Svetlana Koroteeva ID: 300432399

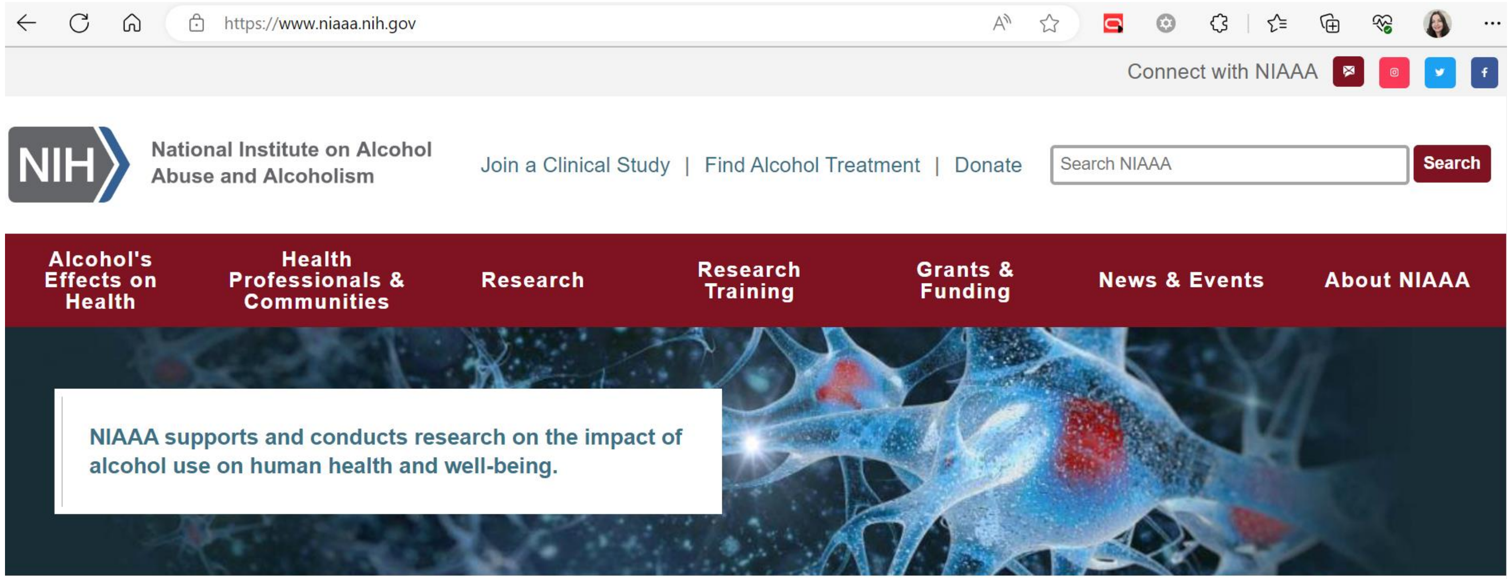
AIML 427 - Big Data

Assignment 3

Personal disorders classifiers

based on NESARC data

ABOUT DATASET



1

Tape Location	Source Code	Frequency	Item value and description
	SECTION 1: BACKGROUND INFORMATION		
1-5	IDNUM		UNIQUE ID NUMBER WITH NO ALPHABETICS
		43093	1-43093. Unique Identification number
6-10	PSU		PSU
		43093	1001-56017. Psu
11-14	STRATUM		STRATUM
		43093	101-5605. Stratum
15-32	WEIGHT		FINAL WEIGHT [Format: XXXXX.XXXXXXXXXXXX]
		43093	398.03738221-57902.204788. Weighting factor
33-34	CDAY		DATE OF INTERVIEW: DAY
		43093	1-31. Day
35-36	CMON		DATE OF INTERVIEW: MONTH
		43093	1-12. Month
37-40	CYEAR		DATE OF INTERVIEW: YEAR
		36992	2001. 2001
		6101	2002. 2002
41-41	REGION		CENSUS REGION
		8209	1. Northeast
		8991	2. Midwest
		16156	3. South
		9737	4. West

