AI - ASSIGNMENT 4 REPORT

DRISHTI DE

MT20075

Contents

1. Summary

In this assignment, we have done 5 main attempts in the data after thorough analysis on the Job Suggestions Data given with this assignment.

Attempt 1: On Original 34 Job Labels Data with total of 38 feature columns
Attempt 2: Reducing the Labels from 34 to 9 classes (Optimal number of clusters on the Data)
Attempt 3: Standard Scaling the Attempt 2 data with 9 manually labelled classes
Attempt 4: Assigning the data to 9 (optimal) clusters labelled through KMeans Algorithm
Attempt 5: Standard Scaling on the cluster formed 9 labels

The Job Data has been modified and prepared accordingly based on the 5 attempts. Also to find the best parameters for the Scikit-Learn ANN MLP Classifier Model, Grid Search CV has been performed to get the best score among all the MLP Classifier Parameters.

The Model performance has been tested on 4 different splits in the data, which are, 60-40, 70-30, 80-20 and 90-10 for train and test data respectively to create splits X_train, y_train, X_test and y_test.

2. Data Preparation and Modification

The data preparations and modifications can be mainly categorized into two types:

1. Using normal unscaled data to check for model performance in Job Role Prediction:
    a. Original 34 Suggested Job Roles Label Data
    b. Manually labelled data on 9 class labels (optimum number of clusters)
    c. KMeans assigned clusters on optimal number of cluster class labels

2. Using standard scaling on the Data:
    a. Standard Scaler on the manually labelled data - to check improvement in performance
    b. Standard Scaling on the KMeans assigned clusters

3. Other modifications on the data include Feature Importance based modifications based on the methods utilised above to check the mean and variance plot of the important features according to those models.

3. Experiments Performed

The experiments performed can be distinguished in detail based on the different attempts made on the data and model performance to achieve the end goal of suggesting a suitable job role to the student candidate based on their data in the form of 38 feature columns.

3.1 Attempt 1

In Attempt 1, the original data is taken into account with its original 34 labels. This forms the baseline of all the attempts, preparations and modifications done on the data. Here the categorical data were converted into their numerical counterparts using the pd.Factorial() command on the different categorical attributes.
This resultant data is then fed into Grid search on MLP Classifier estimator to obtain the best parameters on this data. After acquiring the best parameters and an estimated best score on the data, we run the model with those parameters on different train and test splits, namely, 60-40, 70-30, 80-20 and 90-10.

3.2 Attempt 2

In Attempt 2, the data has been standard scaled and fed into the KMeans Algorithm model to get the optimum number of clusters, which was 9 clusters. Therefore, the 34 job labels were converged into 9 labels as follows:

The class labels will be merged on the basis of these combinations:

1. Security Engineer
    - Network Security Administrator
    - Network Security Engineer
    - Systems Security Administrator
    - Network Engineer
    - Portal Administrator
2. Business Analyst
    - CRM Business Analyst
    - Business System Analyst
    - E-commerce Analyst
    - Business Intelligence Analyst
3. Analyst

    - Systems Analyst
    - Information Security Analyst
    - Programmer Analyst
4. Developer
    - Software Developer
    - Database Developer
    - Web Developer
    - CRM Technical Developer
    - Applications Developer
    - Mobile Applications Developer
5. UX Design
    - UX Designer
    - Design and UX
6. Software
    - Software Engineer
    - Software Systems Engineer
    - Software Quality Assurance
    - Quality Assurance Associate
    - Project Manager
7. Technical Supports Engineer
    - Technical Support
    - Technical Services/Helpdesk/Tech Support
    - Technical Engineer
    - Solutions Architect
8. Data Engineer
    - Database Administrator
    - Database Manager
    - Data Architect
9. Information Technology
    - Information Technology Manager
    - Information Technology Auditor

These new labels are then appended into a copy of the original data and the categorical attributes are then converted to numerical data. This resultant data is then fed into the MLP classifier model along with gridsearch to obtain the best parameters for this data. After obtaining the best parameters, The model is then evaluated for different train test splits, namely, 60-40, 70-30, 80-20 and 90-10.

3.3 Attempt 3

In this attempt, the manually labelled data was scaled to check model performance on the 8 manual classes. The Grid search for this attempt gave different best parameters for the best score on the MLP classifier model. The obtained parameters were loaded into the MLP classifier to check for the four different train-test splits.

3.4 Attempt 4

In this attempt, instead of manually assigning the data, we have assigned based on the clusters that were obtained from the k-means algorithm model. These cluster labels were then added to the data as a separate column and then used the unscaled data to find best parameters for MLP classifier using grid search technique. After analysing the best score using grid search, the parameters obtained from this technique were fed into the MLP classifier to test the model evaluation for four different train test splits.

3.5 Attempt 5

In this attempt, the new data that was created in attempt 4 was scaled and then fed into the grid search model to find the optimal parameters and the best scope. These parameters were then fed into the MLP classifier for the four different train test splits.

Analysis on 2 different experiments, that is, firstly on the manually labelled data and secondly on the cluster label data, were analysed based on their feature importance on different data attributes. The mean and standard deviations of the important features were then analysed using line plots data visualisation. This gives us more insights on which attributes were used while training and testing the MLP classifiers on the data created and modified.

4. Code with Outputs

(Jupyter Notebook file in PDF Format)

AI_Assignment4_MT20075

November 16, 2021

1 Loading the Data...

[2]:
```
# imports for the program
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
import sys
np.set_printoptions(threshold=np.inf)
```

[3]:
```
data = pd.read_csv('/content/drive/MyDrive/AI/roo_data.csv')
data
```

[3]:
```
       Acedamic percentage in Operating Systems  ...
Suggested Job Role
0                                            69  ...
Database Developer
1                                            78  ...
Portal Administrator
2                                            71  ...
Portal Administrator
3                                            76  ...              Systems
Security Administrator
4                                            92  ...                    Business
Systems Analyst
...                                         ...  ...
...
19995                                        83  ...
Technical Engineer
19996                                        80  ...
E-Commerce Analyst
19997                                        83  ...               Business
Intelligence Analyst
19998                                        68  ...  Software Quality Assurance
(QA) / Testing
19999                                        73  ...
Applications Developer
```

```
[20000 rows x 39 columns]
```

2 Data Analysis

[4]:
```
# checking the unique number of job roles in this data
print(data['Suggested Job Role'].value_counts())
print("\nThe number of unique suggested job roles: ", len(np.
 ↪array(data['Suggested Job Role'].unique())))
```

```
Network Security Administrator                 1112
Network Security Engineer                       630
Network Engineer                                621
Project Manager                                 602
Database Administrator                          593
Portal Administrator                            593
Information Technology Manager                  591
Software Engineer                               590
UX Designer                                     589
Design & UX                                     588
Software Developer                              587
CRM Business Analyst                            584
Business Systems Analyst                        582
Database Developer                              581
Solutions Architect                             578
Software Systems Engineer                       575
Software Quality Assurance (QA) / Testing       571
Database Manager                                570
Web Developer                                   570
CRM Technical Developer                         567
Quality Assurance Associate                     565
Technical Support                               565
Data Architect                                  564
Systems Security Administrator                  562
Technical Services/Help Desk/Tech Support       558
Information Technology Auditor                  558
Technical Engineer                              557
Applications Developer                          551
Systems Analyst                                 550
E-Commerce Analyst                              546
Information Security Analyst                    543
Business Intelligence Analyst                   540
Mobile Applications Developer                   538
Programmer Analyst                              529
Name: Suggested Job Role, dtype: int64
```

```
The number of unique suggested job roles:  34
```

The classes that are used here to later calculate the confusion matrix and class-wise accuracies of 34 classes are as follows:

[5]:
```
data['Suggested Job Role'].unique()
```

[5]:
```
array(['Database Developer', 'Portal Administrator',
       'Systems Security Administrator', 'Business Systems Analyst',
       'Software Systems Engineer', 'Business Intelligence Analyst',
       'CRM Technical Developer', 'Mobile Applications Developer',
       'UX Designer', 'Quality Assurance Associate', 'Web Developer',
       'Information Security Analyst', 'CRM Business Analyst',
       'Technical Support', 'Project Manager',
       'Information Technology Manager', 'Programmer Analyst',
       'Design & UX', 'Solutions Architect', 'Systems Analyst',
       'Network Security Administrator', 'Data Architect',
       'Software Developer', 'E-Commerce Analyst',
       'Technical Services/Help Desk/Tech Support',
       'Information Technology Auditor', 'Database Manager',
       'Applications Developer', 'Database Administrator',
       'Network Engineer', 'Software Engineer', 'Technical Engineer',
       'Network Security Engineer',
       'Software Quality Assurance (QA) / Testing'], dtype=object)
```

Converting the categorical data into numerical values using pandas factorize

[6]:
```
# copying the original dataframe and do the modifications here
df = data.copy(deep=True)
df['can work long time before system?'] = pd.factorize(df['can work long time␣
 ↪before system?'])[0]
df['self-learning capability?'] = pd.factorize(df['self-learning capability?
 ↪'])[0]
df['Extra-courses did'] = pd.factorize(df['Extra-courses did'])[0]
df['certifications'] = pd.factorize(df['certifications'])[0]
df['workshops'] = pd.factorize(df['workshops'])[0]
df['talenttests taken?'] = pd.factorize(df['talenttests taken?'])[0]
df['olympiads'] = pd.factorize(df['olympiads'])[0]
df['reading and writing skills'] = pd.factorize(df['reading and writing␣
 ↪skills'])[0]
df['memory capability score'] = pd.factorize(df['memory capability score'])[0]
df['Interested subjects'] = pd.factorize(df['Interested subjects'])[0]
df['interested career area '] = pd.factorize(df['interested career area '])[0]
df['Job/Higher Studies?'] = pd.factorize(df['Job/Higher Studies?'])[0]
df['Type of company want to settle in?'] = pd.factorize(df['Type of company␣
 ↪want to settle in?'])[0]
df['Taken inputs from seniors or elders'] = pd.factorize(df['Taken inputs from␣
 ↪seniors or elders'])[0]
df['interested in games'] = pd.factorize(df['interested in games'])[0]
```

```
df['Interested Type of Books'] = pd.factorize(df['Interested Type of Books'])[0]
df['Salary Range Expected'] = pd.factorize(df['Salary Range Expected'])[0]
df['In a Realtionship?'] = pd.factorize(df['In a Realtionship?'])[0]
df['Gentle or Tuff behaviour?'] = pd.factorize(df['Gentle or Tuff behaviour?
 ↪'])[0]
df['Management or Technical'] = pd.factorize(df['Management or Technical'])[0]
df['Salary/work'] = pd.factorize(df['Salary/work'])[0]
df['hard/smart worker'] = pd.factorize(df['hard/smart worker'])[0]
df['worked in teams ever?'] = pd.factorize(df['worked in teams ever?'])[0]
df['Introvert'] = pd.factorize(df['Introvert'])[0]
df['Suggested Job Role'] = pd.factorize(df['Suggested Job Role'])[0]
df
```

[6]:
```
       Acedamic percentage in Operating Systems  ...  Suggested Job Role
0                                            69  ...                   0
1                                            78  ...                   1
2                                            71  ...                   1
3                                            76  ...                   2
4                                            92  ...                   3
...                                         ...  ...                 ...
19995                                        83  ...                  31
19996                                        80  ...                  23
19997                                        83  ...                   5
19998                                        68  ...                  33
19999                                        73  ...                  27

[20000 rows x 39 columns]
```

3 Attempt 1 - On Normal 34 Class Label Data

[7]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
```

[8]:
```
# do on df dataframe
# Using Grid Search CV to obtain the best parameters and achieve the best score
# with the MLP Classifier of Scikit Learn ANN
param_grid = {
    'hidden_layer_sizes' : [20,50,100,150,200],
    'activation' : ['identity', 'logistic', 'tanh', 'relu'],
    'solver' : ['lbfgs', 'sgd', 'adam'],
    'learning_rate' : ['constant', 'invscaling', 'adaptive'],
    'max_iter' : [100,200,300],
    'early_stopping' : [True]
}
```

[ ]:
```
mlpc = MLPClassifier()
clf = GridSearchCV(mlpc, param_grid,cv = 5, n_jobs = -1, verbose=10)
```

```python
clf.fit(df.drop(['Suggested Job Role'],axis=1),df['Suggested Job Role'])
print(clf.best_params_)
print(clf.best_score_)
```

```
Fitting 5 folds for each of 540 candidates, totalling 2700 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    5.2s
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:    9.8s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   14.7s
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:   17.8s
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed:   37.2s
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed:   43.9s
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done  57 tasks      | elapsed:  1.7min
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed:  2.1min
[Parallel(n_jobs=-1)]: Done  81 tasks      | elapsed:  2.7min
[Parallel(n_jobs=-1)]: Done  94 tasks      | elapsed:  3.1min
[Parallel(n_jobs=-1)]: Done 109 tasks      | elapsed:  3.6min
[Parallel(n_jobs=-1)]: Done 124 tasks      | elapsed:  4.4min
[Parallel(n_jobs=-1)]: Done 141 tasks      | elapsed:  5.0min
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed:  5.7min
[Parallel(n_jobs=-1)]: Done 177 tasks      | elapsed:  6.6min
[Parallel(n_jobs=-1)]: Done 196 tasks      | elapsed:  7.3min
[Parallel(n_jobs=-1)]: Done 217 tasks      | elapsed:  8.4min
[Parallel(n_jobs=-1)]: Done 238 tasks      | elapsed:  9.1min
[Parallel(n_jobs=-1)]: Done 261 tasks      | elapsed: 10.7min
[Parallel(n_jobs=-1)]: Done 284 tasks      | elapsed: 11.5min
[Parallel(n_jobs=-1)]: Done 309 tasks      | elapsed: 13.3min
[Parallel(n_jobs=-1)]: Done 334 tasks      | elapsed: 14.6min
[Parallel(n_jobs=-1)]: Done 361 tasks      | elapsed: 16.2min
[Parallel(n_jobs=-1)]: Done 388 tasks      | elapsed: 17.9min
[Parallel(n_jobs=-1)]: Done 417 tasks      | elapsed: 20.0min
[Parallel(n_jobs=-1)]: Done 446 tasks      | elapsed: 22.5min
[Parallel(n_jobs=-1)]: Done 477 tasks      | elapsed: 24.3min
[Parallel(n_jobs=-1)]: Done 508 tasks      | elapsed: 26.8min
[Parallel(n_jobs=-1)]: Done 541 tasks      | elapsed: 30.2min
[Parallel(n_jobs=-1)]: Done 574 tasks      | elapsed: 33.3min
[Parallel(n_jobs=-1)]: Done 609 tasks      | elapsed: 35.4min
[Parallel(n_jobs=-1)]: Done 644 tasks      | elapsed: 38.6min
[Parallel(n_jobs=-1)]: Done 681 tasks      | elapsed: 42.6min
[Parallel(n_jobs=-1)]: Done 718 tasks      | elapsed: 43.2min
[Parallel(n_jobs=-1)]: Done 757 tasks      | elapsed: 44.3min
[Parallel(n_jobs=-1)]: Done 796 tasks      | elapsed: 45.5min
[Parallel(n_jobs=-1)]: Done 837 tasks      | elapsed: 47.4min
[Parallel(n_jobs=-1)]: Done 878 tasks      | elapsed: 49.8min
[Parallel(n_jobs=-1)]: Done 921 tasks      | elapsed: 52.5min
```

