

Bioinformatics CS-6643

Final Project Report

Major Depression Disorder Analysis with Gene Expression and Demographic Symptom Data

Instructor: Dr. Brett McKinney

Author: Selim Karaoglu

In this project, we conduct data analysis experiments on Major Depression Disorder data provided by Le et al.'s research. This data is provided in two sets: gene expressions and demographic symptoms of subjects. Following their research, we recreated a similar scenario to understand the importance of different genes and symptoms. To measure the importance of these features, we trained different AI models and present the results of these experiments.

# 1. Introduction

This project focuses on the analysis of Major Depression Disorder (MDD) dataset presented by Le et. al. in their research [1]. They adopt a co-expression network to detect the significant gene-expression modules on RNA-Seq data based on 79 healthy control (HC) and 78 MDD, in total 157 subjects. Within this co-expression network they utilize centrality metric to see the importance of individual gene expressions in different modules. Similar to this research, we conducted experiments on gene expression data, however; instead of using a co-expression network, we employed univariate, recursive and model-based feature selection methods to uncover the importance of different genes. Moustafavi et. al.'s work stated that there is no single gene expression association that indicates MDD, but joint gene expression yields significant results [2]. Parallel to these findings, there are overwhelming evidence that the MDD is heritable [3,4], therefore gene expressions can show importance on MDD diagnosis classification tasks. Following this idea, we conducted experiments with a subset of gene expressions from several feature selection method results. To achieve this goal, the feature selection plays a critical role since the joint gene expression is designed at this step of our experiment. There are several feature selection methods adopted in this project and their results are compared to see which gene expressions found significant by each method. The gene expressions that appear the most significant are grouped together. After the creation of the joint gene expression subset, we train different AI models to make binary classification on the subject's diagnosis, in other words, we experiment to see if the joint gene expression subset can help detect MDD. To explain our work in detail, first we present some background information.

2. Background

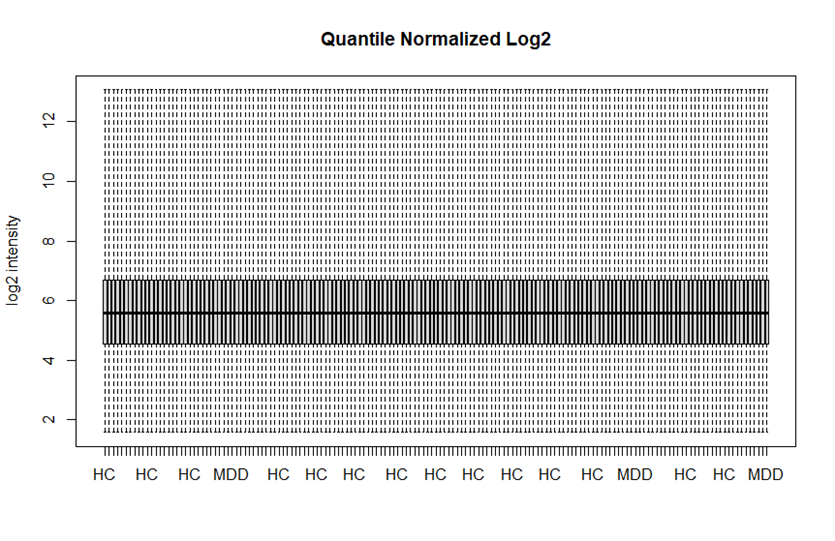
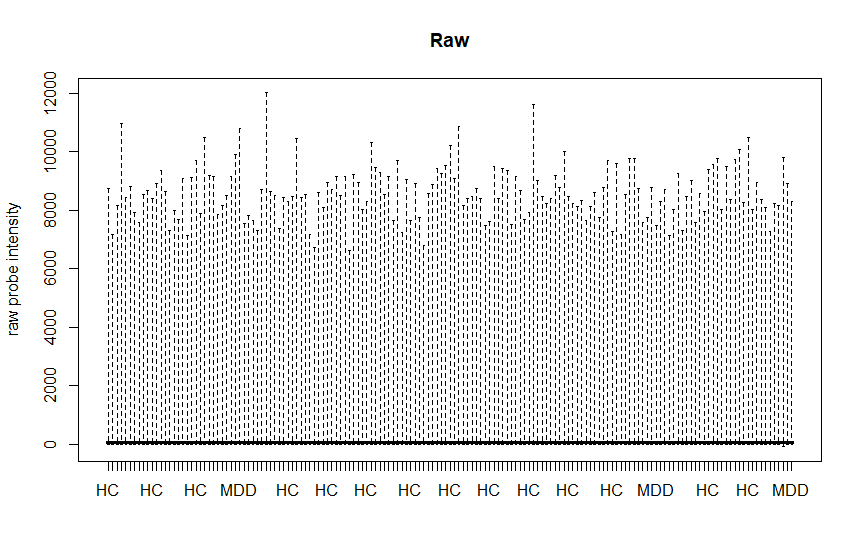
In this section, we brought clarity to the techniques employed in this project. First, we present the data explored in the Le et. al.’s paper. After gathering this data, several pre-processing methods applied to the gene expression set to achieve better results. Application of the pre-processing leads our research to conduct feature selection experiments with the gene selection and therefore we conclude this section with explaining all different feature selection methods utilized for this work.

## 2.a. Data

This data is collected by Le et al. to experiment on their project to create a co-expression network of gene expression and see the importance of modular expression profiles on MDD diagnosis. Their RNA-seq computation workflow follows these steps: pre-processing the raw data, applying CoV filtering, constructing a co-expression network and pruning the dynamic tree iteratively. Lastly, they perform false discovery threshold test. The data is collected from 160 subjects between ages 18-55 from the Laureate Psychiatric Clinic and Hospital. 93 of the subjects are female with 52 MDD diagnosed patients and 41 healthy controls. 3 of the samples are taken out since they were outliers. The morning blood samples were gathered from subjects and RNA expressions are obtained from these samples with sequencing depths around 30 million reads. The researchers removed the gene expressions with low counts, quantile normalized the values, removed the 3 outlier samples, and applied batch effect correction and coefficient of variation (CoV) filtering for CoV values below 0.8. The resulting data contains 8923 different gene expressions for 157 subjects and there are 40 features in the demographic symptom data. The demographic symptom data contains demographic information like age and sex and scores from different test results. There are some negative score values and some empty fields in the demographic symptom data that needs to be addressed before applying any feature importance detection method. To conduct our experiments, first we apply some required pre-processing methods on the data.

## 2.b Pre-Processing

The data provided in Le et al.’s project is applied thresholding, quantile normalization, outlier detection, batch effect correction and CoV filtering. Even after the filtering process, the data balance is quite unfavorable for classification methods since there are 8923 features for only 157 samples. Therefore, we applied further calculations to eliminate gene expressions that does not show significance. First the data is transformed with log2 transformation to prevent the effects of extremities and then normalized with quantile normalization. Below figure suggests that the probe intensities show extreme differences in the original data, but the transformed and normalized data values show the probe intensities are very similar.



Like Le et al.’s method, we also applied coefficients of variation filtering, however, they filtered samples with standard deviation value higher than 0.8, in contrast we filter samples higher than 0.45 standard deviation. With the CoV filtering and removing the single sample with 0 standard deviation, the feature size of the remaining data is 5587 for 157 samples.

