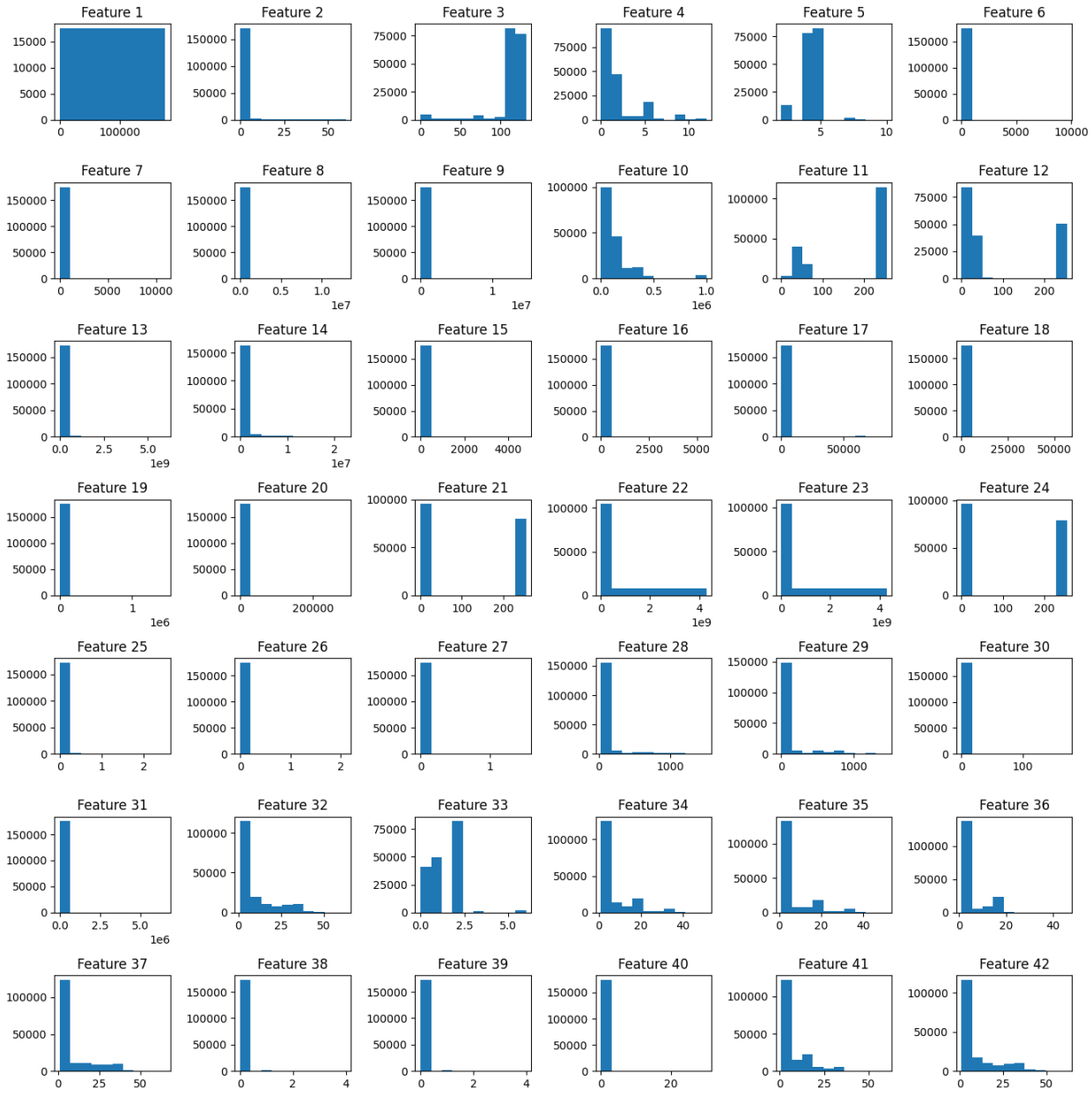
Final Project

610397193 Yazdan Zandiye Vakili

1. Openning & Visuallizing the Dataset

در این قسمت دیتاست را load کرده و برچسب ها را جدا کردیم و به چهار دسته X_train, y_train, X_test, y_test تقسیم کردیم.
سه عدد ویژگی Nominal در دیتاست موجود بود که مجبور بودیم به عدد تبدیلشان کنیم برای انجام مراحل که اسم آن ها به ترتیب Proto, Service, State بود که هر کدام را با توجه به تعداد انواع مقادیر دریافتی اش به اعداد صفر تا اندازه تعداد انواع آن ویژگی، اصطلاحا Map کردیم و می توانید در قسمت ابتدایی کد نحوه دریافت تعداد این سه ویژگی و Map کردن آن را ببینید. تعداد انواع مقادیر هر ویژگی به ترتیب برای Proto برابر 133 ، برای Service برابر 13 و برای State برابر 11 عدد بود.