National Epidemiologic Survey of Drug Use and Health Code Book

DATASET specifics and problems

S2AQ6F	S2AQ6G	S2AQ6H	S2AQ6I	WINEECF	S2AQ7A	S2AQ7B	S2AQ7CR	S2AQ7D	S2AQ7E	S2AQ7F	S2AQ7G	S2AQ7H	S2AQ7I	LIQRECF	S
					2										
10	11	3	1	0.125	2										
					1	10	1	1	2	10	11	1	2	0.4	
6	11	2	1	0.11	2										
10	11	1	1	0.135	1	9	35	1	1	9	11	3	1	0.17	
					1	8	1	5	8	10	8	1	2	0.4	
6	11	1	1	0.135	2										
					1	4	3	2	3	8	11	1	1	0.4	
					2										
					1	4	1	1	3	9	11	1	1	0.4	

43094 instances

Categorical values

Blank or populated with 9,99,999 values

Sometimes several columns represent a class

50.0% missing data in 2183 features from 3008

DATASET CLEANING and PREPARING

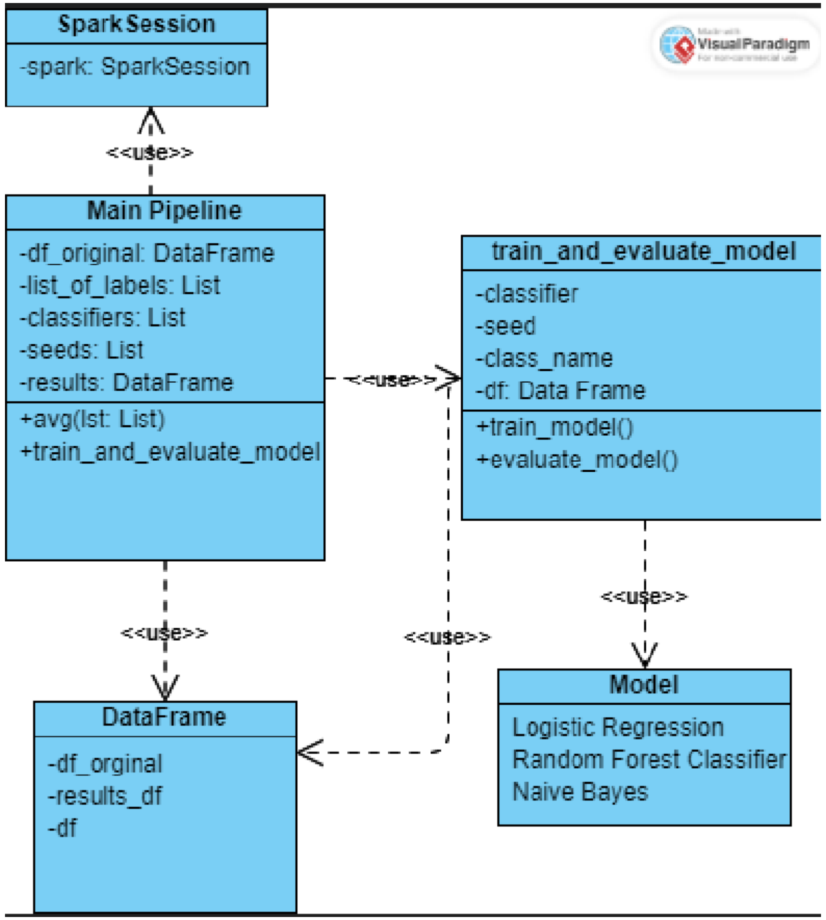
```
df = pd.read_csv(r"data\nesarc_cleaned.csv")
print(df.shape)
df = df.replace(r'^\s*$', np.nan, regex=True)
df = df.replace({'99': pd.np.nan})
df = df.replace({'9': pd.np.nan})
```

652 columns left after deleting those columns which have missing data more then 30%

```
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")
```

```
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}")
```

Baseline model



Accuracy 100%?