The data pre-processing part of this project is conducted in R. After the pre-processing, the data is saved and stored as “filtered.csv” and imported in Python environment for further experimenting. We utilized “pandas” library to create a data frame of the filtered data and separated the diagnosis column as targets. The demographic symptom data does not need any pre-processing; however, we excluded diagnosis, primary diagnosis and batch columns, diagnosis column is assigned as targets after converted to binary values (MDD = 1 and HC = 0). There were some blank entries in the demographic symptom data, before the implementation, these blank values are filled with the mean value of the corresponding column. Below figures show value distributions of each feature for filtered gene expression data and demographic symptom data respectively where x axis shows features and y axis shows their values.

A picture containing text, outdoor, plant

Description automatically generatedChart, box and whisker chart

Description automatically generated

Filtered gene expression dataset does not have any negative values, this is important for feature selection since chi square and mutual information methods do not support negative values. Demographic symptom data does contain negative score values, but we do not alter those values since they are based on scoring metric results.

For training the AI models, we need to split data into training and test subsets. Since there are only 157 samples in our data, we split random 125 samples as training set and 32 samples as test set. This training - test split is only applied for classification part.

## 2.c Feature Selection

In this part, we provide information about the different feature selection methods we adopted in this project and how they measure the significance of the features. The feature selection methods used in this project are chi square, ANOVA f-value, mutual information, Recursive Feature Elimination with Cross Validation (RFECV), false positive rate, false discovery rate, family-wise error, L1 penalty and Tree Based model. All these methods are available in “sklearn” library for Python, for L1 penalty method we utilized Linear Support Vector Classifier (SVC) and for tree-based model we utilized Random Forest classifier.

**Chi square:** This method works by computing the chi squared statistics between each non-negative feature and targets. The chi square can be calculated with:

**χ2 = ∑ (O − E)2 / E**

We use this score to get the desired number of features sorted by their chi square values. This test measures dependance factor among stochastic variables by comparing the observed and expected values, therefore by using this test we can measure which features are more important.

**ANOVA f-value:** ANOVA stands for “analysis of variants” and compares if the means of more than 2 independent groups to see if there is significant difference. The f-value in ANOVA represents the ratio of variation of sample means to variation within samples. The ANOVA f-value calculation focuses on sum of the squares and can be formulated as:

F = ∑kj=1∑lj=1(x¯j − xj)2 / k−1

where x represents data points, x¯j is the mean and k-1 is the degrees of freedom. Similar to chi square method, we use this method to extract the features that shows the most significance.

**Mutual Information:** This method estimates the mutual information for a discrete target variable by measuring the dependency between features. If the variables are independent, the mutual information is expected to be zero, if mutual information is not zero, higher values show higher dependencies.

**Recursive Feature Elimination with Cross Validation:** RFECV method uses an external estimator that assigns weights for each feature. We implemented a Linear SVC to be the external estimator and assign weights through the RFECV process. This method follows a top-down approach to eliminate features by recursively considering smaller subsets of data. In this method, Linear SVC is trained on the initial data and importance of features are obtained with accuracy score of the classifier. Following, least important features are pruned from the dataset and this process is repeated in a loop to find the optimal number of features.

**False Positive Rate:** False positiverate (FPR) is the probability of a test being falsely rejecting the null hypothesis, calculated by dividing the number of false positives (FP) with number of false positives and number of true negatives (TN) summed.

FPR = FP / (FP + TN)

**False Discovery Rate:** False discovery rate (FDR) is similar to FPR, but instead of taking true negatives, true positives are taken into consideration. Similar to the q-value thresholding applied with FDR in Sullivan et al.’s paper [5], we set the significance to 0.1 in our experiment.

FDR = FP / (FP + TP)

**Family-wise Error:** Family-wise error rate refers to the probability of resulting with false positive through a multiple hypothesis test.

FWE = P(FP ≥ 1)

# 3. Methods

The filtered gene expression data and demographic symptom data features are further examined for their importance by using the feature selection methods mentioned in the previous section. After filtering the gene expressions and demographic symptoms, we trained AI models to measure their accuracy on classifying the MDD diagnosis. First, we present the feature selection method results and what features show significance.

## 3.a Selecting Important Features

### 3.a.I Demographic Symptom Features

The chi square test can’t be applied to dataset with negative values; hence this test is not applicable for demographic symptom data. The selected features by the methods’ are presented in the table below. After achieving results of each model, we trained a Gaussian Naïve Bayes (GNB) classifier to train on the subset of features selected by each method and the weighted average accuracy results are placed after features for each method.

|  |  |  |
| --- | --- | --- |
| Method | Selected Features | GNB accuracy |
| ANOVA f-value 10 best features | ‘shaps\_score', 'madrs\_score', 'hamd\_hamd17\_score', 'hamd\_hamd21\_score', 'hama\_score', 'poms\_score\_fatigue', 'poms\_score\_tmd', 'qids\_sleep\_score', 'qids\_appetite\_score', 'qids\_score' |  |
| ANOVA f-value 4 best features (%1) | 'madrs\_score', 'hamd\_hamd17\_score', 'hamd\_hamd21\_score', 'hama\_score' | %94 |
| Mutual information 10 best features | ‘shaps\_score', 'psqi\_score', 'madrs\_score', 'hamd\_hamd17\_score', 'hamd\_hamd21\_score', 'hama\_score', 'poms\_score\_depression', 'poms\_score\_tmd', 'qids\_appetite\_score', 'qids\_score' |  |
| Mutual information 4 best features (%1) | 'madrs\_score', 'hamd\_hamd17\_score', 'hamd\_hamd21\_score', 'hama\_score' | %94 |
| RFECV | ‘psqi\_score’, ‘hamd\_hamd21\_score’ | %94 |

An important note here is that the ANOVA f-value and Mutual information select same features on best %1, and they show a little variance on the selection of best 10 features.Chart, line chart

Description automatically generated RFECV result suggests the optimal number of features is 2 with ‘psqi\_score’ and ‘hamd\_hamd21\_score’. Figure near shows the cross validation accuracies for different number of features, as the figure suggests, the optimal number of features detected by RFECV method is 2.

### 3.a.II Gene Expression Features

The Mutual information method is not included in FPR, FDR and FWE due to NoneType values occurring in the calculations, in addition to that, chi square is excluded from FDR and FWE since there is no significant difference in p-values in these methods. The feature selection methods’ results are:

