return word\_features

def extract\_features(document):

document\_words = set( )

features = {}

for word in word\_features:

features['contains(%s)' % word] = (word in document\_words)

return features

word\_features = get\_word\_features(get\_words\_in\_tweets(tweets))

training\_set = nltk.classify.apply\_features(extract\_features, tweets)

classifier = nltk.NaiveBayesClassifier.train(training\_set)

# optional to save your classifier so you can load it elsewhere without having to rebuild training set every time

save\_classifier = open("tweetposneg.pickle","wb")

pickle.dump(classifier, save\_classifier)

save\_classifier.close()

# optional load from classifier that was saved previously

# classifier\_f = open("naivebayes.pickle", "rb")

# classifier = pickle.load(classifier\_f)

# classifier\_f.close()

runtweets = [] # setup to import a list of tweets here if you wish into a python list

if len(sys.argv) > 1: # if param passed 4 name of text file w/ list of tweets

tweetfile = sys.argv[1]

with open(tweetfile, "r") as ins:

for line in ins:

runtweets.append(line)

runtweets.append('Why the silence? It’s been 2 business days since Johanna Escalated this with no response. Looks bad.') # test tweet incase

poscount = 0

negcount = 0

for tweett in runtweets:

valued = classifier.classify(extract\_features(tweett.split()))

print (valued)

if valued == 'negative':

negcount = negcount + 1

else:

poscount = poscount + 1

print ('Positive count: %s \nNegative count: %s' % (poscount,negcount))

exit()

**Project 3: Extracting the positive and negative words in probability:**

import nltk

from nltk.probability import FreqDist, ELEProbDist

from nltk.classify.util import apply\_features,accuracy

def get\_words\_in\_tweets(tweets):

all\_words = []

for (words, sentiment) in tweets:

all\_words.extend(words)

return all\_words

def get\_word\_features(wordlist):

wordlist = FreqDist(wordlist)

word\_features = wordlist.keys()

return word\_features

pos\_tweets=[('I love this car','positive'),

('This view is amazing','positive'),

('I feel great this morning','positive'),

('I am so excited about the concert','positive'),

('He is my best friend','positive')]

neg\_tweets=[('I do not like this car','negative'),

('This view is horrible','negative'),

('I feel tired this morning','negative'),

('I am not looking forward to the concert','negative'),

('He is my enemy','negative')]

tweets=[]

for(words,sentiment)in pos\_tweets+neg\_tweets:

words\_filtered=[e.lower() for e in words.split() if len(e)>=3]

tweets.append((words\_filtered,sentiment))

test\_pos\_tweets=[('I feel happy this morning','positive'),

('Larry is my friend','positive')]

test\_neg\_tweets=[('I do not like that man','negative'),

('This view is horrible','negative'),

('The house is not great','negative'),

('Your song is annoying','negative')]

test\_tweets=[]

for(test\_words,test\_sentiment)in test\_pos\_tweets+test\_neg\_tweets:

test\_words\_filtered=[e.lower() for e in test\_words.split() if len(e)>=3]

test\_tweets.append((test\_words\_filtered,test\_sentiment))

word\_features = get\_word\_features(get\_words\_in\_tweets(tweets))

def extract\_features(document):

document\_words = set(document)

features = {}

for word in word\_features:

features['contains(%s)' % word] = (word in document\_words)

return features

training\_set = apply\_features(extract\_features, tweets)

test\_training\_set=apply\_features(extract\_features, test\_tweets)

classifier = nltk.classify.NaiveBayesClassifier.train(training\_set)

tweet = 'The house is not great'

print (classifier.classify(extract\_features(tweet.split())))

classifier.show\_most\_informative\_features(5)

print (nltk.classify.util.accuracy(classifier,test\_training\_set))

OpenCv:

Installation :

1. pip install opencv-python

links:

1. <http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html>
2. <http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html>
3. http://opencv-python-tutroals.readthedocs.io/en/latest/py\_tutorials/py\_feature2d/py\_table\_of\_contents\_feature2d/py\_table\_of\_contents\_feature2d.html

code: finding corner detections

import numpy as np

import cv2

img = cv2.imread('sample2.png')

gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

gray = np.float32(gray)

corners = cv2.goodFeaturesToTrack(gray, 100, 0.01, 10)

corners = np.int0(corners)

for corner in corners:

x,y = corner.ravel()

cv2.circle(img,(x,y),3,255,-1)

cv2.imwrite('subpixel5.png',img)

cv2.imshow('Corner',img)

cv2.waitKey(10000)

cv2.destroyAllWindows()

code : findind edge detection:

import cv2

import numpy as np

filename = 'sample2.png'

img = cv2.imread(filename)

gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

gray = np.float32(gray)

dst = cv2.cornerHarris(gray,2,3,0.08)

#result is dilated for marking the corners, not important

dst = cv2.dilate(dst,None)

# Threshold for an optimal value, it may vary depending on the image.

img[dst>0.01\*dst.max()]=[0,0,255]

cv2.imshow('dst',img)

if cv2.waitKey(0) & 0xff == 27:

cv2.destroyAllWindows()

imp link:

https://pythonprogramming.net/drawing-writing-python-opencv-tutorial/

**Finding the vectors on image(like lines):**

import numpy as np

from PIL import Image

from PIL import Image

import os, sys

img = Image.open('D:/For BVP/NLP/map\_vector.png').convert('RGBA')

arr = np.array(img)

# record the original shape

shape = arr.shape

# make a 1-dimensional view of arr

flat\_arr = arr.ravel()

# convert it to a matrix

vector = np.matrix(flat\_arr)

# do something to the vector

vector[:,::10] = 128

# reform a numpy array of the original shape

arr2 = np.asarray(vector).reshape(shape)

# make a PIL image

img2 = Image.fromarray(arr2, 'RGBA')

img2.show()

**Clustring**

**K Means Clustering:**

setwd("D:/data")

getwd()

data = read.csv("data.csv")

View(data)

library(ggplot2)

mydata <- data

wss <- (nrow(mydata)-1)\*sum(apply(mydata,2,var))

for (i in 2:15) wss[i] <- sum(kmeans(mydata,

centers=i)$withinss)

plot(1:15, wss, type="b", xlab="Number of Clusters",

ylab="Within groups sum of squares",

main="Assessing the Optimal Number of Clusters with the Elbow Method",

pch=20, cex=2)

set.seed(7)

km2 = kmeans(data, 6, nstart=100)

plot(data, col =(km2$cluster +1) , main="K-Means result with 6 clusters", pch=20, cex=2)

**3d data classification by kmeans:**

data <- read.csv("D:/data/data.csv")

# Reading LAS file

# Setring the xyz coordinates and subsetting the data

xyz<-subset(data)

# Finding clusters

clLAS<-kmeans(xyz, 3)

# Set the id vector

id<-as.factor(clLAS$cluster)

data$shape <- id

head(data)

table(data$shape)

tail(data)

**SCClust:**

**Project 1: Find the specific size clusters using the SCClustering**

setwd("D:/data")

getwd()

data = read.csv("Associate\_lat\_lon2.csv")

library(ggplot2)

library(scclust)

my\_data <- data[,c(-1,-2,-3)]

mydata1 <- my\_data

# Construct distance metric

my\_dist <- distances(my\_data,

dist\_variables = c("Lat", "Lon"))

# Make clustering with at least 3 data points in each cluster

my\_clustering <- sc\_clustering(my\_dist, 16)

# Check so clustering satisfies constraints

check\_clustering(my\_clustering, 16)

# > TRUE

# Get statistics about the clustering

get\_clustering\_stats(my\_dist, my\_clustering)

# Make clustering with at least 8 points in total of which at least

# one must be "A", two must be "B" and five can be any type

my\_refined\_clustering <- hierarchical\_clustering(my\_dist,

size\_constraint = 16,

existing\_clustering = my\_clustering)

# Make clustering from scratch with `hierarchical\_clustering`

my\_other\_clustering <- hierarchical\_clustering(my\_dist, 16)

data$group <- my\_other\_clustering

write.csv(data,"Sol\_Associate\_lat\_lon.csv")