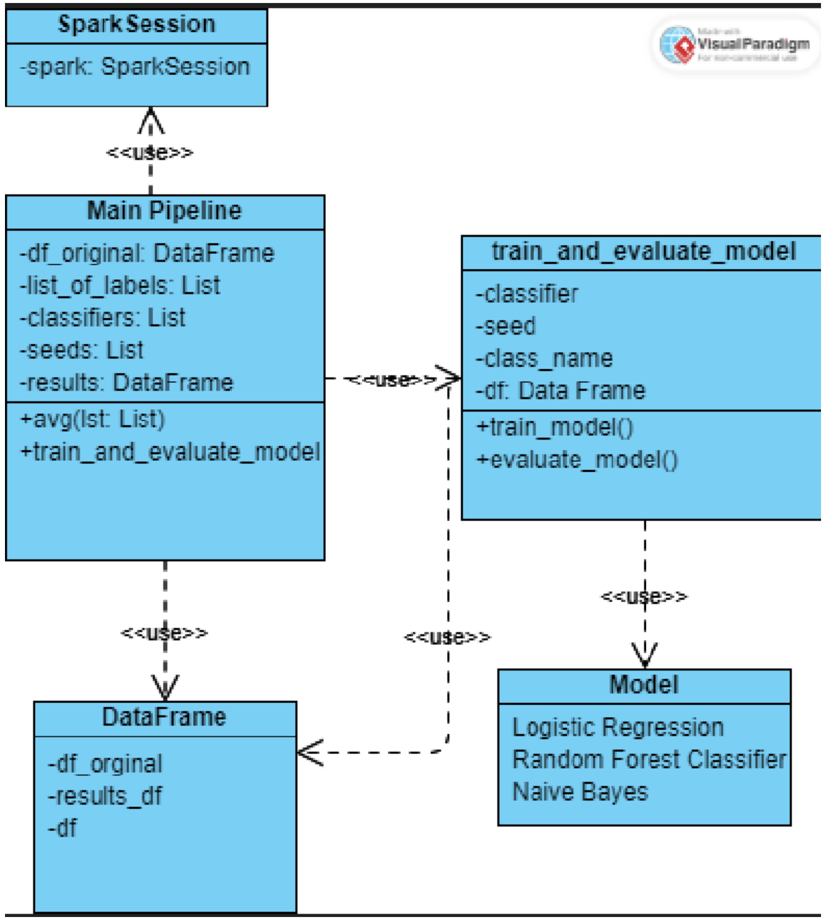
Feature selection

```
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
```

362 columns left

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

Baseline model



Train and evaluate the model pseudocode

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.

Baseline model performance 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

Normalized data model

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

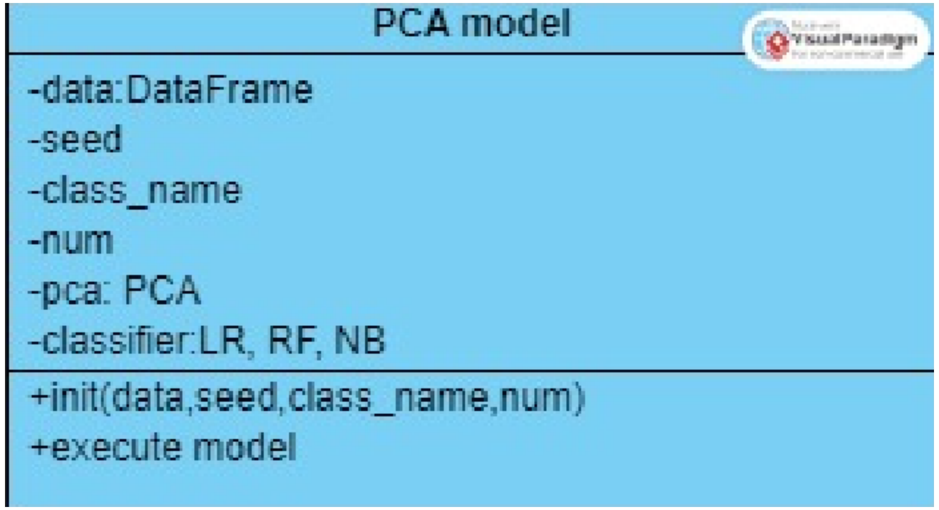
(Applied for both training and test data)

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

Why scaling did not work?

