Project Report

November 21, 2021

0.0.1 Introduction

In this project, I implemented artificial neural nets to predict the career of an individual on the basis of features provided. I used pandas library to read and peform operations CSV files and scikit for training models and retrieving data.I tried to make the model more accurate by making tweaks in the data. I merged some features, dropped some, or encoded them to gain accuracy.

0.0.2 Code

The code of the program is in a file 'Code.ipynb' attached along with this file. To run the program, open the file in a Jupyter notebook and then run the entire kernel. It will become more clear with the below explanations and codes.

In the below code, I have imported the 'pandas' library and MLPClassifier from the 'sklearn'(scikit) library. I have imported some other modules too from the sklearn which are used in he code below.

[1]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix
import pandas as pd
import numpy as np
import random
```

In the below code, I have read the csv file into a pandas dataframe df. The pandas dataframe makes many operations on the data convenient and fast.

[2]:
```
path = 'AI/ML/data.csv'
df = pd.read_csv(path)
```

This is an object of label encoder(). It will be used further in code to change objects of other data types to integer data type.

[3]:
```
le = LabelEncoder()
```

Here, I merged several roles and provided generalisation to the various classes. I generalised all the suggested jobs to 7 general roles. The new roles are related to their chidren and contains them as a whole. It will help in training the model and getting desired output.

[4]:
```
df = df.replace(to_replace =["Business Intelligence Analyst", "Business Systems␣
↪Analyst","CRM Business Analyst","E-Commerce Analyst","Systems␣
↪Analyst","Systems Security Administrator"], value ="Analyst")

df=df.replace(to_replace =["CRM Technical Developer","Mobile Applications␣
↪Developer","Web Developer"],value="Applications Developer")

df=df.replace(to_replace=["Data Architect","Database Administrator","Database␣
↪Developer","Database Manager"],value="Database Engineer")

df=df.replace(to_replace=["Design & UX","UX Designer"],value="Applications␣
↪Developer")

df=df.replace(to_replace=["Information Security Analyst","Information␣
↪Technology Auditor","Information Technology Manager"],value="Analyst")

df=df.replace(to_replace=["Network Engineer","Network Security␣
↪Administrator","Network Security Engineer"],value="Technical Engineer")

df=df.replace(to_replace=["Portal Administrator","Programmer Analyst","Project␣
↪Manager","Quality Assurance Associate"],value="Project Manager")

df=df.replace(to_replace=["Software Developer","Software Quality Assurance (QA)␣
↪/ Testing","Software Systems Engineer","Solutions␣
↪Architect"],value="Software Engineer")

df=df.replace(to_replace=["Technical Services/Help Desk/Tech␣
↪Support","Technical Support"],value="Technical Engineer")
```

In this part, I removed the blankspace and other symbols present in the name of the columns using the replace method of columns of a dataframe to underscore as they will provide unnecessary errors while referencing them using their name with dataframes.

[5]:
```
df.columns = [c.replace(' ', '_') for c in df.columns]

df.columns = [c.replace('?', '') for c in df.columns]

df.columns = [c.replace('-', '_') for c in df.columns]

df.columns = [c.replace('/', '_') for c in df.columns]
```

In this part, I merged some columns, applied some other arithmetic operations on others and dropped some unnecessary ones to decrease the number of features which will judge the predicted output. For instance, I added talents tests and olympiad and merged both the columns. I similarly did it for others.

[6]:
```
sum_column = df["talenttests_taken"] + df["olympiads"]

df["talenttests_taken"] = sum_column

df.drop('hackathons', axis=1, inplace=True)

df.drop('olympiads', axis=1, inplace=True)

df.drop("Interested_Type_of_Books",axis=1,inplace=True)

sum_column = df["Extra_courses_did"] + df["certifications"]

df["Extra_courses_did"] = sum_column

df.drop('certifications', axis=1, inplace=True)

df.drop('Taken_inputs_from_seniors_or_elders',axis=1, inplace=True)

df.drop('interested_in_games',axis=1, inplace=True)

df.drop('Job_Higher_Studies',axis=1, inplace=True)

df.drop('In_a_Realtionship',axis=1, inplace=True)

df.drop('Gentle_or_Tuff_behaviour',axis=1, inplace=True)
```

As you can observe, object of label encoder initialised earlier is used here. All the columns present here have data types other than integer. As ANN is a mathematical model, objects will create possible errors while training. I used a label encoder to remove this problem as it will encode the objects and provide them a unique integer value. This integer value will be used to refer these objects in later steps.

[7]:
```
df.can_work_long_time_before_system=le.fit_transform(df.
↪can_work_long_time_before_system)

df.self_learning_capability=le.fit_transform(df.self_learning_capability)