```
[Parallel(n_jobs=-1)]: Done 964 tasks      | elapsed: 55.7min
[Parallel(n_jobs=-1)]: Done 1009 tasks      | elapsed: 59.4min
[Parallel(n_jobs=-1)]: Done 1054 tasks      | elapsed: 63.4min
[Parallel(n_jobs=-1)]: Done 1101 tasks      | elapsed: 68.3min
[Parallel(n_jobs=-1)]: Done 1148 tasks      | elapsed: 73.3min
[Parallel(n_jobs=-1)]: Done 1197 tasks      | elapsed: 79.3min
[Parallel(n_jobs=-1)]: Done 1246 tasks      | elapsed: 86.3min
[Parallel(n_jobs=-1)]: Done 1297 tasks      | elapsed: 93.7min
[Parallel(n_jobs=-1)]: Done 1348 tasks      | elapsed: 101.8min
[Parallel(n_jobs=-1)]: Done 1401 tasks      | elapsed: 103.2min
[Parallel(n_jobs=-1)]: Done 1454 tasks      | elapsed: 104.6min
[Parallel(n_jobs=-1)]: Done 1509 tasks      | elapsed: 106.8min
[Parallel(n_jobs=-1)]: Done 1564 tasks      | elapsed: 109.9min
[Parallel(n_jobs=-1)]: Done 1621 tasks      | elapsed: 113.3min
[Parallel(n_jobs=-1)]: Done 1678 tasks      | elapsed: 117.3min
[Parallel(n_jobs=-1)]: Done 1737 tasks      | elapsed: 122.6min
[Parallel(n_jobs=-1)]: Done 1796 tasks      | elapsed: 129.2min
[Parallel(n_jobs=-1)]: Done 1857 tasks      | elapsed: 135.8min
[Parallel(n_jobs=-1)]: Done 1918 tasks      | elapsed: 144.1min
[Parallel(n_jobs=-1)]: Done 1981 tasks      | elapsed: 153.7min
[Parallel(n_jobs=-1)]: Done 2044 tasks      | elapsed: 161.8min
[Parallel(n_jobs=-1)]: Done 2109 tasks      | elapsed: 163.0min
[Parallel(n_jobs=-1)]: Done 2174 tasks      | elapsed: 164.9min
[Parallel(n_jobs=-1)]: Done 2241 tasks      | elapsed: 167.7min
[Parallel(n_jobs=-1)]: Done 2308 tasks      | elapsed: 171.1min
[Parallel(n_jobs=-1)]: Done 2377 tasks      | elapsed: 175.8min
[Parallel(n_jobs=-1)]: Done 2446 tasks      | elapsed: 180.4min
[Parallel(n_jobs=-1)]: Done 2517 tasks      | elapsed: 186.2min
[Parallel(n_jobs=-1)]: Done 2588 tasks      | elapsed: 192.9min
[Parallel(n_jobs=-1)]: Done 2661 tasks      | elapsed: 200.2min
[Parallel(n_jobs=-1)]: Done 2700 out of 2700 | elapsed: 204.8min finished

{'activation': 'tanh', 'early_stopping': True, 'hidden_layer_sizes': 150,
'learning_rate': 'invscaling', 'max_iter': 200, 'solver': 'adam'}
0.05595
```

Here we see the affect of multiple classes on the accuracy of a classification problem. In the presence of 34 classes of Suggested Job Roles, it becomes difficult for the ANN MLP Classifier to classify each data into their respective job roles given as target values. Hence making the accuracy around 5% which is very low.

Checking for different train-test splits: 60-40, 70-30, 80-20 and 90-10 respectively.

3.1 Checking for 4 different Train-test splits

[9]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

```
[ ]: test_sizes = [0.4,0.3,0.2,0.1]
     for ts in test_sizes:
       print('\033[1m'+"For Train-test split: "+␣
      ↪str(int(100-(ts*100)))+"-"+str(int(ts*100))+ '\033[0m' +"\n")
       X_train, X_test, y_train, y_test = train_test_split(df.drop(['Suggested Job␣
      ↪Role'],axis=1),df['Suggested Job Role'], stratify=df['Suggested Job Role'],␣
      ↪test_size=ts, random_state=1)
       mlpc_1 = MLPClassifier(hidden_layer_sizes=150, activation='tanh',␣
      ↪solver='adam',batch_size='auto', learning_rate='invscaling', max_iter=200,␣
      ↪verbose=False, early_stopping=True)
       mlpc_1.fit(X_train,y_train)
       y_pred = mlpc_1.predict(X_test)
       print("Accuracy: ",mlpc_1.score(X_test,y_test))
       print("\nConfusion Matrix for each class: ")
       matrix = confusion_matrix(y_test,y_pred,labels=np.array(df['Suggested Job␣
      ↪Role'].unique()))
       print(np.array(matrix))
       print("\nClasswise Accuracies")
       print(matrix.diagonal()/matrix.sum(axis=1))
       print("--------------------------------------------------------------")
```

```
For Train-test split: 60-40

Accuracy:  0.055375

Confusion Matrix for each class:
[[  0   0   0   0   0   0   0   0   0   0   0   0   3   0   0   0   0   0
    0   0 229   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3   0   0   0   0   0
    0   0 234   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   6   0   0   0   0   0
    0   0 219   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0
    0   0 231   0   0   0   0   0   0   0   0   0   0   0   1   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3   0   0   0   0   0
    0   0 227   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   5   0   0   0   0   0
    0   0 210   0   0   0   0   0   0   0   0   0   0   0   1   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   4   0   0   0   0   0
    0   0 223   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   2   0   0   0   1   0
    0   0 212   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   5   0   0   0   0   0
    0   0 231   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   2   0   0   0   0   0
    0   0 224   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   2   0   0   0   0   0
```

```
    0   0 225   0   0   0   0   0   0   0   0   0   0   0   1   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   5   0   0   0   0   0
    0   0 212   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   7   0   0   0   0   0
    0   0 226   0   0   0   0   0   0   0   0   0   0   0   1   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   5   0   0   0   0   0
    0   0 221   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   5   0   0   0   0   0
    0   0 236   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   4   0   0   0   0   0
    0   0 232   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3   0   0   0   0   0
    0   0 209   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0
    0   0 231   0   0   0   0   0   0   0   0   0   0   0   3   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   2   0   0   0   0   0
    0   0 228   0   0   0   0   0   0   0   0   0   0   0   1   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0 220   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   7   0   0   0   0   0
    0   0 436   0   0   0   0   0   0   0   0   0   0   0   2   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3   0   0   0   0   0
    0   0 222   0   0   0   0   0   0   0   0   0   0   0   1   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   5   0   0   1   0   0
    0   0 229   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0
    0   0 217   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   2   0   0   0   0   1
    0   0 219   0   0   0   0   0   0   0   0   0   0   0   1   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   1   0   0
    0   0 220   0   0   0   0   0   0   0   0   0   0   0   1   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   5   0   0   0   0   0
    0   0 223   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   5   0   0   0   0   0
    0   0 215   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   2   0   0   0   0   0
    0   0 235   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   2   0   0   0   0   0
    0   0 246   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   5   0   0   0   0   0
    0   0 231   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   2   0   0   0   0   0
    0   0 221   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0
    0   0 251   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   4   0   0   0   0   0
    0   0 225   0   0   0   0   0   0   0   0   0   0   0   0   0]]
```

```
Classwise Accuracies
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.02991453 0.         0.         0.         0.         0.
 0.         0.         0.97977528 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         ]
-------------------------------------------------------------
```

For Train-test split: 70-30

```
Accuracy:  0.056

Confusion Matrix for each class:
[[  0   0   0   0   0   0   0   0   1   0   0   2   0   0   0   0   0   0
    0   0 171   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   1   0   0   4   0   0   0   0   0   0
    0   0 173   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0 169   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   2   0   0   1   0   0   0   0   0   0
    0   0 172   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   1   0   0   2   0   0   0   0   0   0
    0   0 170   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0
    0   0 161   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   2   0   0   0   0   0   0
    0   0 168   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   5   0   0   0   0   0   0
    0   0 156   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   1   0   0   1   0   0   0   0   0   0
    1   0 174   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   1   0   0   0   0   0   0   0   3   0   0   0   0   0   0
    0   0 166   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0 171   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   1   0   0   3   0   0   0   0   0   0
    0   0 159   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0
    0   0 174   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0
    0   0 169   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   1   0   0   1   0   0   0   0   0   0
    0   0 179   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0 177   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   3   0   0   0   0   0   0
    0   0 156   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   3   0   0   0   0   0   0
```

```
    0   0 173   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   3   0   0   0   0   0   0
    0   0 170   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0
    0   0 164   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   2   0   0   0   0   0   0
    0   0 332   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0 169   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   2   0   0   0   0   0   0
    0   0 174   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   6   0   0   0   0   0   0
    0   0 158   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0
    0   0 166   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   2   0   0   4   0   0   0   0   0   0
    0   0 161   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   4   0   0   0   0   0   0
    0   0 167   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   1   0   0   1   0   0   0   0   0   0
    0   0 163   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   2   0   0   0   0   0   0
    0   0 176   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    1   0 185   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0 177   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0
    0   0 166   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   1   0   0   2   0   0   0   0   0   0
    0   0 186   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   2   0   0   0   0   0   0
    0   0 169   0   0   0   0   0   0   0   0   0   0   0   0   0]]

Classwise Accuracies
[0.         0.         0.         0.         0.         0.
 0.         0.         0.00564972 0.         0.         0.01840491
 0.         0.         0.         0.         0.         0.
 0.         0.         0.99401198 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
--------------------------------------------------------------
```

For Train-test split: 80-20

```
Accuracy:  0.0555

Confusion Matrix for each class:
[[  0   0   0   0   0   0   0   2   0   0   0   0   0   0   0   0   0   0
```

```
   0   0 113   0   0   0   0   0   0   0   0   1   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0 118   0   0   0   0   0   0   0   0   1   0   0   0   0]
 [  0   0   0   0   0   0   0   2   0   0   0   0   0   0   0   0   0   0
   0   0 110   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   2   0   0   0   0   0   0   0   0   0   0
   0   0 114   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   2   0   0   0   0   0   0   0   0   0   0
   0   0 113   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   2   0   0   0   0   0   0   0   0   0   0
   0   0 106   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   3   0   0   0   0   0   0   0   0   0   0
   0   0 109   0   0   0   0   0   0   0   0   1   0   0   0   0]
 [  0   0   0   0   0   0   0   2   0   0   0   0   0   0   0   0   0   0
   0   0 106   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   5   0   0   0   0   0   0   0   0   0   0
   0   0 112   0   0   0   0   0   0   0   0   1   0   0   0   0]
 [  0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0
   0   0 111   0   0   0   0   0   0   0   0   1   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0 112   0   0   0   0   0   0   0   0   2   0   0   0   0]
 [  0   0   0   0   0   0   0   3   0   0   0   0   0   0   0   0   0   0
   0   0 106   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   2   0   0   0   0   0   0   0   0   0   0
   0   0 114   0   0   0   0   0   0   0   0   1   0   0   0   0]
 [  0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0
   0   0 112   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   2   0   0   0   0   0   0   0   0   0   0
   0   0 118   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   2   0   0   0   0   0   0   0   0   0   0
   0   0 115   0   0   0   0   0   0   0   0   1   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0 106   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   4   0   0   0   0   0   0   0   0   0   0
   0   0 114   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   2   0   0   0   0   0   0   0   0   0   0
   0   0 113   0   0   0   0   0   0   0   0   1   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0 109   0   0   0   0   0   0   0   0   1   0   0   0   0]
 [  0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0
   0   0 220   0   0   0   0   0   0   0   0   1   0   0   0   0]
 [  0   0   0   0   0   0   0   2   0   0   0   0   0   0   0   0   0   0
   0   0 111   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0 115   0   0   0   0   0   0   0   0   2   0   0   0   0]
 [  0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0
   0   0 108   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0
```

```
   0   0 110   0   0   0   0   0   0   0   0   1   0   0   0   0]
 [  0   0   0   0   0   0   0   2   0   0   0   0   0   0   0   0   0   0
   0   0 110   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0
   0   0 112   0   0   0   0   0   0   0   0   1   0   0   0   0]
 [  0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0
   0   0 109   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   2   0   0   0   0   0   0   0   0   0   0
   0   0 117   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0
   0   0 123   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   3   0   0   0   0   0   0   0   0   0   0
   0   0 115   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0
   0   0 109   0   0   0   0   0   0   0   0   1   0   0   0   0]
 [  0   0   0   0   0   1   0   1   0   0   0   0   0   0   0   0   0   0
   0   0 123   0   0   0   0   0   0   0   0   1   0   0   0   0]
 [  0   0   0   0   0   1   0   3   0   0   0   0   0   0   0   0   0   0
   0   0 110   0   0   0   0   0   0   0   0   0   0   0   0   0]]

Classwise Accuracies
[0.         0.         0.         0.         0.         0.
 0.         0.01851852 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.99099099 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
--------------------------------------------------------------
```

For Train-test split: 90-10

