Svetlana Koroteeva ID: 300432399

Assignment 3 – Individual part

AIML 427 - Big Data

2023 T1

(a) Describe the task including the details of the input data (data source, data size, the original number of features and instances, feature types, whether missing values exist, etc.) and the expected output of the system;

The main goal of the project to make classifier to diagnose 14 personal disorders based on existing NESARC dataset. As there are several label columns the final pipeline is going to be elaborated during the process.

NESARC was a survey of 43,093 participants that covered alcohol, drug and psychiatric disorders, risk factors, and consequences. Wave 1 of the NESARC was conducted in 2001-2002 in a frame of the National Epidemiologic Survey on Alcohol and Related Conditions (NESARC) was conducted to provide information from 43,093 American adults on the common mental substance and psychiatric disorders as defined in DSM-IV [1]. It was used since in multiple projects. The row dataset was loaded from a github project[2].

"nesarc_pds.csv" is a raw dataset and does not specifically have class labels. For the project columns of interest were selected as class labels. Original data has 3008 features and 43094 instances and should be reduced for the assignment purpose.

In the assignment we will use this data to predict DSM-IV diagnosis [3] indicated in the last part of the data description as different types of personality disorders.
Some of the diagnoses are represented by several columns. Diagnoses part of the data is fully populated. However, the set contains poorly populated columns. It was checked on the initial stage of the project if some columns and rows have high percent of missing data. Simple analysis with pandas methods

```
threshold=0.5
missing_columns,lst = count_missing_data(df, axis='columns', threshold=threshold)
print(f"Number of columns with >= {threshold*100}% missing data: {missing_columns}")
```

showed that.
Number of rows with >= 50.0% missing data: 41933
Number of columns with >= 50.0% missing data: 2183
All the features have categorical values.

(b) Describe all the pre-processing steps applied to the download data file(s) to obtain the dataset that is used as input of your program,

First of all, we will assign the features of interest as potential labels. Some of the columns should be joined later.

As our goal is to use massive data and not required to research specifical feature we can remove all the columns with missing data more than 30%.(Recommended maximum percent of missing data is 25-30%.). This reduced the dataset in 5 times. After deletion the set size was reduced to 65 MB.

In the dataset unknown values set as 9 ,99,999,9999. We replace them to Nan as well as the "white spaces".

```
df = pd.read_csv(r"data\nesarc_cleaned.csv")
print(df.shape)
df = df.replace(r'^\s*$', np.nan, regex=True)
df = df.replace({'99': pd.np.nan})
df = df.replace({'9': pd.np.nan})
```

As the next step the most frequent value was imputed to all Nan cells with SimpleImputer method of sklearn library. (Other option "mean" is only suitable for continuous data).

```
imputer = SimpleImputer(strategy='most_frequent')
df_imputed = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)
df_imputed = df_imputed.reindex(columns=df.columns)
df_imputed.to_csv("imputed_nesarc.csv")
```

Next, we prepare diagnoses class labels, joining some of them by most often value. After a simple modification we will have 652 columns which include 14 label columns. This dataset was saved as "nesarc_final.csv" dataset. The first fit to a basic Logistic Regression model showed 100% of accuracy. It might happen that some of the features are fully correlated with the corresponding diagnose labels. However , we are more interested to discover less obvious patterns in the data via this research, so removal was required. To find them feature selection technique based on Pearson's correlation was conducted.

```
    elif method == 'pearson':
        for feature in feature_columns:
            score = corr_matrix.loc[feature, 'indexed_lab']
            feature_scores[score]=feature

        sorted_d = dict(sorted(feature_scores.items(),reverse=True))
        feature_dict = get_first_n_elements(sorted_d, num_features)

        selected_feature_names = [feature_dict[el] for el in feature_dict.keys() ]

        return selected_feature_names
```

Class	Features
AVOIDPDX2	DEPPDDX2, S1Q211, S1Q212, S1Q16, DOBY, S1Q20, MARITAL, S1Q14C4, S1Q7A9, SPOUSE
DEPPDDX2	AVOIDPDX2, S1Q16, S1Q212, S1Q211, S1Q14C4, S1Q20, S1Q7A1, S1Q8A, S3EQ8B, S3EQ8C
OBCOMDX2	S1Q211, S1Q212, S1Q6A, S1Q7A9, DOBY, S4BQ11, S1Q20, S1Q14C1, S1Q131, S4BQ7C
DEP	S1Q211, S1Q212, S1Q16, S1Q20, SEX, S4BQ11, S4BQ7C, S1Q7A1, S4BQ12, S4BQ7B

PAN	S1Q211, S1Q212, S1Q16, S1Q20, SEX, S1Q7A9, S1Q1D3, S1Q7A1, S4BQ11, S11BQ11
AGORA	S1Q211, S1Q212, S1Q16, S1Q20, SEX, S1Q7A9, S1Q7A1, S2AQ16B, S4BQ11, S4BQ7C
SPECPHOB	SEX, S1Q211, S1Q212, S1Q16, S1Q20, S4BQ11, S1Q7A9, DOBY, S4BQ7C, S1Q14C1
ANX	DEP, PARADX2, S1Q211, S1Q212, S1Q20, S1Q16, SEX, S1Q1C, S1Q7A1, S1Q7A9

The analysis based of ranking confirmed that several groups of features are highly correlated to labels .They were discovered and for the purpose of investigation removed.
The a3_correlation.py script was run on the dataset to identify covariant features. This dataset was saved as “nesarc_final_lack.csv” which contains 348 features and 14 label columns.
The names of label columns refer to the following diagnoses:

Depression - DEP,
Panic disorder - PAN,
Anxiety -ANX,
Dependent personality - DEPPDDX2 ,
Obsessive-Compulsive - OBCOMDX2,
Paranoid personality - PARADX2 ,
Histrionic personality disorder - HISTDX2,
Agoraphobia - AGORA,
Social phobia- SOCPHOB,
Conduct disorder - CONDUCTONLY ,
Antisocial personality disorder - ANTISOCDX2,
Schizoid personality disorder - SCHIZDX2
Avoidant personality disorder - AVOIDPDX2

(c) (15 marks) Describe the program using UML class diagrams and/or pseudo-code;

Base model pseudo-code:

The pseudocode is suitable for each classifier model and does not include pre-processing steps. However, ideally preprocessing script should be written in a single function called “clean_and_impute” and applied to train and test data inside of model function.

