MICE BEHAVIOR PREDICTION USING PROTEIN EXPRESSIONS

BHAVYA SHREE ARCOT MURUGESHAN

CWID-20016312

Department of Mathematical Sciences, Stevens Institute of Technology, Hoboken, NJ

Project Supervisor: Dr. Hadi Safari Katesari

ABSTRACT

Statistical analysis is a research tool used to investigate trends, patterns, and relationships using quantitative data. The goal is to use the protein expression data to predict mice's behavior by applying various statistical methods like ANOVA, categorical data analysis, regression etc,. The accuracy metric used in the project gives us a 100% accuracy while categorizing the data.

CHAPTER 1: INTRODUCTION

The main focus of this project is to apply various statistical methods intrtoduced in this course. Using the dataset on mice protein expression, I will be first analyzing the data to filter out null values and find the correlation and dependency between the various categories in the data. Then I will compare two independent samples using Mann-Whitney test , perform Kruskal- Wallis ANOVA test, Nemenyi's test, Chi squared test of independence to understand the statistical properties of the dataset. Later I will be implementing Forward and Backward elimination methods to select the best features from the data to fit a logistic, Ridge, and polynomial regression models. The ultimate goal will be to predict the mouse behavior(expression) based on the protein expressions.

CHAPTER 2: METHODOLOGY

Mann-Whitney Test:

The Mann-Whitney test is used for comparing differences between two independent groups. It tests whether the two groups come from same population and does not assume any specific distribution for calculating test statistics and p values.

Kruskal-Walis Test:

The Kruskal Walis Test is a generalization of Mann-Whitney test that observations are assumed to be independent but no particular distribution from, such as normal, is assumed. The observations are pooled together and ranked.

Nemenyi Test:

The Nemenyi test is a post-hoc test intended to find the groups of data that differ after a global statistical test has rejected the null hypothesis that the performance of the comparisons on the groups of data is similar. The test makes pair-wise tests of performance.

Chi Squared Test of Independence:

A Chi-Square Test of Independence is used to determine whether or not there is a significant association between two categorical variables.

Forward and Backward Step-wise Selection:

Forward stepwise selection begins with a model containing no predictors, and then adds predictors to the model, one-at-a-time, until all of the predictors are in the model. Backward stepwise selection begins with the full least squares model containing all p predictors, and then iteratively removes the least useful predictor, one-at-a-time.

Logistic Regression:

Logistic regression is a statistical technique (model) which by using a logistic function models a binary response (dependent) variable, such as tail/head, pass/fail, win/lose, and so on. We label these binary data with 0/1.

Ridge Regression:

Ridge regression is a method of estimating the coefficients of multiple-regression models in scenarios where the independent variables are highly correlated.

Polynomial Regression:

polynomial regression is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modelled as an nth degree polynomial in x. Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y.

CHAPTER 3: DATA DESCRIPTION

The dataset consists of the expression levels of 77 protein modifications that produced detectable signals in the nuclear fraction of cortex. There are 38 control mice and 34 trisomic mice (Down syndrome), for a total of 72 mice. In the experiments, 15 measurements were registered of each protein per mouse. Therefore, for control mice, there are 38x15, or 570 measurements, and for trisomic mice, there are 34x15, or 510 measurements. The dataset contains a total of 1080 measurements per protein. Each measurement is an independent sample per mouse.

The eight classes of mice are described based on features such as genotype, behavior and treatment. According to genotype, mice can be control or trisomic. According to behavior, some mice have been stimulated to learn (context-shock) and others have not (shock-

context) and in order to assess the effect of the drug memantine in recovering the ability to learn in trisomic mice, some mice have been injected with the drug and others have not.

We observe that there are 77 protein expressions generated by the treatment given to the mice. Out of these we will choose the best 20 proteins and predict if the mice have been stimulated the ability to learn or not.