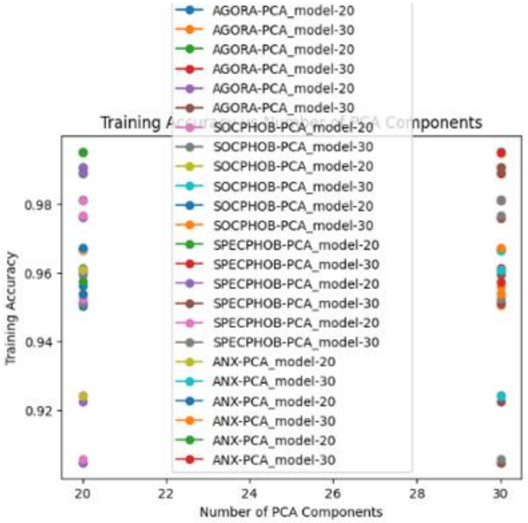
- Data is in the same magnitude
- Data is categorical

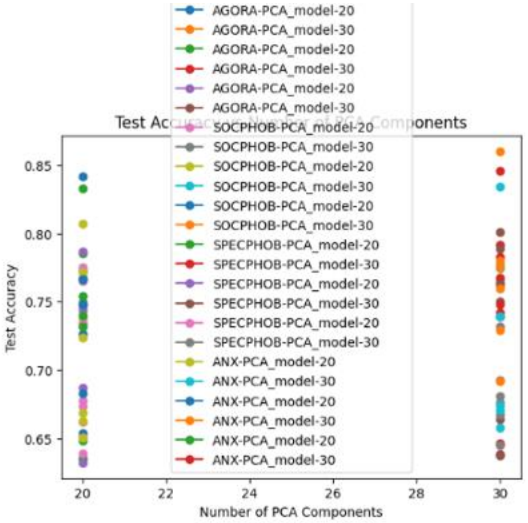
PCA model

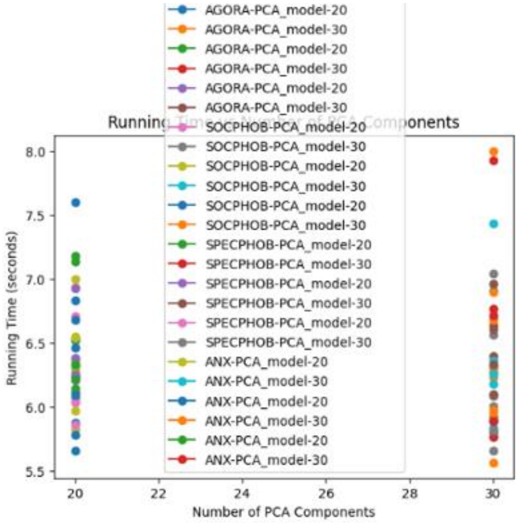


Classifier	Class	Seed	Num PCA Comp	Train Accuracy	Test Accuracy	RunTime
PCA_model	CONDUCTONLY	987	3	0.990304	0.540806	15.59032
PCA_model	CONDUCTONLY	987	10	0.990304	0.57854	9.47362
PCA_model	CONDUCTONLY	1052	3	0.990637	0.536487	7.497615
PCA_model	CONDUCTONLY	1052	10	0.990637	0.53302	8.667549
PCA_model	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	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