|  |  |  |
| --- | --- | --- |
| Method | Selected Features | GNB accuracy |
| **χ2 10 best features** | 'CD83', 'DPEP3', 'FAM129B', 'HMGN2P46', 'MDGA1', 'NKX1-1', 'RARS2', 'TNC', 'UBD', 'ZDHHC20' | %69 |
| **χ2 56 best features (%1)** | 'ABCE1', 'AHCTF1P1', 'AHSG', 'AP4B1-AS1', 'BIRC5', 'C1orf35', 'CD83', 'CDC14B', 'CDC20', 'CHMP1B2P', 'CHP1', 'CHSY3', 'DDHD2', 'DDX26B-AS1', 'DDX58', 'DPEP3', 'EHF', 'ELK4', 'FAM129B', 'FLJ31356', 'FOXRED2', 'GSC', 'HMGN2P46', 'HNRNPA1P10', 'ICOS', 'IFI16', 'LINC01001', 'LUC7L2', 'MCTS2P', 'MDGA1', 'METTL4', 'MITF', 'MRPS26', 'MYO1A', 'NAA20', 'NKX1-1', 'NR1H2', 'OIP5', 'PAPL', 'POGZ', 'PPP2R5C', 'PTP4A2', 'RARS2', 'SS18L2', 'SUMO3', 'TAF1D', 'TNC', 'TOMM34', 'UBD', 'WDR76', 'ZDHHC20', 'ZFP36L2', 'ZMYM6NB', 'ZNF225', 'ZNF518B', 'ZNF549' | %62 |
| ANOVA f-value 10 best features | 'ARFGAP1', 'BCL2L12', 'CBL', 'FAM138A', 'IRF2BPL', 'KANTR', 'MDGA1', 'NPFF', 'UBD', 'ZDHHC20' | %70 |
| ANOVA f-value 56 best features (%1) | 'ABCE1', 'AP1G2', 'ARFGAP1', 'ATXN7L2', 'BCL2L12', 'C16orf74', 'C1orf35', 'CBL', 'CCAR2', 'CD83', 'CDC20', 'CDK5RAP2', 'CSPG4', 'DPEP3', 'DYNLRB1', 'EIF2D', 'ELP3', 'EXOSC2', 'FAM129B', 'FAM138A', 'FAM193A', 'HMGN2P46', 'HNRNPUL2', 'ICOS', 'IRF2BPL', 'ISX', 'KANTR', 'KPNA2', 'LEUTX', 'LRRC37BP1', 'MCM3AP-AS1', 'MDGA1', 'METAP1D', 'NDUFB1', 'NGRN', 'NPFF', 'PANX1', 'POGZ', 'PRPF31', 'RARS2', 'RCSD1', 'SETDB2', 'SRSF5', 'SS18L2', 'STX16-NPEPL1', 'SUPT7L', 'TADA3', 'TOMM34', 'TRPC5', 'TSC2', 'TSR3', 'UBD', 'ZDHHC20', 'ZFP36L2', 'ZMYM6NB', 'ZNF658' | %66 |
| ANOVA f-value 6 best features (%0.1) | 'ARFGAP1', 'FAM138A', 'IRF2BPL', 'MDGA1', 'NPFF', 'ZDHHC20' | %69 |
| Mutual information 10 best features | 'AASS', 'ARPC4', 'BAZ2A', 'FUNDC1', 'MKNK1-AS1', 'RNASEH1-AS1', 'RPL15', 'SNX18', 'SRSF7', 'ZMYM6NB' | %70 |
| Mutual information 56 best features (%1) | 'AASS', 'ANAPC4', 'ARPC4', 'BAZ2A', 'C9orf116', 'CASP10', 'CCAR2', 'COPB2', 'COPS6', 'DNAJC11', 'EIF3A', 'FCGBP', 'FOXN3-AS1', 'FUNDC1', 'GSC', 'HADH', 'HNRNPA1P10', 'HSPE1', 'IREB2', 'KBTBD3', 'KDM1B', 'LEUTX', 'LIMK2', '8-Mar', 'MIIP', 'MKNK1-AS1', 'MPO', 'MRPS31', 'N4BP2L1', 'NAB2', 'NUDT6', 'PCYT1A', 'PTCH1', 'RCSD1', 'RNASEH1-AS1', 'RNASEH2B-AS1', 'RPL15', 'RRP9', 'SEC61A2', 'SEPW1', 'SLC35A5', 'SNX18', 'SRSF7', 'STYXL1', 'SUPT7L', 'TAB3', 'TBCA', 'TRIM39', 'TYRP1', 'UBL4B', 'UBR4', 'VPS18', 'WSCD1', 'ZBTB11-AS1', 'ZFP91-CNTF', 'ZMYM6NB' | %46 |
| FPR with **χ2** | 'ABCE1', 'AHCTF1P1', 'AHSG', 'AP4B1-AS1', 'BIRC5', 'C1orf35', 'CD83', 'CDC14B', 'CDC20', 'DDHD2', 'DDX26B-AS1', 'DPEP3', 'EHF', 'FAM129B', 'FLJ31356', 'FOXRED2', 'GSC', 'HMGN2P46', 'HNRNPA1P10', 'ICOS', 'IFI16', 'LINC01001', 'LUC7L2', 'MDGA1', 'METTL4', 'MRPS26', 'NAA20', 'NKX1-1', 'NR1H2', 'OIP5', 'PAPL', 'POGZ', 'RARS2', 'SS18L2', 'SUMO3', 'TNC', 'UBD', 'WDR76', 'ZDHHC20', 'ZFP36L2', 'ZMYM6NB', 'ZNF225', 'ZNF518B', 'ZNF549' | %62 |
| FPR with ANOVA f-value | 'ABCE1', 'ARFGAP1', 'ATXN7L2', 'BCL2L12', 'C16orf74', 'C1orf35', 'CBL', 'CCAR2', 'CD83', 'CDK5RAP2', 'CSPG4', 'DPEP3', 'DYNLRB1', 'EIF2D', 'ELP3', 'FAM138A', 'FAM193A', 'HMGN2P46', 'HNRNPUL2', 'ICOS', 'IRF2BPL', 'ISX', 'KANTR', 'LEUTX', 'MCM3AP-AS1', 'MDGA1', 'NDUFB1', 'NGRN', 'NPFF', 'PANX1', 'POGZ', 'RARS2', 'RCSD1', 'SETDB2', 'SRSF5', 'SS18L2', 'STX16-NPEPL1', 'SUPT7L', 'TADA3', 'TOMM34', 'TRPC5', 'TSC2', 'TSR3', 'UBD', 'ZDHHC20', 'ZFP36L2', 'ZMYM6NB', 'ZNF658' | %66 |
| FDR with ANOVA f-value | 'ABCE1', 'ARFGAP1', 'ATXN7L2', 'BCL2L12', 'C16orf74', 'CBL', 'CCAR2', 'CD83', 'CDK5RAP2', 'CSPG4', 'DPEP3', 'DYNLRB1', 'EIF2D', 'FAM138A', 'FAM193A', 'HMGN2P46', 'HNRNPUL2', 'ICOS', 'IRF2BPL', 'ISX', 'KANTR', 'MDGA1', 'NDUFB1', 'NGRN', 'NPFF', 'PANX1', 'POGZ', 'RARS2', 'RCSD1', 'SETDB2', 'SRSF5', 'SS18L2', 'STX16-NPEPL1', 'SUPT7L', 'TADA3', 'TOMM34', 'TRPC5', 'TSC2', 'TSR3', 'UBD', 'ZDHHC20', 'ZFP36L2', 'ZMYM6NB', 'ZNF658' | %70 |
| FWE with ANOVA f-value | 'ARFGAP1', 'MDGA1', 'NPFF', 'ZDHHC20' | %62 |
| RFECV | ‘CDC14B ‘, ‘ELP3’, ‘MICU1’, ‘TUSC3’ | %66 |
| L1 penalty (c=0.1) | 'GSC', 'HMGN2P46', 'MDGA1', 'PANX1', 'UBD' | %65 |
| Tree-based | 'C6orf118', 'ELK4', 'FAM109A', 'FOXL1', 'N4BP2', 'UBE2Q1', 'ZDHHC20', 'ZFYVE27' | %69 |

Chart

Description automatically generated

RFECV results show that the accuracy increases throughout the cross validation process, but dips right before it reaches to complete as the figure on the right suggests. The optimal number of features is 4 and selected gene expressions are ‘CDC14B ‘, ‘ELP3’, ‘MICU1’ and ‘TUSC3’. It is important to note that the accuracies given in the table are weighted accuracy results of GNB model trained on 125 samples with number of features filtered in each method. The accuracies vary between %62 and %70 (excluding one %46) although different classification methods might yield better results with a selection of features. Before starting model selection process, we finalize the feature selection by creating a joint gene expression data subset. To do this, the gene expressions are counted and ordered with their appearances in out feature selection method results. Our experiments showed that the genes ‘MDGA1’ and ‘ZDHHC20’ both selected by 10 out of our methods, ‘UBD’ selected by 8, ‘HMGN2P46' and 'ZMYM6NB’ selected by 7, 'ARFGAP1', 'NPFF', 'CD83', 'DPEP3' and 'RARS2' selected by 6 feature selection methods. We created a joint gene expression subset by filtering data with only these 10 gene expressions.