```
Accuracy:  0.0545

Confusion Matrix for each class:
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  58   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
   0   0  58   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  56   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  58   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
   0   0  57   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  54   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
   0   0  56   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

```
   0   0  54   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  59   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
   0   0  55   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  57   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  54   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  58   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  57   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
   0   0  59   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  59   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  53   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  59   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
   0   0  57   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  55   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   1
   0   0 109   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  56   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  59   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
   0   0  54   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  56   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
   0   0  55   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2
   0   0  55   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  55   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  59   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  62   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0  59   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
```

```
    0    0  55    0    0    0    0    0    0    0    0    0    0    0    0    0]
 [  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0  63    0    0    0    0    0    0    0    0    0    0    0    0    0]
 [  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0  57    0    0    0    0    0    0    0    0    0    0    0    0    0]]

Classwise Accuracies
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.98198198 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         ]
--------------------------------------------------------------
```

4 Reducing the number of Classes to a lower value

Trying to transform the data using standard scaler in order to assess the data clusters well

[ ]:
```python
X = df.drop(['Suggested Job Role'],axis=1)
```

[11]:
```python
from sklearn.preprocessing import StandardScaler
```

[ ]:
```python
scaler = StandardScaler()
X = scaler.fit_transform(X)
print(pd.DataFrame(X))
```

```
             0         1         2  ...        35        36        37
0     -0.793450 -1.380807  0.096941  ... -1.004711 -1.005415 -1.009748
1      0.098925 -1.479803 -0.396421  ... -1.004711  0.994615  0.990347
2     -0.595145  0.896086  1.379680  ... -1.004711  0.994615  0.990347
3     -0.099381  0.995082 -1.679160  ...  0.995311 -1.005415  0.990347
4      1.487064 -1.479803  1.281008  ... -1.004711 -1.005415  0.990347
...         ...       ...       ...  ...       ...       ...       ...
19995  0.594689 -0.984826 -1.481815  ...  0.995311 -1.005415 -1.009748
19996  0.297230 -0.786835  0.590302  ... -1.004711  0.994615  0.990347
19997  0.594689 -0.687840  0.294285  ... -1.004711  0.994615  0.990347
19998 -0.892603  0.995082  1.379680  ...  0.995311 -1.005415 -1.009748
19999 -0.396839  0.005128 -0.297748  ... -1.004711 -1.005415 -1.009748

[20000 rows x 38 columns]
```

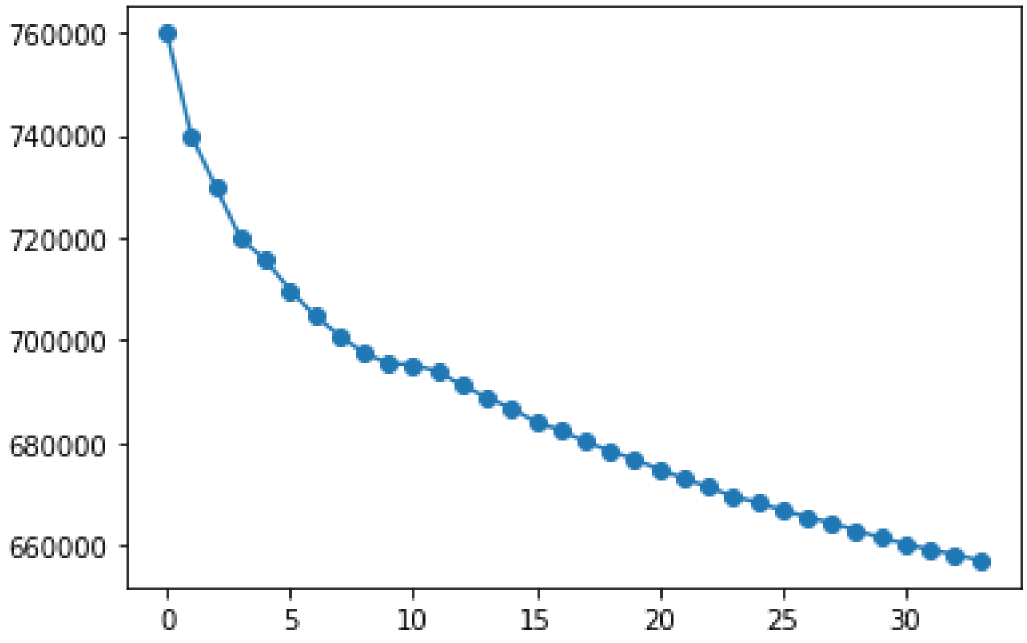
In order to convert this data into a fewer class based clasification problem, we need to understand the optimum number of clusters in which the data can be best summed up with.

This can be done using the Elbow Method using K-Means Clustering Model to find the optimum number of clusters

[40]:
```python
from sklearn.cluster import KMeans
```

[ ]:
```
# X = df.drop(['Suggested Job Role'],axis=1)
wcss = []
for c in range(1,35):
  kmeans = KMeans(n_clusters=c, random_state=0).fit(X)
  wcss.append(kmeans.inertia_)

plt.plot(wcss,marker='o')
plt.show()
```



On the basis of this graph we see that the elbow forms at around the range of 7-9, therefore let us convert this data into a 8-9 class classification task to check the accuracy of our model.

[10]:
```
data
```

[10]:
```
       Acedamic percentage in Operating Systems  ...
Suggested Job Role
0                                            69  ...
Database Developer
1                                            78  ...
Portal Administrator
2                                            71  ...
Portal Administrator
3                                            76  ...                 Systems
Security Administrator
4                                            92  ...                        Business
Systems Analyst
...                                         ...  ...
```

```
...
19995                                        83  ...
Technical Engineer
19996                                        80  ...
E-Commerce Analyst
19997                                        83  ...                Business
Intelligence Analyst
19998                                        68  ...  Software Quality Assurance
(QA) / Testing
19999                                        73  ...
Applications Developer

[20000 rows x 39 columns]
```

[12]:
```
data_copy = data.copy(deep=True)
data_copy
```

[12]:
```
       Acedamic percentage in Operating Systems  ...
Suggested Job Role
0                                            69  ...
Database Developer
1                                            78  ...
Portal Administrator
2                                            71  ...
Portal Administrator
3                                            76  ...               Systems
Security Administrator
4                                            92  ...                     Business
Systems Analyst
...                                         ...  ...
...
19995                                        83  ...
Technical Engineer
19996                                        80  ...
E-Commerce Analyst
19997                                        83  ...                Business
Intelligence Analyst
19998                                        68  ...  Software Quality Assurance
(QA) / Testing
19999                                        73  ...
Applications Developer

[20000 rows x 39 columns]
```

The class labels will be merged on the basis of these combinations: 1. Security Engineer * Network Security Administrator * Network Security Engineer * Systems Security Administrator * Network Engineer * Portal Administrator 2. Business Analyst * CRM Business Analyst * Business System Analyst * E-commerce Analyst * Business Intelligence Analyst 3. Analyst * Systems Analyst * Information Security Analyst * Programmer Analyst 4. Developer * Software Developer

* Database Developer * Web Developer * CRM Technical Developer * Applications Developer * Mobile Applications Developer 5. UX Design * UX Designer * Design and UX 6. Software * Software Engineer * Software Systems Engineer * Software Quality Assurance * Quality Assurance Associate * Project Manager 7. Technical Supports Engineer * Technical Support * Technical Services/Helpdesk/Tech Support * Technical Engineer * Solutions Architect 8. Data Engineer * Database Administrator * Database Manager * Data Architect 9. Information Technology * Information Technology Manager * Information Technology Auditor

[15]:
```
data_copy = pd.DataFrame(data_copy)
data
```

[15]:
```
       Acedamic percentage in Operating Systems  ...
Suggested Job Role
0                                            69  ...
Database Developer
1                                            78  ...
Portal Administrator
2                                            71  ...
Portal Administrator
3                                            76  ...                 Systems
Security Administrator
4                                            92  ...                       Business
Systems Analyst
...                                         ...  ...
...
19995                                        83  ...
Technical Engineer
19996                                        80  ...
E-Commerce Analyst
19997                                        83  ...                  Business
Intelligence Analyst
19998                                        68  ...  Software Quality Assurance
(QA) / Testing
19999                                        73  ...
Applications Developer

[20000 rows x 39 columns]
```

[16]:
```
df_new = pd.DataFrame()
#temp_df = pd.DataFrame()
#-----------------------LABEL 1--------------------------------
temp_df = pd.DataFrame(data_copy.loc[(data_copy['Suggested Job Role']=='Network␣
 ↪Security Administrator') | (data_copy['Suggested Job Role']=='Network␣
 ↪Security Engineer') | (data_copy['Suggested Job Role']=='Network Engineer')␣
 ↪| (data_copy['Suggested Job Role']=='Systems Security Administrator')  |␣
 ↪(data_copy['Suggested Job Role']=='Portal Administrator')])
temp_df['New Job Label'] = ['Security Engineer']*len(temp_df)
#print(temp_df['Suggested Job Role'].value_counts())
df_new = df_new.append(temp_df)
```

```
#------------------------LABEL 2---------------------------------
temp_df = pd.DataFrame(data_copy.loc[(data_copy['Suggested Job Role']=='CRM␣
↪Business Analyst') | (data_copy['Suggested Job Role']=='Business Systems␣
↪Analyst') | (data_copy['Suggested Job Role']=='E-Commerce Analyst') |␣
↪(data_copy['Suggested Job Role']=='Business Intelligence Analyst')])
temp_df['New Job Label'] = ['Business Analyst']*len(temp_df)
#print(temp_df['Suggested Job Role'].value_counts())
df_new = df_new.append(temp_df)

#------------------------LABEL 3---------------------------------
temp_df = pd.DataFrame(data_copy.loc[(data_copy['Suggested Job Role']=='Systems␣
↪Analyst') | (data_copy['Suggested Job Role']=='Information Security␣
↪Analyst') | (data_copy['Suggested Job Role']=='Programmer Analyst')])
temp_df['New Job Label'] = ['Analyst']*len(temp_df)
#print(temp_df['Suggested Job Role'].value_counts())
df_new = df_new.append(temp_df)

#------------------------LABEL 4---------------------------------
temp_df = pd.DataFrame(data_copy.loc[(data_copy['Suggested Job␣
↪Role']=='Software Developer') | (data_copy['Suggested Job Role']=='Database␣
↪Developer') | (data_copy['Suggested Job Role']=='Web Developer') |␣
↪(data_copy['Suggested Job Role']=='CRM Technical Developer') |␣
↪(data_copy['Suggested Job Role']=='Applications Developer') |␣
↪(data_copy['Suggested Job Role']=='Mobile Applications Developer')])
temp_df['New Job Label'] = ['Developer']*len(temp_df)
#print(temp_df['Suggested Job Role'].value_counts())
df_new = df_new.append(temp_df)

#------------------------LABEL 5---------------------------------
temp_df = pd.DataFrame(data_copy.loc[(data_copy['Suggested Job Role']=='UX␣
↪Designer') | (data_copy['Suggested Job Role']=='Design & UX')])
temp_df['New Job Label'] = ['UX Design']*len(temp_df)
#print(temp_df['Suggested Job Role'].value_counts())
df_new = df_new.append(temp_df)

#------------------------LABEL 6---------------------------------
temp_df = pd.DataFrame(data_copy.loc[(data_copy['Suggested Job␣
↪Role']=='Software Engineer') | (data_copy['Suggested Job Role']=='Software␣
↪Systems Engineer') | (data_copy['Suggested Job Role']=='Software Quality␣
↪Assurance (QA) / Testing') | (data_copy['Suggested Job Role']=='Quality␣
↪Assurance Associate') | (data_copy['Suggested Job Role']=='Project␣
↪Manager')])
temp_df['New Job Label'] = ['Software']*len(temp_df)
#print(temp_df['Suggested Job Role'].value_counts())
df_new = df_new.append(temp_df)
```

```
#------------------------LABEL 7---------------------------------
temp_df = pd.DataFrame(data_copy.loc[(data_copy['Suggested Job␣
 ↪Role']=='Technical Support') | (data_copy['Suggested Job Role']=='Technical␣
 ↪Services/Help Desk/Tech Support') | (data_copy['Suggested Job␣
 ↪Role']=='Technical Engineer') | (data_copy['Suggested Job Role']=='Solutions␣
 ↪Architect')])
temp_df['New Job Label'] = ['Technical Supports Engineer']*len(temp_df)
#print(temp_df['Suggested Job Role'].value_counts())
df_new = df_new.append(temp_df)

#------------------------LABEL 8---------------------------------
temp_df = pd.DataFrame(data_copy.loc[(data_copy['Suggested Job␣
 ↪Role']=='Database Administrator') | (data_copy['Suggested Job␣
 ↪Role']=='Database Manager') | (data_copy['Suggested Job Role']=='Data␣
 ↪Architect')])
temp_df['New Job Label'] = ['Data Engineer']*len(temp_df)
#print(temp_df['Suggested Job Role'].value_counts())
df_new = df_new.append(temp_df)

#------------------------LABEL 9---------------------------------
temp_df = pd.DataFrame(data_copy.loc[(data_copy['Suggested Job␣
 ↪Role']=='Information Technology Manager') | (data_copy['Suggested Job␣
 ↪Role']=='Information Technology Auditor')])
temp_df['New Job Label'] = ['Information Technology']*len(temp_df)
#print(temp_df['Suggested Job Role'].value_counts())
df_new = df_new.append(temp_df)


#print(df_new['Suggested Job Role'].value_counts())
print("Original class count: ",len(np.array(df_new['Suggested Job Role'].
 ↪unique())))
print("New class count: ",len(np.array(df_new['New Job Label'].unique())))
print("\n")
print(df_new)
```

```
Original class count:  34
New class count:  9


       Acedamic percentage in Operating Systems  ...            New Job Label
1                                            78  ...        Security Engineer
2                                            71  ...        Security Engineer
3                                            76  ...        Security Engineer
32                                           80  ...        Security Engineer
38                                           70  ...        Security Engineer
...                                         ...  ...                      ...
19925                                        89  ...   Information Technology
```

```
19934                                                     67  ...  Information Technology
19973                                                     64  ...  Information Technology
19986                                                     74  ...  Information Technology
19991                                                     77  ...  Information Technology

[20000 rows x 40 columns]
```

Here the different classes for which the confusion matrices and class-wise accuracies will be calculated later are as follows:

[17]:
```
df_new['New Job Label'].unique()
```

[17]:
```
array(['Security Engineer', 'Business Analyst', 'Analyst', 'Developer',
       'UX Design', 'Software', 'Technical Supports Engineer',
       'Data Engineer', 'Information Technology'], dtype=object)
```

