Predicting the possibility of Personality Match using Psychometric Traits

Data Mining and Warehousing

Team Members

1.) C. Anirudh Bhardwaj 14 BCE 1130
2.) Megha Mishra 14 BCE 1252

Prof: Dr. Sweetlin Hemalatha

Abstract

Incompatibility between traits of managers and employees situated at different levels of an organization is an ever prevalent problem. Through this project, we seek to reduce this incompatibility by employing a Psychology based approach which will help us in predicting whether a manager and employee would be compatible or not. We use two approaches to predict the probability of compatibility, namely an Ensemble based on Gradient Boosted Trees and a neural network approach with Multilayer Perceptron Networks. This project uses both these approaches to predict the outcome. The classification occurs on the basis of 6 attributes observed for each individual and one output target variable.

DATA

We have modelled and created a synthetic dataset with the help of Domain Knowledge, based upon DISC theory of classification of individuals based on different traits.

We have created 1000 entries, with 6 attributes each representing the different people and their corresponding attributes. We have a split according to 80:20 split of Test and Train Data.

CODE

The project was created using Juipyter Notebooks running on Anaconda Package Management System. The code was created in Python 2.7 with Numpy, Scikit Learn and Pandas Libraries used.

ATTRIBUTES

The attributes used in the data are: Ambition, adaptability, faith, emotional, decisiveness, dominance

The attributes used for input data into the classifier are:

Person 1: Ambition, adaptability, faith, emotional, decisiveness, dominance

Person 2: Ambition, adaptability, faith, emotional, decisiveness, dominance

And 1 output variable to determine whether the individuals are compatible or not.

Hence, the characteristics of the data are: 12 input variable and 1 output variable

---

Dataset_genreator

```
import matplotlib.pyplot as plt
```

```
import pandas as pd
import numpy as np

ambition= []
adaptibility= []
faith = []
emotional = []
decisivness = []
dominance =[]

for i in range(1,1000,1):

    x1 = int(np.random.randint(1,10))
    x2 = int(np.random.randint(1,10))
    x3 = int(np.random.randint(1,10))
    x4 = int(np.random.randint(1,10))
    x5 = int(np.random.randint(1,10))
    x6 = int(np.random.randint(1,10))



    ambition.append(x1)
    dominance.append(x2)
    decisivness.append(x3)
    adaptibility.append(x4)
    faith.append(x5)
    emotional.append(x6)

data_full = { 'Ambition' : ambition , 'Dominance' : dominance , 'Decisivness' : decisivness ,
'Adaptibility' : adaptibility ,'Faith' : faith , 'Emotional' : emotional }


df = pd.DataFrame(data=data_full)
```

```
df.to_csv('Research_dataset.csv',Index=False)

df.head()
```

Distance_Matrix

```
import pandas as pd
import numpy as np

df = pd.read_csv('Research_dataset.csv')

df.drop(df.columns[[0]], axis=1,inplace=True)

s = (1000,1000)

v = np.zeros(s)

val = pd.DataFrame(data=v)

df.head()

def func():
    for i in df.index:

        for j in df.index:

            x14 = y11 = 10 - df.iloc[i,0]      #Adaptibility
```

```
        x11 = y12 = 10 - df.iloc[i,1]      #Ambition
        x12 = y13 = 10 - df.iloc[i,2]      #Desicveness
        x13 = y14 = 10 - df.iloc[i,3]      #Dominance
        x15 = y15 = 10 - df.iloc[i,4]      #Emotional
        x16 = y16 = 10 - df.iloc[i,5]      #faith

        x24 = y21 = df.iloc[j,0]      #Adaptibility
        x21 = y22 = df.iloc[j,1]      #Ambition
        x22 = y23 = df.iloc[j,2]      #Desicveness
        x23 = y24 = df.iloc[j,3]      #Dominance
        x25 = y25 = df.iloc[j,4]      #Emotional
        x26 = y26 = df.iloc[j,5]      #faith

        forumla = (y11-y21)*(y11-y21) + (y12-y22)*(y12-y22) + (y13-y23)*(y13-y23)+ (y14-
y24)*(y14-y24) + (y15-y25)*(y15-y25) + (y16-y26)*(y16-y26);
        val.loc[i,j] =  forumla**(1/2.0);



import time
start_time = time.time()
print(start_time)

func()

val.to_csv('Utility_val.csv')

end_time = time.time()
print("--- %s seconds ---" % ( end_time - start_time))

val.head()
```

```
val.info()

val.describe()
```

Binarising_Utility

```
import pandas as pd
import numpy as np
df = pd.read_csv('Utility_val.csv')

df.drop(df.columns[[0]], axis=1,inplace=True)
s = (1000,1000)

v = np.zeros(s)

val = pd.DataFrame(data=v)
df.head()



thresholf_val=3.606;



def func(threshold_val,count):
    for i in df.index:
        count = count + 1
        for j in df.index:

            if(df.iloc[i,j] <threshold_val):
```

```
            forumla = 1
            val.loc[i,j] = forumla


        elif(df.iloc[i,j]>=threshold_val):

            forumla = 0
            val.loc[i,j] = forumla

Th_val = 3.606              #Setting Threshold value
count = 0;

import time
start_time = time.time()

func(Th_val,count)

val.to_csv('Binarized_Utility_val.csv')

end_time = time.time()
print("--- %s seconds ---" % ( end_time - start_time))

val.info()

val.describe()
```