## 3.b Model Selection

In this part of the project, we select and train different AI models on gene expression and demographic symptom data to compare which method results with the highest accuracy. Our previous experiment showed that with feature selection applied, GNB classification can achieve %94 accuracy. This is mainly due to some of the scoring metric results can correlate to MDD and achieving high accuracies with demographic symptoms are easier to achieve. In contrast, selected gene expressions data accuracies are not satisfactory and needs improvement. Here we concentrate on classifications with joint gene expression data and select models suitable for this data. One of the limitations with the joint gene expression data is the target type, due to the diagnosis being MDD or healthy, we use binary targets and adopted methods can be applicable for binary classification. In addition, Naïve Bayes classifications are based on the assumption that the features are i.i.d. (independent and identically distributed), however this assumption might not be suitable for gene expressions. Furthermore, the data is built with continuous values, therefore any classification method that favors discrete features are not expected to yield good results.

**Gaussian Naïve Bayes** classifier is based on the assumption of continuous values are sampled from a normal distribution. Calculates the densities using standard deviation and mean and fits the data to a Gaussian distribution. This method does not take any parameters on training.

**K-Nearest Neighbor** classifier compares distances of k number of nearest neighbors. This method requires k as a parameter to train. To find the best k values for a classification task, we cross validated through different k values and present the results of the k values that achieved the highest weighted average precision score.

**Linear SVC** is alinear method that uses L1 or L2 penalty metrics to calculate errors and takes C value as a parameter. For this experiment, we employed the L1 penalty since we used the same method in feature selection and cross validated through different C values and present the results that achieved the highest accuracy.

**RBF Kernel SVC** uses Radial Basis Function as kernel in SVC and takes gamma and C as parameters. Here the gamma is the kernel coefficient, and the C value is a regularization parameter. The cross validation applied for both parameters to select the optimal values that yield the best results.

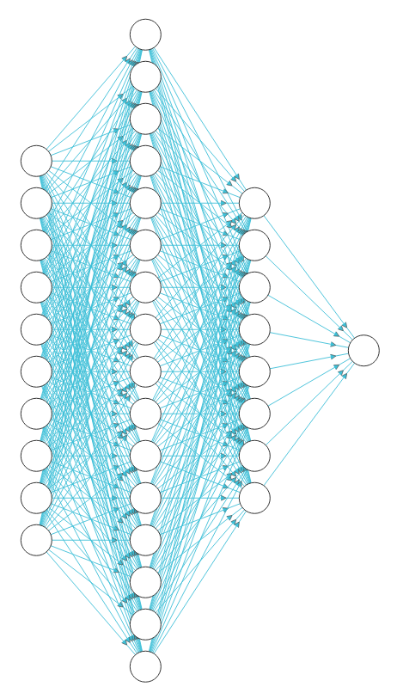
**Logistic Regression** is another method used for classification tasks and it does not take any parameters. This method predicts a binary output given a set of independent variables and uses the Log-loss function to achieve that.

**Random Forest** classifier fits several decision tree classifiers on sub-samples. To control over-fitting and improve accuracy this method adopts averaging. For this experiment, random forest criterion is set to Gini index, number of estimators is set to 100 and enabled bootstrapping.

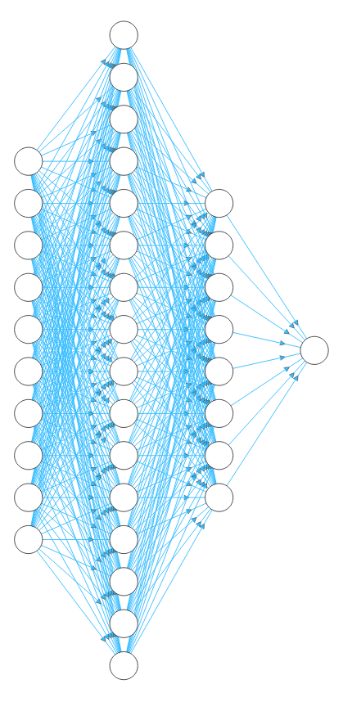
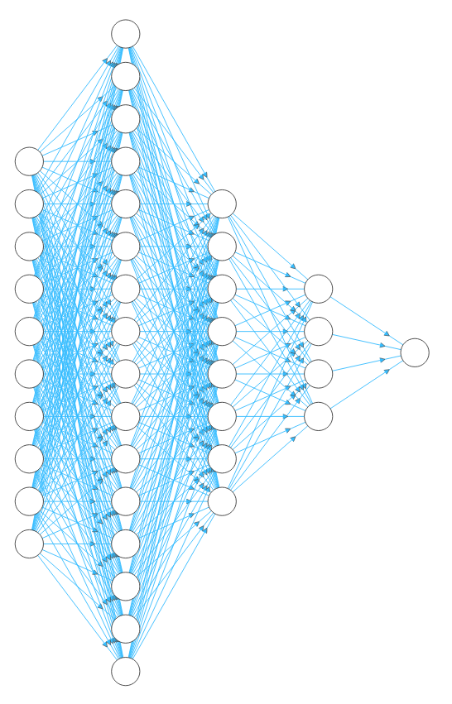
**AdaBoost** classifier is a meta-estimator based on decision trees like random forest. This method fits additional instances with close to original but still wrong weights to make algorithm focus on correct weights. The number of estimators is set to 50 and learning rate is assigned 1. Implementation of this method follows Zhu et al.’s implementation [6].

**Quadratic Discriminant Analysis** classifier has a quadratic decision boundary instead of a linear one. This quadratic boundary is created by using the Gaussian assumption on fitting the densities to data.

**Simple Dense Neural Networks** are built by creating neural network models with fully connected nodes in each hidden layer. For this experiment we cross validated over several different neural network structures by altering the number of hidden layers, the number of nodes in each layer and the activation functions. We selected 3 structures that performed the best amongst the similar model structures. First model contains input layer with 10 nodes, 2 hidden layers with 16 and 8 nodes and Rectified Linear Unit (ReLU) activation function and sigmoid output layer. The optimizer employed for all of the Simple DNN models is Adam and the loss function is binary cross-entropy. The left figure below shows the first NN architecture design. Second model is built with 10 nodes input layer, 3 hidden layers with 16, 8 and 4 nodes and ReLU activation functions and sigmoid output layer. Middle figure presented below shows this models architecture. The last model is similar to the first model structure, the first layer is input layer with 10 nodes, the second layer has 16 nodes and uses ReLU activation function, following this first hidden layer, we apply some dropout on the edges with rate of 0.2, the second hidden layer has 8 nodes and Hyperbolic Tangent (TanH) activation function and the output layer has sigmoid function. The figure on the right shows this models architecture design.