همچنین برای کار کرددن با برچسب ها هم نیاز به Map کردن آن ها به اعداد 0 تا 9 (به دلیل اینکه 10 عدد برچسب متفاوت داشتیم) بود که در این قسمت انجام دادیم و قابل مشاهده است.
بعد از انجام این کار ها با استفاده از کتاب خانه matplotlib برای هر کدام از ویژگی های موجود یک نمودار پراکندگی رسم کردیم که در زیر می توانید آن را مشاهده کنید.



Pre-processing .2

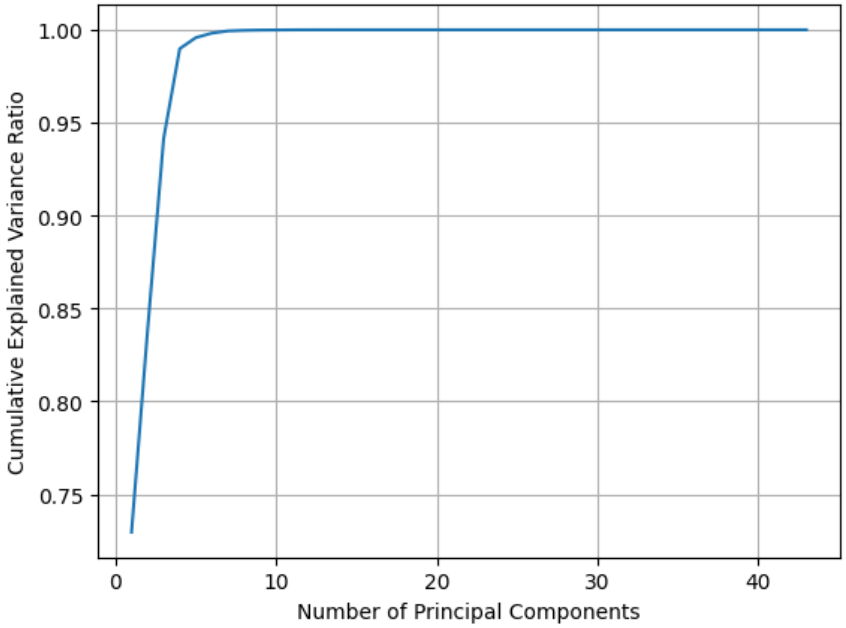
در این قسمت با توجه به مباحثی که در طول این ترم یادگرفتم با استفاده از سه روش عمل کاهش ابعاد را بر روی دیتا ست انجام دادم که آن سه روش عبارت اند از :