#Ploting

plot(my\_other\_clustering)

points(my\_other\_clustering, col = 1:65, pch = 8, cex = 2)

#frequency finding

table(my\_refined\_clustering)

table(my\_other\_clustering)

**Finalised code:**

setwd("D:/data")

getwd()

data = read.csv("Associate\_lat\_lon\_general.csv")

library(ggplot2)

library(scclust)

my\_data <- data[,c(-1,-2,-3,-6)]

mydata1 <- my\_data

resourcesize = 16

# Construct distance metric

my\_dist <- distances(my\_data,

dist\_variables = c("Lat", "Lon"))

# Make clustering with at least 3 data points in each cluster

my\_clustering <- sc\_clustering(my\_dist, resourcesize)

# Check so clustering satisfies constraints

check\_clustering(my\_clustering, resourcesize)

# > TRUE

# Get statistics about the clustering

get\_clustering\_stats(my\_dist, my\_clustering)

# Make clustering with at least 8 points in total of which at least

# one must be "A", two must be "B" and five can be any type

my\_refined\_clustering <- hierarchical\_clustering(my\_dist,

size\_constraint = resourcesize,

existing\_clustering = my\_clustering)

# Make clustering from scratch with `hierarchical\_clustering`

my\_other\_clustering <- hierarchical\_clustering(my\_dist, resourcesize)

data$group <- my\_other\_clustering

#write.csv(data[,-1],"Sol\_1.csv")

#Ploting

plot(my\_other\_clustering)

points(my\_other\_clustering, col = 1:65, pch = 8, cex = 2)

#frequency finding

#frequency table

x <- table(my\_other\_clustering)

y <- as.data.frame(x)

#size of cluster which has length more than 16

z <- y$Freq[y$Freq>resourcesize]

# name of cluster which has length more than 16

z1<-which(y$Freq == z)

z2 <- z1-1

#data not having cluster size more than 16

data1 <- data[data$group!=z2,]

#data having cluster size more than 16

data2 <- data[data$group==z2,]

#data1 clustering

# Construct distance metric

my\_dist1 <- distances(data1,

dist\_variables = c("Lat", "Lon"))

# Make clustering from scratch with `hierarchical\_clustering`

my\_other\_clustering\_data1 <- hierarchical\_clustering(my\_dist1, resourcesize)

table(my\_other\_clustering)

#making data1 CSV file

data1$group <- my\_other\_clustering\_data1

final\_data1 <- data1[,-1]

#write.csv(data[,-1],"Sol\_1.csv")

#data2 clustering

lendata2 <- nrow(data2)

clusterdata2size <- lendata2 - resourcesize

my\_dist2 <- distances(data2,

dist\_variables = c("Lat", "Lon"))

# Make clustering from scratch with `hierarchical\_clustering`

my\_other\_clustering\_data2 <- hierarchical\_clustering(my\_dist2, clusterdata2size)

table(my\_other\_clustering\_data2)

#data frame for frequency table of data2

table\_my\_other\_clustering\_data2 <- table(my\_other\_clustering\_data2)

data\_my\_other\_clustering\_data2 <-as.data.frame(table\_my\_other\_clustering\_data2)

#replacing the cluster size 0,1 with z2 and z1.

my\_other\_clustering\_data2[my\_other\_clustering\_data2 == 0] <- z2

my\_other\_clustering\_data2[my\_other\_clustering\_data2 == 1] <- z1

my\_other\_clustering\_data2 <- as.integer(my\_other\_clustering\_data2)

#making data2 CSV file

data2$group <- my\_other\_clustering\_data2

final\_data2 <- data2[,-1]

#combinig two data frames

final\_data <- rbind(final\_data1, final\_data2)

#writing CSV file

write.csv(final\_data,"Sol\_gereral\_shift.csv",row.names = FALSE)