[18]:
```
# copying the new (reduced class) dataframe and do the modifications here -␣
 ↪converting categorical to numerical data for further analysis and␣
 ↪computations
copy_df = df_new.copy(deep=True)
#print(copy_df)
copy_df['can work long time before system?'] = pd.factorize(copy_df['can work␣
 ↪long time before system?'])[0]
copy_df['self-learning capability?'] = pd.factorize(copy_df['self-learning␣
 ↪capability?'])[0]
copy_df['Extra-courses did'] = pd.factorize(copy_df['Extra-courses did'])[0]
copy_df['certifications'] = pd.factorize(copy_df['certifications'])[0]
copy_df['workshops'] = pd.factorize(copy_df['workshops'])[0]
copy_df['talenttests taken?'] = pd.factorize(copy_df['talenttests taken?'])[0]
copy_df['olympiads'] = pd.factorize(copy_df['olympiads'])[0]
copy_df['reading and writing skills'] = pd.factorize(copy_df['reading and␣
 ↪writing skills'])[0]
copy_df['memory capability score'] = pd.factorize(copy_df['memory capability␣
 ↪score'])[0]
copy_df['Interested subjects'] = pd.factorize(copy_df['Interested subjects'])[0]
copy_df['interested career area '] = pd.factorize(copy_df['interested career␣
 ↪area '])[0]
copy_df['Job/Higher Studies?'] = pd.factorize(copy_df['Job/Higher Studies?'])[0]
copy_df['Type of company want to settle in?'] = pd.factorize(copy_df['Type of␣
 ↪company want to settle in?'])[0]
copy_df['Taken inputs from seniors or elders'] = pd.factorize(copy_df['Taken␣
 ↪inputs from seniors or elders'])[0]
copy_df['interested in games'] = pd.factorize(copy_df['interested in games'])[0]
copy_df['Interested Type of Books'] = pd.factorize(copy_df['Interested Type of␣
 ↪Books'])[0]
copy_df['Salary Range Expected'] = pd.factorize(copy_df['Salary Range␣
 ↪Expected'])[0]
copy_df['In a Realtionship?'] = pd.factorize(copy_df['In a Realtionship?'])[0]
```

```
copy_df['Gentle or Tuff behaviour?'] = pd.factorize(copy_df['Gentle or Tuff␣
 ↪behaviour?'])[0]
copy_df['Management or Technical'] = pd.factorize(copy_df['Management or␣
 ↪Technical'])[0]
copy_df['Salary/work'] = pd.factorize(copy_df['Salary/work'])[0]
copy_df['hard/smart worker'] = pd.factorize(copy_df['hard/smart worker'])[0]
copy_df['worked in teams ever?'] = pd.factorize(copy_df['worked in teams ever?
 ↪'])[0]
copy_df['Introvert'] = pd.factorize(copy_df['Introvert'])[0]
copy_df['Suggested Job Role'] = pd.factorize(copy_df['Suggested Job Role'])[0]
copy_df['New Job Label'] = pd.factorize(copy_df['New Job Label'])[0]
copy_df
```

[18]:
```
       Acedamic percentage in Operating Systems  ...  New Job Label
1                                            78  ...              0
2                                            71  ...              0
3                                            76  ...              0
32                                           80  ...              0
38                                           70  ...              0
...                                         ...  ...            ...
19925                                        89  ...              8
19934                                        67  ...              8
19973                                        64  ...              8
19986                                        74  ...              8
19991                                        77  ...              8

[20000 rows x 40 columns]
```

5 Attempt 2 - On Manually Combined 9 class data

Now building model on this and finding optimal parameters to train and evaluate Scikit-Learn ANN MLP Classifier Model

[ ]:
```
mlpc2 = MLPClassifier()
clf2 = GridSearchCV(mlpc2, param_grid,cv = 5, n_jobs = -1, verbose=10)
clf2.fit(copy_df.drop(['Suggested Job Role','New Job␣
 ↪Label'],axis=1),copy_df['New Job Label'])
print(clf2.best_params_)
print(clf2.best_score_)
```

```
Fitting 5 folds for each of 540 candidates, totalling 2700 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    4.7s
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:    8.2s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   13.5s
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:   18.4s
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed:   36.5s
```

```
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed:   44.2s
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done  57 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done  81 tasks      | elapsed:  2.4min
[Parallel(n_jobs=-1)]: Done  94 tasks      | elapsed:  2.7min
[Parallel(n_jobs=-1)]: Done 109 tasks      | elapsed:  3.3min
[Parallel(n_jobs=-1)]: Done 124 tasks      | elapsed:  3.9min
[Parallel(n_jobs=-1)]: Done 141 tasks      | elapsed:  4.5min
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed:  5.0min
[Parallel(n_jobs=-1)]: Done 177 tasks      | elapsed:  5.8min
[Parallel(n_jobs=-1)]: Done 196 tasks      | elapsed:  6.4min
[Parallel(n_jobs=-1)]: Done 217 tasks      | elapsed:  7.4min
[Parallel(n_jobs=-1)]: Done 238 tasks      | elapsed:  8.1min
[Parallel(n_jobs=-1)]: Done 261 tasks      | elapsed:  9.5min
[Parallel(n_jobs=-1)]: Done 284 tasks      | elapsed: 10.2min
[Parallel(n_jobs=-1)]: Done 309 tasks      | elapsed: 11.9min
[Parallel(n_jobs=-1)]: Done 334 tasks      | elapsed: 13.2min
[Parallel(n_jobs=-1)]: Done 361 tasks      | elapsed: 14.7min
[Parallel(n_jobs=-1)]: Done 388 tasks      | elapsed: 16.4min
[Parallel(n_jobs=-1)]: Done 417 tasks      | elapsed: 18.4min
[Parallel(n_jobs=-1)]: Done 446 tasks      | elapsed: 20.9min
[Parallel(n_jobs=-1)]: Done 477 tasks      | elapsed: 22.8min
[Parallel(n_jobs=-1)]: Done 508 tasks      | elapsed: 25.5min
[Parallel(n_jobs=-1)]: Done 541 tasks      | elapsed: 29.0min
[Parallel(n_jobs=-1)]: Done 574 tasks      | elapsed: 32.2min
[Parallel(n_jobs=-1)]: Done 609 tasks      | elapsed: 34.9min
[Parallel(n_jobs=-1)]: Done 644 tasks      | elapsed: 38.2min
[Parallel(n_jobs=-1)]: Done 681 tasks      | elapsed: 41.8min
[Parallel(n_jobs=-1)]: Done 718 tasks      | elapsed: 42.5min
[Parallel(n_jobs=-1)]: Done 757 tasks      | elapsed: 43.6min
[Parallel(n_jobs=-1)]: Done 796 tasks      | elapsed: 45.1min
[Parallel(n_jobs=-1)]: Done 837 tasks      | elapsed: 46.9min
[Parallel(n_jobs=-1)]: Done 878 tasks      | elapsed: 49.2min
[Parallel(n_jobs=-1)]: Done 921 tasks      | elapsed: 52.1min
[Parallel(n_jobs=-1)]: Done 964 tasks      | elapsed: 55.6min
[Parallel(n_jobs=-1)]: Done 1009 tasks      | elapsed: 59.7min
[Parallel(n_jobs=-1)]: Done 1054 tasks      | elapsed: 64.2min
[Parallel(n_jobs=-1)]: Done 1101 tasks      | elapsed: 70.2min
[Parallel(n_jobs=-1)]: Done 1148 tasks      | elapsed: 76.2min
[Parallel(n_jobs=-1)]: Done 1197 tasks      | elapsed: 83.6min
[Parallel(n_jobs=-1)]: Done 1246 tasks      | elapsed: 92.2min
[Parallel(n_jobs=-1)]: Done 1297 tasks      | elapsed: 101.1min
[Parallel(n_jobs=-1)]: Done 1348 tasks      | elapsed: 111.4min
[Parallel(n_jobs=-1)]: Done 1401 tasks      | elapsed: 112.6min
[Parallel(n_jobs=-1)]: Done 1454 tasks      | elapsed: 113.7min
[Parallel(n_jobs=-1)]: Done 1509 tasks      | elapsed: 116.0min
```

```
[Parallel(n_jobs=-1)]: Done 1564 tasks      | elapsed: 118.9min
[Parallel(n_jobs=-1)]: Done 1621 tasks      | elapsed: 122.2min
[Parallel(n_jobs=-1)]: Done 1678 tasks      | elapsed: 126.6min
[Parallel(n_jobs=-1)]: Done 1737 tasks      | elapsed: 132.4min
[Parallel(n_jobs=-1)]: Done 1796 tasks      | elapsed: 140.0min
[Parallel(n_jobs=-1)]: Done 1857 tasks      | elapsed: 147.5min
[Parallel(n_jobs=-1)]: Done 1918 tasks      | elapsed: 157.3min
[Parallel(n_jobs=-1)]: Done 1981 tasks      | elapsed: 168.7min
[Parallel(n_jobs=-1)]: Done 2044 tasks      | elapsed: 178.7min
[Parallel(n_jobs=-1)]: Done 2109 tasks      | elapsed: 179.7min
[Parallel(n_jobs=-1)]: Done 2174 tasks      | elapsed: 181.6min
[Parallel(n_jobs=-1)]: Done 2241 tasks      | elapsed: 184.7min
[Parallel(n_jobs=-1)]: Done 2308 tasks      | elapsed: 187.9min
[Parallel(n_jobs=-1)]: Done 2377 tasks      | elapsed: 192.7min
[Parallel(n_jobs=-1)]: Done 2446 tasks      | elapsed: 197.9min
[Parallel(n_jobs=-1)]: Done 2517 tasks      | elapsed: 203.9min
[Parallel(n_jobs=-1)]: Done 2588 tasks      | elapsed: 211.2min
[Parallel(n_jobs=-1)]: Done 2661 tasks      | elapsed: 219.3min
[Parallel(n_jobs=-1)]: Done 2700 out of 2700 | elapsed: 223.8min finished

{'activation': 'relu', 'early_stopping': True, 'hidden_layer_sizes': 200,
'learning_rate': 'adaptive', 'max_iter': 200, 'solver': 'sgd'}
0.17824999999999996
```

5.1 Checking for 4 different Train-test splits

```
[ ]: test_sizes = [0.4,0.3,0.2,0.1]
     for ts2 in test_sizes:
       print('\033[1m'+"For Train-test split: "+␣
     ↪str(int(100-(ts2*100)))+"-"+str(int(ts2*100))+ '\033[0m' +"\n")
       X_train, X_test, y_train, y_test = train_test_split(copy_df.drop(['Suggested␣
     ↪Job Role','New Job Label'],axis=1),copy_df['New Job Label'],␣
     ↪stratify=copy_df['New Job Label'], test_size=ts2, random_state=1)
       mlpc_2 = MLPClassifier(hidden_layer_sizes=200, activation='relu',␣
     ↪solver='sgd',batch_size='auto', learning_rate='adaptive', max_iter=200,␣
     ↪verbose=False, early_stopping=True)
       mlpc_2.fit(X_train,y_train)
       y_pred = mlpc_2.predict(X_test)
       print("Accuracy: ",mlpc_2.score(X_test,y_test))
       print("\nConfusion Matrix for each class: ")
       matrix = confusion_matrix(y_test,y_pred,labels=np.array(copy_df['New Job␣
     ↪Label'].unique()))
       print(np.array(matrix))
       print("\nClasswise Accuracies")
       print(matrix.diagonal()/matrix.sum(axis=1))
       print("--------------------------------------------------------------")
```

For Train-test split: 60-40

```
Accuracy:  0.171

Confusion Matrix for each class:
[[ 355    0    0 1046    0    6    0    0    0]
 [ 224    0    0  673    0    4    0    0    0]
 [ 160    0    0  489    0    0    0    0    0]
 [ 343    0    0 1008    0    6    0    0    0]
 [ 120    0    0  351    0    0    0    0    0]
 [ 291    0    0  865    0    5    0    0    0]
 [ 229    0    0  674    0    0    0    0    0]
 [ 158    0    0  531    0    2    0    0    0]
 [ 125    0    0  334    0    1    0    0    0]]

Classwise Accuracies
[0.25230988 0.         0.         0.74281503 0.         0.00430663
 0.         0.         0.        ]
--------------------------------------------------------------
```

For Train-test split: 70-30

```
Accuracy:  0.17616666666666667

Confusion Matrix for each class:
[[1055    0    0    0    0    0    0    0    0]
 [ 674    0    0    2    0    0    0    0    0]
 [ 486    0    0    1    0    0    0    0    0]
 [1015    1    0    2    0    0    0    0    0]
 [ 352    0    0    1    0    0    0    0    0]
 [ 870    0    0    1    0    0    0    0    0]
 [ 676    0    0    1    0    0    0    0    0]
 [ 517    0    0    1    0    0    0    0    0]
 [ 345    0    0    0    0    0    0    0    0]]

Classwise Accuracies
[1.         0.         0.         0.00196464 0.         0.
 0.         0.         0.        ]
--------------------------------------------------------------
```

For Train-test split: 80-20

```
Accuracy:  0.16975

Confusion Matrix for each class:
[[   3    0    0 697    0    0    4    0    0]
 [   1    0    0 448    0    1    0    0    0]
 [   0    0    0 324    0    0    0    0    0]
 [   1    0    0 676    0    1    1    0    0]
 [   0    0    0 235    0    0    0    0    0]
 [  0   0   0 581   0   0   0   0   0]
 [  1   0   0 451   0   0   0   0   0]
 [  1   0   0 344   0   0   0   0   0]
 [  0   0   0 230   0   0   0   0   0]]

Classwise Accuracies
[0.00426136 0.         0.         0.99558174 0.         0.
 0.         0.         0.        ]
---------------------------------------------------------------
For Train-test split: 90-10

Accuracy:  0.1695

Confusion Matrix for each class:
[[ 16   0   0 336   0   0   0   0   0]
 [  4   0   0 221   0   0   0   0   0]
 [  9   0   0 153   0   0   0   0   0]
 [ 16   0   0 323   0   0   0   0   0]
 [  4   0   0 114   0   0   0   0   0]
 [ 20   0   0 270   0   0   0   0   0]
 [  6   0   0 220   0   0   0   0   0]
 [  9   0   0 164   0   0   0   0   0]
 [  9   0   0 106   0   0   0   0   0]]

Classwise Accuracies
[0.04545455 0.         0.         0.95280236 0.         0.
 0.         0.         0.        ]
---------------------------------------------------------------
```