Import necessary libraries:

pyspark.ml.feature.VectorAssembler/StandardScaler,
pyspark.ml.classification.LogisticRegression/RandomForestClassifier/NaiveBayes
pyspark.ml import Pipeline
pyspark.sql import SparkSession
and time

Start SparkSession
Read CSV file to RDD

Define functions

  Average calculator
  Train and evaluate the model.

Set label list:

list_of_labels = [
  "CONDUCTONLY",
  "ANTISOCDX2",
  "AVOIDPDX2",
  "DEPPDDX2",
  "OBCOMDX2",
  "PARADX2",
  "SCHIZDX2",

"HISTDX2",
"DEP",
"PAN",
"AGORA",
"SOCPHOB",
"SPECPHOB",
"ANX"
]

Set list of classifiers
classifiers = [
"LogisticRegression",
"RandomForestClassifier",
"NaiveBayes"
]

Set seeds
seeds = [987, 1052, 777]

Initialize empty results list
For each label in list_of_labels
    For each classifier in classifiers
        Initialize lists for storing results
        For each seed in seeds
            Read the data from the CSV file
            Prepare the feature columns
            Train and evaluate the model
                Start measuring the time.
                Split the dataset into training and test sets.
                (Clean and impute test and train data) #not implemented yet
                Create an instance of classifier
                Create a pipeline and add the classifier.
                Create Vector assembler object
                Transform training and test data
                Create a Scaler object #this part is for normalizing script only
                Fit a scaler on the training data
                Transform training and test data with scaler
                Fit the pipeline on the training data
                Use the trained model to make predictions.
                Calculate the accuracy of the training predictions
                Use the trained model to make predictions on test data.
                Calculate the accuracy of the test predictions
                Stop measuring time and calculate the running time.
                Return the training accuracy, test accuracy, and running time.
            Append the results to corresponded lists

Calculate average results
Add the average results to the results list
Create the Spark DataFrame from the results list
Save the DataFrame as a CSV file

Base model UML class diagram:

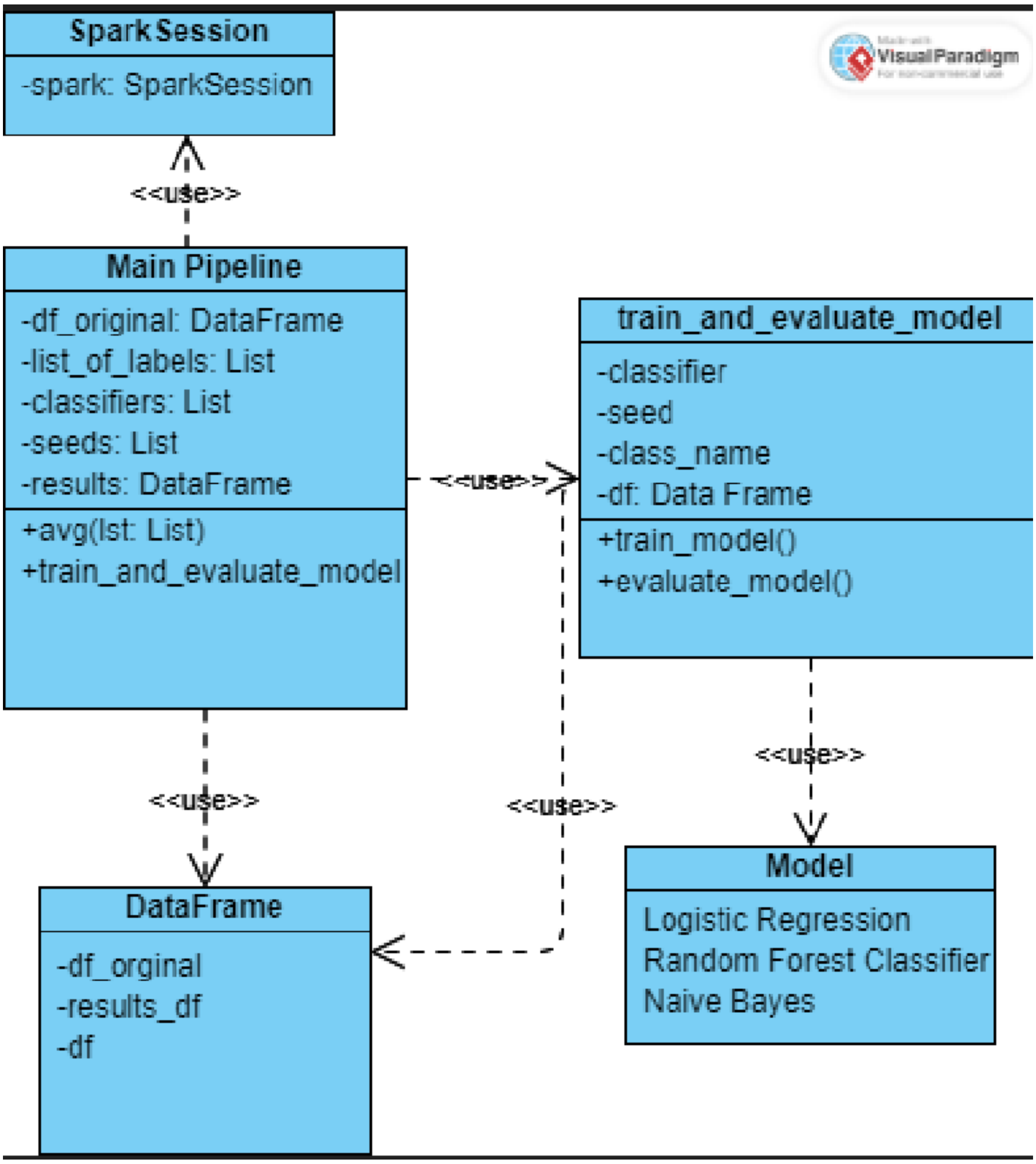

Fig 1. Baseline Model UML class diagram

(d) Describe how to install and run the program step by step in a Readme.txt file.