LOADING THE REQUIRED LIBRARIES

In [31]:
```
import pandas as pd  # library used for data manipulation and analysis
import numpy as np  # library used for working with arrays
import matplotlib.pyplot as plt  # library for plots and visualizations
import seaborn as sns  # library for visualizations
%matplotlib inline
from sklearn.preprocessing import LabelEncoder
from scipy.stats import kruskal
from scipy.stats import mannwhitneyu
from scipy.stats import chisquare
from statsmodels.stats.multicomp import pairwise_tukeyhsd
from sklearn.feature_selection import SelectKBest, chi2, f_classif
from scipy.stats import chi2, chi2_contingency
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
import joblib
import sys
import scipy.stats as stats  # this library contains a large number of probability
from sklearn.linear_model import LogisticRegression, LinearRegression, Ridge
from sklearn.metrics import r2_score
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.metrics import mean_absolute_error,mean_squared_error
from pingouin import multivariate_normality
from sklearn.model_selection import KFold, cross_val_score
import scikit_posthocs as sp
```

Import the dataset and obtain the general statistics. Checking for the null values and drop them.

In [32]:
```
mice_df = pd.read_csv("Data_Cortex_Nuclear.csv")
mice_df.head(10)
```

Out[32]:

	MouseID	DYRK1A_N	ITSN1_N	BDNF_N	NR1_N	NR2A_N	pAKT_N	pBRAF_N	pCAMKII_N
0	309_1	0.503644	0.747193	0.430175	2.816329	5.990152	0.218830	0.177565	2.373744
1	309_2	0.514617	0.689064	0.411770	2.789514	5.685038	0.211636	0.172817	2.292150
2	309_3	0.509183	0.730247	0.418309	2.687201	5.622059	0.209011	0.175722	2.283337
3	309_4	0.442107	0.617076	0.358626	2.466947	4.979503	0.222886	0.176463	2.152301
4	309_5	0.434940	0.617430	0.358802	2.365785	4.718679	0.213106	0.173627	2.134014
5	309_6	0.447506	0.628176	0.367388	2.385939	4.807635	0.218578	0.176233	2.141282
6	309_7	0.428033	0.573696	0.342709	2.334224	4.473130	0.225173	0.184004	2.012414
7	309_8	0.416923	0.564036	0.327703	2.260135	4.268735	0.214834	0.179668	2.007985
8	309_9	0.386311	0.538428	0.317720	2.125725	4.063950	0.207222	0.167778	1.861514
9	309_10	0.380827	0.499294	0.362462	2.096266	3.598587	0.227649	0.188093	1.717861

10 rows × 82 columns

In [33]:
```
mice_df.describe()
```

Out[33]:

	DYRK1A_N	ITSN1_N	BDNF_N	NR1_N	NR2A_N	pAKT_N	pBRAF_N
count	1077.000000	1077.000000	1077.000000	1077.000000	1077.000000	1077.000000	1077.000000
mean	0.425810	0.617102	0.319088	2.297269	3.843934	0.233168	0.181846
std	0.249362	0.251640	0.049383	0.347293	0.933100	0.041634	0.027042
min	0.145327	0.245359	0.115181	1.330831	1.737540	0.063236	0.064043
25%	0.288121	0.473361	0.287444	2.057411	3.155678	0.205755	0.164595
50%	0.366378	0.565782	0.316564	2.296546	3.760855	0.231177	0.182302
75%	0.487711	0.698032	0.348197	2.528481	4.440011	0.257261	0.197418
max	2.516367	2.602662	0.497160	3.757641	8.482553	0.539050	0.317066

8 rows × 77 columns

In [34]:
```
mice_df = mice_df.dropna()
mice_df.shape
```

Out[34]:
```
(552, 82)
```

Converting the string data columns - behavior, genotype, treatment and class to numeric and assigning binary codes to each of these categories. These numeric values will help us find the correlation between all the columns in the dataset.