**Dropout 0.2**



# 4. Results

In our experiment, we use the gene expression and demographic symptom data on MDD. First we applied quantile normalization and log2 transformation on the gene expression data. After these, we filtered out the features with CoV values lower than 0.45 and 0 standard deviation. This filtered data contains 5587 features for 157 samples. Since training an AI model to classify MDD diagnosis on this many features would be a hard task, we employed feature selection methods to identify significant features. By conducting experiments with several methods and comparing their results, we created a subset of gene expression data with 10 features selected by the most common appearance observed in feature selection methods’ results. We hypothesized that this joint gene expression data can yield better classification results compared to the filtered data. To see if this hypothesis is true, here we present the classification results based on models’ weighted average precision scores.

|  |  |  |
| --- | --- | --- |
| Classification Method | Filtered Data Accuracy | Joint gene expression accuracy |
| Decision Tree Classifier | %75 (highest in 10-fold CV) | %66.7 (highest in 10-fold CV) |
| Gaussian Naïve Bayes | %56 | %76 |
| K-Nearest Neighbor | %60 | %77 |
| Linear SVC | %49 | %72 |
| RBF kernel SVC | %19 | %81 |
| Logistic Regression | %39 | %72 |
| Random Forest Classifier | %52 | %71 |
| AdaBoost | %35 | %75 |
| Quadratic Discriminant Analysis | %41 | %62 |
| Simple DNN first model | - | %73 |
| Simple DNN second model | - | %76 |
| Simple DNN third model | - | %79 |

As the results suggest, all of the classification methods yielded higher weighted average precision accuracies with the joint gene expression data, the only exception being the Decision Tree classifier. It is important to note that the results shown in the table are the highest results out of 10-fold CV results, when we average these results the joint gene expression performance is improved slightly. The best classification result belongs to the RBF kernel SVC, in addition to that this model shows the most significant improvement from %19 to %81. Following the RBF kernel SVC, the runner-up for our experiment is the third Simple DNN model with %79 accuracy. K-Nearest Neighbor classification takes the third place with %77. Even though our classification accuracies significantly increased with the joint gene expression, the results are still far from being satisfactory. This can be related to several different reasons. First of all, the sample size of 157 might not be enough for complicated relations between gene expressions to be uncovered by classification methods. In addition to that, we know that gene expressions by themselves are not enough to indicate the MDD diagnosis within our current knowledge. Finally, there might be some relations unrevealed within the feature selection methods applied in this project such as the closeness distance adopted in the Le et al.’s research. With considering these factors, the improvement of the classification methods still shows promise for further experimentation.

# 5. Future Work & Discussions

For this experiment, the biggest obstacle that prevents the training of classification models is the sample size. In order to this experiment to be improved and to obtain more accurate classification models, the sample size needs to be increased. In addition to this, we implemented some of the state-of-art classification methods that are widely accepted and utilized in different tasks, however there are other classification methods that can be experimented with. That also refers to the NN design. The only NN type employed in this work is the Simple Dense NN, although there are several other types of networks can be implemented. The Simple DNN can be best compared to a Sparse NN. Besides, RNN’s can yield good results for this task. In addition to these architectures, the Encoder - Decoder NN can be implemented where Encoder network tries to create realistic data points and the Decoder network tries to separate the real and the artificial data points. Finally, we think implementation of PCA and ICA can contribute to the feature selection process and can be included in this project in future.

# References [1] Le T., Savitz J., Suzuki H., Misaki M., Teague K., White B., Marino J., Wiley G., Gaffney P., Drevets W., McKinney B. and Bodurka J., “Identification and replication of RNA-Seq gene network modules associated with depression severity”. Translational Psychiatry (2018)8:180, DOI 10.1038/s41398-018-0234-3 [2] Mostafavi S., Battle A., Zhu X., Potash J., Weissman M., Shi J., Beckman K., Haudenschild C., McCormick C., Mei R., Gameroff M., Gindes H., Adams P., Goes F., Mondimore F., MacKinnon D., Notes L., Schweizer B., Furman D., Montgomery S., Urban A., Koller D. and Levinson D., "Type I interferon signaling genes in recurrent major depression: increased expression detected by whole-blood RNA sequencing". Mol Psychiatry. 2014 Dec; 19(12): 1267–1274.

[3] Sullivan P., Neale M. and Kendler K., “Genetic epidemiology of major depression: review and meta-analysis”. Am J Psychiatry. 2000;157:1552–1562.

[4] Glahn D., Curran J., Winkler A., Carless M., Kent J., Charlesworth J. et al., “High dimensional endophenotype ranking in the search for major depression risk genes”. Biological psychiatry. 2012;71(1):6–14. Epub 2011/10/11.

[5] Sullivan P., de Geus E., Willemsen G., James M., Smit J., Zandbelt T. et al., “Genome-wide association for major depressive disorder: a possible role for the presynaptic protein piccolo”. Molecular psychiatry. 2009;14(4):359–75. Epub 2008/12/10.

[6] Zhu J., Zou H., Rosset S. and Hastie T., “Multi-class adaboost”. Statistics and Its Interface, 2. 349-360. 2009.

# Appendix

## 1. R Code

load("sense.filtered.cpm.Rdata")

subject.attrs <- read.csv(**"Demographic\_symptom.csv"**, stringsAsFactors = FALSE)

library(dplyr)

phenos.df <- subject.attrs %>%

filter(X %in% colnames(sense.filtered.cpm)) %>%

dplyr::select(X, Diag)

colnames(phenos.df)

mddPheno <- as.factor(phenos.df$Diag)

boxplot(sense.filtered.cpm,range=0,ylab="raw probe intensity", main="Raw", names=mddPheno)

hist(sense.filtered.cpm[,1], freq=F, ylab="density", xlab="raw probe intensity", main="Raw Data Density for Sample 1")

boxplot(log2(sense.filtered.cpm), range=0,ylab="log2 intensity", main="Log2 Transformed", names=mddPheno)

hist(log2(sense.filtered.cpm[,1]), freq=F, ylab="density", xlab="log2 probe intensity", main="log2 Data Density for Sample 1")

library(BiocManager)

library(preprocessCore)

mddExprData\_quantile <- normalize.quantiles(sense.filtered.cpm)

boxplot(mddExprData\_quantile,range=0,ylab="raw intensity", main="Quantile Normalized")

mddExprData\_quantileLog2 <- log2(mddExprData\_quantile)

colnames(mddExprData\_quantileLog2) <- mddPheno

boxplot(mddExprData\_quantileLog2,range=0,ylab="log2 intensity", main="Quantile Normalized Log2")

hist(log2(mddExprData\_quantileLog2[,1]), freq=F, ylab="density", xlab="log2 probe intensity", main="log2 Quantile Normalized for Sample 1")

## 2. Python code

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import Normalizer

from sklearn.preprocessing import FunctionTransformer

from sklearn.covariance import EmpiricalCovariance

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import LinearSVC

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.naive\_bayes import BernoulliNB

from sklearn.naive\_bayes import GaussianNB

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.discriminant\_analysis import QuadraticDiscriminantAnalysis

from tensorflow.keras import Sequential

from tensorflow.keras.layers import Dense

from sklearn.feature\_selection import chi2, f\_classif, mutual\_info\_classif

from sklearn.feature\_selection import SelectKBest

from sklearn.feature\_selection import SelectPercentile

from sklearn.feature\_selection import SelectFpr

from sklearn.feature\_selection import SelectFdr

from sklearn.feature\_selection import SelectFwe

from sklearn.feature\_selection import RFECV

from sklearn.feature\_selection import SelectFromModel

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import StratifiedKFold

from sklearn.metrics import classification\_report

demog = pd.read\_csv("Demographic\_symptom.csv")

demog.rename(columns = {'Unnamed: 0':'si'}, inplace = True)

Xd = demog

yd = pd.get\_dummies(demog["Diag"])["MDD"].tolist()

Xd.index = Xd["si"]

del Xd["Diag"]

del Xd["batch"]

del Xd["PrimaryDiagnosis"]

del Xd["si"]