6 Standard Scaling the Manually Labelled Data to evaluate ANN Model Performance

[21]:
```python
Xhat = copy_df.drop(['Suggested Job Role','New Job Label'],axis=1)
scaler2 = StandardScaler()
Xhat = scaler2.fit_transform(Xhat)
print(pd.DataFrame(Xhat))
```

```
             0         1         2   ...        35        36        37
0      0.098925 -1.479803 -0.396421  ... -1.004711 -0.994615 -0.990347
1     -0.595145  0.896086  1.379680  ... -1.004711 -0.994615 -0.990347
2     -0.099381  0.995082 -1.679160  ...  0.995311  1.005415 -0.990347
3      0.297230  0.797091 -0.889782  ... -1.004711 -0.994615  1.009748
4     -0.694298 -0.588844 -1.185799  ...  0.995311 -0.994615 -0.990347
...         ...       ...       ...  ...       ...       ...       ...
19995  1.189605 -1.281812  1.379680  ... -1.004711 -0.994615 -0.990347
19996 -0.991756 -1.083821  1.281008  ...  0.995311 -0.994615  1.009748
19997 -1.289214 -1.083821 -0.495093  ... -1.004711 -0.994615  1.009748
```

```
19998 -0.297686 -0.489849 -1.185799  ... -1.004711 -0.994615  1.009748
19999 -0.000228  1.391063 -0.297748  ... -1.004711 -0.994615 -0.990347

[20000 rows x 38 columns]
```

7 Attempt 3 - Standard Scaling On Manually Combined 9 class data

Now building model on this and finding optimal parameters using Grid Search to train and evaluate Scikit-Learn ANN MLP Classifier Model

[ ]:
```
mlpc3 = MLPClassifier()
clf3 = GridSearchCV(mlpc3, param_grid,cv = 5, n_jobs = -1, verbose=10)
clf3.fit(Xhat,copy_df['New Job Label'])
print(clf3.best_params_)
print(clf3.best_score_)
```

```
Fitting 5 folds for each of 540 candidates, totalling 2700 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    2.2s
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:    3.2s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    7.4s
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:   10.8s
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed:   16.3s
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed:   21.6s
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:   26.4s
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:   33.0s
[Parallel(n_jobs=-1)]: Done  57 tasks      | elapsed:   38.2s
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed:   44.0s
[Parallel(n_jobs=-1)]: Done  81 tasks      | elapsed:   51.2s
[Parallel(n_jobs=-1)]: Done  94 tasks      | elapsed:   58.2s
[Parallel(n_jobs=-1)]: Done 109 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done 124 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done 141 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed:  2.0min
[Parallel(n_jobs=-1)]: Done 177 tasks      | elapsed:  2.3min
[Parallel(n_jobs=-1)]: Done 196 tasks      | elapsed:  2.4min
[Parallel(n_jobs=-1)]: Done 217 tasks      | elapsed:  2.6min
[Parallel(n_jobs=-1)]: Done 238 tasks      | elapsed:  3.0min
[Parallel(n_jobs=-1)]: Done 261 tasks      | elapsed:  3.4min
[Parallel(n_jobs=-1)]: Done 284 tasks      | elapsed:  3.8min
[Parallel(n_jobs=-1)]: Done 309 tasks      | elapsed:  4.1min
[Parallel(n_jobs=-1)]: Done 334 tasks      | elapsed:  4.4min
[Parallel(n_jobs=-1)]: Done 361 tasks      | elapsed:  4.7min
[Parallel(n_jobs=-1)]: Done 388 tasks      | elapsed:  5.4min
[Parallel(n_jobs=-1)]: Done 417 tasks      | elapsed:  5.9min
[Parallel(n_jobs=-1)]: Done 446 tasks      | elapsed:  6.4min
[Parallel(n_jobs=-1)]: Done 477 tasks      | elapsed:  6.8min
```

```
[Parallel(n_jobs=-1)]: Done 508 tasks      | elapsed:  7.5min
[Parallel(n_jobs=-1)]: Done 541 tasks      | elapsed:  8.4min
[Parallel(n_jobs=-1)]: Done 574 tasks      | elapsed:  9.0min
[Parallel(n_jobs=-1)]: Done 609 tasks      | elapsed:  9.6min
[Parallel(n_jobs=-1)]: Done 644 tasks      | elapsed: 10.5min
[Parallel(n_jobs=-1)]: Done 681 tasks      | elapsed: 11.8min
[Parallel(n_jobs=-1)]: Done 718 tasks      | elapsed: 12.8min
[Parallel(n_jobs=-1)]: Done 757 tasks      | elapsed: 14.0min
[Parallel(n_jobs=-1)]: Done 796 tasks      | elapsed: 15.2min
[Parallel(n_jobs=-1)]: Done 837 tasks      | elapsed: 17.0min
[Parallel(n_jobs=-1)]: Done 878 tasks      | elapsed: 19.0min
[Parallel(n_jobs=-1)]: Done 921 tasks      | elapsed: 21.5min
[Parallel(n_jobs=-1)]: Done 964 tasks      | elapsed: 24.5min
[Parallel(n_jobs=-1)]: Done 1009 tasks      | elapsed: 27.8min
[Parallel(n_jobs=-1)]: Done 1054 tasks      | elapsed: 31.5min
[Parallel(n_jobs=-1)]: Done 1101 tasks      | elapsed: 35.0min
[Parallel(n_jobs=-1)]: Done 1148 tasks      | elapsed: 36.9min
[Parallel(n_jobs=-1)]: Done 1197 tasks      | elapsed: 39.5min
[Parallel(n_jobs=-1)]: Done 1246 tasks      | elapsed: 42.7min
[Parallel(n_jobs=-1)]: Done 1297 tasks      | elapsed: 45.1min
[Parallel(n_jobs=-1)]: Done 1348 tasks      | elapsed: 49.2min
[Parallel(n_jobs=-1)]: Done 1401 tasks      | elapsed: 50.9min
[Parallel(n_jobs=-1)]: Done 1454 tasks      | elapsed: 52.6min
[Parallel(n_jobs=-1)]: Done 1509 tasks      | elapsed: 55.5min
[Parallel(n_jobs=-1)]: Done 1564 tasks      | elapsed: 58.9min
[Parallel(n_jobs=-1)]: Done 1621 tasks      | elapsed: 62.9min
[Parallel(n_jobs=-1)]: Done 1678 tasks      | elapsed: 67.9min
[Parallel(n_jobs=-1)]: Done 1737 tasks      | elapsed: 74.4min
[Parallel(n_jobs=-1)]: Done 1796 tasks      | elapsed: 82.8min
[Parallel(n_jobs=-1)]: Done 1857 tasks      | elapsed: 90.9min
[Parallel(n_jobs=-1)]: Done 1918 tasks      | elapsed: 101.3min
[Parallel(n_jobs=-1)]: Done 1981 tasks      | elapsed: 113.2min
[Parallel(n_jobs=-1)]: Done 2044 tasks      | elapsed: 123.4min
[Parallel(n_jobs=-1)]: Done 2109 tasks      | elapsed: 125.1min
[Parallel(n_jobs=-1)]: Done 2174 tasks      | elapsed: 127.2min
[Parallel(n_jobs=-1)]: Done 2241 tasks      | elapsed: 130.0min
[Parallel(n_jobs=-1)]: Done 2308 tasks      | elapsed: 132.9min
[Parallel(n_jobs=-1)]: Done 2377 tasks      | elapsed: 137.3min
[Parallel(n_jobs=-1)]: Done 2446 tasks      | elapsed: 142.0min
[Parallel(n_jobs=-1)]: Done 2517 tasks      | elapsed: 147.8min
[Parallel(n_jobs=-1)]: Done 2588 tasks      | elapsed: 154.6min
[Parallel(n_jobs=-1)]: Done 2661 tasks      | elapsed: 162.2min
[Parallel(n_jobs=-1)]: Done 2700 out of 2700 | elapsed: 167.0min finished

{'activation': 'logistic', 'early_stopping': True, 'hidden_layer_sizes': 100,
'learning_rate': 'adaptive', 'max_iter': 200, 'solver': 'sgd'}
0.1783
```

7.1 Checking for 4 different Train-test splits

```
[ ]: test_sizes = [0.4,0.3,0.2,0.1]
     for ts3 in test_sizes:
       print('\033[1m'+"For Train-test split: "+␣
      ↪str(int(100-(ts3*100)))+"-"+str(int(ts3*100))+ '\033[0m' +"\n")
       X_train, X_test, y_train, y_test = train_test_split(Xhat,copy_df['New Job␣
      ↪Label'], stratify=copy_df['New Job Label'], test_size=ts3, random_state=1)
       mlpc_3 = MLPClassifier(hidden_layer_sizes=100, activation='logistic',␣
      ↪solver='sgd',batch_size='auto', learning_rate='adaptive', max_iter=200,␣
      ↪verbose=False, early_stopping=True)
       mlpc_3.fit(X_train,y_train)
       y_pred = mlpc_3.predict(X_test)
       print("Accuracy: ",mlpc_3.score(X_test,y_test))
       print("\nConfusion Matrix for each class: ")
       matrix = confusion_matrix(y_test,y_pred,labels=np.array(copy_df['New Job␣
      ↪Label'].unique()))
       print(np.array(matrix))
       print("\nClasswise Accuracies")
       print(matrix.diagonal()/matrix.sum(axis=1))
       print("--------------------------------------------------------------")
```

```
For Train-test split: 60-40

Accuracy:  0.171875

Confusion Matrix for each class:
[[1009    0    0  360    0   38    0    0    0]
 [ 650    0    0  235    0   16    0    0    0]
 [ 474    0    0  164    0   11    0    0    0]
 [ 990    0    0  339    0   28    0    0    0]
 [ 339    0    0  122    0   10    0    0    0]
 [ 836    0    0  298    0   27    0    0    0]
 [ 643    0    0  243    0   17    0    0    0]
 [ 481    0    0  204    0    6    0    0    0]
 [ 332    0    0  124    0    4    0    0    0]]

Classwise Accuracies
[0.71712864 0.         0.         0.24981577 0.         0.02325581
 0.         0.         0.        ]
--------------------------------------------------------------
For Train-test split: 70-30

Accuracy:  0.17983333333333335

Confusion Matrix for each class:
[[585   0   0 430   0  40   0   0   0]
```

```
 [372   0   0 293   0  11   0   0   0]
 [268   0   0 206   0  13   0   0   0]
 [538   0   0 456   0  24   0   0   0]
 [195   0   0 147   0  11   0   0   0]
 [491   0   0 342   0  38   0   0   0]
 [367   0   0 292   0  18   0   0   0]
 [291   0   0 207   0  20   0   0   0]
 [187   0   0 150   0   8   0   0   0]]

Classwise Accuracies
[0.55450237 0.         0.         0.44793713 0.         0.04362801
 0.         0.         0.        ]
--------------------------------------------------------------
```
For Train-test split: 80-20

```
Accuracy:  0.17375

Confusion Matrix for each class:
[[415   0   0 275   0  14   0   0   0]
 [252   0   0 188   0  10   0   0   0]
 [195   0   0 123   0   6   0   0   0]
 [392   0   0 270   0  17   0   0   0]
 [132   0   0  95   0   8   0   0   0]
 [339   0   0 232   0  10   0   0   0]
 [265   0   0 176   0  11   0   0   0]
 [204   0   0 132   0   9   0   0   0]
 [147   0   0  76   0   7   0   0   0]]

Classwise Accuracies
[0.58948864 0.         0.         0.39764359 0.         0.0172117
 0.         0.         0.        ]
--------------------------------------------------------------
```
For Train-test split: 90-10

```
Accuracy:  0.1695

Confusion Matrix for each class:
[[173   0   0 179   0   0   0   0   0]
 [104   0   0 121   0   0   0   0   0]
 [ 83   0   0  79   0   0   0   0   0]
 [171   0   0 166   0   2   0   0   0]
 [ 57   0   0  60   0   1   0   0   0]
 [142   0   0 148   0   0   0   0   0]
 [118   0   0 108   0   0   0   0   0]
 [ 84   0   0  89   0   0   0   0   0]
 [ 56   0   0  59   0   0   0   0   0]]

Classwise Accuracies
```

```
[0.49147727 0.         0.         0.48967552 0.         0.
 0.         0.         0.        ]
-------------------------------------------------------------
```

8 Computing Feature Importance based on label manual clustering

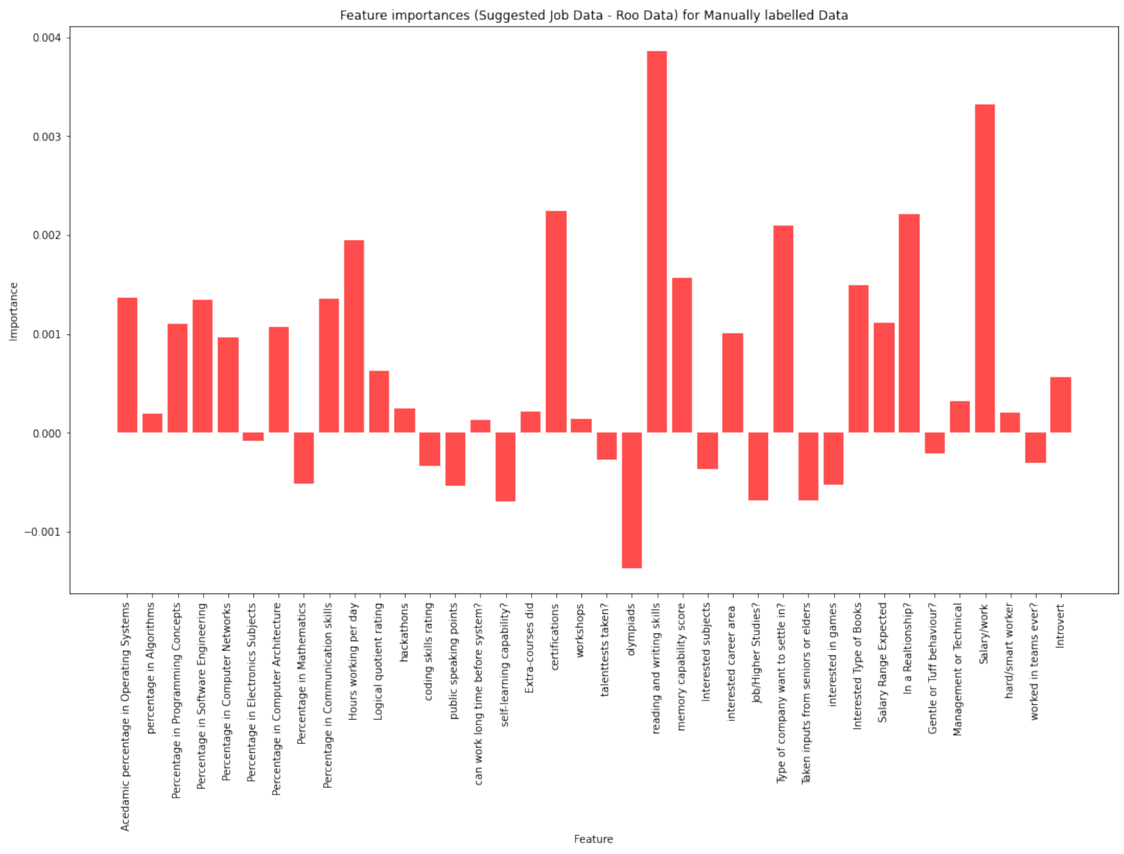
Now let us see which features were considered important in the case of this data.