Please look at **README.txt** file.

(***e) Compare and discuss the results (including the training and***

test accuracy, the running time, the model, etc. depending on the program) of the program with and without normalising/scaling data.

The basic classifier was run on cluster and results were written as csv file.
As it was indicated in pseudocode and UML diagram 3 Classifiers: LogisticRegression, RandomForestClassifier and NaiveBayes are used inside the model.
In order to use other type, you can just add other model name as it is imported to the "classifiers" list.
Besides, the data for the model was split with different seeds and average values from each run populated the table. Diagnose codes are listed in the column called "class". The model was run 9 times for each class with 3 different seeds and 3 different classifier.

Table1. Baseline model results

Classifier	Class	Training Accuracy	Test Accuracy	Running Time
LogisticRegression	CONDUCTONLY	0.990667586	0.989592767	112.1902
RandomForestClassifier	CONDUCTONLY	0.99051317	0.990032873	5.487722
NaiveBayes	CONDUCTONLY	0.656224096	0.655213452	1.641517
LogisticRegression	ANTISOCDX2	0.969575959	0.967224738	6.093356
RandomForestClassifier	ANTISOCDX2	0.967115992	0.967691178	5.507843
NaiveBayes	ANTISOCDX2	0.377409928	0.377975363	1.567013
LogisticRegression	AVOIDPDX2	0.981357332	0.978460826	5.791655
RandomForestClassifier	AVOIDPDX2	0.976702169	0.978020777	5.373419
NaiveBayes	AVOIDPDX2	0.48252516	0.478837491	1.602829
LogisticRegression	DEPPDDX2	0.997473886	0.993372754	5.482821
RandomForestClassifier	DEPPDDX2	0.995102155	0.995391874	5.322338
NaiveBayes	DEPPDDX2	0.579333386	0.573646909	1.564391
LogisticRegression	OBCOMDX2	0.929256918	0.928884067	63.72542
RandomForestClassifier	OBCOMDX2	0.924017117	0.925880876	5.896053
NaiveBayes	OBCOMDX2	0.566107277	0.568573574	1.557272
LogisticRegression	PARADX2	0.960453252	0.956274904	31.97484
RandomForestClassifier	PARADX2	0.952003334	0.95205446	5.555039
NaiveBayes	PARADX2	0.468847952	0.465787744	1.55604
LogisticRegression	SCHIZDX2	0.969013407	0.96574947	11.88612
RandomForestClassifier	SCHIZDX2	0.966818199	0.967354737	5.785848
NaiveBayes	SCHIZDX2	0.443474276	0.442387207	1.530819
LogisticRegression	HISTDX2	0.98184267	0.979470615	8.140685
RandomForestClassifier	HISTDX2	0.981114557	0.981567106	5.388974
NaiveBayes	HISTDX2	0.440072916	0.438513734	1.679145
LogisticRegression	DEP	0.96430306	0.962487361	5.547238
RandomForestClassifier	DEP	0.960861411	0.961271192	5.809699
NaiveBayes	DEP	0.454409618	0.452656348	1.606098
LogisticRegression	PAN	0.961103948	0.961373783	91.32235
RandomForestClassifier	PAN	0.960398003	0.962798029	5.433159
NaiveBayes	PAN	0.5795131	0.574596789	1.53932
LogisticRegression	AGORA	0.991483888	0.988350137	5.757849

RandomForestClassifier	AGORA	0.989046047	0.989126913	5.768109
NaiveBayes	AGORA	0.497957224	0.495926297	1.556853
LogisticRegression	SOCPHOB	0.958489535	0.956687832	5.668611
RandomForestClassifier	SOCPHOB	0.9539558	0.954202907	5.426215
NaiveBayes	SOCPHOB	0.57103262	0.565116502	1.57947
LogisticRegression	SPECPHOB	0.91044881	0.911124845	6.091499
RandomForestClassifier	SPECPHOB	0.906179735	0.907966538	5.711357
NaiveBayes	SPECPHOB	0.609364059	0.604160556	1.673929
LogisticRegression	ANX	0.959923758	0.960572018	100.5981
RandomForestClassifier	ANX	0.957529998	0.959536721	5.635833
NaiveBayes	ANX	0.40817785	0.407871796	1.699034

The basic model shows good performance for most classes. Logistic Regression and RandomForest perform without big difference. From the other hand, Naive Bayes shows lower accuracies compared to the other classifiers. It might happened that features are dependant from each other. Logistic Regression shows quite high running time for some of the features as ANX ,PAN, CONDUCTONLY. Some of the diagnoses were predicted with a higher accuracy for example depression (DEP), panic disorder (PAN), anxiety (ANX), and others.

For normaliziation the following code added after features to vectors transformation

```
scaler = StandardScaler(inputCol="features", outputCol="scaled_features")
scaler_model = scaler.fit(df)
df = scaler_model.transform(df)
```

The model was run on cluster the same way as previous. And showed absolutely the same accuracies. From the other hand running time increased. It might happen due to additional scaling step in the script. The reason why scaling did not impact on accuracy might be that all values in similar scale and as we know categorical. Normalization is normally applied to continuous numerical features to ensure that they are on a similar scale.