df.Extra_courses_did=le.fit_transform(df.Extra_courses_did)

df.workshops=le.fit_transform(df.workshops)

df.talenttests_taken=le.fit_transform(df.talenttests_taken)

df.reading_and_writing_skills=le.fit_transform(df.reading_and_writing_skills)

df.memory_capability_score=le.fit_transform(df.memory_capability_score)
```

```
df.Interested_subjects=le.fit_transform(df.Interested_subjects)

df.interested_career_area=le.fit_transform(df.interested_career_area)

df.Type_of_company_want_to_settle_in=le.fit_transform(df.
↪Type_of_company_want_to_settle_in)

df.Salary_Range_Expected=le.fit_transform(df.Salary_Range_Expected)

df.Management_or_Technical=le.fit_transform(df.Management_or_Technical)

df.Salary_work=le.fit_transform(df.Salary_work)

df.hard_smart_worker=le.fit_transform(df.hard_smart_worker)

df.worked_in_teams_ever=le.fit_transform(df.worked_in_teams_ever)

df.Introvert=le.fit_transform(df.Introvert)

df.Suggested_Job_Role=le.fit_transform(df.Suggested_Job_Role)
```

In this part of code, I generalised some specific columns which were containing numbers of wide ranges. I generalised them by classyfying into three categories - high, mid and low. I further encoded them as 1,2 and 3 respectively. The targeted columns were those which were containing percentage of person in various domains. Instead of considering numbers from [0,100], I have to consider only three numbers. This will help in better training of the model.

[8]:
```
df.percentage_in_Algorithms=df.percentage_in_Algorithms.floordiv(10)

df.loc[df.percentage_in_Algorithms < 6 , "percentage_in_Algorithms"] = 1

df.loc[df.percentage_in_Algorithms > 8 , "percentage_in_Algorithms"] = 3

df.loc[df.percentage_in_Algorithms >=6 , "percentage_in_Algorithms"] = 2

df.Acedamic_percentage_in_Operating_Systems=df.
↪Acedamic_percentage_in_Operating_Systems.floordiv(10)

df.loc[df.Acedamic_percentage_in_Operating_Systems < 6 ,␣
↪"Acedamic_percentage_in_Operating_Systems"] = 1

df.loc[df.Acedamic_percentage_in_Operating_Systems > 8 ,␣
↪"Acedamic_percentage_in_Operating_Systems"] = 3

df.loc[df.Acedamic_percentage_in_Operating_Systems >=6 ,␣
↪"Acedamic_percentage_in_Operating_Systems"] = 2
```

```
df.Percentage_in_Programming_Concepts=df.Percentage_in_Programming_Concepts.
 ↪floordiv(10)

df.loc[df.Percentage_in_Programming_Concepts < 6 ,␣
 ↪"Percentage_in_Programming_Concepts"] = 1

df.loc[df.Percentage_in_Programming_Concepts > 8 ,␣
 ↪"Percentage_in_Programming_Concepts"] = 3

df.loc[df.Percentage_in_Programming_Concepts >=6 ,␣
 ↪"Percentage_in_Programming_Concepts"] = 2

df.Percentage_in_Software_Engineering=df.Percentage_in_Software_Engineering.
 ↪floordiv(10)

df.loc[df.Percentage_in_Software_Engineering < 6 ,␣
 ↪"Percentage_in_Software_Engineering"] = 1

df.loc[df.Percentage_in_Software_Engineering > 8 ,␣
 ↪"Percentage_in_Software_Engineering"] = 3

df.loc[df.Percentage_in_Software_Engineering >=6 ,␣
 ↪"Percentage_in_Software_Engineering"] = 2

df.Percentage_in_Computer_Networks=df.Percentage_in_Computer_Networks.
 ↪floordiv(10)

df.loc[df.Percentage_in_Computer_Networks < 6 ,␣
 ↪"Percentage_in_Computer_Networks"] = 1

df.loc[df.Percentage_in_Computer_Networks > 8 ,␣
 ↪"Percentage_in_Computer_Networks"] = 3

df.loc[df.Percentage_in_Computer_Networks >=6 ,␣
 ↪"Percentage_in_Computer_Networks"] = 2

df.Percentage_in_Electronics_Subjects=df.Percentage_in_Electronics_Subjects.
 ↪floordiv(10)

df.loc[df.Percentage_in_Electronics_Subjects < 6 ,␣
 ↪"Percentage_in_Electronics_Subjects"] = 1

df.loc[df.Percentage_in_Electronics_Subjects > 8 ,␣
 ↪"Percentage_in_Electronics_Subjects"] = 3
```