[20]:
```
from sklearn.metrics import accuracy_score
```

[37]:
```
X_train, X_test, y_train, y_test = train_test_split(Xhat,copy_df['New Job␣
 ↪Label'], stratify=copy_df['New Job Label'],  random_state=1)
mlpc3 = MLPClassifier(hidden_layer_sizes=100, activation='logistic',␣
 ↪solver='sgd',batch_size='auto', learning_rate='adaptive', max_iter=200,␣
 ↪verbose=False, early_stopping=True)
mlpc3.fit(Xhat,copy_df['New Job Label'])
y_pred = mlpc3.predict(X_test)
```

[25]:
```
def get_feature_importance(j, n,clf):
  s = accuracy_score(y_test, y_pred) # baseline score
  total = 0.0
  for i in range(n):
    perm = np.random.permutation(range(X_test.shape[0]))
    X_test_ = X_test.copy()
    X_test_[:, j] = X_test[perm, j]
    y_pred_ = clf.predict(X_test_)
    s_ij = accuracy_score(y_test, y_pred_)
    total += s_ij
  return s - total / n
```

[39]:
```
# Feature importances
f = []
for j in range(Xhat.shape[1]):
  f_j = get_feature_importance(j, 100, mlpc3)
  f.append(f_j)
# Plot
plt.figure(figsize=(18, 10))
plt.bar(range(Xhat.shape[1]), f, color="r", alpha=0.7)
plt.xticks(range(X_test.shape[1]), np.array(copy_df.drop(['Suggested Job␣
 ↪Role','New Job Label'],axis=1).columns), rotation=90)
plt.xlabel("Feature")
plt.ylabel("Importance")
plt.title("Feature importances (Suggested Job Data - Roo Data) for Manually␣
 ↪labelled Data")
plt.show()
```

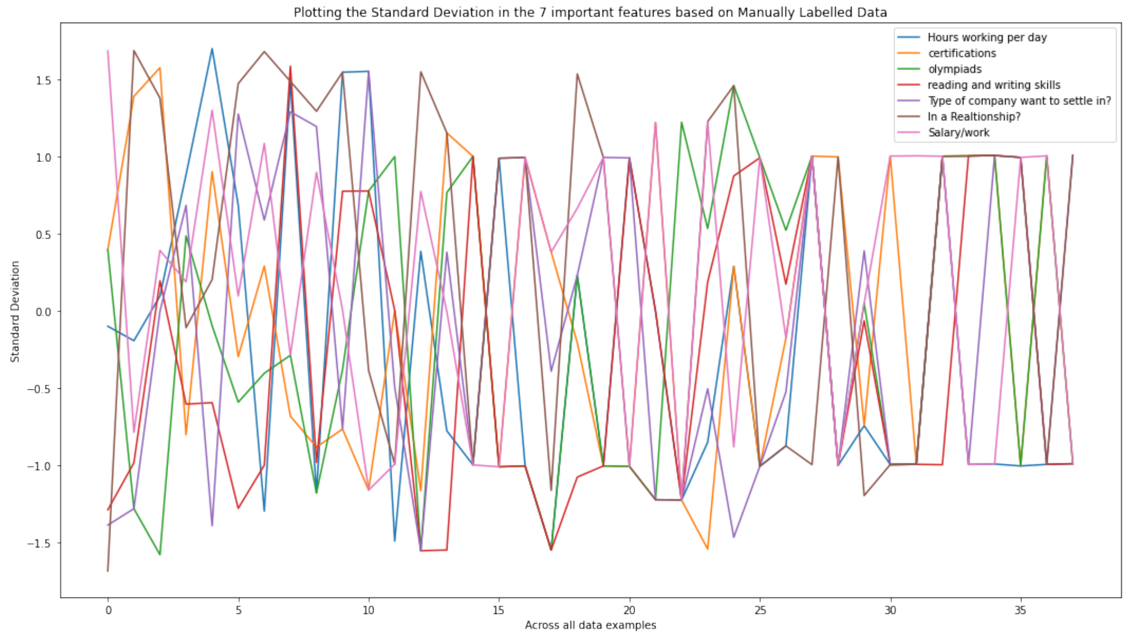


[55]:
```
print('\033[1m'+"The important features according to the manually labelled data␣
↪and model are: "+ '\033[0m')
features_index = [9,17,20,21,26,31,34]
for col_num in features_index:
  # print(col_num)
  print(np.array(copy_df.drop(['Suggested Job Role','New Job Label'],axis=1).
↪columns)[col_num])
```

```
The important features according to the manually labelled data and model
are:
Hours working per day
certifications
olympiads
reading and writing skills
Type of company want to settle in?
In a Realtionship?
Salary/work
```

[58]:
```
# plotting the standard deviations for the 8 important attributes or features␣
↪according to the MLP ANN Classifier
fig, ax = plt.subplots(figsize=(18, 10))
```

```
plt.plot(Xhat[9],label = 'Hours working per day')
plt.plot(Xhat[17],label = 'certifications')
plt.plot(Xhat[20],label = 'olympiads')
plt.plot(Xhat[21],label = 'reading and writing skills')
plt.plot(Xhat[26],label = 'Type of company want to settle in?')
plt.plot(Xhat[31],label = 'In a Realtionship?')
plt.plot(Xhat[34],label = 'Salary/work')
plt.legend()
plt.title("Plotting the Standard Deviation in the 7 important features based on␣
 ↪Manually Labelled Data")
plt.ylabel("Standard Deviation")
plt.xlabel("Across all data examples")
plt.show()
```



9 Checking based on K Means cluster labels

Since the optimum number of clusters obtained after applying the KMeans algorithm Within Cluster Sum Of Squares (WCSS) Elbow Method is equal to 9 therefore we will be generating 9 cluster labels for the data and check the model performance for the same.

[41]:
```
kmeans2 = KMeans(n_clusters=9).fit(df.drop(['Suggested Job␣
 ↪Role'],axis=1),df['Suggested Job Role'])
new_cluster_data = pd.DataFrame(df.drop(['Suggested Job Role'],axis=1))
new_cluster_data['Job Labels'] = data['Suggested Job Role']
new_cluster_data['Cluster Labels'] = kmeans2.labels_
```

```
new_cluster_data
```

[41]:
```
       Acedamic percentage in Operating Systems  ...  Cluster Labels
0                                             69  ...               7
1                                             78  ...               6
2                                             71  ...               4
3                                             76  ...               1
4                                             92  ...               3
...                                          ...  ...             ...
19995                                         83  ...               7
19996                                         80  ...               1
19997                                         83  ...               4
19998                                         68  ...               1
19999                                         73  ...               2

[20000 rows x 40 columns]
```

Let us now check the different job roles that were clustered together as same cluster labels...

[42]:
```
job_cluster_dict = {}
num_jobs_in_clusters = {}
for c in range(len(new_cluster_data['Cluster Labels'].unique())):
  #print(c)
  job_array = np.array(new_cluster_data.loc[new_cluster_data['Cluster Labels']␣
 ↪== c]['Job Labels'].unique())
  job_cluster_dict[c] = job_array
  num_jobs_in_clusters[c] = len(job_array)
#print(job_cluster_dict)
print(num_jobs_in_clusters)
```

```
{0: 34, 1: 34, 2: 34, 3: 34, 4: 34, 5: 34, 6: 34, 7: 34, 8: 34}
```

This analysis done above shows that each cluster formed contains all the 34 original job classes which hinders the process of predicting and suggesting a suitable job role to students.

10 Attempt 4 - On KMeans Cluster Labelled Data

Now building model on this and finding optimal parameters using Grid Search to train and evaluate Scikit-Learn ANN MLP Classifier Model

[ ]:
```
mlpc4 = MLPClassifier()
clf4 = GridSearchCV(mlpc4, param_grid,cv = 5, n_jobs = -1, verbose=10)
clf4.fit(new_cluster_data.drop(['Job Labels','Cluster␣
 ↪Labels'],axis=1),new_cluster_data['Cluster Labels'])
print(clf4.best_params_)
print(clf4.best_score_)
```

```
Fitting 5 folds for each of 540 candidates, totalling 2700 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    4.2s
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:    7.6s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   13.4s
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:   19.8s
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed:   37.6s
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed:   45.6s
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done  57 tasks      | elapsed:  1.7min
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed:  2.1min
[Parallel(n_jobs=-1)]: Done  81 tasks      | elapsed:  2.6min
[Parallel(n_jobs=-1)]: Done  94 tasks      | elapsed:  3.0min
[Parallel(n_jobs=-1)]: Done 109 tasks      | elapsed:  3.5min
[Parallel(n_jobs=-1)]: Done 124 tasks      | elapsed:  4.2min
[Parallel(n_jobs=-1)]: Done 141 tasks      | elapsed:  4.7min
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed:  5.3min
[Parallel(n_jobs=-1)]: Done 177 tasks      | elapsed:  6.1min
[Parallel(n_jobs=-1)]: Done 196 tasks      | elapsed:  6.7min
[Parallel(n_jobs=-1)]: Done 217 tasks      | elapsed:  7.8min
[Parallel(n_jobs=-1)]: Done 238 tasks      | elapsed:  8.5min
[Parallel(n_jobs=-1)]: Done 261 tasks      | elapsed:  9.8min
[Parallel(n_jobs=-1)]: Done 284 tasks      | elapsed: 10.5min
[Parallel(n_jobs=-1)]: Done 309 tasks      | elapsed: 12.2min
[Parallel(n_jobs=-1)]: Done 334 tasks      | elapsed: 13.4min
[Parallel(n_jobs=-1)]: Done 361 tasks      | elapsed: 15.0min
[Parallel(n_jobs=-1)]: Done 388 tasks      | elapsed: 16.5min
[Parallel(n_jobs=-1)]: Done 417 tasks      | elapsed: 18.3min
[Parallel(n_jobs=-1)]: Done 446 tasks      | elapsed: 20.6min
[Parallel(n_jobs=-1)]: Done 477 tasks      | elapsed: 22.1min
[Parallel(n_jobs=-1)]: Done 508 tasks      | elapsed: 24.5min
[Parallel(n_jobs=-1)]: Done 541 tasks      | elapsed: 27.2min
[Parallel(n_jobs=-1)]: Done 574 tasks      | elapsed: 30.1min
[Parallel(n_jobs=-1)]: Done 609 tasks      | elapsed: 32.4min
[Parallel(n_jobs=-1)]: Done 644 tasks      | elapsed: 35.3min
[Parallel(n_jobs=-1)]: Done 681 tasks      | elapsed: 39.1min
[Parallel(n_jobs=-1)]: Done 718 tasks      | elapsed: 41.1min
[Parallel(n_jobs=-1)]: Done 757 tasks      | elapsed: 42.5min
[Parallel(n_jobs=-1)]: Done 796 tasks      | elapsed: 44.4min
[Parallel(n_jobs=-1)]: Done 837 tasks      | elapsed: 47.1min
[Parallel(n_jobs=-1)]: Done 878 tasks      | elapsed: 50.1min
[Parallel(n_jobs=-1)]: Done 921 tasks      | elapsed: 53.3min
[Parallel(n_jobs=-1)]: Done 964 tasks      | elapsed: 57.8min
[Parallel(n_jobs=-1)]: Done 1009 tasks      | elapsed: 62.6min
[Parallel(n_jobs=-1)]: Done 1054 tasks      | elapsed: 67.7min
[Parallel(n_jobs=-1)]: Done 1101 tasks      | elapsed: 74.7min
[Parallel(n_jobs=-1)]: Done 1148 tasks      | elapsed: 81.8min
[Parallel(n_jobs=-1)]: Done 1197 tasks      | elapsed: 89.8min
```

```
[Parallel(n_jobs=-1)]: Done 1246 tasks      | elapsed: 99.4min
[Parallel(n_jobs=-1)]: Done 1297 tasks      | elapsed: 110.0min
[Parallel(n_jobs=-1)]: Done 1348 tasks      | elapsed: 121.0min
[Parallel(n_jobs=-1)]: Done 1401 tasks      | elapsed: 123.1min
[Parallel(n_jobs=-1)]: Done 1454 tasks      | elapsed: 124.9min
[Parallel(n_jobs=-1)]: Done 1509 tasks      | elapsed: 128.0min
[Parallel(n_jobs=-1)]: Done 1564 tasks      | elapsed: 131.8min
[Parallel(n_jobs=-1)]: Done 1621 tasks      | elapsed: 136.3min
[Parallel(n_jobs=-1)]: Done 1678 tasks      | elapsed: 142.0min
[Parallel(n_jobs=-1)]: Done 1737 tasks      | elapsed: 148.7min
[Parallel(n_jobs=-1)]: Done 1796 tasks      | elapsed: 157.8min
[Parallel(n_jobs=-1)]: Done 1857 tasks      | elapsed: 166.5min
[Parallel(n_jobs=-1)]: Done 1918 tasks      | elapsed: 177.5min
[Parallel(n_jobs=-1)]: Done 1981 tasks      | elapsed: 190.6min
[Parallel(n_jobs=-1)]: Done 2044 tasks      | elapsed: 201.3min
[Parallel(n_jobs=-1)]: Done 2109 tasks      | elapsed: 203.5min
[Parallel(n_jobs=-1)]: Done 2174 tasks      | elapsed: 206.0min
[Parallel(n_jobs=-1)]: Done 2241 tasks      | elapsed: 209.2min
[Parallel(n_jobs=-1)]: Done 2308 tasks      | elapsed: 212.7min
[Parallel(n_jobs=-1)]: Done 2377 tasks      | elapsed: 217.6min
[Parallel(n_jobs=-1)]: Done 2446 tasks      | elapsed: 222.8min
[Parallel(n_jobs=-1)]: Done 2517 tasks      | elapsed: 229.0min
[Parallel(n_jobs=-1)]: Done 2588 tasks      | elapsed: 236.2min
[Parallel(n_jobs=-1)]: Done 2661 tasks      | elapsed: 244.4min
[Parallel(n_jobs=-1)]: Done 2700 out of 2700 | elapsed: 249.6min finished

{'activation': 'logistic', 'early_stopping': True, 'hidden_layer_sizes': 200,
'learning_rate': 'adaptive', 'max_iter': 100, 'solver': 'adam'}
0.8121499999999999
```