In [35]:
```
mice_df['Behavior_Code'] = mice_df['Behavior'].astype('category').cat.codes
mice_df['Genotype_Code'] = mice_df['Genotype'].astype('category').cat.codes
mice_df['Treatment_Code'] = mice_df['Treatment'].astype('category').cat.codes
mice_df['class_Code'] = mice_df['class'].astype('category').cat.codes
mice_df
```

Out[35]:

	MouseID	DYRK1A_N	ITSN1_N	BDNF_N	NR1_N	NR2A_N	pAKT_N	pBRAF_N	pCAMKII
75	3415_1	0.649781	0.828696	0.405862	2.921435	5.167979	0.207174	0.176640	3.7280
76	3415_2	0.616481	0.841974	0.388584	2.862575	5.194163	0.223433	0.167725	3.6482
77	3415_3	0.637424	0.852882	0.400561	2.968155	5.350820	0.208790	0.173261	3.8145
78	3415_4	0.576815	0.755390	0.348346	2.624901	4.727509	0.205892	0.161192	3.7785
79	3415_5	0.542545	0.757917	0.350051	2.634509	4.735602	0.210526	0.165671	3.8719
...	...	...	...	...	...	...	...	...	
1045	3525_11	0.382149	0.595792	0.399101	2.527574	4.629493	0.317810	0.213235	6.2195
1046	3525_12	0.387947	0.586510	0.431650	2.527041	4.781082	0.316903	0.222111	6.3077
1047	3525_13	0.343695	0.562547	0.361243	2.412885	3.949361	0.330158	0.225620	6.0601
1048	3525_14	0.346594	0.551467	0.390353	2.445052	4.236201	0.343610	0.229239	6.0233
1049	3525_15	0.365907	0.552861	0.386275	2.473812	4.347236	0.353783	0.232784	6.0853

552 rows × 86 columns

In [36]:
```
mice_corr_matrix=mice_df.corr()
mice_corr_matrix
```

Out[36]:

	DYRK1A_N	ITSN1_N	BDNF_N	NR1_N	NR2A_N	pAKT_N	pBRAF_N	pCA
DYRK1A_N	1.000000	0.922362	0.382784	0.254820	0.288369	-0.127104	-0.026650	-0
ITSN1_N	0.922362	1.000000	0.535981	0.430789	0.428279	-0.046645	0.031511	-0
BDNF_N	0.382784	0.535981	1.000000	0.832862	0.747559	0.437506	0.499886	0
NR1_N	0.254820	0.430789	0.832862	1.000000	0.899797	0.360170	0.398750	0
NR2A_N	0.288369	0.428279	0.747559	0.899797	1.000000	0.269293	0.254087	0
...	...	...	...	...	...	...	...	
CaNA_N	0.463305	0.526255	0.341437	0.258619	0.183099	-0.163782	-0.093022	-0
Behavior_Code	-0.619505	-0.550178	-0.106760	-0.012038	-0.038808	0.429839	0.335645	0
Genotype_Code	0.398890	0.470320	0.075261	-0.116341	-0.100587	0.150839	0.067259	0
Treatment_Code	-0.246033	-0.158021	0.103583	0.090734	0.017308	-0.113461	-0.137639	-0
class_Code	0.024673	0.142265	0.043714	-0.091457	-0.106153	0.309994	0.184463	0

81 rows × 81 columns

CHAPTER 4: ANALYSIS AND RESULTS

4.1 COMPARING TWO SAMPLES

We first check if the given data is normal or not. We will use multivariate normality to check if all the columns in out dataset are normally distributed or not.

In [37]:
```
#Checking for Normality

mice_copy_df = mice_df.copy()
nonNumericColumns = ['MouseID', 'Genotype', 'Treatment', 'Behavior', 'Behavior_Cod
mice_copy_df = mice_copy_df.drop(nonNumericColumns, axis=1)
multivariate_normality(mice_copy_df, alpha=.05)
```

Out[37]:
```
HZResults(hz=2208, pval=nan, normal=False)
```

Therefore, the data is not Normal.

Since our samples are independent and we have an unknown distribution, we can use the Mann-Whitney test to compare two samples, behavior code and the protein expression CaNa_N of our dataset.