Table 2. Baseline model with scaling results

Classifier	Class	Training Accuracy	Test Accuracy	Running Time
LogisticRegression	CONDUCTONLY	0.990667586	0.989592767	218.5083
RandomForestClassifier	CONDUCTONLY	0.99051317	0.990032873	7.837038
NaiveBayes	CONDUCTONLY	0.656224096	0.655213452	1.776618
LogisticRegression	ANTISOCDX2	0.969575959	0.967224738	130.0013
RandomForestClassifier	ANTISOCDX2	0.967082938	0.967794684	6.647616
NaiveBayes	ANTISOCDX2	0.377409928	0.377975363	1.734752
LogisticRegression	AVOIDPDX2	0.981357332	0.978460826	195.757
RandomForestClassifier	AVOIDPDX2	0.976735232	0.977968997	5.920451
NaiveBayes	AVOIDPDX2	0.48252516	0.478837491	1.904354
LogisticRegression	DEPPDDX2	0.997473886	0.993372754	128.5787
RandomForestClassifier	DEPPDDX2	0.995080095	0.995391874	5.969145
NaiveBayes	DEPPDDX2	0.579333386	0.573646909	1.989378
LogisticRegression	OBCOMDX2	0.929256918	0.928884067	91.06342
RandomForestClassifier	OBCOMDX2	0.923939894	0.925803164	7.3722

NaiveBayes	OBCOMDX2	0.566107277	0.568573574	2.175725
LogisticRegression	PARADX2	0.960453252	0.956274904	62.034
RandomForestClassifier	PARADX2	0.951694414	0.952028674	6.980798
NaiveBayes	PARADX2	0.468847952	0.465787744	1.920981
LogisticRegression	SCHIZDX2	0.969013407	0.96574947	95.91298
RandomForestClassifier	SCHIZDX2	0.966785116	0.967354737	6.470842
NaiveBayes	SCHIZDX2	0.443474276	0.442387207	1.728125
LogisticRegression	HISTDX2	0.98184267	0.979470615	10.65026
RandomForestClassifier	HISTDX2	0.981114557	0.981567106	5.805281
NaiveBayes	HISTDX2	0.440072916	0.438513734	1.633691
LogisticRegression	DEP	0.96430306	0.962487361	5.789421
RandomForestClassifier	DEP	0.960971696	0.961530076	5.865344
NaiveBayes	DEP	0.454409618	0.452656348	1.630901
LogisticRegression	PAN	0.961103948	0.961373783	85.92358
RandomForestClassifier	PAN	0.960386972	0.962798029	5.9271
NaiveBayes	PAN	0.5795131	0.574596789	1.626236
LogisticRegression	AGORA	0.991483888	0.988350137	5.668263
RandomForestClassifier	AGORA	0.989046047	0.989126913	5.793369
NaiveBayes	AGORA	0.497957224	0.495926297	1.725305
LogisticRegression	SOCPHOB	0.958489535	0.956687832	6.325853
RandomForestClassifier	SOCPHOB	0.953691041	0.954202972	5.873572
NaiveBayes	SOCPHOB	0.57103262	0.565116502	1.682777
LogisticRegression	SPECPHOB	0.91044881	0.911124845	6.528693
RandomForestClassifier	SPECPHOB	0.905870878	0.907811096	5.763771
NaiveBayes	SPECPHOB	0.609364059	0.604160556	1.650015
LogisticRegression	ANX	0.959923758	0.960572018	86.57909
RandomForestClassifier	ANX	0.957375568	0.959510781	5.860768
NaiveBayes	ANX	0.40817785	0.407871796	1.632331

(f) Compare and discuss the results (including the training and test accuracy, the running time, the model, etc. depending on the program) of the program with and without transforming data using PCA

For this model "train and evaluate" class replaced by PCA model, which includes both PCA and one of the classifiers used for previous part.

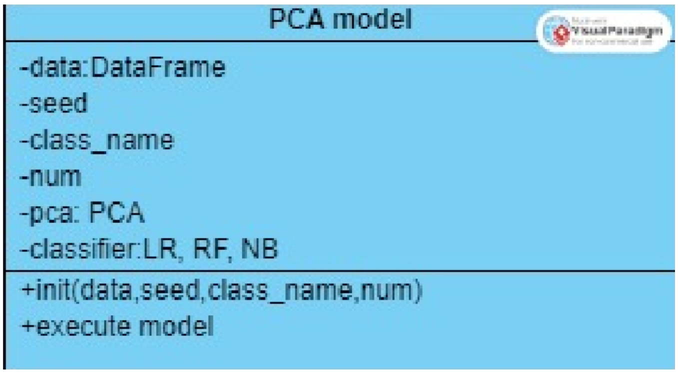

Fig. 2 PCA embedded model

PCA was used along with Logistic regression for the assignment as embedded method . The results are also extracted for different seeds and different numbers of classes. Besides, PCA is also taking the number of components as a parameter. For our model 3,10,20,30 components were used to obtain the results (see Table 3 in the Appendix)

We can observe that test accuracy is very low, and it is much lower then training accuracy overall, which can indicate significant overfitting. Testing accuracy increase with increasing of number of components from 3 to 20 significantly and to 30 slightly. We can also notice that this tendency is not always true. For some of the classes (e.g., "ANTISOCDX2"), increasing the number of PCA components does not significantly improve test accuracy. Optimal number of PCA components may vary depending on the class and the data distribution.

The runtime is increasing slightly with the number of PCA components increases. It is quite good in general however we cannot compare the runtime with other assignment models as this PCA-Logistic regression model was not run on cluster (due to cluster technical issues)

The poor results of the model can occur by the several issues. First is the fact that values are categorical as PCA is more suitable for numerical data. Second reason could be caused by non-linear relationship between features as PCA is linear method. And third are outliers: PCA is sensitive to outliers as it maximizes the variance in the data.

To summarize, PCA can help reduce the dimensionality of the data, but it might not capture all the relevant information for the classification task, other feature selection as lasso techniques could be considered. Besides it worth to check how non-linear dimensionality reduction techniques such as manifold learning algorithms (t-SNE, Isomap) are performing on this data, as they can capture non-linear relationships. To improve existing model the outliers could be investigated and cut as well.

Other trials: LASSO

To compare with baseline model and PCA with other model and following recommendations from PCA part, Lasso dimensionality reduction method was applied to the existing data. Lasso does not require other "co-model" as both reduce features and output the trained model in the same time.

From the lasso model results (Table 3) we can see that accuracy is good overall, however running time is quite high from 100 to 200 seconds per model run. Training accuracy is slightly higher than test which indicate slight overfitting. We can observe a negative correlation between accuracy and running time when run with different class labels: accuracy and running time both declined for some of the class labels. It might happen by a couple of reasons. First some classes, which show better results are more balanced. Second, feature distribution could vary across the label and some of them can have more complex data patterns which require longer time to converge. We can improve the results for the lasso model by preparing data for each class label and balance the number of positive and negative instances inside of each class. Approximately 250 features were selected by this model.

