MACHINE LEARNING AND DEEP LEARNING

EEPE – 34

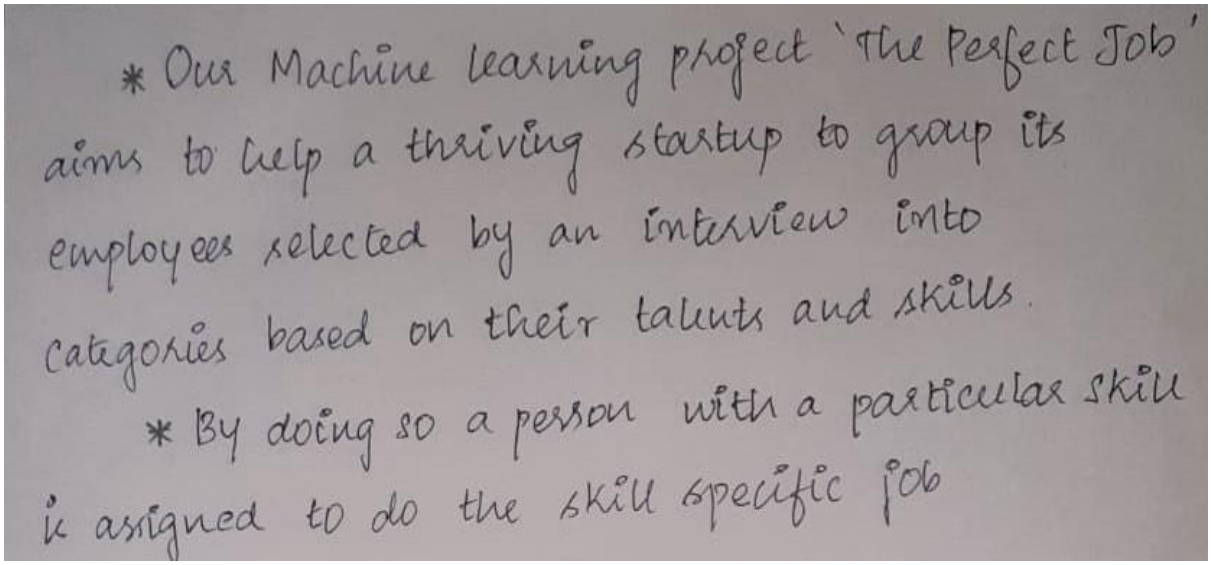
THE PERFECT JOB

GROUP - E

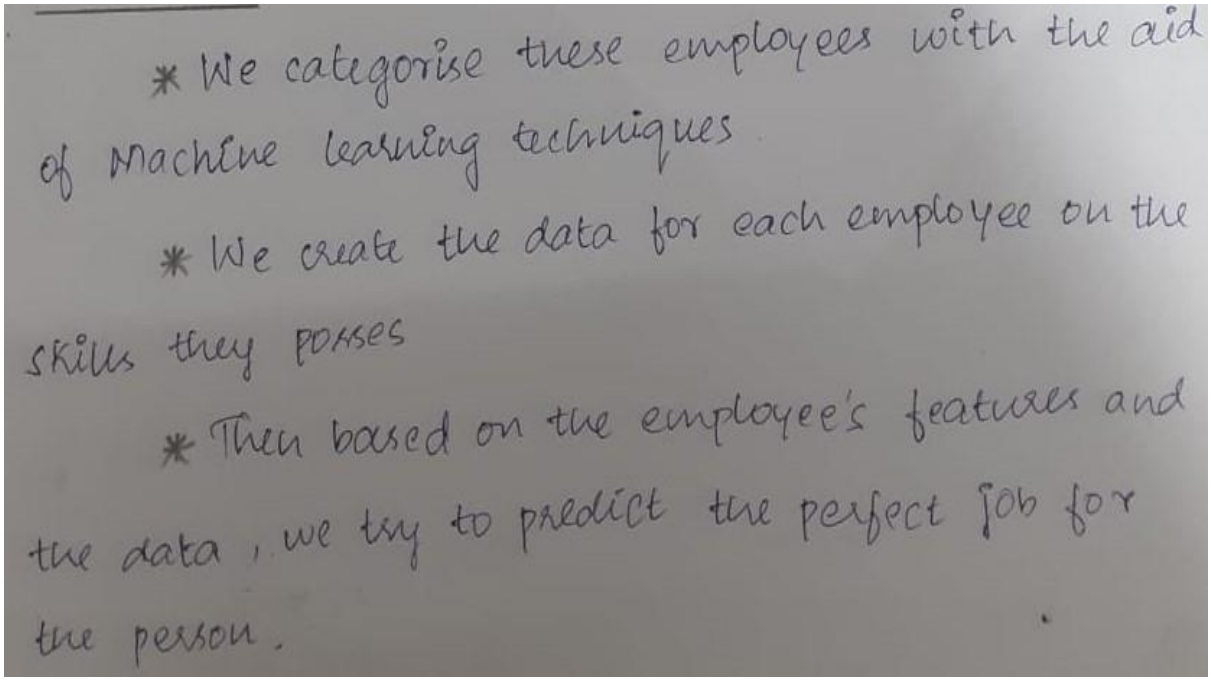
TEAM MEMBERS:

1. Pranav Kumar A V - 107119091
2. Muhilan R - 107119073
3. Prethivan B - 107119093
4. Dinesh Kumar P – 107119087

AIM:



Overview:



Theory:

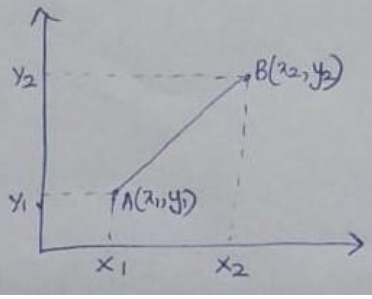
* As we are going to find a suitable role for a person, there are many roles available for him. Implies we are going to have more than two class labels. So this is a multiclass classification.

* We have preferred KNN algorithm to do multiclass classification. Because,
  * We don't have to perform training.
  * It is easy to understand and implement.
  * As there are many distance metrics to find the distance between two points, we can pick any one from it.

  Eg: Euclidean distance, Manhattan distance, Hamming distance, etc.,

KNN Algorithm:

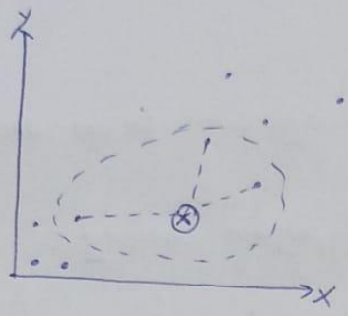
1. First, load the data and select the value for k, ie, the nearest data points.
2. For each data, calculate the distance between this data and each row of rest of the data with many different distance metrics available. We opted for euclidean distance for this project.



$$D_{AB} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

* Take K nearest neighbours as per our calculated euclidean distance.