#One hot encoding feature "sex" 1 - Female and 0 - Male

Xd["sex"]=pd.get\_dummies(Xd["sex"])["Female"]

Xd = Xd.fillna(Xd.mean())

Xd.head()

filtered = pd.read\_csv("filtered.csv")

filtered = filtered.transpose()

filtered = filtered.rename(columns=filtered.iloc[0]).drop(filtered.index[0])

X = X.iloc[: , :5587]

y = filtered.iloc[:, -1]

y = y.tolist()

plt.figure(figsize=(15,4))

sns.boxplot(data=X)

plt.figure(figsize=(15,4))

sns.boxplot(data=Xd)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

Xd\_train, Xd\_test, yd\_train, yd\_test = train\_test\_split(Xd, yd, test\_size=0.2, random\_state=42)

print("Length of the dataset, X: %d" % len(X\_train))

print("Length of the targets, y: %d" % len(y\_train))

print("Length of first row of X: %d" % len(X\_train.iloc[0]))

dsk10f = SelectKBest(f\_classif, k=10)

dsk10f.fit\_transform(Xd, yd)

Xd\_k10f = Xd.iloc[:,dsk10f.get\_support(indices=True)]

Xd\_k10f.keys()

dsk10m = SelectKBest(mutual\_info\_classif, k=10)

dsk10m.fit\_transform(Xd, yd)

Xd\_k10m = Xd.iloc[:,dsk10m.get\_support(indices=True)]

Xd\_k10m.keys()

dsp1f = SelectPercentile(f\_classif, percentile=10)

dsp1f.fit\_transform(Xd, yd)

Xd\_1pf = Xd.iloc[:,dsp1f.get\_support(indices=True)]

Xd\_1pf.keys()

dsp1m = SelectPercentile(mutual\_info\_classif, percentile=10)

dsp1m.fit\_transform(Xd, yd)

Xd\_1pm = Xd.iloc[:,dsp1m.get\_support(indices=True)]

Xd\_1pm.keys()

Xd\_train\_1pm, Xd\_test\_1pm, yd\_train\_1pm, yd\_test\_1pm = train\_test\_split(Xd\_1pm, yd, test\_size=0.2, random\_state=42)

gnb\_d1pm = GaussianNB()

gnb\_d1pm.fit(Xd\_train\_1pm, yd\_train\_1pm)

gnb\_p\_d1pm = gnb\_d1pm.predict(Xd\_test\_1pm)

print("Gaussian Naive Bayes Classifier report for RFECV: \n\n", classification\_report(yd\_test\_1pm, gnb\_p\_d1pm))

gnb = GaussianNB()

drfecv = RFECV(

estimator=svc,

step=1,

cv=StratifiedKFold(2),

scoring="accuracy",

min\_features\_to\_select=2,

)

drfecv.fit(Xd, yd)

print("Optimal number of features : %d" % drfecv.n\_features\_)

plt.figure()

plt.xlabel("Number of features selected")

plt.ylabel("Cross validation score (accuracy)")

plt.plot(

range(2, len(drfecv.grid\_scores\_) + 2),

drfecv.grid\_scores\_,

)

plt.show()

drfecv.ranking\_

drfefn = []

for i in range(len(drfecv.support\_)):

if drfecv.support\_[i] == True:

print(Xd.keys().tolist()[i] + " "+ str(i))

drfefn.append(Xd.keys().tolist()[i])

Xd\_train\_rfe, Xd\_test\_rfe, yd\_train\_rfe, yd\_test\_rfe = train\_test\_split(Xd\_rfe, yd, test\_size=0.2, random\_state=42)

gnb\_drfe = GaussianNB()

gnb\_drfe.fit(Xd\_train\_rfe, yd\_train\_rfe)

gnb\_p\_drfe = gnb\_drfe.predict(Xd\_test\_rfe)

print("Gaussian Naive Bayes Classifier report for RFECV: \n\n", classification\_report(yd\_test\_rfe, gnb\_p\_drfe))

sk10 = SelectKBest(chi2, k=10)

sk10.fit\_transform(X, y)

X\_k10 = X.iloc[:,sk10.get\_support(indices=True)]

X\_p.keys()

sk10f = SelectKBest(f\_classif, k=10)

sk10f.fit\_transform(X, y)

X\_k10f = X.iloc[:,sk10f.get\_support(indices=True)]

X\_k10f.keys()

sk10m = SelectKBest(mutual\_info\_classif, k=10)

sk10m.fit\_transform(X, y)

X\_k10m = X.iloc[:,sk10m.get\_support(indices=True)]

X\_k10m.keys()

X\_train\_k10, X\_test\_k10, y\_train\_k10, y\_test\_k10 = train\_test\_split(X\_k10, y, test\_size=0.2, random\_state=42)

X\_train\_k10f, X\_test\_k10f, y\_train\_k10f, y\_test\_k10f = train\_test\_split(X\_k10f, y, test\_size=0.2, random\_state=42)

X\_train\_k10m, X\_test\_k10m, y\_train\_k10m, y\_test\_k10m = train\_test\_split(X\_k10m, y, test\_size=0.2, random\_state=42)

gnb1,gnb2,gnb3 = GaussianNB(),GaussianNB(),GaussianNB()

gnb1.fit(X\_train\_k10, y\_train\_k10)

gnb2.fit(X\_train\_k10f, y\_train\_k10f)

gnb3.fit(X\_train\_k10m, y\_train\_k10m)

gnb\_k10 = gnb1.predict(X\_test\_k10)

print("Gaussian Naive Bayes Classifier report for 10-best with chi square: \n\n", classification\_report(y\_test\_k10, gnb\_k10))

gnb\_k10f = gnb2.predict(X\_test\_k10f)

print("Gaussian Naive Bayes Classifier report for 10-best with f-value: \n\n", classification\_report(y\_test\_k10f, gnb\_k10f))

gnb\_k10m = gnb3.predict(X\_test\_k10m)

print("Gaussian Naive Bayes Classifier report for 10-best with mutual info: \n\n", classification\_report(y\_test\_k10m, gnb\_k10m))

sp1 = SelectPercentile(chi2, percentile=1)

sp1.fit\_transform(X, y)

X\_1p = X.iloc[:,sp1.get\_support(indices=True)]

X\_1p.keys()

sp1f = SelectPercentile(f\_classif, percentile=1)

sp1f.fit\_transform(X, y)

X\_1pf = X.iloc[:,sp1f.get\_support(indices=True)]

X\_1pf.keys()

sp1m = SelectPercentile(mutual\_info\_classif, percentile=1)

sp1m.fit\_transform(X, y)

X\_1pm = X.iloc[:,sp1m.get\_support(indices=True)]

X\_1pm.keys()

X\_train\_1p, X\_test\_1p, y\_train\_1p, y\_test\_1p = train\_test\_split(X\_1p, y, test\_size=0.2, random\_state=42)

X\_train\_1pf, X\_test\_1pf, y\_train\_1pf, y\_test\_1pf = train\_test\_split(X\_1pf, y, test\_size=0.2, random\_state=42)