In [38]:
```
#Comparing two Independent Samples

stat, p = mannwhitneyu(mice_df.Behavior_Code, mice_df.CaNA_N)
print('Statistics=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
    print('H0 : Behavior_Code and CaNA_N are drawn from same distribution (fail to
else:
    print('H1 : Behavior_Code and CaNA_N are drawn from different distribution (re
```

```
Statistics=24831.000, p=0.000
H1 : Behavior_Code and CaNA_N are drawn from different distribution (reject H0)
```

Since p < 0.05 we reject Ho. Therefore, we conclude that the columns protein type CaNa_N and behavior code follow different statistical distributions.

4.2 ANALYSIS OF VARIANCE - ONE WAY

We shall now analyze the variance between all the 77 protein groups and the behavior of the mice. Since there are more than two groups to analyze, we will use Kruskal-Wallis test

In [39]:
```
#ANOVA-KW Test

stat, p = kruskal(mice_df.Behavior_Code, mice_df.DYRK1A_N, mice_df.ITSN1_N, mice_d
)
print('Statistics=%.3f, p=%.3f' % (stat, p))
alpha = 0.05
if p > alpha:
    print('H0 : Means are equal across above groups of protein (fail to reject H0)
else:
    print('H1 : Means are different across above groups of protein (reject H0)')
```

```
Statistics=39358.152, p=0.000
H1 : Means are different across above groups of protein (reject H0)
```

Since p < 0.05 we reject Ho. Therefore, we conclude that the means are different between all these categories analyzed.

We shall now find which two groups have different means using Nemenyi's test. We will analyze each of the protein expression paired with the behavior code to find which exact pair is different.