PCA .1.2

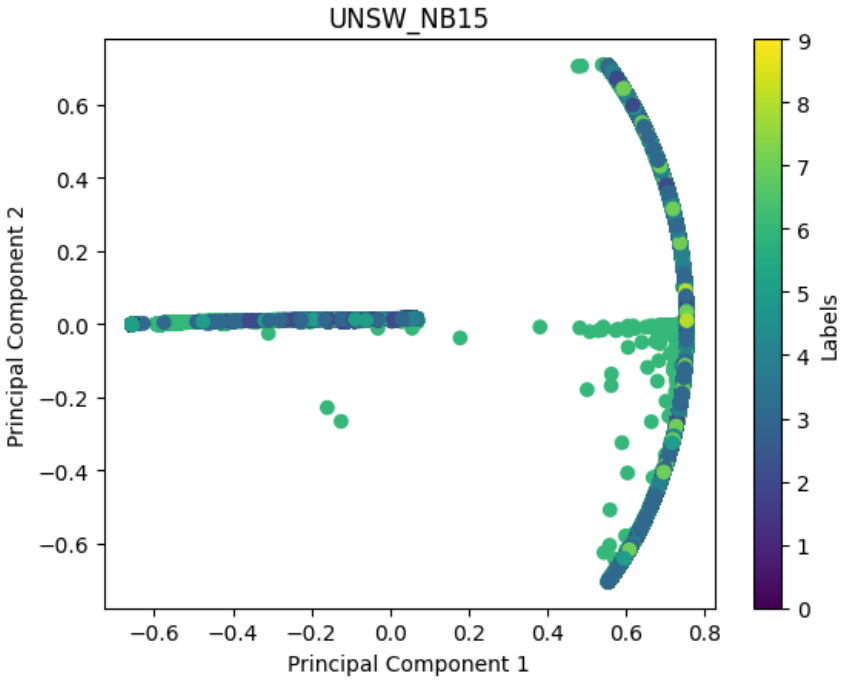
Recursive Feature Elimination .2.2

Univariate Selection .3.2

برای انجام عمل PCA ابتدا نمودار Cumulative Explained Variance Ratio را رسم کردم و با توجه به نمودار عدد 9 را برای تعداد ویژگی های نهایی انتخاب کردم که بعد از انجام این عمل به نمودار زیر برحسب دو ویژگی اول و دوم رسیدم.



نمودار Cumulative Explained Variance Ratio



نمودار رکورد ها بعد از تبدیل بر حسب دو ویژگی اول

برای دو روش دیگر با توجه به شـهودی که از نمودار ها در بخش انجام PCA گرفتم، عدد 15 را انتخاب کـردم و بـر روی هـر دو روش بـعـدی اعـمـال کـردم و در نـهـایـت بـرای روش Recursive Feature Selection اندیس های 3, 4, 15, 23, 24, 25, 26, 29, 31, 32, 35, 37, 38, 39, 42 را انتخاب کردم و فقط با این اندیس ها در دیتا های Train و Test کار کردم و هم چنین برای روش Univariate Selection به اندیس های 0, 2, 7, 8, 9, 10, 12, 13, 16, 17, 18, 19, 21, 22, 27 در نهایت رسیدم.

نکته : در ادامه برای هر روش ارائه شده از تمام داده های کاهش داده شده با هر سه روش استفاده کردم و نتایج را گزارش نمودم.

3. Training Section

تصمیم گرفتم که از 5 روش استفاده کنم که به ترتیب آن ها را گزارش می کنم.

1.3. KNN :

در این قسمت با استفاده از 5 folds cross-validation مقدار بهتر برای تعداد همسایگان را از بین انتخاب های موجود که به مدل داده بودم (7و9و11و13و15) بهترین را انتخاب کردم و تمام Metric های خواسته شده را گزارش کردم.

1.1.3. PCA :

```
Results with PCA :
Best number of K : 13
Accuracy : 0.4777850653451878
Precision : 0.5817609703892407
Recall : 0.4777850653451878
F1 Score : 0.5084717814720889
Confusion Matrix : [[    4    15   125   380    42     3   104     4     0     0]
 [    4     6   124   392    11     6    35     5     0     0]
 [   12   191   339  1437   522    47  1489    50     2     0]
 [   33   159   479  5355  1463    55  3465   122     1     0]
 [   26    34   295  2755  1148   169  1579    56     0     0]
 [    3   108    14   442  5454  9950  1951   949     0     0]
 [   57     8   226  9304  4571   873 21750   195    16     0]
 [    6    73   100  1192   388    18   937   782     0     0]
 [    0     7    22   133    79    10   105    19     3     0]
 [    0     0     0    19     7     0    15     3     0     0]]
```