**Method2:**

setwd("D:/data")

getwd()

data = read.csv("Evening.csv")

library(ggplot2)

library(scclust)

my\_data <- data[,c(-1,-2,-3,-6)]

mydata1 <- my\_data

resourcesize = 16

# Construct distance metric

my\_dist <- distances(my\_data,

dist\_variables = c("Lat", "Lon"))

# Make clustering with at least 3 data points in each cluster

my\_clustering <- sc\_clustering(my\_dist, resourcesize)

# Check so clustering satisfies constraints

check\_clustering(my\_clustering, resourcesize)

# > TRUE

# Get statistics about the clustering

get\_clustering\_stats(my\_dist, my\_clustering)

# Make clustering with at least 8 points in total of which at least

# one must be "A", two must be "B" and five can be any type

my\_refined\_clustering <- hierarchical\_clustering(my\_dist,

size\_constraint = resourcesize,

existing\_clustering = my\_clustering)

# Make clustering from scratch with `hierarchical\_clustering`

my\_other\_clustering <- hierarchical\_clustering(my\_dist, resourcesize)

data$group <- my\_other\_clustering

write.csv(data[,-1],"Sol\_1.csv")

#Ploting

plot(my\_other\_clustering)

points(my\_other\_clustering, col = 1:65, pch = 8, cex = 2)

#frequency finding

#frequency table

x <- table(my\_other\_clustering)

y <- as.data.frame(x)

#size of cluster which has length more than 16

z <- y$Freq[y$Freq>resourcesize]

# name of cluster which has length more than 16

z1<-which(y$Freq == z)

z2 <- z1-1

#data not having cluster size more than 16

data1 <- data[data$group!=z2,]

#data having cluster size more than 16

data2 <- data[data$group==z2,]

#data1 clustering

# Construct distance metric

my\_dist1 <- distances(data1,

dist\_variables = c("Lat", "Lon"))

# Make clustering from scratch with `hierarchical\_clustering`

my\_other\_clustering\_data1 <- hierarchical\_clustering(my\_dist1, resourcesize)

table(my\_other\_clustering)

#making data1 CSV file

data1$group <- my\_other\_clustering\_data1

final\_data1 <- data1[,-1]

#write.csv(data[,-1],"Sol\_1.csv")

#data2 clustering

lendata2 <- nrow(data2)

clusterdata2size <- lendata2 - resourcesize

data2\_split1 <- head(data2,resourcesize)

data2\_split2 <- tail(data2,clusterdata2size)

#making new cluster numbers

data2\_size1 <- nrow(data)

cluster\_number <- floor(data2\_size1/16)

#replacing the cluster size 0,1 with z2 and z1.

data2\_split1$group[data2\_split1$group == z2] <- cluster\_number

data2\_split2$group[data2\_split2$group == z2] <- cluster\_number + 1

final\_data2 <- rbind(data2\_split1, data2\_split2)

final\_data2 <- final\_data2[,-1]

#combinig two data frames

final\_data <- rbind(final\_data1, final\_data2)

#writing CSV file

write.csv(final\_data,"Sol\_Evening\_shift.csv",row.names = FALSE)

**Data Preparation and Missing data treating**

setwd('D:/ML/1\_Data\_Preprocessing')

getwd()

data = read.csv('Data.csv')

#Method1

# Taking care of missing data

data$Age[is.na(data$Age)] <- mean(data$Age,na.rm=TRUE)

data$Salary[is.na(data$Salary)] <- mean(data$Salary,na.rm=TRUE)

#Method2

# Taking care of missing data

dataset$Age = ifelse(is.na(dataset$Age),

ave(dataset$Age, FUN = function(x) mean(x, na.rm = TRUE)),

dataset$Age)

dataset$Salary = ifelse(is.na(dataset$Salary),

ave(dataset$Salary, FUN = function(x) mean(x, na.rm = TRUE)),

dataset$Salary)

**Data Preprocessing**

**Splitting data into training and testing:**

# Data Preprocessing Template

setwd("D:/ML/1\_Data\_Preprocessing")

getwd()

# Importing the dataset

data = read.csv('Data.csv')

# Splitting the dataset into the Training set and Test set

# install.packages('caTools')

library(caTools)

set.seed(123)