X\_train\_1pm, X\_test\_1pm, y\_train\_1pm, y\_test\_1pm = train\_test\_split(X\_1pm, y, test\_size=0.2, random\_state=42)

gnb4,gnb5,gnb6 = GaussianNB(),GaussianNB(),GaussianNB()

gnb4.fit(X\_train\_1p, y\_train\_1p)

gnb5.fit(X\_train\_1pf, y\_train\_1pf)

gnb6.fit(X\_train\_1pm, y\_train\_1pm)

gnb\_1p = gnb4.predict(X\_test\_1p)

print("Gaussian Naive Bayes Classifier report for best %1 with chi square: \n\n", classification\_report(y\_test\_1p, gnb\_1p))

gnb\_1pf = gnb5.predict(X\_test\_1pf)

print("Gaussian Naive Bayes Classifier report for best %1 with f-value: \n\n", classification\_report(y\_test\_1pf, gnb\_1pf))

gnb\_1pm = gnb6.predict(X\_test\_1pm)

print("Gaussian Naive Bayes Classifier report for best %1 with mutual info: \n\n", classification\_report(y\_test\_1pm, gnb\_1pm))

sp01f = SelectPercentile(f\_classif, percentile=0.1)

sp01f.fit\_transform(X, y)

X\_01pf = X.iloc[:,sp01f.get\_support(indices=True)]

X\_01pf.keys()

X\_train\_01pf, X\_test\_01pf, y\_train\_01pf, y\_test\_01pf = train\_test\_split(X\_01pf, y, test\_size=0.2, random\_state=42)

gnb7 = GaussianNB()

gnb7.fit(X\_train\_01pf, y\_train\_01pf)

gnb\_p\_01pf = gnb7.predict(X\_test\_01pf)

print("Gaussian Naive Bayes Classifier report for best %0.1 with f-value: \n\n", classification\_report(y\_test\_01pf, gnb\_p\_01pf))

fpr = SelectFpr(chi2, alpha=0.79)

fpr.fit\_transform(X, y)

fprf = SelectFpr(f\_classif, alpha=0.001)

fprf.fit\_transform(X, y)

#fprm = SelectFpr(mutual\_info\_classif, alpha=1)

#fprm.fit\_transform(X, y)

X\_fpr = X.iloc[:,fpr.get\_support(indices=True)]

X\_fprf = X.iloc[:,fprf.get\_support(indices=True)]

#X\_fprm = X.iloc[:,fprm.get\_support(indices=True)]

X\_fpr.keys()

X\_fprf.keys()

X\_train\_fpr, X\_test\_fpr, y\_train\_fpr, y\_test\_fpr = train\_test\_split(X\_fpr, y, test\_size=0.2, random\_state=42)

X\_train\_fprf, X\_test\_fprf, y\_train\_fprf, y\_test\_fprf = train\_test\_split(X\_fprf, y, test\_size=0.2, random\_state=42)

gnb\_fpr,gnb\_fprf = GaussianNB(),GaussianNB()

gnb\_fpr.fit(X\_train\_fpr, y\_train\_fpr)

gnb\_fprf.fit(X\_train\_fprf, y\_train\_fprf)

gnb\_p\_fpr = gnb\_fpr.predict(X\_test\_fpr)

print("Gaussian Naive Bayes Classifier report for FPR chi square: \n\n", classification\_report(y\_test\_fpr, gnb\_p\_fpr))

gnb\_p\_fprf = gnb\_fprf.predict(X\_test\_fprf)

print("Gaussian Naive Bayes Classifier report for FPR f-value: \n\n", classification\_report(y\_test\_fprf, gnb\_p\_fprf))

fdr = SelectFdr(chi2, alpha=0.99)

fdr.fit\_transform(X, y)

fdrf = SelectFdr(f\_classif, alpha=0.11)

fdrf.fit\_transform(X, y)

#fdrm = SelectFdr(mutual\_info\_classif, alpha=1)

#fdrm.fit\_transform(X, y)

X\_fdr = X.iloc[:,fdr.get\_support(indices=True)]

X\_fdrf = X.iloc[:,fdrf.get\_support(indices=True)]

#X\_fdrm = X.iloc[:,fdrm.get\_support(indices=True)]

X\_fdr.keys()

X\_fdrf.keys()

X\_train\_fdrf, X\_test\_fdrf, y\_train\_fdrf, y\_test\_fdrf = train\_test\_split(X\_fdrf, y, test\_size=0.2, random\_state=42)

gnb\_fdrf = GaussianNB()

gnb\_fdrf.fit(X\_train\_fdrf, y\_train\_fdrf)

gnb\_p\_fdrf = gnb\_fdrf.predict(X\_test\_fdrf)

print("Gaussian Naive Bayes Classifier report for best FDR f-value: \n\n", classification\_report(y\_test\_fdrf, gnb\_p\_fdrf))

fwe = SelectFwe(chi2, alpha=0.99)

fwe.fit\_transform(X, y)

fwef = SelectFwe(f\_classif, alpha=0.99)

fwef.fit\_transform(X, y)

#fwem = SelectFwe(mutual\_info\_classif, alpha=1)

#fwem.fit\_transform(X, y)

X\_fwe = X.iloc[:,fwe.get\_support(indices=True)]

X\_fwef = X.iloc[:,fwef.get\_support(indices=True)]

#X\_fwem = X.iloc[:,fwem.get\_support(indices=True)]

X\_fwe.keys()

X\_fwef.keys()

X\_train\_fwef, X\_test\_fwef, y\_train\_fwef, y\_test\_fwef = train\_test\_split(X\_fwef, y, test\_size=0.2, random\_state=42)

gnb\_fwef = GaussianNB()

gnb\_fwef.fit(X\_train\_fwef, y\_train\_fwef)

gnb\_p\_fwef = gnb\_fwef.predict(X\_test\_fwef)

print("Gaussian Naive Bayes Classifier report for best FWE f-value: \n\n", classification\_report(y\_test\_fwef, gnb\_p\_fwef))

gnb = GaussianNB()

rfecv = RFECV(

estimator=svc,

step=1,

cv=StratifiedKFold(2),

scoring="accuracy",

min\_features\_to\_select=2,

)

rfecv.fit(X, y)

print("Optimal number of features : %d" % rfecv.n\_features\_)

plt.figure()

plt.xlabel("Number of features selected")

plt.ylabel("Cross validation score (accuracy)")

plt.plot(

range(2, len(rfecv.grid\_scores\_) + 2),

rfecv.grid\_scores\_,

)

plt.show()

rfecv.ranking\_

rfegn = []

for i in range(len(rfecv.support\_)):

if rfecv.support\_[i] == True:

print(X.keys().tolist()[i] + " "+ str(i))

rfegn.append(X.keys().tolist()[i])

X\_train\_rfe, X\_test\_rfe, y\_train\_rfe, y\_test\_rfe = train\_test\_split(X\_rfe, y, test\_size=0.2, random\_state=42)

gnb\_rfe = GaussianNB()

gnb\_rfe.fit(X\_train\_rfe, y\_train\_rfe)

gnb\_p\_rfe = gnb\_rfe.predict(X\_test\_rfe)

print("Gaussian Naive Bayes Classifier report for RFECV: \n\n", classification\_report(y\_test\_rfe, gnb\_p\_rfe))

sfm = SelectFromModel(estimator=LinearSVC(C=0.1, penalty="l1", dual=False),threshold=0.1).fit(X, y)

X\_sfm = X.iloc[:,sfm.get\_support(indices=True)]

X\_sfm

X\_train\_sfm, X\_test\_sfm, y\_train\_sfm, y\_test\_sfm = train\_test\_split(X\_sfm, y, test\_size=0.2, random\_state=42)

gnb8 = GaussianNB()

gnb8.fit(X\_train\_sfm, y\_train\_sfm)

gnb\_p\_sfm = gnb8.predict(X\_test\_sfm)

print("Gaussian Naive Bayes Classifier report for SFM L1: \n\n", classification\_report(y\_test\_sfm, gnb\_p\_sfm))

sfmt = SelectFromModel(estimator=RandomForestClassifier(),threshold=0.015).fit(X, y)