: RFE .2.1.3

```
Results with RFE :
Best number of K : 11
Accuracy : 0.6553223533984356
Precision : 0.7124004540674701
Recall : 0.6553223533984356
F1 Score : 0.6750032129656204
Confusion Matrix : [[    0    65   187   165    91   132    37     0     0     0]
 [    1     2   178   161    66   132    35     8     0     0]
 [  264   754   554  1499   308   142   432   123    13     0]
 [  257   666   698  6148   947   289  1664   449    14     0]
 [   35    98   404   584  2795   280  1739   123     4     0]
 [    1     9   182   530   205 17711   193    34     6     0]
 [  204     9   198  3050  7605    67 24862   963    42     0]
 [   34    94    60   630   511     3   321  1841     2     0]
 [    0     0    14    36   122     2    39   124    41     0]
 [    0     0     0    31     4     0     4     4     1     0]]
```

: US .3.1.3

```
Results with US :
Best number of K : 15
Accuracy : 0.5957829276587475
Precision : 0.6770696963926571
Recall : 0.5957829276587475
F1 Score : 0.6199009660946486
Confusion Matrix : [[    3   113   127   393    18     1    22     0     0     0]
 [    4    15   116   414     8     4    22     0     0     0]
 [    8  1435   356  1495   243    32   499    18     3     0]
 [   28  1252   482  5430  1208    39  2631    57     5     0]
 [   23   192   268  2763  1263     5  1495    39    14     0]
 [    1     4    23   330   149 17975   382     6     1     0]
 [   42     3   162  9194  4478    18 22918   142    43     0]
 [    4   189    45  1080   280    11   813  1071     3     0]
 [    0     6    18   144    85     8    90     6    21     0]
 [    0     0     0    19     5     0    18     2     0     0]]
```

نکته : جالب توجه بود که الگوریتمی مانند KNN با RFE نتیجه بهتری نسبت به سایر روش ها می دهد و به حدود 66 درصد می رسد.

2.3. Random Forest :

در این قســـمت یک Random Forest با تعداد 100 درخت و Random State برابر با 42 را برای هر کدام از سه روش کاهش ابعاد در نظر گرفتم و نتایج را گزارش کردم.

2.3.1. PCA :

```
Results with PCA :
Accuracy : 0.4849390273526697
Precision : 0.5864070704448366
Recall : 0.4849390273526697
F1 Score : 0.5151134937089797
confusion Matrix : [[    4    11    79   390    63     0   122     8     0     0]
 [    6    11    71   389    44     2    52     8     0     0]
 [    9    94   163  1185   737    16  1817    57     9     2]
 [   32    92   335  4330  1399    10  4646   261    25     2]
 [   27    26   206  1935  1661    13  1910   245    37     2]
 [    1     3  1254   247  4003  9872  2057   177  1257     0]
 [   72   108   429  5815  6298    42 22786  1259   158    33]
 [   10    45    40   633   366     1  1307  1080    13     1]
 [    0     2     9    73    89     6   146    34    19     0]
 [    0     0     0    12     6     0    24     2     0     0]]
```

2.3.2. RFE :

```
Results with RFE :
Accuracy : 0.45119759024437645
Precision : 0.4983893488926834
Recall : 0.45119759024437645
F1 Score : 0.28449996089081786
confusion Matrix : [[    0     2     0     8     2     0   665     0     0     0]
 [    0     4     0    18     6     0   555     0     0     0]
 [    0     8     4    85    27     0  3965     0     0     0]
 [    0     6     0   102    47     0 10977     0     0     0]
 [    0     4     2    55    40     0  5961     0     0     0]
...
 [    0     1    22    22    11     1 36943     0     0     0]
 [    0     0     0    14     1     0  3481     0     0     0]
 [    0     0     0     0     0     0   378     0     0     0]
 [    0     0     0     0     0     0    44     0     0     0]]
```

: US .3.2.3

```
Results with US :
Accuracy : 0.4391609580721955
Precision : 0.5841249474058042
Recall : 0.4391609580721955
F1 Score : 0.4532524573086317
confusion Matrix : [[    1     1     8   383    12     0   165     0   107     0]
 [    0     4     8   376    20     0    66     0   109     0]
 [    4     2    36   848   115     0  2950    21   113     0]
 [    2     2    24  4240   258     0  6377     7   222     0]
 [    4     2    13  1755  1442     0  2589     6   251     0]
 [    0     0  1994   365    97  5261  2643     6  8505     0]
 [    3     1    10  6561  5940     0 24380    82    23     0]
 [    0     0     2   414   192     0  2122   764     2     0]
 [    0     0     0    55    68     1   213    12    29     0]
 [    0     0     0    12     3     0    29     0     0     0]]
```