```
df.loc[df.Percentage_in_Electronics_Subjects >=6 ,␣
↪"Percentage_in_Electronics_Subjects"] = 2

df.Percentage_in_Computer_Architecture=df.Percentage_in_Computer_Architecture.
↪floordiv(10)

df.loc[df.Percentage_in_Computer_Architecture < 6 ,␣
↪"Percentage_in_Computer_Architecture"] = 1

df.loc[df.Percentage_in_Computer_Architecture > 8 ,␣
↪"Percentage_in_Computer_Architecture"] = 3

df.loc[df.Percentage_in_Computer_Architecture >=6 ,␣
↪"Percentage_in_Computer_Architecture"] = 2

df.Percentage_in_Communication_skills=df.Percentage_in_Communication_skills.
↪floordiv(10)

df.loc[df.Percentage_in_Communication_skills < 6,␣
↪"Percentage_in_Communication_skills"] = 1

df.loc[df.Percentage_in_Communication_skills > 8 ,␣
↪"Percentage_in_Communication_skills"] = 3

df.loc[df.Percentage_in_Communication_skills >=6 ,␣
↪"Percentage_in_Communication_skills"] = 2

df.Percentage_in_Mathematics=df.Percentage_in_Mathematics.floordiv(10)

df.loc[df.Percentage_in_Mathematics < 6 , "Percentage_in_Mathematics"] = 1

df.loc[df.Percentage_in_Mathematics > 8 , "Percentage_in_Mathematics"] = 3

df.loc[df.Percentage_in_Mathematics >=6 , "Percentage_in_Mathematics"] = 2
```

I did a similar task here as I did in the previous case. In this case, I specified the dividends accordingly as they have different ranges and dividing data in a common way remove the effectiveness.

[9]:
```
df.Hours_working_per_day=df.Hours_working_per_day.floordiv(3)

df.Logical_quotient_rating=df.Logical_quotient_rating.floordiv(2)

df.coding_skills_rating=df.coding_skills_rating.floordiv(2)

df.public_speaking_points=df.public_speaking_points.floordiv(2)

df.workshops=df.workshops.floordiv(3)
```

```
df.Interested_subjects=df.Interested_subjects.floordiv(2)

df.Type_of_company_want_to_settle_in=df.Type_of_company_want_to_settle_in.
↪floordiv(3)
```

In the next few lines, I distributed the data into training and testing set with a ratio of 90:10 and then seperated the features and the column to be predicted. I distributed using random to prevent any kind of bias in datasets.

[10]:
```
points = list(range(len(df)))
```

[11]:
```
trainSize = int(0.9*len(df))
testSize  = int(0.1*len(df))
```

[12]:
```
trainingSet = random.sample(points,trainSize)

for x in trainingSet:
    points.remove(x)
```

[13]:
```
testSet = points
TrainingSet = df.drop(testSet)
TestingSet   = df.drop(trainingSet)
```

As you can observe, I removed the 'Suggested_Job_Role' column from the TrainingX and TestingX using drop function and named them as TrainingY and TestingY.

[14]:
```
TrainingY = TrainingSet['Suggested_Job_Role']
TestingY = TestingSet['Suggested_Job_Role']
```

[15]:
```
TrainingX = TrainingSet.drop('Suggested_Job_Role',axis="columns")
TestingX = TestingSet.drop('Suggested_Job_Role',axis="columns")
```

In this step, I am training my model with the training set. I used MLPclassifier of scikit library with 10000 iterations. I tried many other features too but I am getting the best results by letting all of them to default. I specified the iterations specifically as it was unable to converge for the default value. In the next step, I am finding the accuracy of my model by using the inbuilt function 'score'. It provides the accuracy of model by printing a float between 0 and 1.

[16]:
```
classifier = MLPClassifier(max_iter=10000).fit(TrainingX, TrainingY)
```

[17]:
```
print(classifier.score(TestingX, TestingY))
```