#Method1

split <- sample(2,nrow(data),replace = TRUE, prob = c(0.8,0.2))

train\_set <- data[split == 1, ]

test\_set <- data[split == 2, ]

#Method2

split = sample.split(dataset$DependentVariable, SplitRatio = 0.8)

training\_set = subset(dataset, split == TRUE)

test\_set = subset(dataset, split == FALSE)

**Converting data in to factor variables and assign names to dummy variables**

setwd("D:/ML/1\_Data\_Preprocessing")

getwd()

# Importing the dataset

data = read.csv('Data.csv')

# Splitting the dataset into the Training set and Test set

# install.packages('caTools')

library(caTools)

set.seed(123)

#Method1

# Taking care of missing data

data$Age[is.na(data$Age)] <- mean(data$Age,na.rm=TRUE)

data$Salary[is.na(data$Salary)] <- mean(data$Salary,na.rm=TRUE)

# Encoding categorical data

data$Country = factor(data$Country,

levels = c('France', 'Spain', 'Germany'),

labels = c(1, 2, 3))

data$Purchased = factor(data$Purchased,

levels = c('No', 'Yes'),

labels = c(0, 1))

**Simple Linear Regression:**

# Importing the dataset

data = read.csv('D:/ML/3\_Regression/Simple\_Linear\_Regression/Salary\_Data.csv')

# Splitting the dataset into the Training set and Test set

# install.packages('caTools')

set.seed(123)

#Method1

split <- sample(2,nrow(data),replace = TRUE, prob = c(0.8,0.2))

train\_set <- data[split == 1, ]

test\_set <- data[split == 2, ]

# Feature Scaling

# training\_set = scale(training\_set)

# test\_set = scale(test\_set)

# Fitting Simple Linear Regression to the Training set

regressor = lm(formula = Salary ~ YearsExperience,

data = train\_set)

Summary(regressor)

# Predicting the Test set results

y\_pred = predict(regressor, newdata = test\_set)

# Visualising the Training set results

library(ggplot2)

ggplot() +

geom\_point(aes(x = train\_set$YearsExperience, y = train\_set$Salary),

colour = 'red') +

geom\_line(aes(x = train\_set$YearsExperience, y = predict(regressor, newdata = train\_set)),

colour = 'blue') +

ggtitle('Salary vs Experience (Training set)') +

xlab('Years of experience') +

ylab('Salary')

# Visualising the Test set results

library(ggplot2)

ggplot() +

geom\_point(aes(x = test\_set$YearsExperience, y = test\_set$Salary),

colour = 'red') +

geom\_line(aes(x = train\_set$YearsExperience, y = predict(regressor, newdata = train\_set)),

colour = 'blue') +

ggtitle('Salary vs Experience (Test set)') +

xlab('Years of experience') +

ylab('Salary')

**# Multiple Linear Regression**

# Importing the dataset

data = read.csv('D:/ML/3\_Regression/Multiple\_Linear\_Regression/50\_Startups.csv')

# Encoding categorical data

data$State = factor(data$State,

levels = c('New York', 'California', 'Florida'),

labels = c(1, 2, 3))

# Splitting the dataset into the Training set and Test set

# install.packages('caTools')

set.seed(123)

#Method1

split <- sample(2,nrow(data),replace = TRUE, prob = c(0.8,0.2))

train\_set <- data[split == 1, ]

test\_set <- data[split == 2, ]

# Feature Scaling

# training\_set = scale(training\_set)

# test\_set = scale(test\_set)

# Fitting Multiple Linear Regression to the Training set

regressor = lm(formula = Profit ~ .,

data = train\_set)

summary(regressor)

# Predicting the Test set results

y\_pred = predict(regressor, newdata = test\_set)

test\_set$y\_pred <- y\_pred

**Deploying the R code in AzureML platform**

data <- read.csv("Associate\_lat\_lon\_general.csv")

data <- data[,c(-1,-2,-3,-6,-7)]

data <- subset(data,select = c("Lat", "Lon"))

Lat <- data$Lat

Lon <- data$Lon

PredictGroup <- function(dataLat,datalon)