همانطور که می بینید این روش بر روی PCA جواب بهتری داد هر چند خیلی محســـوس نبود و در کل بر خلاف اینکه این مدل از مدل های Ensemle محسـوب می شـود و انتظار نتیجه خوبی را داریم ولی در کل بر روی هیچ کدام از روش های کاهش ابعاد نتیجه قابل قبولی ارائه نداد.

Extreme Learning Model .3.3

این مدل در بین دیگر مدل های استفاده شده یک مدل جدید می باشد به همین دلیل در کتاب خانه های موجود این Classifier موجود نبود و در کتاب خانه های Open Source ای که توســـط افراد دیگر گذاشته شده بود قابل دسترس بود ولی به علت راحتی اجرا توسط مصحح پروژه از آن ها استفاده نکردم و از یک ELM Classifier پیاده سازی شده توسط یک کاربر در وبسایت Kaggle به این لینک استفاده نمودم. به نظر این کد با توجه به کامنت هایی که در وبسـایت گرفته اسـت سـالم اسـت و کار می کند، هر چند برای اســتفاده خودم مجبور به تغییر روش ضرب داخلی موجود در آن شــده که به خطا Singular Matrix برخورد نکنم، ولی نتایج خیلی عجیبی به ما می دهد و برای هر سـه روش کاهش ابعاد نتیجه 100 درصـد را داد!!! من با توجه به مریض احوال بودنم و وقت کم نتوانسـتم بیشـتر از این پیگیری کنم تا ببینم دقیقا موضـوع این مدل از چه قرار اسـت ولی برای هر سـه مدل این نتیجه یکسـان بود و کد را در پروژه می توانید مشاهده کنید.

: PCA .1.3.3

```
PCA Results :
Accuracy : 1.000000
```

: RFE .2.3.3

```
RFE Results :
Accuracy: 1.000000
```

: US .3.3.3

```
US Results :
Accuracy: 1.000000
```

Stacking Model .4.3

با وجه به اینکه این روش یک روش Ensemble محسوب می شود می خواستم که حتما از آن در این پروژه استفاده کنم و در ابتدا با توجه به مطالب گفته شده در کلاس در طول ترم یک مدل با KNN, Random Forest, SVM ساختم و تمام مدل ها را از ابتدا ساختم تا Train شود ولی بعد از گذشت حدود 100 دقیقه جوابی نگرفتم و به دلیل محدودیت وقت نتوانستم آن را ادامه دهم.

4. Stacking Model

```python
from sklearn.ensemble import StackingClassifier, RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

# With PCA
base_classifiers_pca = [
    ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
    ('knn', KNeighborsClassifier(n_neighbors=13)),
    ('SVC', SVC()),
    ]

meta_estimator = RandomForestClassifier(n_estimators=100, random_state=42)
stacking_classifier_pca = StackingClassifier(estimators=base_classifiers_pca, final_estimator=meta_estimator)
stacking_classifier_pca.fit(X_train_pca, y_train)
y_pred_st_pca = stacking_classifier_pca.predict(X_test_pca)

print("Results with PCA : ")
print("Accuracy : " + str(accuracy_score(y_test, y_predict_st_pca)))
print("Precision : " + str(precision_score(y_test, y_predict_st_pca, average='weighted')))
print("Recall : " + str(recall_score(y_test, y_predict_st_pca, average='weighted')))
print("F1 Score : " + str(f1_score(y_test, y_predict_st_pca, average='weighted')))
print("confusion Matrix : " + str(confusion_matrix(y_test, y_predict_st_pca)))
```

97m 45.0s

بعد از آن تصـمیم گرفتم که از مدل های Train شـده مراحل قبلی اسـتفاده کنم و فقط SVM را جدید بگذارم تا Train بشـود و همچنین Meta Estimator هم با توجه به اینکه مدل KNN یک مدل Lazy Learner یا Instance-based است شاید سریع تر انجام گیرد ولی باز بعد از گذشت حدود 90 دقیقه به جوابی نرسـید و باز هم محدودیت وقت باعث شـد که سرانجام آن را متوقف کنم.