```
0.228
```

Now, I am predicting the TestingY using predict method and then I will use it to find the confusion matrix by comparing it with the konown values of TestingY.

[18]:
```
arr=classifier.predict(TestingX)
```

Here, I am using inbulit method to fund confusion matrix. It takes the pedicted and original value of testingY to generate the matrix. By definition a confusion matrix C is such that Cij is equal to the number of observations known to be in group i and predicted to be in group j. So, testingY will act as known and arr will be the one predicted by our model.

[19]:
```
cm=confusion_matrix(TestingY, arr)
```

Now, you can see that a matrix with rows and columns equal to number of classes is generated. This is our required confusion matrix.

[20]:
```
confusion_matrix(TestingY, arr)
```

[20]:
```
array([[297,  53,  19,  18,  50,  66],
       [196,  45,   6,   7,  35,  45],
       [128,  29,   8,   9,  35,  33],
       [117,  22,   9,   8,  25,  31],
       [173,  29,   3,   9,  33,  45],
       [254,  40,   9,   8,  41,  65]])
```

I am using the confusion matrix now to find the class-wise accuracies. In the next line, I used astype function to specify the data type I require. Then, I find the sum and used newaxis funcion of numpy to get aggregate. The required values will be now stored on the diagonals of the matrix. I got these values by using the diagonal function on the matrix. It stored all the values in an array. You can observe an array at the end of the code which represents accuracy of the value corresponding to its index.

[21]:
```
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
```

[22]:
```
cm.diagonal()
```

[22]:
```
array([0.59045726, 0.13473054, 0.03305785, 0.03773585, 0.1130137 ,
       0.1558753 ])
```

Now, I am repeating the same steps for 80:20 ratio for training and testing set. You will observe that the accuracies will decrease with a decrease in training set.

[23]:
```
points1 = list(range(len(df)))
```

[24]:
```
trainSize = int(0.8*len(df))
testSize  = int(0.2*len(df))
```

[25]:
```
trainingSet = random.sample(points1,trainSize)

for x in trainingSet:
    points1.remove(x)
```

[26]:
```
testSet = points1
TrainingSet = df.drop(testSet)
```

```
TestingSet   = df.drop(trainingSet)
```

[27]:
```
TrainingY = TrainingSet['Suggested_Job_Role']
TestingY = TestingSet['Suggested_Job_Role']
```

[28]:
```
TrainingX = TrainingSet.drop('Suggested_Job_Role',axis="columns")
TestingX = TestingSet.drop('Suggested_Job_Role',axis="columns")
```

[29]:
```
classifier = MLPClassifier(max_iter=10000).fit(TrainingX, TrainingY)
```

[30]:
```
print(classifier.score(TestingX, TestingY))
```

```
0.21775
```

[31]:
```
arr=classifier.predict(TestingX)
```

[32]:
```
cm=confusion_matrix(TestingY, arr)
```

[33]:
```
confusion_matrix(TestingY, arr)
```

[33]:
```
array([[490, 124,  37,  30,  57, 251],
       [335,  87,  20,  20,  42, 179],
       [241,  48,  19,   6,  20, 112],
       [209,  57,  18,  16,  25, 145],
       [266,  72,  20,  11,  38, 174],
       [398, 108,  39,  19,  46, 221]])
```

[34]:
```
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
```

[35]:
```
cm.diagonal()
```

[35]:
```
array([0.49544995, 0.12737921, 0.0426009 , 0.03404255, 0.06540448,
       0.26594465])
```

Now, I am repeating the same steps for 70:30 ratio for training and testing set. You will observe that the accuracies will decrease with a decrease in training set. The aaccuracies will be lowest in this case.

[36]:
```
points3 = list(range(len(df)))
```

[37]:
```
trainSize = int(0.6*len(df))
testSize  = int(0.4*len(df))
```

[38]:
```
trainingSet = random.sample(points3,trainSize)

for x in trainingSet:
    points3.remove(x)
```

[39]:
```
testSet = points3
TrainingSet = df.drop(testSet)
TestingSet   = df.drop(trainingSet)
```

[40]:
```
TrainingY = TrainingSet['Suggested_Job_Role']
TestingY = TestingSet['Suggested_Job_Role']
```

[41]:
```
TrainingX = TrainingSet.drop('Suggested_Job_Role',axis="columns")
TestingX = TestingSet.drop('Suggested_Job_Role',axis="columns")
```

[42]:
```
classifier = MLPClassifier(max_iter=10000).fit(TrainingX, TrainingY)
```

[43]:
```
print(classifier.score(TestingX, TestingY))
```

```
0.204375
```

[44]:
```
arr=classifier.predict(TestingX)
```

[45]:
```
cm=confusion_matrix(TestingY, arr)
```

[46]:
```
confusion_matrix(TestingY, arr)
```

[46]:
```
array([[709, 471, 157,  62, 111, 523],
       [390, 310, 116,  56,  74, 355],
       [266, 236,  82,  23,  46, 254],
       [306, 225,  84,  24,  52, 252],
       [383, 297, 115,  37,  70, 305],
       [475, 408, 138,  59,  89, 440]])
```

[47]:
```
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
```

[48]:
```
cm.diagonal()
```

[48]:
```
array([0.3487457 , 0.23827825, 0.09040794, 0.02545069, 0.05799503,
       0.27346178])
```

0.0.3 Conclusion

This assignment provides a deep understanding of ANN models, data modelling, and training ML models. I tried various configurations for training and got various results. The ratio of division of data also plays a vital role in the output and accuracies of the model. Confusion matrix and class wise accuracies give more insights about the model and scope of improvement in it. The accuracies are not as expected but it was a great learning experience.