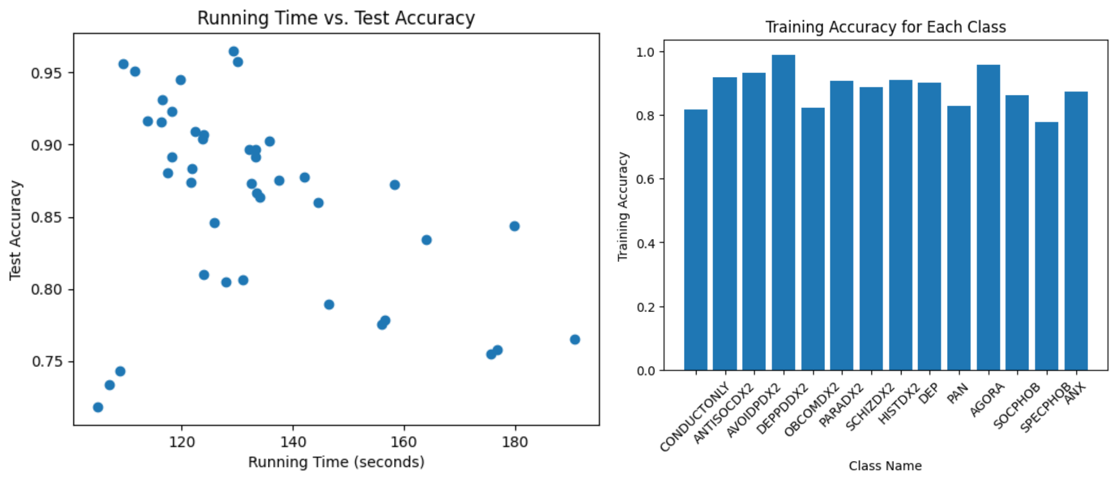

Fig.3 Lasso model viz

Table 3. Lasso model results

Classifier	Class	Training Accuracy	Test Accuracy	Running Time
LassoModel	CONDUCTONLY	0.822943	0.669563	2178.175
LassoModel	CONDUCTONLY	0.819701	0.702053	816.4868
LassoModel	CONDUCTONLY	0.8054	0.741755	67.62742
LassoModel	ANTISOCDX2	0.92088	0.89089	54.61191
LassoModel	ANTISOCDX2	0.921784	0.897888	96.69292
LassoModel	ANTISOCDX2	0.919783	0.894333	213.7744
LassoModel	AVOIDPDX2	0.937045	0.904231	2640.55
LassoModel	AVOIDPDX2	0.926154	0.93131	1902.463
LassoModel	AVOIDPDX2	0.926006	0.933031	2425.387
LassoModel	DEPPDDX2	0.973404	0.968524	611.2313
LassoModel	DEPPDDX2	0.97205	0.977754	126.0041
LassoModel	DEPPDDX2	0.982945	0.974614	52.99104
LassoModel	OBCOMDX2	0.8196	0.815061	60.30946
LassoModel	OBCOMDX2	0.820118	0.814153	585.0147
LassoModel	OBCOMDX2	0.822259	0.803384	53.29897
LassoModel	PARADX2	0.908832	0.888895	50.89624
LassoModel	PARADX2	0.900406	0.909279	55.01105
LassoModel	PARADX2	0.903902	0.900569	55.01244
LassoModel	SCHIZDX2	0.879759	0.875474	460.267
LassoModel	SCHIZDX2	0.884901	0.859646	51.04546
LassoModel	SCHIZDX2	0.881882	0.85929	49.91426
LassoModel	HISTDX2	0.906157	0.888968	47.93953
LassoModel	HISTDX2	0.903338	0.889588	47.15973
LassoModel	HISTDX2	0.905929	0.887631	455.1718
LassoModel	DEP	0.897137	0.885678	54.16138
LassoModel	DEP	0.89962	0.879657	50.81025
LassoModel	DEP	0.901487	0.874257	509.4852

LassoModel	PAN	0.822515	0.784747	50.23395
LassoModel	PAN	0.798135	0.790759	48.82116
LassoModel	PAN	0.82491	0.779445	50.2057
LassoModel	AGORA	0.955502	0.945943	46.36825
LassoModel	AGORA	0.953192	0.941195	482.6968
LassoModel	AGORA	0.9616	0.924161	45.74782
LassoModel	SOCPHOB	0.864406	0.830926	49.63039
LassoModel	SOCPHOB	0.858642	0.844606	49.82849
LassoModel	SOCPHOB	0.858638	0.842761	181.8951
LassoModel	SPECPHOB	0.774298	0.759591	991.9275
LassoModel	SPECPHOB	0.77233	0.763994	113.9712
LassoModel	SPECPHOB	0.771198	0.768571	554.5387
LassoModel	ANX	0.874162	0.863877	827.2188
LassoModel	ANX	0.872603	0.872251	746.81
LassoModel	ANX	0.873543	0.87066	620.4173

Conclusion and future work.

In the frame of the assignment several binary classifiers were built – Basic classifier, Classifier with normalized data, PCA+Logistic Regression Classifier and LASSO regression classifier The best performance was shown by baseline model with Logistic Regression and Random Forest. Feature selection with Pearson correlation was applied on data cleaning stage to investigate highly correlated features issue.

The possible directions for future work:

1) Implement “clean_and_impute function” (instead of several scripts);
2) Ballance data;
3) Investigate and remove outliers;
4) Parameters tuning;
5) Investigate non-linearity and apply non-linear manifold learning algorithms;
6) Build multiclass classifier instead of 14 binary classifiers.