4. Stacking Model

```
from sklearn.ensemble import StackingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

# With PCA
base_classifiers_pca = [
    ('rf', random_forest_pca),
    ('knn', KNeighborsClassifier(n_neighbors=13)),
    ('SVC', SVC()),
    ]

meta_estimator = KNeighborsClassifier(n_neighbors=13)
stacking_classifier_pca = StackingClassifier(estimators=base_classifiers_pca, final_estimator=meta_estimator)
stacking_classifier_pca.fit(X_train_pca, y_train)
y_pred_st_pca = stacking_classifier_pca.predict(X_test_pca)

print("Results with PCA : ")
print("Accuracy : " + str(accuracy_score(y_test, y_predict_st_pca)))
print("Precision : " + str(precision_score(y_test, y_predict_st_pca, average='weighted')))
print("Recall : " + str(recall_score(y_test, y_predict_st_pca, average='weighted')))
print("F1 Score : " + str(f1_score(y_test, y_predict_st_pca, average='weighted')))
print("confusion Matrix : " + str(confusion_matrix(y_test, y_predict_st_pca)))
```

87m 50.0s

MLP .3.5

در انتها با توجه با روش خود مقاله تصـمیم گرفتم که MLP را هم تسـت کنم و برای هر روش دو بار شـبکه را سـاخته و Train کردم، یکبار با 3 لایه مخفی 128 و 64و 128 و یک بار هم با الهام از ELM Classifier با یک لایه مخفی 1000 تایی و نتایج بدین صورت بود.

.1.3.5 PCA with 3 hidden layers :

```
Results of PCA :
Accuracy : 0.446594276830394
Precision : 0.5029473978711182
Recall : 0.446594276830394
F1 Score : 0.4383095092042069
Confusion Matrix : [[    0     0     0   485     0   157    35     0     0     0]
 [    0     0     1   382     0   158    42     0     0     0]
 [    0     0    48  3095    31   204   711     0     0     0]
 [    0     0    95  6438    55   352  4192     0     0     0]
 [    0     0     1  3440     4   549  2068     0     0     0]
 [    0     0     4 13758     0  4822   287     0     0     0]
 [    0     0     0  9415     1  2127 25457     0     0     0]
 [    0     0    15  2102     7   384   988     0     0     0]
 [    0     0     0   248     0    25   105     0     0     0]
 [    0     0     0    21     0     0    23     0     0     0]]
```

: PCA with 1 hidden layer .2.3.5

```
Results of PCA :
Accuracy : 0.5940339114803479
Precision : 0.4982248275983675
Recall : 0.5940339114803479
F1 Score : 0.524817878299853
Confusion Matrix : [[    0     0     0    27     0   603    47     0     0     0]
 [    0     0     1    49     0   491    42     0     0     0]
 [    0     0    51   451    24  2703   860     0     0     0]
 [    0     0   102  2459    30  2977  5564     0     0     0]
 [    0     0     0  1114     1  2205  2742     0     0     0]
 [    0     0     4   184     1 18289   393     0     0     0]
 [    0     0     2  4247     0  4643 28108     0     0     0]
 [    0     0    15   559     5  1580  1337     0     0     0]
 [    0     0     0    61     0   185   132     0     0     0]
 [    0     0     0     8     0     6    30     0     0     0]]
```

: RFE with 3 hidden layers .3.3.5

```
Results of RFE :
Accuracy : 0.5965238303454307
Precision : 0.6922214937179877
Recall : 0.5965238303454307
F1 Score : 0.6057752250220491
Confusion Matrix : [[    0     0     0   621     2     0    16    38     0     0]
 [    0     0     0   513     8     0    22    40     0     0]
 [    0     0    35  3222   213    42   433   144     0     0]
 [    0     0    70  8507   266    36  2089   164     0     0]
 [    0     0     0  4078   913     4   994    73     0     0]
 [    0     0     3   422   117 18146   175     8     0     0]
 [    0     0     0 12963  3710    47 20223    57     0     0]
 [    0     0    10  1751    33    27   386  1289     0     0]
 [    0     0     0   203    84    28    43    20     0     0]
 [    0     0     0    25     6     0    13     0     0     0]]
```