X\_sfmt = X.iloc[:,sfmt.get\_support(indices=True)]

X\_sfmt.keys()

X\_train\_sfmt, X\_test\_sfmt, y\_train\_sfmt, y\_test\_sfmt = train\_test\_split(X\_sfmt, y, test\_size=0.2, random\_state=42)

gnb\_sfmt = GaussianNB()

gnb\_sfmt.fit(X\_train\_sfmt, y\_train\_sfmt)

gnb\_p\_sfmt = gnb\_sfmt.predict(X\_test\_sfmt)

print("Gaussian Naive Bayes Classifier report for SFM Tree: \n\n", classification\_report(y\_test\_sfmt, gnb\_p\_sfmt))

keys = [X\_01pf, X\_1p, X\_1pf, X\_1pm, X\_fdr, X\_fdrf, X\_fpr, X\_fprf, X\_fwe, X\_fwef, X\_k10, X\_k10f, X\_k10m, X\_rfe, X\_sfm, X\_sfmt]

nkeys = []

for i in keys:

for j in i.keys().tolist():

nkeys.append(j)

from collections import Counter

c = Counter(nkeys)

gns = np.asarray(c.most\_common(10)).T[0].tolist()

print(gns)

c.most\_common(15)

Xn\_train, Xn\_test, yn\_train, yn\_test = train\_test\_split(Xn, y, test\_size=0.2, random\_state=42)

ngnb = GaussianNB()

ngnb.fit(Xn\_train, y\_train)

ngnb\_p = ngnb.predict(Xn\_test)

print("Gaussian Naive Bayes Classifier report for selected genes: \n\n", classification\_report(yn\_test, ngnb\_p))

clf = DecisionTreeClassifier(random\_state=0)

cross\_val\_score(clf, X\_train, y\_train, cv=10)

nclf = DecisionTreeClassifier(random\_state=0)

cross\_val\_score(nclf, Xn\_train, yn\_train, cv=10)

gnb = GaussianNB()

gnb.fit(X\_train, y\_train)

gnb\_p = gnb.predict(X\_test)

print("Gaussian Naive Bayes Classifier report: \n\n", classification\_report(y\_test, gnb\_p))

ngnb = GaussianNB()

ngnb.fit(Xn\_train, yn\_train)

ngnb\_p = ngnb.predict(Xn\_test)

print("Gaussian Naive Bayes Classifier report for selected genes: \n\n", classification\_report(yn\_test, ngnb\_p))

knn = KNeighborsClassifier(13)

knn.fit(X\_train, y\_train)

knn\_p = knn.predict(X\_test)

print("K-Nearest Neighbor Classifier report: \n\n", classification\_report(y\_test, knn\_p))

nknn = KNeighborsClassifier(6)

nknn.fit(Xn\_train, yn\_train)

nknn\_p = nknn.predict(Xn\_test)

print("K-Nearest Neighbor Classifier report for selected genes: \n\n", classification\_report(yn\_test, nknn\_p))

lsvc = LinearSVC(C=0.1, penalty="l1", dual=False)

lsvc.fit(X\_train, y\_train)

lsvc\_p = lsvc.predict(X\_test)

print("Linear SVC report: \n\n", classification\_report(y\_test, lsvc\_p))

nlsvc = LinearSVC(C=0.25, penalty="l1", dual=False)

nlsvc.fit(Xn\_train, yn\_train)

nlsvc\_p = nlsvc.predict(Xn\_test)

print("Linear SVC report for selected genes: \n\n", classification\_report(yn\_test, nlsvc\_p))

rsvc = SVC(gamma=0.15, C=1)

rsvc.fit(X\_train, y\_train)

rsvc\_p = rsvc.predict(X\_test)

print("RBF kernel SVC report: \n\n", classification\_report(y\_test, rsvc\_p))

nrsvc = SVC(gamma=8.2, C=1)

nrsvc.fit(Xn\_train, yn\_train)

nrsvc\_p = nrsvc.predict(Xn\_test)

print("RBF kernel SVC report for selected genes: \n\n", classification\_report(yn\_test, nrsvc\_p))

nn = Sequential()

nn.add(Dense(16, activation = 'relu', input\_dim=10))

nn.add(Dense(8, activation = 'relu'))

nn.add(Dense(1, activation = 'sigmoid'))

nn.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

nn.fit(Xn\_train, yn\_train, batch\_size = 10, epochs = 1000)

yn\_pred = nn.predict(Xn\_test)

yn\_pred = (yn\_pred > 0.5)

print ("Simple DNN report for selected genes:\n\n",classification\_report(yn\_test, yn\_pred))

nn = Sequential()

nn.add(Dense(16, activation = 'relu', input\_dim=10))

nn.add(Dense(8, activation = 'relu'))

nn.add(Dense(4, activation = 'relu'))

nn.add(Dense(1, activation = 'sigmoid'))

nn.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

nn.fit(Xn\_train, yn\_train, batch\_size = 10, epochs = 1000, verbose=0)

yn\_pred = nn.predict(Xn\_test)

yn\_pred = (yn\_pred > 0.5)

print ("Simple DNN report for selected genes:\n\n",classification\_report(yn\_test, yn\_pred))

nn22 = Sequential()

nn22.add(Dense(16, activation = 'relu', input\_dim=10))

nn22.add(layers.Dropout(0.2))

nn22.add(Dense(8, activation = 'tanh'))

nn22.add(Dense(1, activation = 'sigmoid'))

nn22.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

nn22.fit(Xn\_train, yn\_train, batch\_size = 10, epochs = 1000, verbose=0)

yn\_pred = nn22.predict(Xn\_test)

yn\_pred = (yn\_pred > 0.5)

print ("Simple DNN report for selected genes:\n\n",classification\_report(yn\_test, yn\_pred))

lr = LogisticRegression(random\_state=0).fit(X\_train, y\_train)

lr\_p = lr.predict(X\_test)

print("Logistic Regression report: \n\n", classification\_report(y\_test, lr\_p))

nlr = LogisticRegression(random\_state=0).fit(Xn\_train, yn\_train)

nlr\_p = nlr.predict(Xn\_test)

print("Logistic Regression report for selected genes: \n\n", classification\_report(yn\_test, nlr\_p))

rfc = RandomForestClassifier().fit(X\_train, y\_train)

rfc\_p = rfc.predict(X\_test)

print("Random Forest Classifier report: \n\n", classification\_report(y\_test, rfc\_p))

nrfc = RandomForestClassifier().fit(Xn\_train, yn\_train)

nrfc\_p = nrfc.predict(Xn\_test)

print("Random Forest Classifier report for selected genes: \n\n", classification\_report(yn\_test, nrfc\_p))

ada = AdaBoostClassifier().fit(X\_train, y\_train)

ada\_p = ada.predict(X\_test)

print("AdaBoost Classifier report: \n\n", classification\_report(y\_test, ada\_p))

nada = AdaBoostClassifier().fit(Xn\_train, yn\_train)

nada\_p = nada.predict(Xn\_test)

print("AdaBoost Classifier report for selected genes: \n\n", classification\_report(yn\_test, nada\_p))

qda = QuadraticDiscriminantAnalysis().fit(X\_train, y\_train)

qda\_p = qda.predict(X\_test)

print("QDA report: \n\n", classification\_report(y\_test, qda\_p))

nqda = QuadraticDiscriminantAnalysis().fit(Xn\_train, yn\_train)

nqda\_p = nqda.predict(Xn\_test)

print("QDA report for selected genes: \n\n", classification\_report(yn\_test, nqda\_p))