In [57]:
```
#Nemenyi Test

# columnArr = ["DYRK1A_N", "ITSN1_N", "BDNF_N", "NR1_N", "NR2A_N"]
columnArr = []
for column in mice_df.columns:
    if column not in nonNumericColumns and column != 'Behavior_Code':
# for column in columnArr:
        stats = sp.posthoc_nemenyi(a=mice_df, val_col=column, group_col="Behavior_
        p = stats[0][1]
        print("\nColumn Name - " + column)
        print('p-value - ', p)
        if p > alpha:
            print('fail to reject')
        else:
            print('reject')
```

```
Column Name - DYRK1A_N
p-value -  3.0045657183785035e-55
reject

Column Name - ITSN1_N
p-value -  1.845332232113005e-39
reject

Column Name - BDNF_N
p-value -  0.04694600791671962
reject

Column Name - NR1_N
p-value -  0.7199031635060402
reject

Column Name - NR2A_N
p-value -  0.4125357120146196
reject

Column Name - pAKT_N
p-value -  2.860133038093946e-22
reject

Column Name - pBRAF_N
p-value -  6.034139033046804e-15
reject

Column Name - pCAMKII_N
p-value -  1.5053058897649213e-23
reject

Column Name - pCREB_N
p-value -  1.4034077411846187e-06
reject

Column Name - pELK_N
p-value -  1.02750582699361e-09
reject

Column Name - pERK_N
p-value -  1.2701714810465476e-76
reject

Column Name - pJNK_N
p-value -  1.292382015805957e-14
reject

Column Name - PKCA_N
p-value -  1.8346044615879238e-14
reject

Column Name - pMEK_N
p-value -  1.8680057435894033e-20
reject

Column Name - pNR1_N
p-value -  0.005075203207212833
reject

Column Name - pNR2A_N
p-value -  5.5781185071950025e-22
reject
```

```
Column Name - pNR2B_N
p-value -  0.0046085405568562655
reject

Column Name - pPKCAB_N
p-value -  1.3045297014286467e-47
reject

Column Name - pRSK_N
p-value -  0.12127786297362632
reject

Column Name - AKT_N
p-value -  4.211374083687951e-25
reject

Column Name - BRAF_N
p-value -  1.907772971221085e-49
reject

Column Name - CAMKII_N
p-value -  5.855092550019354e-12
reject

Column Name - CREB_N
p-value -  9.507963040991152e-09
reject

Column Name - ELK_N
p-value -  0.4652314595514173
reject

Column Name - ERK_N
p-value -  0.0006078618575143319
reject

Column Name - GSK3B_N
p-value -  6.801115955978333e-40
reject

Column Name - JNK_N
p-value -  0.9525202557721986
reject

Column Name - MEK_N
p-value -  0.0013854925452430314
reject

Column Name - TRKA_N
p-value -  0.01040963257636983
reject

Column Name - RSK_N
p-value -  0.0001327064632281158
reject

Column Name - APP_N
p-value -  0.00019244718530457135
reject

Column Name - Bcatenin_N
p-value -  0.03452538637600175
reject
```

```
Column Name - SOD1_N
p-value -  1.0890478666733965e-91
reject

Column Name - MTOR_N
p-value -  2.663434027969689e-20
reject

Column Name - P38_N
p-value -  1.9012749364353567e-56
reject

Column Name - pMTOR_N
p-value -  1.9075589853502443e-31
reject

Column Name - DSCR1_N
p-value -  7.989872348104197e-10
reject

Column Name - AMPKA_N
p-value -  7.259684911963541e-07
reject

Column Name - NR2B_N
p-value -  1.709698440019489e-22
reject

Column Name - pNUMB_N
p-value -  9.919895780294397e-19
reject

Column Name - RAPTOR_N
p-value -  2.1736545145653152e-18
reject

Column Name - TIAM1_N
p-value -  6.061063559513842e-05
reject

Column Name - pP70S6_N
p-value -  1.913897571605295e-09
reject

Column Name - NUMB_N
p-value -  3.1907702195568846e-11
reject

Column Name - P70S6_N
p-value -  0.17472138370174747
reject

Column Name - pGSK3B_N
p-value -  9.611669567159576e-24
reject

Column Name - pPKCG_N
p-value -  1.9012889001086966e-06
reject

Column Name - CDK5_N
p-value -  2.5499244338770645e-09
reject
```

```
Column Name - S6_N
p-value -  5.807196606617656e-23
reject

Column Name - ADARB1_N
p-value -  4.5325823418870364e-05
reject

Column Name - AcetylH3K9_N
p-value -  0.42049935986250786
reject

Column Name - RRP1_N
p-value -  0.7131219981174703
reject

Column Name - BAX_N
p-value -  0.59204163712848
reject

Column Name - ARC_N
p-value -  3.067068600260275e-65
reject

Column Name - ERBB4_N
p-value -  1.781505414810313e-18
reject

Column Name - nNOS_N
p-value -  3.0075549726964385e-27
reject

Column Name - Tau_N
p-value -  0.08402556354064315
reject

Column Name - GFAP_N
p-value -  1.3855388344165918e-10
reject

Column Name - GluR3_N
p-value -  0.07637552567877609
reject

Column Name - GluR4_N
p-value -  0.8219127379245709
reject

Column Name - IL1B_N
p-value -  4.138058085448347e-51
reject

Column Name - P3525_N
p-value -  0.6866208006227865
reject

Column Name - pCASP9_N
p-value -  1.3182313385741716e-07
reject

Column Name - PSD95_N
p-value -  1.6461837558963803e-16
reject
```

```
Column Name - SNCA_N
p-value -  6.60322019394497e-51
reject

Column Name - Ubiquitin_N
p-value -  9.86883242467711e-50
reject

Column Name - pGSK3B_Tyr216_N
p-value -  2.2444084710822514e-09
reject

Column Name - SHH_N
p-value -  2.6880538832424318e-12
reject

Column Name - BAD_N
p-value -  5.096653232081863e-07
reject

Column Name - BCL2_N
p-value -  7.400969667005206e-18
reject

Column Name - pS6_N
p-value -  3.067068600260275e-65
reject

Column Name - pCFOS_N
p-value -  8.250859129155023e-05
reject

Column Name - SYP_N
p-value -  0.07124881434906748
reject

Column Name - H3AcK18_N
p-value -  8.267052032291881e-08
reject

Column Name - EGR1_N
p-value -  1.6468098122834173e-28
reject

Column Name - H3MeK4_N
p-value -  1.803662948068017e-20
reject

Column Name - CaNA_N
p-value -  1.226473120002031e-64
reject

Column Name - Genotype_Code
p-value -  0.046618753600974325
reject

Column Name - Treatment_Code
p-value -  0.5683473994576402
reject

Column Name - class_Code
p-value -  1.9623147867109678e-17
reject
```