10.1 Checking for 4 different Train-test splits

```
[ ]: test_sizes = [0.4,0.3,0.2,0.1]
     for ts4 in test_sizes:
       print('\033[1m'+"For Train-test split: "+␣
     ↪str(int(100-(ts4*100)))+"-"+str(int(ts4*100))+ '\033[0m' +"\n")
       X_train, X_test, y_train, y_test = train_test_split(new_cluster_data.
     ↪drop(['Job Labels','Cluster Labels'],axis=1),new_cluster_data['Cluster␣
     ↪Labels'], stratify=new_cluster_data['Cluster Labels'], test_size=ts4,␣
     ↪random_state=1)
       mlpc_4 = MLPClassifier(hidden_layer_sizes=200, activation='logistic',␣
     ↪solver='adam',batch_size='auto', learning_rate='adaptive', max_iter=100,␣
     ↪verbose=False, early_stopping=True)
       mlpc_4.fit(X_train,y_train)
       y_pred = mlpc_4.predict(X_test)
       print("Accuracy: ",mlpc_4.score(X_test,y_test))
       print("\nConfusion Matrix for each class: ")
```

```
matrix = confusion_matrix(y_test,y_pred,labels=np.array(copy_df['New Job␣
↪Label'].unique()))
print(np.array(matrix))
print("\nClasswise Accuracies")
print(matrix.diagonal()/matrix.sum(axis=1))
print("--------------------------------------------------------------")
```

```
For Train-test split: 60-40

Accuracy:  0.804

Confusion Matrix for each class:
[[794  21   4  17  27   0  16   3   9]
 [ 82 433 114  21   3  40  41  69  75]
 [  6  30 708  29  60   5  26   7  21]
 [ 17   4  28 755  10  16  21  18  18]
 [ 47   0  75  14 586  41  45  39  51]
 [ 16  19  18  12  45 742  18   8  12]
 [  6   8  11   5   8   1 872   0   7]
 [  8  25  11  12  35   1  15 716  21]
 [  1  19   7  10  29   1   8   1 826]]

Classwise Accuracies
[0.89113356 0.49316629 0.79372197 0.85118377 0.65256125 0.83370787
 0.94989107 0.84834123 0.91574279]
--------------------------------------------------------------
For Train-test split: 70-30

Accuracy:  0.8033333333333333

Confusion Matrix for each class:
[[543  44   1  10  35  14   9   1  11]
 [ 27 459  48   6   0  27  22  32  38]
 [  1  59 493  23  57  12   9   5  10]
 [ 13   7  18 565  10  24   6  13   9]
 [ 26   3  53   8 435  53  22  32  41]
 [  0  21   4   6  18 613   3   1   1]
 [  4  24  10   6   9  16 595   9  16]
 [  5  40   6  10  23  11   5 526   7]
 [  1  35   4   7  28  10   0   1 591]]

Classwise Accuracies
[0.81287425 0.69650986 0.73692078 0.84962406 0.64635958 0.91904048
 0.86357039 0.83096367 0.87296898]
--------------------------------------------------------------
For Train-test split: 80-20
```

```
Accuracy:  0.81525

Confusion Matrix for each class:
[[386  26   1   3  10   2   9   2   7]
 [ 21 294  29   7   0  13  15  32  28]
 [  2  38 332  18  28   5   9   7   7]
 [ 10   8  10 383   1   8   9   8   6]
 [ 25   0  44   9 267  23  19  32  30]
 [  4  17   4  10  14 390   3   2   1]
 [  2  11   3   5   2   0 429   2   5]
 [  3  25   1   6   7   1   6 367   6]
 [  0  18   4   3  10   1   2   0 413]]

Classwise Accuracies
[0.86547085 0.66970387 0.74439462 0.86455982 0.59465479 0.87640449
 0.93464052 0.86966825 0.91574279]
--------------------------------------------------------------
For Train-test split: 90-10

Accuracy:  0.808

Confusion Matrix for each class:
[[195   8   0   7  10   0   1   0   2]
 [ 10 128  29  11   1   7   7  12  15]
 [  1  11 179  11  17   2   1   0   1]
 [  2   1   1 209   1   4   0   3   1]
 [  9   0  23   6 145   6   5  12  18]
 [  2   8   4   4  11 191   1   0   1]
 [  2   6   6   6   1   1 205   0   2]
 [  5   9   3   9   8   3   3 167   4]
 [  0   8   3   8   8   2   0   0 197]]

Classwise Accuracies
[0.87443946 0.58181818 0.80269058 0.94144144 0.64732143 0.86036036
 0.89519651 0.79146919 0.87168142]
--------------------------------------------------------------
```

11 Standard Scaling the Data for KMeans Optimal 9 clusters

[43]:
```
# new_cluster_data.drop(['Job Labels','Cluster Labels'],axis=1)
Xhat2 = new_cluster_data.drop(['Job Labels','Cluster Labels'],axis=1)
scaler2 = StandardScaler()
Xhat2 = scaler2.fit_transform(Xhat2)
print(pd.DataFrame(Xhat2))
```

```
             0         1         2   ...        35        36        37
0     -0.793450 -1.380807  0.096941  ... -1.004711 -1.005415 -1.009748
```

```
1      0.098925 -1.479803 -0.396421  ... -1.004711  0.994615  0.990347
2     -0.595145  0.896086  1.379680  ... -1.004711  0.994615  0.990347
3     -0.099381  0.995082 -1.679160  ...  0.995311 -1.005415  0.990347
4      1.487064 -1.479803  1.281008  ... -1.004711 -1.005415  0.990347
...         ...       ...       ...  ...       ...       ...       ...
19995  0.594689 -0.984826 -1.481815  ...  0.995311 -1.005415 -1.009748
19996  0.297230 -0.786835  0.590302  ... -1.004711  0.994615  0.990347
19997  0.594689 -0.687840  0.294285  ... -1.004711  0.994615  0.990347
19998 -0.892603  0.995082  1.379680  ...  0.995311 -1.005415 -1.009748
19999 -0.396839  0.005128 -0.297748  ... -1.004711 -1.005415 -1.009748

[20000 rows x 38 columns]
```

12 Attempt 5 - Standard Scaling on KMeans Cluster Labelled Data

Now building model on this and finding optimal parameters using Grid Search to train and evaluate Scikit-Learn ANN MLP Classifier Model

[ ]:
```
mlpc5 = MLPClassifier()
clf5 = GridSearchCV(mlpc5, param_grid,cv = 5, n_jobs = -1, verbose=10)
clf5.fit(Xhat2,new_cluster_data['Cluster Labels'])
print(clf5.best_params_)
print(clf5.best_score_)
```

```
Fitting 5 folds for each of 540 candidates, totalling 2700 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:    2.1s
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed:    3.0s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:   12.4s
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:   21.5s
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed:   28.4s
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed:   41.0s
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:   51.3s
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done  57 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done  81 tasks      | elapsed:  1.8min
[Parallel(n_jobs=-1)]: Done  94 tasks      | elapsed:  2.0min
[Parallel(n_jobs=-1)]: Done 109 tasks      | elapsed:  2.4min
[Parallel(n_jobs=-1)]: Done 124 tasks      | elapsed:  3.0min
[Parallel(n_jobs=-1)]: Done 141 tasks      | elapsed:  3.6min
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed:  4.1min
[Parallel(n_jobs=-1)]: Done 177 tasks      | elapsed:  4.6min
[Parallel(n_jobs=-1)]: Done 196 tasks      | elapsed:  5.2min
[Parallel(n_jobs=-1)]: Done 217 tasks      | elapsed:  5.7min
[Parallel(n_jobs=-1)]: Done 238 tasks      | elapsed:  6.5min
[Parallel(n_jobs=-1)]: Done 261 tasks      | elapsed:  7.4min
```

```
[Parallel(n_jobs=-1)]: Done 284 tasks      | elapsed:  8.2min
[Parallel(n_jobs=-1)]: Done 309 tasks      | elapsed:  9.1min
[Parallel(n_jobs=-1)]: Done 334 tasks      | elapsed:  9.9min
[Parallel(n_jobs=-1)]: Done 361 tasks      | elapsed: 11.2min
[Parallel(n_jobs=-1)]: Done 388 tasks      | elapsed: 12.4min
[Parallel(n_jobs=-1)]: Done 417 tasks      | elapsed: 13.7min
[Parallel(n_jobs=-1)]: Done 446 tasks      | elapsed: 15.0min
[Parallel(n_jobs=-1)]: Done 477 tasks      | elapsed: 16.7min
[Parallel(n_jobs=-1)]: Done 508 tasks      | elapsed: 18.8min
[Parallel(n_jobs=-1)]: Done 541 tasks      | elapsed: 20.7min
[Parallel(n_jobs=-1)]: Done 574 tasks      | elapsed: 22.2min
[Parallel(n_jobs=-1)]: Done 609 tasks      | elapsed: 24.6min
[Parallel(n_jobs=-1)]: Done 644 tasks      | elapsed: 27.4min
[Parallel(n_jobs=-1)]: Done 681 tasks      | elapsed: 29.7min
[Parallel(n_jobs=-1)]: Done 718 tasks      | elapsed: 32.2min
[Parallel(n_jobs=-1)]: Done 757 tasks      | elapsed: 33.6min
[Parallel(n_jobs=-1)]: Done 796 tasks      | elapsed: 35.7min
[Parallel(n_jobs=-1)]: Done 837 tasks      | elapsed: 38.9min
[Parallel(n_jobs=-1)]: Done 878 tasks      | elapsed: 41.2min
[Parallel(n_jobs=-1)]: Done 921 tasks      | elapsed: 43.6min
[Parallel(n_jobs=-1)]: Done 964 tasks      | elapsed: 47.4min
[Parallel(n_jobs=-1)]: Done 1009 tasks      | elapsed: 51.3min
[Parallel(n_jobs=-1)]: Done 1054 tasks      | elapsed: 53.9min
[Parallel(n_jobs=-1)]: Done 1101 tasks      | elapsed: 59.7min
[Parallel(n_jobs=-1)]: Done 1148 tasks      | elapsed: 64.6min
[Parallel(n_jobs=-1)]: Done 1197 tasks      | elapsed: 69.8min
[Parallel(n_jobs=-1)]: Done 1246 tasks      | elapsed: 77.1min
[Parallel(n_jobs=-1)]: Done 1297 tasks      | elapsed: 82.7min
[Parallel(n_jobs=-1)]: Done 1348 tasks      | elapsed: 91.6min
[Parallel(n_jobs=-1)]: Done 1401 tasks      | elapsed: 94.5min
[Parallel(n_jobs=-1)]: Done 1454 tasks      | elapsed: 97.0min
[Parallel(n_jobs=-1)]: Done 1509 tasks      | elapsed: 101.2min
[Parallel(n_jobs=-1)]: Done 1564 tasks      | elapsed: 103.9min
[Parallel(n_jobs=-1)]: Done 1621 tasks      | elapsed: 108.1min
[Parallel(n_jobs=-1)]: Done 1678 tasks      | elapsed: 112.0min
[Parallel(n_jobs=-1)]: Done 1737 tasks      | elapsed: 116.7min
[Parallel(n_jobs=-1)]: Done 1796 tasks      | elapsed: 122.1min
[Parallel(n_jobs=-1)]: Done 1857 tasks      | elapsed: 128.9min
[Parallel(n_jobs=-1)]: Done 1918 tasks      | elapsed: 136.3min
[Parallel(n_jobs=-1)]: Done 1981 tasks      | elapsed: 144.3min
[Parallel(n_jobs=-1)]: Done 2044 tasks      | elapsed: 151.8min
[Parallel(n_jobs=-1)]: Done 2109 tasks      | elapsed: 154.8min
[Parallel(n_jobs=-1)]: Done 2174 tasks      | elapsed: 158.7min
[Parallel(n_jobs=-1)]: Done 2241 tasks      | elapsed: 161.7min
[Parallel(n_jobs=-1)]: Done 2308 tasks      | elapsed: 165.8min
[Parallel(n_jobs=-1)]: Done 2377 tasks      | elapsed: 169.4min
[Parallel(n_jobs=-1)]: Done 2446 tasks      | elapsed: 174.4min
[Parallel(n_jobs=-1)]: Done 2517 tasks      | elapsed: 178.8min
```

```
[Parallel(n_jobs=-1)]: Done 2588 tasks      | elapsed: 185.5min
[Parallel(n_jobs=-1)]: Done 2661 tasks      | elapsed: 190.9min
[Parallel(n_jobs=-1)]: Done 2700 out of 2700 | elapsed: 195.4min finished

{'activation': 'identity', 'early_stopping': True, 'hidden_layer_sizes': 200,
'learning_rate': 'invscaling', 'max_iter': 100, 'solver': 'lbfgs'}
0.99215
```