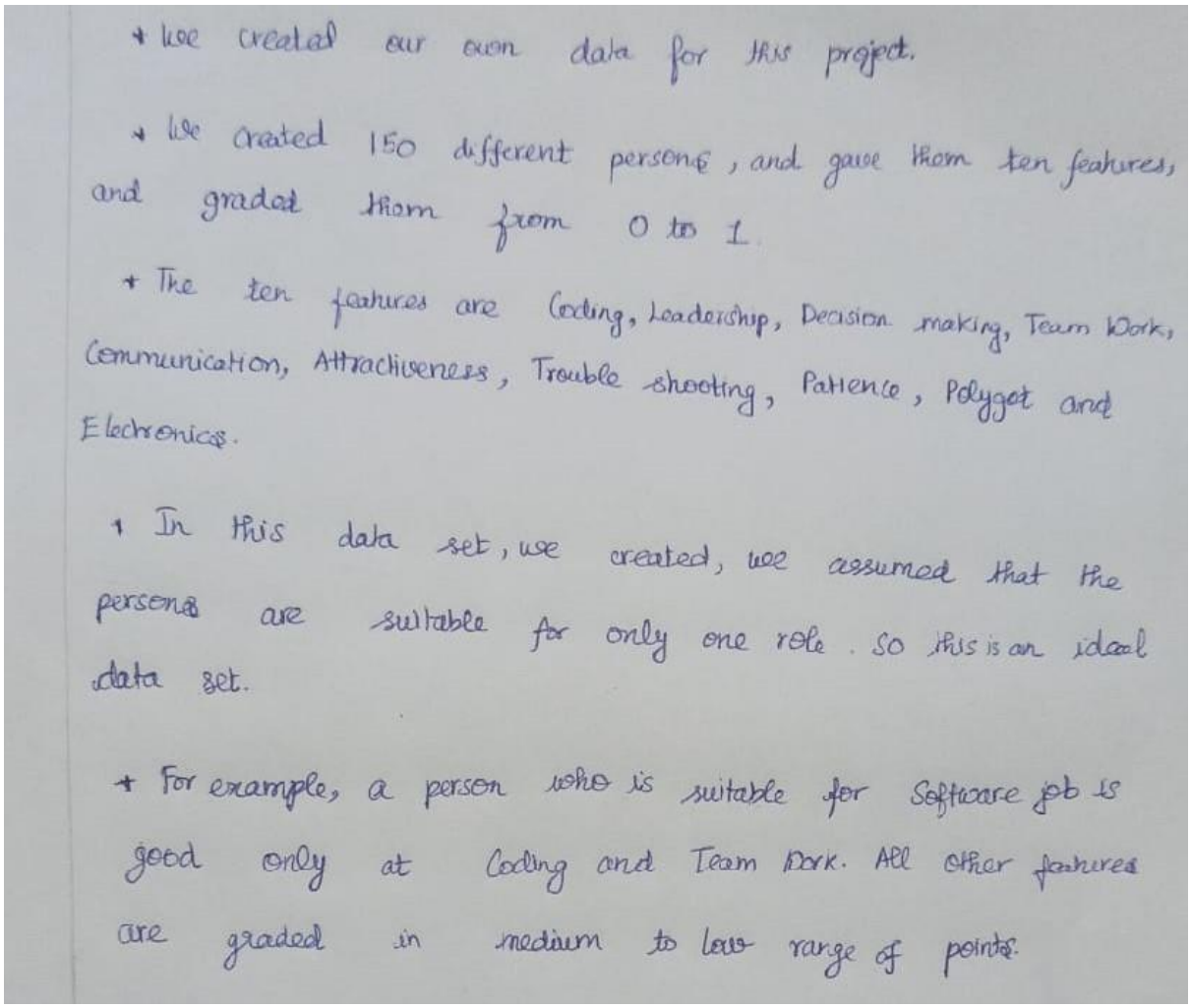
* Among those K neighbours, count the number of data points in each category.



* Our model is ready.

Data:

	A	B	C	D	E	F	G	H	I	J	K	L
1	Name	Coding	Leadership	Decision making	Communication	Attractiveness	Team Work	Trouble Shooting	Patience	Polygot	Electronics	Y
2	Florence	0.03	0.29	0.23	0.72	0.98	0.23	0.15	0.597	0.31	0.06	Receptionist
3	Alec	0.14	0.2	0.3	0.9	0.15	0.11	0.3	0.99	0.94	0.2	Customer Care
4	Joseph	0.98	0.32	0.35	0.25	0.12	0.91	0.64	0.25	0.23	0.2	Software
5	Sylvestr	0.45	0.19	0.21	0.26	0.28	0.562	0.971	0.37	0.23	0.914	Technician
6	Abi	0.11	0.95	0.91	0.55	0.17	0.75	0.14	0.16	0.15	0.12	Management