Therefore, we observe that all the protein means are different from behavior except the following 14 proteins:

- NR1_N
- NR2A_N
- pNR1_N
- pRSK_N
- ELK_N
- JNK_N
- P70S6_N
- AcetylH3K9_N
- RRP1_N
- BAX_N
- GluR3_N
- GluR4_N
- P3525_N
- SYP_N

4.3 CATEGORICAL DATA ANALYSIS

We check if there is relation between the categorical variable treatement and the output categorical variable behavior.

In [41]:
```
#Categorical Analysis

crosstab = pd.crosstab(mice_df["Behavior"], mice_df["Treatment"])
stat, p, dof, expected = chi2_contingency(crosstab)
print('Statistics=%.3f, p=%.3f, dof=%.3f' % (stat, p, dof))
prob = 0.95
critical_value = chi2.ppf(prob, dof)
print('z95=%.3f' % (critical_value))
if abs(stat) >= critical_value:
    print('Dependent (reject H0)')
else:
    print('Independent (fail to reject H0)')
```

```
Statistics=0.236, p=0.627, dof=1.000
z95=3.841
Independent (fail to reject H0)
```

Since p > 0.05 we fail to reject Ho. Therefore, we see that treatment and behavior are independent i.e., irrespective of what treatment the mice has been given, its behavior expression is random.

4.4 LOGISTIC REGRESSION

We now fit the logistic regressioin model to predict the behavior and classify the mice into the classes - context-shock or shock-context.

In [65]:
```
#Logistic Regression

feature_selected_df = refined_mice_df.iloc[:,[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
x_train, x_test, y_train, y_test = train_test_split(feature_selected_df, refined_m
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)
logistic_reg_model = LogisticRegression(random_state = 0,multi_class='multinomial'
logistic_reg_model.fit(x_train, y_train)
print("R-Squared: ", logistic_reg_model.score(x_test, y_test))
y_pred = logistic_reg_model.predict(x_test)
#accuracy_score(y_test,y_pred)
```

```
mae = mean_absolute_error(y_true=y_test,y_pred=y_pred)
mse = mean_squared_error(y_true=y_test,y_pred=y_pred) #default=True
rmse = mean_squared_error(y_true=y_test,y_pred=y_pred,squared=False)

print("Mean Absolute Error:",mae)
print("Mean Squared error:",mse)
print("Root Mean Squared Error:",rmse)
```

```
R-Squared:  1.0
Mean Absolute Error: 0.0
Mean Squared error: 0.0
Root Mean Squared Error: 0.0
```

Since the R squared = 1 and the errors MAE, MSE, RMSE are 0, we have enough evidence to say that our model fits the data accurately.

We see that there is no difference between the test values and the predicted values in the below output.

In [56]:
```
y_test-y_pred
```

Out[56]:
```
235    0
955    0
261    0
317    0
623    0
      ..
826    0
911    0
226    0
999    0
825    0
Name: Behavior_Code, Length: 111, dtype: int8
```

4.5 RESAMPLING METHODS

Since we have 77 protein expressions based on which the behavior can be predicted, we will try to select the best 20 protein expressions that show heavy weightage in predicting the behavior of the mice. We thus use forward and backward selection to filter out 20 best protein expressions.