References

1. National Epidemiologic Survey on Alcohol and Related Conditions-III (NESARC-III) | National Institute on Alcohol Abuse and Alcoholism (NIAAA) (nih.gov) - about NESARC dataset for the Q1
2. wesleyan-machine-learning/data at master · radumas/wesleyan-machine-learning (github.com) – source for download nesarc_pds.csv dataset for Q1
3. DSM-IV codes - Wikipedia - Diagnostic and Statistical Manual of Mental Disorders, 4th Edition
4. PySpark Lasso Regression – Building, Tuning, and Evaluating Lasso Regression with PySpark MLlib - Machine Learning Plus – Lasso regression with pyspark, making vector from features
5. Correlation — PySpark 3.4.0 documentation (apache.org) – Ranking with correlation: chi-square
6. PCA — PySpark 3.4.0 documentation (apache.org) -PCA
7. Visual Paradigm Online (visual-paradigm.com)

Appendix

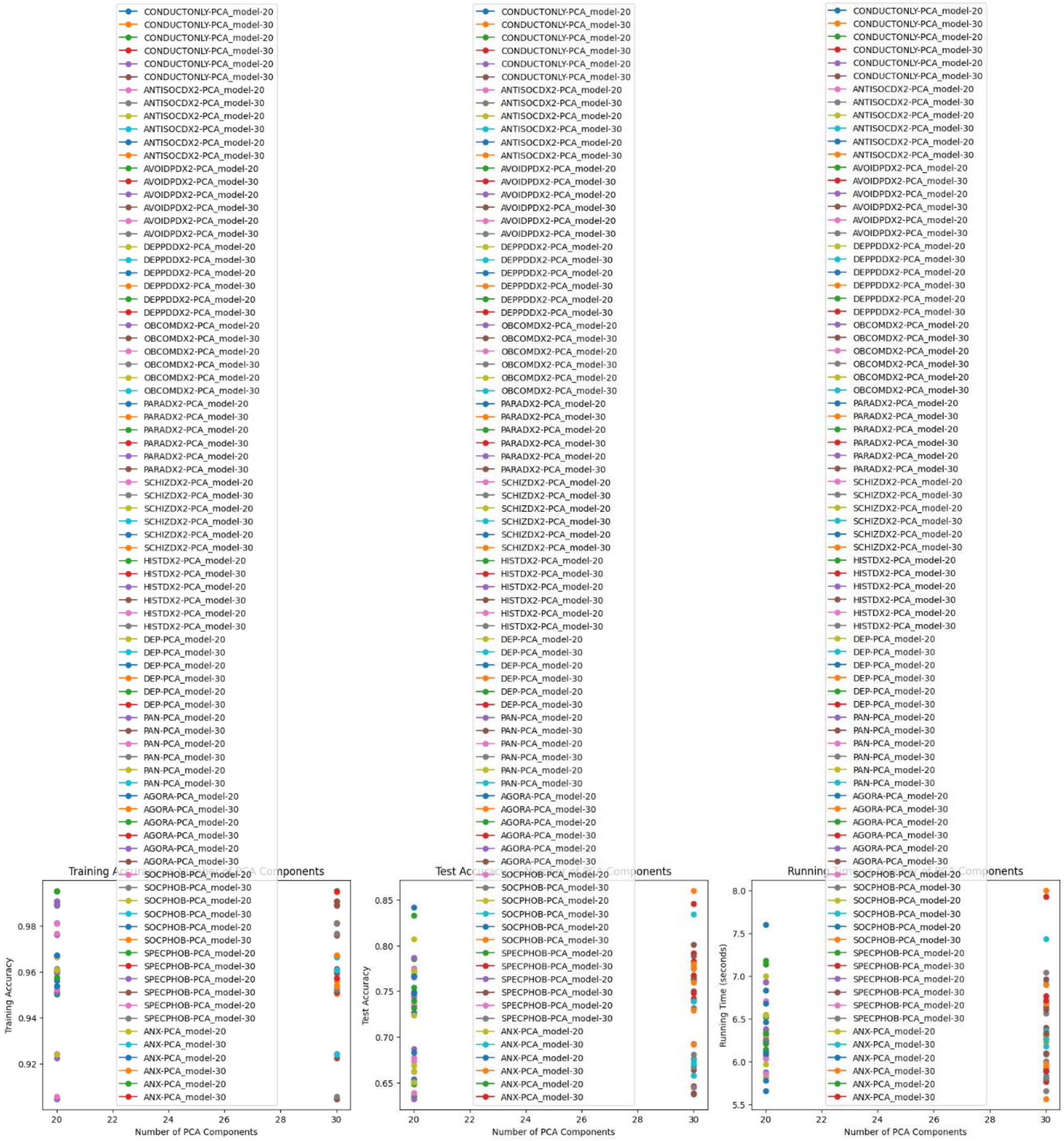

Fig 4. PCA Number of component vs Running time/Test and Train accuracy visualisation

Table 4. PCA results

Classifier	Class	Seed	Num PCA Comp	Train Accuracy	Test Accuracy	RunTime
PCA_model	CONDUCTONLY	987	3	0.990304	0.540806	15.59032
PCA_model	CONDUCTONLY	987	10	0.990304	0.57854	9.47362
PCA_model	CONDUCTONLY	1052	3	0.990637	0.536487	7.497615
PCA_model	CONDUCTONLY	1052	10	0.990637	0.53302	8.667549
PCA_model	CONDUCTONLY	777	3	0.990599	0.515133	6.93837
PCA_model	CONDUCTONLY	777	10	0.990599	0.535507	7.700324
PCA_model	ANTISOCDX2	987	3	0.966675	0.543612	8.508811
PCA_model	ANTISOCDX2	987	10	0.966675	0.59933	8.084162
PCA_model	ANTISOCDX2	1052	3	0.966584	0.549993	7.352689
PCA_model	ANTISOCDX2	1052	10	0.966584	0.607995	8.820627