{

data <- cbind(Lat,Lon)

data <- as.data.frame(data)

library(ggplot2)

library(scclust)

my\_data <- data

mydata1 <- my\_data

resourcesize = 16

# Construct distance metric

my\_dist <- distances(my\_data,

dist\_variables = c("Lat", "Lon"))

# Make clustering with at least 3 data points in each cluster

my\_clustering <- sc\_clustering(my\_dist, resourcesize)

# Check so clustering satisfies constraints

check\_clustering(my\_clustering, resourcesize)

# > TRUE

# Get statistics about the clustering

get\_clustering\_stats(my\_dist, my\_clustering)

# Make clustering with at least 8 points in total of which at least

# one must be "A", two must be "B" and five can be any type

my\_refined\_clustering <- hierarchical\_clustering(my\_dist,

size\_constraint = resourcesize,

existing\_clustering = my\_clustering)

# Make clustering from scratch with `hierarchical\_clustering`

my\_other\_clustering <- hierarchical\_clustering(my\_dist, resourcesize)

data$group <- my\_other\_clustering

write.csv(data[,-1],"Sol\_1.csv")

#Ploting

#plot(my\_other\_clustering)

#points(my\_other\_clustering, col = 1:65, pch = 8, cex = 2)

#frequency finding

#frequency table

x <- table(my\_other\_clustering)

y <- as.data.frame(x)

#size of cluster which has length more than 16

z <- y$Freq[y$Freq>resourcesize]

# name of cluster which has length more than 16

z1<-which(y$Freq == z)

z2 <- z1-1

#data not having cluster size more than 16

data1 <- data[data$group!=z2,]

#data having cluster size more than 16

data2 <- data[data$group==z2,]

#data1 clustering

# Construct distance metric

my\_dist1 <- distances(data1,

dist\_variables = c("Lat", "Lon"))

# Make clustering from scratch with `hierarchical\_clustering`

my\_other\_clustering\_data1 <- hierarchical\_clustering(my\_dist1, resourcesize)

table(my\_other\_clustering)

#making data1 CSV file

data1$group <- my\_other\_clustering\_data1

final\_data1 <- data1

#write.csv(data[,-1],"Sol\_1.csv")

#data2 clustering

lendata2 <- nrow(data2)

clusterdata2size <- lendata2 - resourcesize

data2\_split1 <- head(data2,resourcesize)

data2\_split2 <- tail(data2,clusterdata2size)

#making new cluster numbers

data2\_size1 <- nrow(data)

cluster\_number <- floor(data2\_size1/16)

#replacing the cluster size 0,1 with z2 and z1.

data2\_split1$group[data2\_split1$group == z2] <- cluster\_number

data2\_split2$group[data2\_split2$group == z2] <- cluster\_number + 1

final\_data2 <- rbind(data2\_split1, data2\_split2)

final\_data2 <- final\_data2

#combinig two data frames

final\_data <- rbind(final\_data1, final\_data2)

}

print(PredictGroup(Lat,Lon))

library(AzureML)

library(devtools)

ws <- workspace(

id = "596867b445b746c489e82347148e6e28",

auth = "1xInAn1ce05KQV9gbwD4CzgEcwxFhPmbmJ2z/42gdSnwTClRb4p1w8cLDggY0bbIHEx3qpOZw+32wL17x+Ur0g==",

api\_endpoint = "https://studioapi.azureml.net"

)

EmployeeLocationWebservice <- publishWebService(ws,

fun = PredictGroup,

name = "EmployeeLocationWebservice",

inputSchema = data,

data.frame=TRUE,

outputSchema = list(

group = "numeric"

))

head(EmployeeLocationWebservice)

Windows R server setup in centos

Verify installation

rpm -qi microsoft-r-server-packages-9.1.x86\_64

output:

Name : microsoft-r-server-packages-9.1 Relocations: /usr/lib64

Version : 9.1.0

### Start Revo64

* Revo64
* normalizePath(R.home()

output:

"/usr/lib64/microsoft-r/3.3/lib64/R"