Data_for_algorithms

```
import pandas as pd
import numpy as np
df2 = pd.read_csv('Binarized_Utility_val.csv')

df2.drop(df2.columns[[0]], axis=1,inplace=True)
df2.head()
df = pd.read_csv('Research_dataset.csv')

df.drop(df.columns[[0]], axis=1,inplace=True)

df.head()

i1 = []
i2 = []
i3 = []
i4 = []
i5 = []
i6 = []
i7 = []
i8 = []
i9 = []
i10 = []
i11 = []
i12 = []
Op = []

def func():
    for i in df.index:
```

```
for j in df.index:

    x14 = y11 = df.iloc[i,0]      #Adaptibility
    x11 = y12 = df.iloc[i,1]      #Ambition
    x12 = y13 = df.iloc[i,2]      #Desicveness
    x13 = y14 = df.iloc[i,3]      #Dominance
    x15 = y15 = df.iloc[i,4]      #Emotional
    x16 = y16 = df.iloc[i,5]      #faith

    x24 = y21 = df.iloc[j,0]      #Adaptibility
    x21 = y22 = df.iloc[j,1]      #Ambition
    x22 = y23 = df.iloc[j,2]      #Desicveness
    x23 = y24 = df.iloc[j,3]      #Dominance
    x25 = y25 = df.iloc[j,4]      #Emotional
    x26 = y26 = df.iloc[j,5]      #faith

    x_val = df2.iloc[i,j]



    i1.append(x11)
    i2.append(x12)
    i3.append(x13)
    i4.append(x14)
    i5.append(x15)
    i6.append(x16)
    i7.append(x21)
    i8.append(x22)
    i9.append(x23)
    i10.append(x24)
    i11.append(x25)
```

```
        i12.append(x26)
        Op.append(x_val)
```

```
import time
start_time = time.time()

func()

NN_val_data = {'i1': i1,'i2': i2,'i3': i3,'i4': i4,'i5': i5,'i6': i6,'i7': i7,'i8': i8,'i9': i9,'i10': i10,'i11': i11,'i12': i12,'Op': Op}

NN_val = pd.DataFrame(data=NN_val_data)

NN_val.to_csv('Data_for_ML.csv')

end_time = time.time()
print("--- %s seconds ---" % ( end_time - start_time))

NN_val.info()

NN_val.describe()
```

Classifiers

```
import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.ensemble import VotingClassifier
from sklearn.externals import joblib
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import log_loss

df = pd.read_csv("data_for_ML.csv")
df.drop(df[[0]],axis=1,inplace=True)

df.head()

X = df.drop(['Op'],axis=1).values
y = df['Op'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42
,stratify=y)

steps_GBC = [('ss',StandardScaler()),('pca', PCA()),('GradientBoosterClassifier',
GradientBoostingClassifier())]

GBC = Pipeline(steps_GBC)
GBC.fit(X_train,y_train)
pred_gbc = GBC.predict(X_test)
```

```
pred_gbc_proba = GBC.predict_proba(X_test)

score_gbc = GBC.score(X_test,y_test)
print score_gbc

print(confusion_matrix(y_test,pred_gbc))
print(classification_report(y_test,pred_gbc))
print log_loss(y_test,pred_gbc_proba)

joblib.dump(GBC, 'GBC.pkl')

from sklearn.neural_network import MLPClassifier
steps_MLP = [('ss',StandardScaler()),('pca', PCA()),('MultilayerPerceptronNetwork',
MLPClassifier(hidden_layer_sizes=(64,64,64,64)))]

MLP = Pipeline(steps_MLP)
MLP.fit(X_train,y_train)
pred_mlp = MLP.predict(X_test)
pred_mlp_proba = MLP.predict_proba(X_test)

score_mlp = MLP.score(X_test,y_test)
print score_mlp

print(confusion_matrix(y_test,pred_mlp))
print(classification_report(y_test,pred_mlp))
print log_loss(y_test,pred_mlp_proba)

joblib.dump(MLP, 'MLP.pkl')
```

Output

Gradient Boosting Classifier

```
Accuracy = 0.989108271001

Confusion Matrix:

        [[197401      7]
        [  2167     26]]

Full Classification Report:
     precision    recall  f1-score   support

        0.0       0.99      1.00      0.99    197408
        1.0       0.79      0.01      0.02      2193

avg / total       0.99      0.99      0.98    199601

Log Loss Score  = 0.0321866388263

File Name:  ['GBC.pkl']
```

Multilayer Perceptron Network

```
Accuracy = 0.999113230896
Confusion Matrix:

        [[197361     47]
        [   130   2063]]
             precision    recall  f1-score   support

        0.0       1.00      1.00      1.00    197408
        1.0       0.98      0.94      0.96      2193

avg / total       1.00      1.00      1.00    199601

Log Loss Score = 0.00219309601165

File Name: ['MLP.pkl']
```