PCA_model	ANTISOCDX2	777	3	0.966864	0.547999	8.090194
PCA_model	ANTISOCDX2	777	10	0.966864	0.589064	7.368196
PCA_model	AVOIDPDX2	987	3	0.976206	0.609357	7.426095
PCA_model	AVOIDPDX2	987	10	0.976206	0.663202	7.662394
PCA_model	AVOIDPDX2	1052	3	0.976377	0.600268	8.552086
PCA_model	AVOIDPDX2	1052	10	0.976377	0.650561	6.594305
PCA_model	AVOIDPDX2	777	3	0.976795	0.622738	5.66799
PCA_model	AVOIDPDX2	777	10	0.976795	0.66813	6.088605
PCA_model	DEPPDDX2	987	3	0.995301	0.702621	5.933207
PCA_model	DEPPDDX2	987	10	0.995301	0.741292	6.497824
PCA_model	DEPPDDX2	1052	3	0.994905	0.719222	5.998199
PCA_model	DEPPDDX2	1052	10	0.994905	0.74084	7.113749
PCA_model	DEPPDDX2	777	3	0.995035	0.752889	5.436584
PCA_model	DEPPDDX2	777	10	0.995035	0.767163	7.134051
PCA_model	OBCOMDX2	987	3	0.922629	0.536461	9.577986
PCA_model	OBCOMDX2	987	10	0.922629	0.589467	8.446723
PCA_model	OBCOMDX2	1052	3	0.924334	0.523848	7.110282
PCA_model	OBCOMDX2	1052	10	0.924334	0.600085	8.414352
PCA_model	OBCOMDX2	777	3	0.924195	0.497731	7.833553
PCA_model	OBCOMDX2	777	10	0.924195	0.595712	7.771842
PCA_model	PARADX2	987	3	0.950361	0.61737	9.430489
PCA_model	PARADX2	987	10	0.950361	0.643966	8.652737
PCA_model	PARADX2	1052	3	0.951034	0.605743	7.387407
PCA_model	PARADX2	1052	10	0.951034	0.639137	8.129591
PCA_model	PARADX2	777	3	0.951471	0.611276	8.566049
PCA_model	PARADX2	777	10	0.951471	0.651134	8.98711
PCA_model	SCHIZDX2	987	3	0.96651	0.573769	7.678497
PCA_model	SCHIZDX2	987	10	0.96651	0.615003	7.970901
PCA_model	SCHIZDX2	1052	3	0.966584	0.558939	9.436449
PCA_model	SCHIZDX2	1052	10	0.966584	0.615554	6.597362
PCA_model	SCHIZDX2	777	3	0.967162	0.567773	5.572774
PCA_model	SCHIZDX2	777	10	0.967162	0.600495	5.983799
PCA_model	HISTDX2	987	3	0.981005	0.543771	6.016602
PCA_model	HISTDX2	987	10	0.981005	0.609795	6.461125
PCA_model	HISTDX2	1052	3	0.981373	0.531759	5.378772
PCA_model	HISTDX2	1052	10	0.981373	0.604378	6.430977
PCA_model	HISTDX2	777	3	0.980966	0.540519	6.043427
PCA_model	HISTDX2	777	10	0.980966	0.592284	6.05591
PCA_model	DEP	987	3	0.959858	0.565726	5.81741
PCA_model	DEP	987	10	0.959858	0.680735	5.840774
PCA_model	DEP	1052	3	0.960265	0.555997	5.278391
PCA_model	DEP	1052	10	0.960265	0.687422	6.111811
PCA_model	DEP	777	3	0.961369	0.593235	5.584575
PCA_model	DEP	777	10	0.961369	0.689541	6.144098
PCA_model	PAN	987	3	0.959395	0.557673	6.05492
PCA_model	PAN	987	10	0.959395	0.636464	6.149977
PCA_model	PAN	1052	3	0.960959	0.551398	6.413141

PCA_model	PAN	1052	10	0.960959	0.611741	6.295065
PCA_model	PAN	777	3	0.960806	0.562289	5.757803
PCA_model	PAN	777	10	0.960806	0.602771	6.197093
PCA_model	AGORA	987	3	0.989146	0.556229	5.704956
PCA_model	AGORA	987	10	0.989146	0.676798	6.006685
PCA_model	AGORA	1052	3	0.98885	0.542542	5.506873
PCA_model	AGORA	1052	10	0.98885	0.64244	6.319165
PCA_model	AGORA	777	3	0.989142	0.604606	5.672971
PCA_model	AGORA	777	10	0.989142	0.69598	6.166184
PCA_model	SOCPHOB	987	3	0.952346	0.511911	5.772828
PCA_model	SOCPHOB	987	10	0.952346	0.619489	5.930435
PCA_model	SOCPHOB	1052	3	0.954012	0.533724	5.626266
PCA_model	SOCPHOB	1052	10	0.954012	0.613623	5.946704
PCA_model	SOCPHOB	777	3	0.953722	0.53394	5.762829
PCA_model	SOCPHOB	777	10	0.953722	0.620327	6.080694
PCA_model	SPECPHOB	987	3	0.904792	0.55639	5.874079
PCA_model	SPECPHOB	987	10	0.904759	0.604162	5.837915
PCA_model	SPECPHOB	1052	3	0.904715	0.54972	5.346164
PCA_model	SPECPHOB	1052	10	0.904715	0.60154	5.646935
PCA_model	SPECPHOB	777	3	0.905757	0.558024	5.546048
PCA_model	SPECPHOB	777	10	0.905757	0.594391	5.836307
PCA_model	ANX	987	3	0.957443	0.531793	5.931154
PCA_model	ANX	987	10	0.957443	0.667927	5.810253
PCA_model	ANX	1052	3	0.956228	0.535006	5.526748
PCA_model	ANX	1052	10	0.956228	0.678581	5.590827
PCA_model	ANX	777	3	0.957562	0.544375	5.557214
PCA_model	ANX	777	10	0.957562	0.666809	6.047026
PCA_model	CONDUCTONLY	987	20	0.990304	0.65387	6.462421
PCA_model	CONDUCTONLY	987	30	0.990304	0.645327	6.274596
PCA_model	CONDUCTONLY	1052	20	0.990637	0.648574	6.068475
PCA_model	CONDUCTONLY	1052	30	0.990637	0.646135	6.093417
PCA_model	CONDUCTONLY	777	20	0.990599	0.635532	6.241112
PCA_model	CONDUCTONLY	777	30	0.990599	0.638272	6.634392
PCA_model	ANTISOCDX2	987	20	0.966709	0.765204	6.934179
PCA_model	ANTISOCDX2	987	30	0.966675	0.764749	7.042283
PCA_model	ANTISOCDX2	1052	20	0.966584	0.77157	6.553179
PCA_model	ANTISOCDX2	1052	30	0.966617	0.77742	6.915148
PCA_model	ANTISOCDX2	777	20	0.966963	0.76702	6.830005
PCA_model	ANTISOCDX2	777	30	0.966963	0.76811	6.900295
PCA_model	AVOIDPDX2	987	20	0.97614	0.749729	7.181622
PCA_model	AVOIDPDX2	987	30	0.976008	0.783141	6.769742
PCA_model	AVOIDPDX2	1052	20	0.976278	0.773263	6.928752
PCA_model	AVOIDPDX2	1052	30	0.976212	0.801328	6.965388
PCA_model	AVOIDPDX2	777	20	0.976762	0.775243	6.706599
PCA_model	AVOIDPDX2	777	30	0.976596	0.791975	6.704781
PCA_model	DEPPDDX2	987	20	0.995301	0.807481	7.00138
PCA_model	DEPPDDX2	987	30	0.995268	0.834399	7.433594