In [16]:
```
#Best Feature using Forward Selection

scaler = StandardScaler()
refined_mice_df = mice_df.copy()
if 'Behavior_Code' in nonNumericColumns:
    nonNumericColumns.remove('Behavior_Code')
if 'Behavior' in nonNumericColumns:
    nonNumericColumns.remove('Behavior')
refined_mice_df = refined_mice_df.drop(nonNumericColumns, axis=1)
x_train, x_test, y_train, y_test = train_test_split(refined_mice_df.drop(labels=['
                                                                  , refined_mice_df[
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)
classifier = LogisticRegression(random_state = 0,multi_class='multinomial', solver
classifier = LogisticRegression(random_state = 0)
sfs = SFS(classifier, k_features=20, forward=True, floating=True, verbose=2, scori
sfs = sfs.fit(np.array(x_train), y_train)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  80 out of  80 | elapsed:    2.8s finished

[2022-12-16 19:54:11] Features: 1/20 -- score: 0.9364242873508998[Parallel(n_jobs=
1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  79 out of  79 | elapsed:    3.5s finished

[2022-12-16 19:54:15] Features: 2/20 -- score: 1.0[Parallel(n_jobs=1)]: Using back
end SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  78 out of  78 | elapsed:    3.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s finished

[2022-12-16 19:54:18] Features: 3/20 -- score: 1.0[Parallel(n_jobs=1)]: Using back
end SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  77 out of  77 | elapsed:    3.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s finished

[2022-12-16 19:54:22] Features: 4/20 -- score: 1.0[Parallel(n_jobs=1)]: Using back
end SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  76 out of  76 | elapsed:    3.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.1s finished

[2022-12-16 19:54:26] Features: 5/20 -- score: 1.0[Parallel(n_jobs=1)]: Using back
end SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  75 out of  75 | elapsed:    4.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.2s finished

[2022-12-16 19:54:31] Features: 6/20 -- score: 1.0[Parallel(n_jobs=1)]: Using back
end SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  74 out of  74 | elapsed:    4.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.3s finished

[2022-12-16 19:54:35] Features: 7/20 -- score: 1.0[Parallel(n_jobs=1)]: Using back
end SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  73 out of  73 | elapsed:    4.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    0.4s finished

[2022-12-16 19:54:41] Features: 8/20 -- score: 1.0[Parallel(n_jobs=1)]: Using back
end SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  72 out of  72 | elapsed:    4.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    0.4s finished
```

```
[2022-12-16 19:54:46] Features: 9/20 -- score: 1.0[Parallel(n_jobs=1)]: Using back
end SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  71 out of  71 | elapsed:    4.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    0.5s finished

[2022-12-16 19:54:51] Features: 10/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    4.7s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    0.6s finished

[2022-12-16 19:54:57] Features: 11/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  69 out of  69 | elapsed:    5.2s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  11 out of  11 | elapsed:    0.6s finished

[2022-12-16 19:55:03] Features: 12/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  68 out of  68 | elapsed:    4.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    0.7s finished

[2022-12-16 19:55:09] Features: 13/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  67 out of  67 | elapsed:    4.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  13 out of  13 | elapsed:    0.8s finished

[2022-12-16 19:55:14] Features: 14/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  66 out of  66 | elapsed:    4.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  14 out of  14 | elapsed:    0.9s finished

[2022-12-16 19:55:20] Features: 15/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  65 out of  65 | elapsed:    4.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:    1.0s finished

[2022-12-16 19:55:26] Features: 16/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  64 out of  64 | elapsed:    5.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  16 out of  16 | elapsed:    1.1s finished
```

```
[2022-12-16 19:55:32] Features: 17/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  63 out of  63 | elapsed:    4.4s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  17 out of  17 | elapsed:    1.1s finished

[2022-12-16 19:55:38] Features: 18/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  62 out of  62 | elapsed:    4.3s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  18 out of  18 | elapsed:    1.1s finished

[2022-12-16 19:55:44] Features: 19/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  61 out of  61 | elapsed:    4.1s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  19 out of  19 | elapsed:    1.1s finished

[2022-12-16 19:55:49] Features: 20/20 -- score: 1.0
```