MORE PCA


Training Accuracy
Number of PCA Components
AGORA-PCA_model-20
AGORA-PCA_model-30
AGORA-PCA_model-20
AGORA-PCA_model-30
AGORA-PCA_model-20
AGORA-PCA_model-30
SOCPHOB-PCA_model-20
SOCPHOB-PCA_model-30
SOCPHOB-PCA_model-20
SOCPHOB-PCA_model-30
SOCPHOB-PCA_model-20
SOCPHOB-PCA_model-30
SPECPHOB-PCA_model-20
SPECPHOB-PCA_model-30
SPECPHOB-PCA_model-20
SPECPHOB-PCA_model-30
SPECPHOB-PCA_model-20
SPECPHOB-PCA_model-30
ANX-PCA_model-20
ANX-PCA_model-30
ANX-PCA_model-20
ANX-PCA_model-30
ANX-PCA_model-20
ANX-PCA_model-30


Test Accuracy
Number of PCA Components
AGORA-PCA_model-20
AGORA-PCA_model-30
AGORA-PCA_model-20
AGORA-PCA_model-30
AGORA-PCA_model-20
AGORA-PCA_model-30
SOCPHOB-PCA_model-20
SOCPHOB-PCA_model-30
SOCPHOB-PCA_model-20
SOCPHOB-PCA_model-30
SOCPHOB-PCA_model-20
SOCPHOB-PCA_model-30
SPECPHOB-PCA_model-20
SPECPHOB-PCA_model-30
SPECPHOB-PCA_model-20
SPECPHOB-PCA_model-30
SPECPHOB-PCA_model-20
SPECPHOB-PCA_model-30
ANX-PCA_model-20
ANX-PCA_model-30
ANX-PCA_model-20
ANX-PCA_model-30
ANX-PCA_model-20
ANX-PCA_model-30


Running Time (seconds)
Number of PCA Components
AGORA-PCA_model-20
AGORA-PCA_model-30
AGORA-PCA_model-20
AGORA-PCA_model-30
AGORA-PCA_model-20
AGORA-PCA_model-30
SOCPHOB-PCA_model-20
SOCPHOB-PCA_model-30
SOCPHOB-PCA_model-20
SOCPHOB-PCA_model-30
SOCPHOB-PCA_model-20
SOCPHOB-PCA_model-30
SPECPHOB-PCA_model-20
SPECPHOB-PCA_model-30
SPECPHOB-PCA_model-20
SPECPHOB-PCA_model-30
SPECPHOB-PCA_model-20
SPECPHOB-PCA_model-30
ANX-PCA_model-20
ANX-PCA_model-30
ANX-PCA_model-20
ANX-PCA_model-30
ANX-PCA_model-20
ANX-PCA_model-30

Why PCA did not work

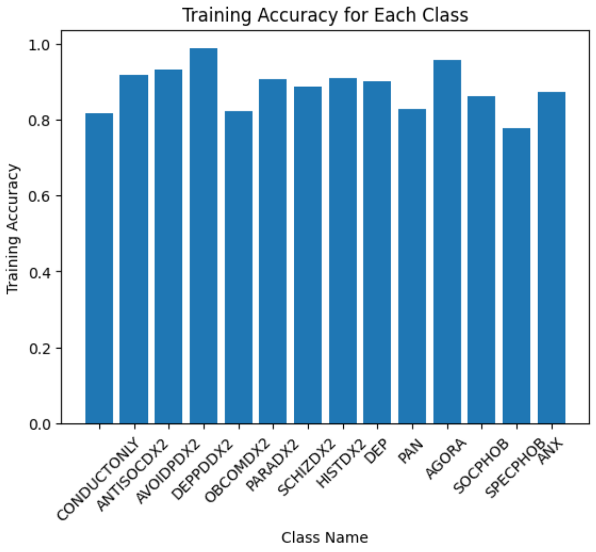
Categorical data

Non-linear relationships

Outliers?

Further trials: Lasso

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



unbalanced data

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)