PCA_model	DEPPDDX2	1052	20	0.994905	0.841856	7.602314
PCA_model	DEPPDDX2	1052	30	0.994872	0.860465	8.00275
PCA_model	DEPPDDX2	777	20	0.995035	0.833182	7.139795
PCA_model	DEPPDDX2	777	30	0.995068	0.845912	7.927685
PCA_model	OBCOMDX2	987	20	0.922629	0.651398	6.371972
PCA_model	OBCOMDX2	987	30	0.922629	0.663745	6.399841
PCA_model	OBCOMDX2	1052	20	0.924301	0.661869	6.032484
PCA_model	OBCOMDX2	1052	30	0.924334	0.667552	6.087562
PCA_model	OBCOMDX2	777	20	0.924228	0.650642	6.272153
PCA_model	OBCOMDX2	777	30	0.924228	0.658078	6.360448
PCA_model	PARADX2	987	20	0.950295	0.74737	6.68001
PCA_model	PARADX2	987	30	0.95046	0.761981	6.228797
PCA_model	PARADX2	1052	20	0.950935	0.754694	6.308718
PCA_model	PARADX2	1052	30	0.951067	0.767385	5.888793
PCA_model	PARADX2	777	20	0.951405	0.747723	6.229868
PCA_model	PARADX2	777	30	0.951471	0.76318	5.889138
PCA_model	SCHIZDX2	987	20	0.96651	0.7262	5.827717
PCA_model	SCHIZDX2	987	30	0.966444	0.731974	6.564595
PCA_model	SCHIZDX2	1052	20	0.966584	0.738313	5.822592
PCA_model	SCHIZDX2	1052	30	0.966551	0.739382	6.305201
PCA_model	SCHIZDX2	777	20	0.967162	0.726799	5.778846
PCA_model	SCHIZDX2	777	30	0.967129	0.72944	6.682925
PCA_model	HISTDX2	987	20	0.981005	0.734029	6.145646
PCA_model	HISTDX2	987	30	0.981005	0.739466	5.917073
PCA_model	HISTDX2	1052	20	0.981373	0.744146	6.20704
PCA_model	HISTDX2	1052	30	0.981373	0.750143	6.60971
PCA_model	HISTDX2	777	20	0.980966	0.734693	6.112098
PCA_model	HISTDX2	777	30	0.980966	0.739589	5.917947
PCA_model	DEP	987	20	0.96009	0.772218	6.317398
PCA_model	DEP	987	30	0.960322	0.77485	5.83541
PCA_model	DEP	1052	20	0.960265	0.766792	5.876925
PCA_model	DEP	1052	30	0.960132	0.780106	5.563653
PCA_model	DEP	777	20	0.961336	0.785314	6.116076
PCA_model	DEP	777	30	0.961336	0.791439	6.324153
PCA_model	PAN	987	20	0.959395	0.686906	6.256224
PCA_model	PAN	987	30	0.959395	0.692813	6.097251
PCA_model	PAN	1052	20	0.960959	0.673536	6.341356
PCA_model	PAN	1052	30	0.960959	0.676355	6.007098
PCA_model	PAN	777	20	0.960806	0.668704	6.36648
PCA_model	PAN	777	30	0.960806	0.67422	5.764261
PCA_model	AGORA	987	20	0.989146	0.766655	6.522008
PCA_model	AGORA	987	30	0.989146	0.77537	6.31117
PCA_model	AGORA	1052	20	0.988817	0.731643	6.531336
PCA_model	AGORA	1052	30	0.98885	0.74232	6.713827
PCA_model	AGORA	777	20	0.989142	0.786672	6.382995
PCA_model	AGORA	777	30	0.989175	0.788735	6.333509
PCA_model	SOCPHOB	987	20	0.952346	0.677264	6.048652

PCA_model	SOCPHOB	987	30	0.952346	0.681218	5.657836
PCA_model	SOCPHOB	1052	20	0.954012	0.662657	5.966885
PCA_model	SOCPHOB	1052	30	0.954012	0.670501	6.256663
PCA_model	SOCPHOB	777	20	0.953722	0.683228	5.656594
PCA_model	SOCPHOB	777	30	0.953756	0.691595	5.975084
PCA_model	SPECPHOB	987	20	0.904759	0.635032	6.333269
PCA_model	SPECPHOB	987	30	0.904825	0.63812	5.765323
PCA_model	SPECPHOB	1052	20	0.904781	0.632093	6.248156
PCA_model	SPECPHOB	1052	30	0.904748	0.637603	5.805026
PCA_model	SPECPHOB	777	20	0.90579	0.638784	5.861352
PCA_model	SPECPHOB	777	30	0.905823	0.644847	5.820113
PCA_model	ANX	987	20	0.957409	0.723833	6.54101
PCA_model	ANX	987	30	0.957409	0.739574	6.177893
PCA_model	ANX	1052	20	0.956195	0.748236	6.092988
PCA_model	ANX	1052	30	0.956063	0.759937	5.956755
PCA_model	ANX	777	20	0.957463	0.739784	6.217742
PCA_model	ANX	777	30	0.957397	0.748179	5.886543