: RFE with 1 hidden layer .4.3.5

```
Results of RFE :
Accuracy : 0.44939999028324346
Precision : 0.2019603512665793
Recall : 0.44939999028324346
F1 Score : 0.2786813200227937
Confusion Matrix : [[    0     0     0     0     0     0   677     0     0     0]
 [    0     0     0     0     0     0   583     0     0     0]
 [    0     0     0     0     0     0  4089     0     0     0]
 [    0     0     0     0     0     0 11132     0     0     0]
 [    0     0     0     0     0     0  6062     0     0     0]
 [    0     0     0     0     0     0 18871     0     0     0]
 [    0     0     0     0     0     0 37000     0     0     0]
 [    0     0     0     0     0     0  3496     0     0     0]
 [    0     0     0     0     0     0   378     0     0     0]
 [    0     0     0     0     0     0    44     0     0     0]]
```

: US with 3 hidden layers .5.3.5

```
Results of US :
Accuracy : 0.6424597969197882
Precision : 0.6845595973886909
Recall : 0.6424597969197882
F1 Score : 0.6340308591992078
Confusion Matrix : [[    0     0     0   605     1     0    37    34     0     0]
 [    0     0     0   495     5     0    47    36     0     0]
 [    0     0     8  2874   181    30   844   152     0     0]
 [    0     0     1  6002   225    30  4712   162     0     0]
 [    0     0     0  2803   920     2  2271    66     0     0]
 [    0     0     0   266   114 18146   337     8     0     0]
 [    0     0     0  6600  3792    12 26530    66     0     0]
 [    0     0     0  1103    21    27  1056  1289     0     0]
 [    0     0     0   137    87    22   105    27     0     0]
 [    0     0     0    13     6     0    25     0     0     0]]
```

: US with 1 hidden layer .6.3.5

```
Results of RFE :
Accuracy : 0.5446484963319244
Precision : 0.6615332616359333
Recall : 0.5446484963319244
F1 Score : 0.5264092565263913
Confusion Matrix : [[    0     0     0   664     0     2    11     0     0     0]
 [    0     0     0   561     0     4    18     0     0     0]
 [    0     0     7  3639    27   138   278     0     0     0]
 [    0     0     2  9603    33   138  1356     0     0     0]
 [    0     0     0  5139    21   386   516     0     0     0]
 [    0     0     0   571     0 18162   138     0     0     0]
 [    0     0     0 17061    85  2805 17049     0     0     0]
 [    0     0     0  3202     4    48   242     0     0     0]
 [    0     0     0   338     0    21    19     0     0     0]
 [    0     0     0    34     0     2     8     0     0     0]]
```

همانطور که مشـــاهده می کنید PCA با ســـه لایه جواب خوبی نداد و با یک لایه 1000 تایی به جواب بهتری رسید، RFE برعکس PCA با یک لایه 1000 تایی جواب خوبی نداشت و با سه لایه مخفی به جواب قابل قبول تری رســـید و در آخر US با یک لایه جوابش تقریبا به اندازه جواب های خوب دو روش قبلی بود ولی با ســـه لایه مخفی به جواب بهتری با حدود 10 درصـــد بهبود رسید.

نکته : همچنین یکبار با Train دوباره MLP موجود 3 لایه با RFE بر روی داده های بدســـت آمده از US به بهبود محسوس ای (حدود 8 درصد) رسیدم که در کد وجود ندارد ولی به نظرم این ایده هم برای بهبود نتیجه می تواند خوب و امیدوار کننده باشد.

نکته : برای اجرا پروژه به آدرس دهی دیتاست ها دقت شود.

صـحبتی با اسـتاد : با توجه به نتایجی که در مدت زمان فقط یک روز گرفتم به نظرم می توانم برای ادامه این پروژه مفید باشـم و به سرانجام برسـانم آن را، اگر امکانش فراهم باشـد خیلی ممنون میشم.