Code :

```
# Importing all the required libraries
from csv import reader
from math import sqrt
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
import seaborn

#Importing the file from my google drive
!gdown --id 1BWl6uXuorD2ZGvkGb81TkgdjHqhReU0o

# Reading the CSV file
def load_csv(filename):
  X = []
  with open(filename, 'r') as file:
```

```
    csv_reader = reader(file)
    for i in csv_reader:
      if not i:
        continue
      X.append(i)
  return X

# Convert Input column to float
def input_col(X, n):
  for i in X:
    i[n] = float(i[n].strip())         # Removing all the White spaces in all rows of the particular
column and to convert to float

# Convert Output column to integer
def output_col(X, n):
  class_values = [i[n] for i in X]  #Getting all the Class Values
  unique = set(class_values)        #Getting all the unique Class names
  global class_names

  #class_names stores the Class names and its corresponding number
  class_names = {}
  for i, value in enumerate(unique):  #Assigning values in class_names
    class_names[value] = i

  #Converting Class names to its corresponding number
  for i in X:
    i[n] = class_names[i[n]]
  return class_names                #Returning the class names

# Calculate the Euclidean distance between two vectors
def calc_distance(test, train):
  dist = 0.0            #This represents the distance between the 2 rows
  for i in range(len(test)-1):         #Calculating distance
    dist += (test[i] - train[i])**2
  return sqrt(dist)             #Returning the root of the distance value

#Getting the neighbors of the particular Row
def neighbors(train, test, num_neighbors):
  distance = []
  for row in train:
    dist = calc_distance(test, row)    #Calculating Euclidian distances
    distance.append([row[-1], dist])   #Appending the distance with its corresponding class
number
  distance.sort(key=lambda x: x[1])    #Sorting "distances" based on its euclidian distance
  neighbors = []
  for i in range(num_neighbors):     #Saving only the n closest variables
    neighbors.append(distance[i][0])
  return neighbors
```

```
#Predicting the Classes
def prediction(train, test_row, num_neighbors):
  neigh = neighbors(train, test_row, num_neighbors)
  predict = max(set(neigh), key=neigh.count)
  return predict

#Testing the Model
def testing(X_train, X_test, y_test, num_neighbors):
  y_hat = []                    #Will hold the Predicted class
  for row in X_test:              #To predict each data with our model
    label = prediction(X_train, row, num_neighbors)
    y_hat.append(label)
  y_tested = [i-j for i,j in zip(y_test,y_hat)]
  true_val = y_tested.count(0)      #Checking how many are Predicted correctly
  tot_val = len(y_test)
  acc = (true_val/tot_val)*100      #Calculating the Accuracy
  print("Accuracy of the model in testing is : {}".format(acc))

#Getting the data
filename = 'Final_Data.csv'
dataset = load_csv(filename)        #Importing the file for Predicting

#Getting only the values from the data
X=[]
for i in range(len(dataset)-1):    #Storing the dataset in list format
  X.append(dataset[i+1][1:])

for i in range(len(X[0])-1):      #Converting numbers to float
  input_col(X, i)

output_col(X, len(X[0])-1)        #convert class column to integers

num_neighbors = 5                 # define model parameter

#Importing same data in different data structure for plotting
test = pd.read_csv(filename)

#Seperating Input and Output data for plotting
XX = test.loc[:,'Coding':'Electronics']
YY = test.loc[:,'Y']

#Converting Class names to Values for Testing
Y_values=[]
for i in YY:
  if i =='Technician':
    i=class_names[i]
  if i =='Customer Care':
```

```
    i=class_names[i]
  if i =='Software':
    i=class_names[i]
  if i =='Receptionist':
    i=class_names[i]
  if i =='Management':
    i=class_names[i]
  Y_values.append(i)

#Separating the datas to train and test data
X_train, X_test, y_train, y_test = train_test_split(XX, Y_values, test_size = 0.3)

#Converting from Pandas DataFrame to List
X_train,X_test = X_train.values.tolist(),X_test.values.tolist()

#Combining X and Y datas of training
X_y_train = []
for i,j in zip(X_train,y_train):
  i.append(j)
  X_y_train.append(i)
```

#Testing the Model:

```
Metrices = testing(X_y_train,X_test,y_test,num_neighbors)

pca_datas = PCA(n_components=2)   #Doing Dimensionality Reduction to plot
pca_plot = pca_datas.fit_transform(XX)     #Fitting the data to plot

#Naming the columns of reduced data
pca_plot_Df = pd.DataFrame(data = pca_plot, columns = ['Principal Component 1', 'Principal
Component 2'])
pca_plot_Df['y'] = YY            #Including Y values in reduced data

#Printing the 1st four values of Plotting data
pca_plot_Df.head()
```

#Plotting the points:

```
seaborn.set(style='darkgrid')
plt.figure(figsize=(10,10))
g=seaborn.scatterplot(x="Principal Component 1", y="Principal Component   2",
                hue="y",style="y",
                data=pca_plot_Df)
```

#Prediction:

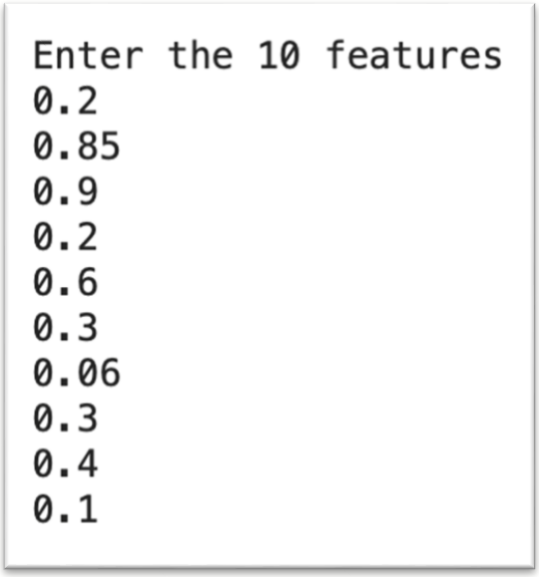
```
#Checking for a Particular model, Sample data for Management-
[0.2,0.85,0.9,0.2,0.6,0.3,0.06,0.3,0.4,0.1]
```

```
print("Enter the 10 features")
row = []
for i in range(10):
  row.append(float(input()))
#Predicting the label number
label = prediction(X, row, num_neighbors)

#Assigning the label number to its corresponding Class name
for i,j in zip(class_names.keys(),class_names.values()):
  if j==label:
    class_label=i

print('So, with these features,the best suitable role for you is "%s"' % (class_label))
```

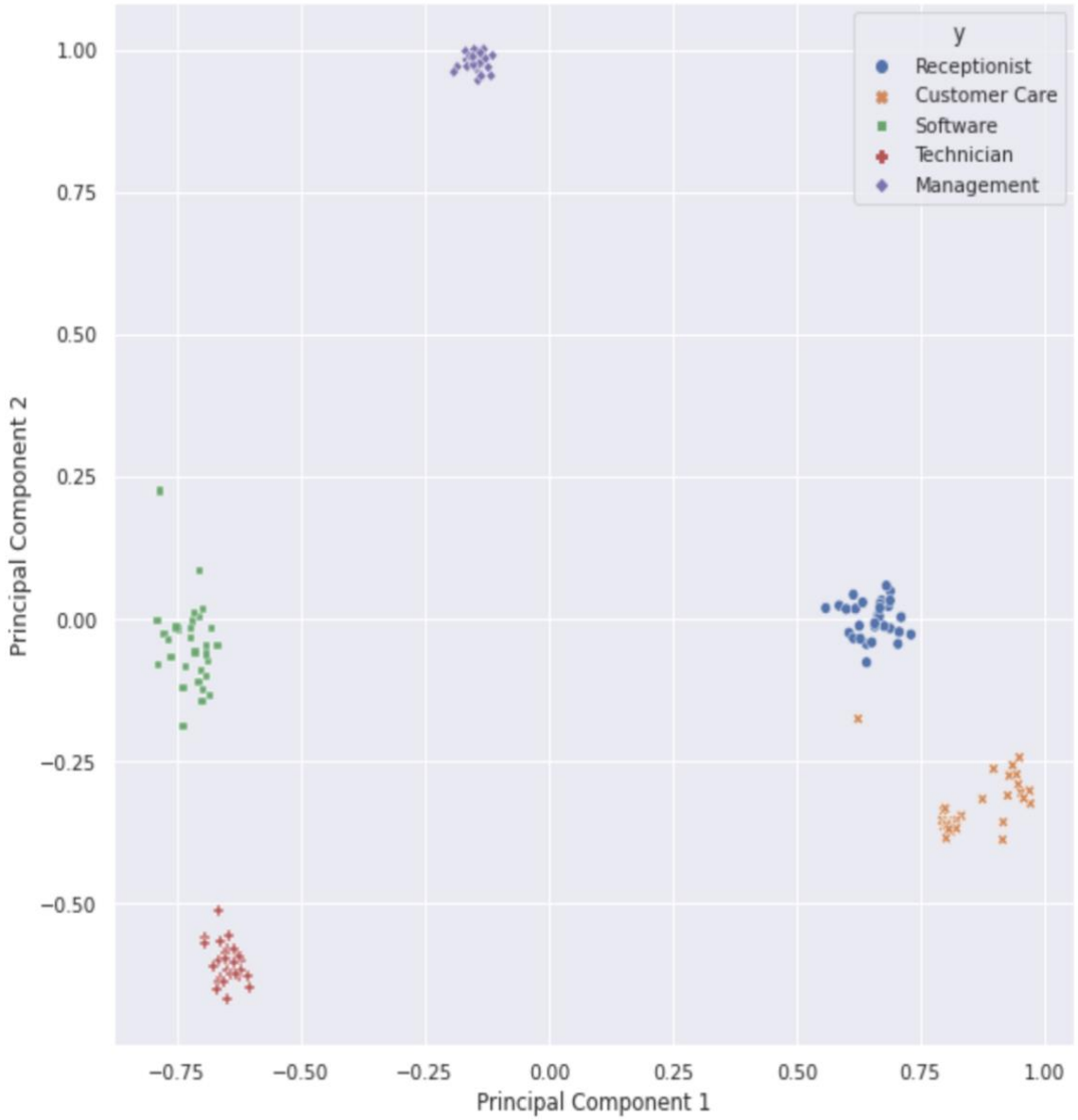
Entering values for Prediction:



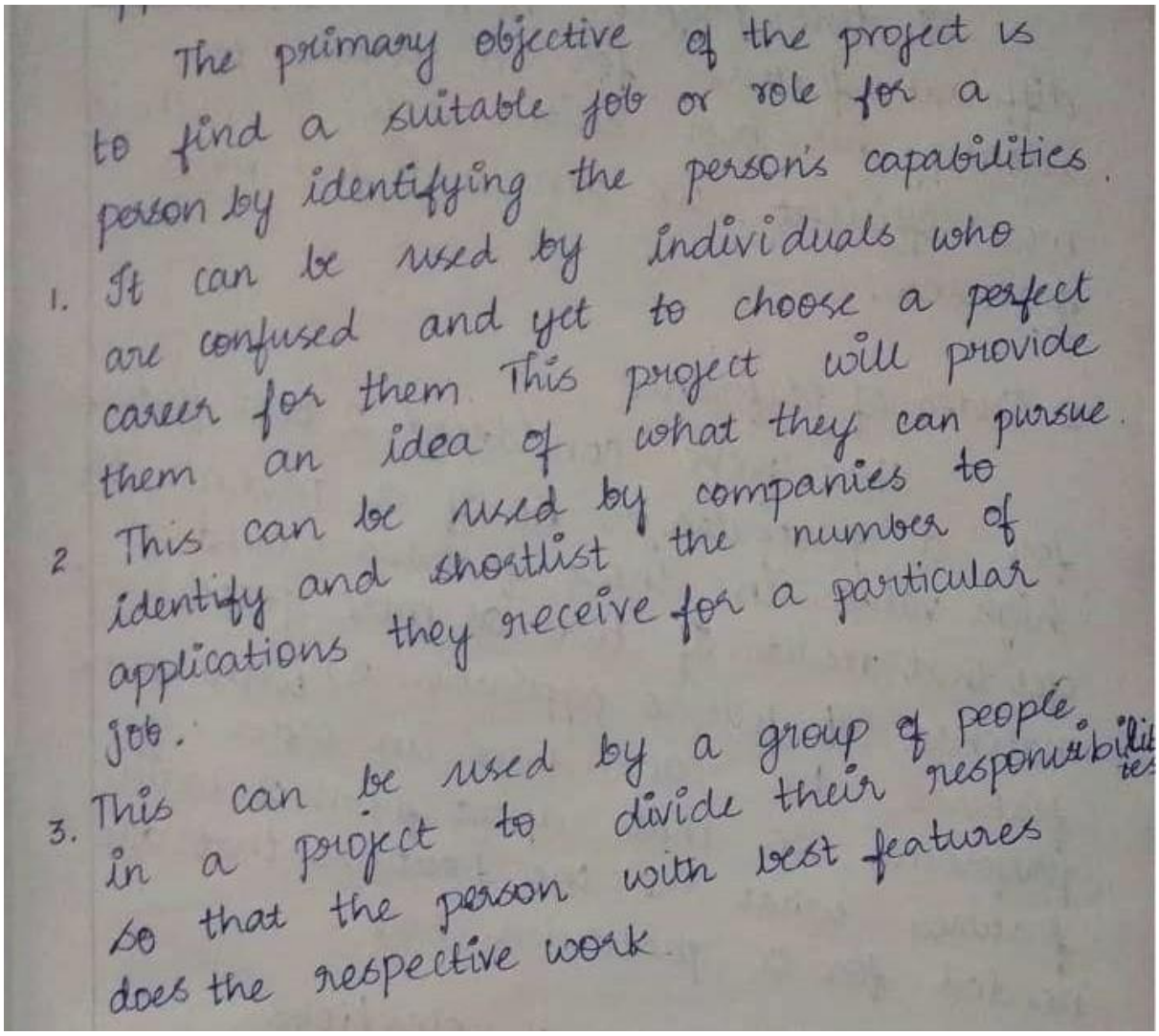
Prediction Output by Machine Learning Algorithm:

Testing Result:

Plot:

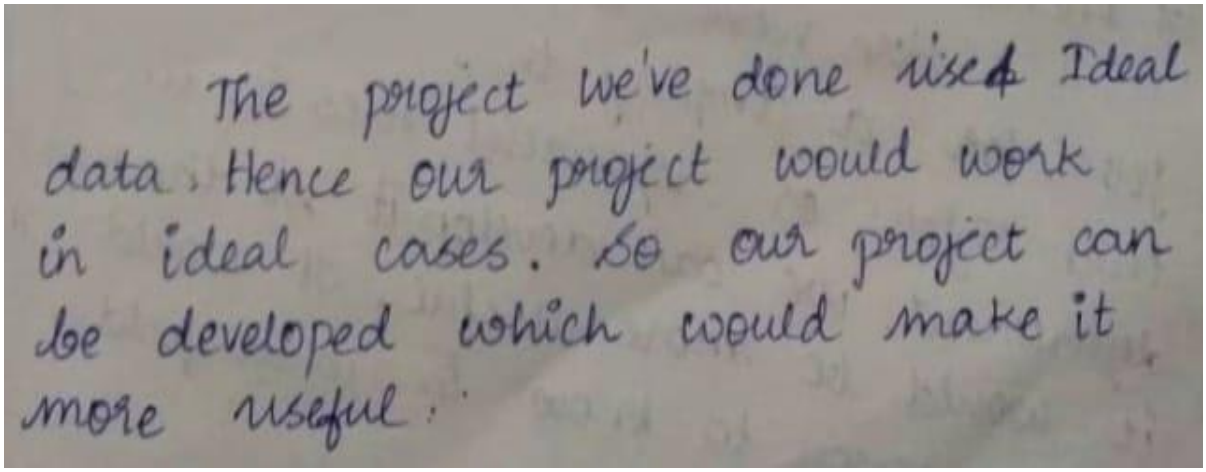

y
Receptionist
Customer Care
Software
Technician
Management
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
Principal Component 2
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
Principal Component 1

Applications:


The primary objective of the project is to find a suitable job or role for a person by identifying the person's capabilities.

1. It can be used by individuals who are confused and yet to choose a perfect career for them. This project will provide them an idea of what they can pursue.
2. This can be used by companies to identify and shortlist the number of applications they receive for a particular job.
3. This can be used by a group of people in a project to divide their responsibilities so that the person with best features does the respective work.

Further Developments:


The project we've done used Ideal data. Hence our project would work in ideal cases. So our project can be developed which would make it more useful.

1. Increase in number of features

In this project we've considered 10 different features for a person to identify his best suited role. To make it more efficient, we can add few more features.

2. Order of preference

Here we've considered 2 best features for a particular job so if a person with high values in those features would get that result. If two or more people's matches (if follows application 2) with same features, we can have an order of preference so that apart from 2 main features what are the features that are needed for a particular role

3. Increase in number of roles/jobs

Here we've considered 5 different jobs. for a company that needs more than 5 jobs or specialist roles in each department, we can develop it so that it would be more useful. It would help a person to know in what field he's a specialist in.