12.1 Checking for 4 different Train-test splits

[44]:
```
test_sizes = [0.4,0.3,0.2,0.1]
for ts5 in test_sizes:
  print('\033[1m'+"For Train-test split: "+␣
 ↪str(int(100-(ts5*100)))+"-"+str(int(ts5*100))+ '\033[0m' +"\n")
  X_train, X_test, y_train, y_test =␣
 ↪train_test_split(Xhat2,new_cluster_data['Cluster Labels'],␣
 ↪stratify=new_cluster_data['Cluster Labels'], test_size=ts5, random_state=1)
  mlpc_5 = MLPClassifier(hidden_layer_sizes=200, activation='identity',␣
 ↪solver='lbfgs',batch_size='auto', learning_rate='invscaling', max_iter=100,␣
 ↪verbose=False, early_stopping=True)
  mlpc_5.fit(X_train,y_train)
  y_pred = mlpc_5.predict(X_test)
  print("Accuracy: ",mlpc_5.score(X_test,y_test))
  print("\nConfusion Matrix for each class: ")
  matrix = confusion_matrix(y_test,y_pred,labels=np.array(copy_df['New Job␣
 ↪Label'].unique()))
  print(np.array(matrix))
  print("\nClasswise Accuracies")
  print(matrix.diagonal()/matrix.sum(axis=1))
  print("--------------------------------------------------------------")
```

```
For Train-test split: 60-40

Accuracy:  0.988875

Confusion Matrix for each class:
[[898   1   0   0   3   2   2   2   0]
 [  2 866   1   1   2   0   0   2   2]
 [  0   1 898   2   0   0   4   1   1]
 [  4   2   1 849   0   1   2   1   0]
 [  1   1   0   2 881   1   1   0   1]
 [  0   3   1   2   1 863   2   2   5]
 [  1   0   0   1   1   1 901   2   2]
 [  0   0   1   1   2   1   2 908   3]
 [  2   3   0   1   0   1   0   1 847]]

Classwise Accuracies
```

```
[0.98898678 0.98858447 0.99007718 0.9872093  0.99211712 0.9817975
 0.99119912 0.98910675 0.99064327]
--------------------------------------------------------------
```
For Train-test split: 70-30

```
Accuracy:  0.9923333333333333

Confusion Matrix for each class:
[[676   0   0   0   2   1   0   2   0]
 [  0 654   0   0   0   1   0   0   2]
 [  0   1 676   0   0   1   0   1   1]
 [  2   2   3 635   0   1   1   0   1]
 [  0   0   0   1 664   0   1   0   0]
 [  0   1   1   1   2 651   1   1   1]
 [  0   1   0   0   1   0 675   2   3]
 [  1   0   1   0   1   1   0 684   1]
 [  1   1   0   0   0   0   0   0 639]]

Classwise Accuracies
[0.99265786 0.99543379 0.99411765 0.98449612 0.996997   0.98786039
 0.98973607 0.99274311 0.99687988]
--------------------------------------------------------------
```
For Train-test split: 80-20

```
Accuracy:  0.99425

Confusion Matrix for each class:
[[453   0   0   0   0   0   0   1   0]
 [  0 436   0   0   1   0   1   0   0]
 [  0   0 452   0   0   0   1   1   0]
 [  1   0   1 425   1   1   0   0   1]
 [  0   0   0   1 440   1   2   0   0]
 [  0   1   0   0   1 435   1   1   0]
 [  1   1   0   0   0   0 452   0   0]
 [  1   0   0   0   1   0   0 457   0]
 [  0   0   0   0   0   0   1   0 427]]

Classwise Accuracies
[0.99779736 0.99543379 0.99559471 0.98837209 0.99099099 0.99088838
 0.99559471 0.9956427  0.99766355]
--------------------------------------------------------------
```
For Train-test split: 90-10

```
Accuracy:  0.996

Confusion Matrix for each class:
[[225   0   0   0   0   1   0   1   0]
 [  0 219   0   0   0   0   0   0   0]
 [  0   0 227   0   0   0   0   0   0]
 [  0   0   1 213   0   0   0   0   1]
 [  0   0   0   1 221   0   0   0   0]
 [  0   0   0   0   0 220   0   0   0]
 [  0   0   0   0   1   0 226   0   0]
 [  0   0   0   0   1   0   0 228   0]
 [  0   1   0   0   0   0   0   0 213]]

Classwise Accuracies
[0.99118943 1.         1.         0.99069767 0.9954955  1.
 0.99559471 0.99563319 0.9953271 ]
----------------------------------------------------------------
```

13 Analysis on the highest accuracy data (i.e. Accuracy of avg. 99.2%)

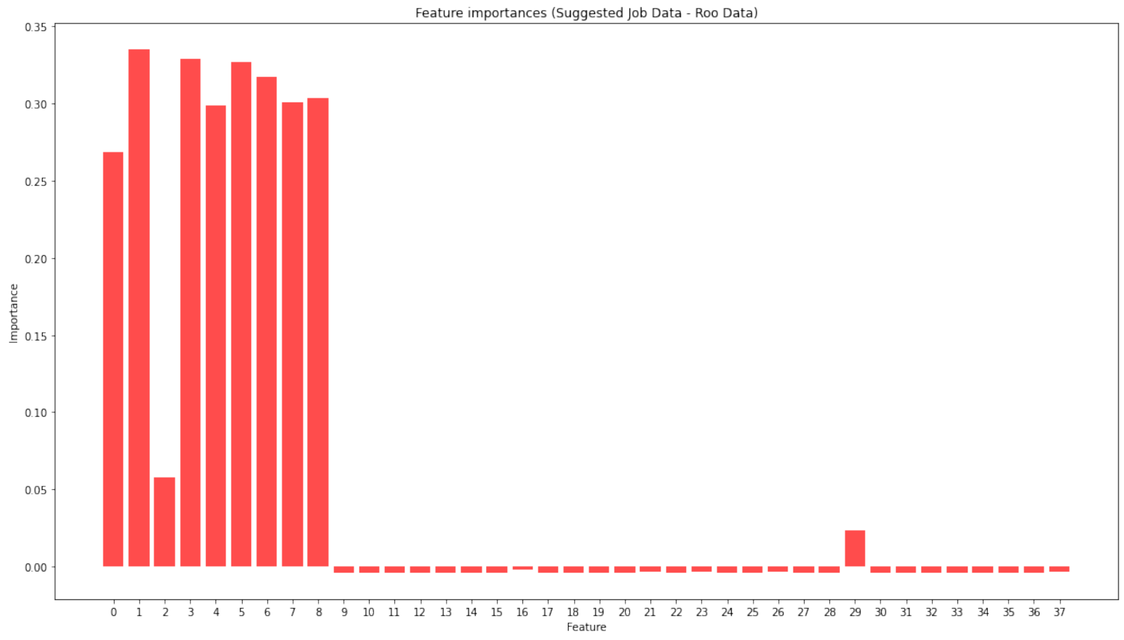
[33]:
```python
from sklearn.metrics import accuracy_score
```

[47]:
```python
mlpc5 = MLPClassifier(hidden_layer_sizes=200, activation='identity',␣
 ↪solver='lbfgs',batch_size='auto', learning_rate='invscaling', max_iter=100,␣
 ↪verbose=False, early_stopping=True)
mlpc5.fit(Xhat2,new_cluster_data['Cluster Labels'])
```

[47]:
```
MLPClassifier(activation='identity', early_stopping=True,
              hidden_layer_sizes=200, learning_rate='invscaling', max_iter=100,
              solver='lbfgs')
```

[48]:
```python
def get_feature_importance(j, n,clf):
  s = accuracy_score(y_test, y_pred) # baseline score
  total = 0.0
  for i in range(n):
    perm = np.random.permutation(range(X_test.shape[0]))
    X_test_ = X_test.copy()
    X_test_[:, j] = X_test[perm, j]
    y_pred_ = clf.predict(X_test_)
    s_ij = accuracy_score(y_test, y_pred_)
    total += s_ij
  return s - total / n
```

[50]:
```python
# Feature importances
f = []
for j in range(Xhat2.shape[1]):
  f_j = get_feature_importance(j, 100, mlpc5)
  f.append(f_j)
# Plot
plt.figure(figsize=(18,10))
plt.bar(range(Xhat2.shape[1]), f, color="r", alpha=0.7)
plt.xticks(ticks=range(X_test.shape[1]))
plt.xlabel("Feature")
plt.ylabel("Importance")
plt.title("Feature importances (Suggested Job Data - Roo Data)")
plt.show()
```



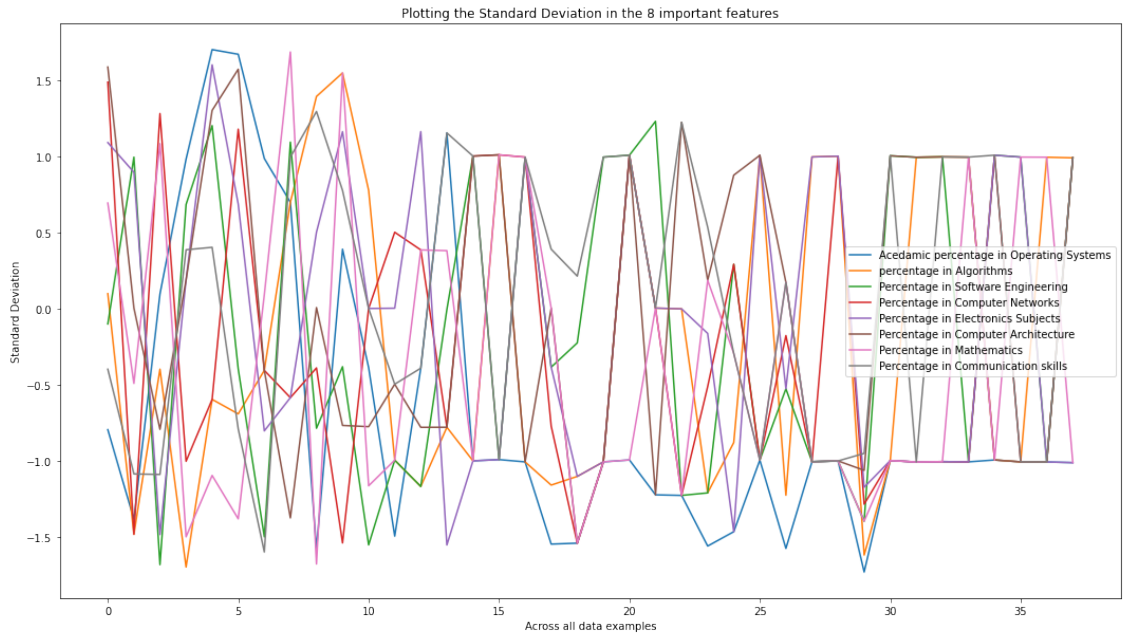
[52]:
```
print('\033[1m'+"The important features according to the highest accuracy data␣
↪and model are: "+ '\033[0m')
features_index = [0,1,3,4,5,6,7,8]
for col_num in features_index:
  # print(col_num)
  print(np.array(new_cluster_data.drop(['Job Labels','Cluster Labels'],axis=1).
↪columns)[col_num])
```

```
The important features according to the highest accuracy data and model are:

Acedamic percentage in Operating Systems
percentage in Algorithms
Percentage in Software Engineering
Percentage in Computer Networks
Percentage in Electronics Subjects
Percentage in Computer Architecture
Percentage in Mathematics
Percentage in Communication skills
```

[53]:
```
# plotting the standard deviations for the 8 important attributes or features␣
↪according to the MLP ANN Classifier
fig, ax = plt.subplots(figsize=(18, 10))
```

```
plt.plot(Xhat2[0],label = 'Acedamic percentage in Operating Systems')
plt.plot(Xhat2[1],label = 'percentage in Algorithms')
plt.plot(Xhat2[3],label = 'Percentage in Software Engineering')
plt.plot(Xhat2[4],label = 'Percentage in Computer Networks')
plt.plot(Xhat2[5],label = 'Percentage in Electronics Subjects')
plt.plot(Xhat2[6],label = 'Percentage in Computer Architecture')
plt.plot(Xhat2[7],label = 'Percentage in Mathematics')
plt.plot(Xhat2[8],label = 'Percentage in Communication skills')
plt.legend()
plt.title("Plotting the Standard Deviation in the 8 important features")
plt.ylabel("Standard Deviation")
plt.xlabel("Across all data examples")
plt.show()
```



Thus the highest accuracy cluster labelled data based model only focusses on segmenting the students based on their marks in different elective subjects therefore it does not serve the purpose of the end goal here, that is, suggesting a suitable job role. For a job role, we need to specifically find a job based on the subjects or courses the student scores the highest in besides other factors like team spirit, coding skills rating, logical quotient rating etc. available as data attributes. Here in this cluster data, the optimal 9 clusters include all the original 34 classes in each of its clusters. Hence, even though it has achieved high accuracy, it cannot be used to suggest a suitable job role to a student candidate.

The concept of machine learning is to use statistical analysis to assign optimal weights. Thus, interfering here with the whole concept to apply manual weights to different features, you need really strong evidence that this is crucial to the process you are trying to model, and for some reason your model is currently missing it. Also attribute weighting is a rare technique while feeding it to different models and hence not much researched.

14 Printing the Jupyter Notebook code as PDF report

[ ]:
```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('AI_Assignment4_MT20075.ipynb')
```

```
File colab_pdf.py already there; not retrieving.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Extracting templates from packages: 100%
```

[ ]:
```

```

5. Results and Analysis

The average model accuracy, confusion matrix and class-wise accuracy for each of the 5 attempts is listed as a sub section after the attempts section. The average model accuracy in of each of the 5 attempts are listed below:

Attempt 1 : around 5% (Detailed output on Page 14)
Attempt 2 : around 17.7% (Detailed output on Page 31)
Attempt 3 : around 17.5% (Detailed output on Page 35)
Attempt 4 : around 81% (Detailed output on Page 43)
Attempt 5 : around 98% (Detailed output on Page 47)

Emphasizing the end goal of suggesting suitable job roles to students based on their inputs in the data on 38 different attributes, from the feature importance plot analysis we observed that the last two attempts on data modification and model evaluation took only the first 9 attributes (refer to the feature importance plot in page 50) and left all the rest attributes while considering the best model and the best accuracy score. Other important attributes like logical quotient rating, coding skills rating, hackathons, self-learning capability, certifications, workshops, interested subjects, interested career area and so on were left out while computing model performance and optimal model score in the last two attempts.

On the other hand, in case of attempt 2 and attempt 3, we see that a mixture of different attributes were considered (refer to the feature importance plot in page 38) even though the accuracy is very low which is 17%. Had some more attributes been taken into account with different variations and emphasis on the model and the target attribute as a whole, the MLP classifier model would have performed better on the manually labelled data. Since the other attributes had fewer categories of inputs, it was difficult for the model to statistically analyse which other attributes would significantly perform well on the manually labelled data.

Since MLP classifiers fire nodes which are computed to be of highest importance to the data, therefore, it cleverly considers the best attributes each time it encounters differences in the data on the basis of the end goal given to it. Thus, attempts 2 and 3 are observed to be closer to the main goal of this assignment rather than attempts 4 and 5 that give better model performance but fail to assign students with a single type of job suitable for the student concerned.