In [44]:
```
sfs.k_feature_idx_
```

Out[44]:
```
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20)
```

In [19]:
```
#Best Feature using Backward Selection

sfs = SFS(classifier, k_features=20, forward=False, floating=False, verbose=2, sco
sfs = sfs.fit(np.array(x_train), y_train)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  80 out of  80 | elapsed:    9.0s finished

[2022-12-16 19:58:18] Features: 79/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  79 out of  79 | elapsed:    9.1s finished

[2022-12-16 19:58:27] Features: 78/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  78 out of  78 | elapsed:    9.7s finished

[2022-12-16 19:58:37] Features: 77/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  77 out of  77 | elapsed:    9.8s finished

[2022-12-16 19:58:47] Features: 76/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  76 out of  76 | elapsed:    9.9s finished

[2022-12-16 19:58:57] Features: 75/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  75 out of  75 | elapsed:   10.2s finished

[2022-12-16 19:59:07] Features: 74/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  74 out of  74 | elapsed:    9.5s finished

[2022-12-16 19:59:17] Features: 73/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  73 out of  73 | elapsed:    9.7s finished

[2022-12-16 19:59:27] Features: 72/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  72 out of  72 | elapsed:    9.0s finished

[2022-12-16 19:59:36] Features: 71/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  71 out of  71 | elapsed:    9.0s finished

[2022-12-16 19:59:45] Features: 70/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  70 out of  70 | elapsed:    8.1s finished

[2022-12-16 19:59:53] Features: 69/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  69 out of  69 | elapsed:    9.2s finished

[2022-12-16 20:00:03] Features: 68/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  68 out of  68 | elapsed:    8.7s finished
```

```
[2022-12-16 20:00:11] Features: 67/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  67 out of  67 | elapsed:    9.2s finished

[2022-12-16 20:00:21] Features: 66/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  66 out of  66 | elapsed:    9.9s finished

[2022-12-16 20:00:31] Features: 65/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  65 out of  65 | elapsed:    8.9s finished

[2022-12-16 20:00:40] Features: 64/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  64 out of  64 | elapsed:    9.0s finished

[2022-12-16 20:00:49] Features: 63/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  63 out of  63 | elapsed:   10.0s finished

[2022-12-16 20:00:59] Features: 62/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  62 out of  62 | elapsed:    7.6s finished

[2022-12-16 20:01:07] Features: 61/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  61 out of  61 | elapsed:    8.9s finished

[2022-12-16 20:01:16] Features: 60/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  60 out of  60 | elapsed:   10.7s finished

[2022-12-16 20:01:27] Features: 59/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  59 out of  59 | elapsed:    9.9s finished

[2022-12-16 20:01:37] Features: 58/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  58 out of  58 | elapsed:    9.7s finished

[2022-12-16 20:01:47] Features: 57/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  57 out of  57 | elapsed:    7.2s finished

[2022-12-16 20:01:54] Features: 56/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  56 out of  56 | elapsed:    8.4s finished

[2022-12-16 20:02:02] Features: 55/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  55 out of  55 | elapsed:    8.1s finished
```

```
[2022-12-16 20:02:11] Features: 54/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  54 out of  54 | elapsed:    8.8s finished

[2022-12-16 20:02:19] Features: 53/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  53 out of  53 | elapsed:    6.1s finished

[2022-12-16 20:02:26] Features: 52/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  52 out of  52 | elapsed:    5.7s finished

[2022-12-16 20:02:32] Features: 51/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  51 out of  51 | elapsed:    5.5s finished

[2022-12-16 20:02:37] Features: 50/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  50 out of  50 | elapsed:    5.2s finished

[2022-12-16 20:02:42] Features: 49/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  49 out of  49 | elapsed:    5.3s finished

[2022-12-16 20:02:48] Features: 48/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  48 out of  48 | elapsed:    5.2s finished

[2022-12-16 20:02:53] Features: 47/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  47 out of  47 | elapsed:    5.3s finished

[2022-12-16 20:02:59] Features: 46/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  46 out of  46 | elapsed:    4.9s finished

[2022-12-16 20:03:04] Features: 45/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  45 out of  45 | elapsed:    4.7s finished

[2022-12-16 20:03:08] Features: 44/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  44 out of  44 | elapsed:    4.5s finished

[2022-12-16 20:03:13] Features: 43/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  43 out of  43 | elapsed:    4.6s finished

[2022-12-16 20:03:18] Features: 42/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
```

```
[Parallel(n_jobs=1)]: Done  42 out of  42 | elapsed:    3.1s finished

[2022-12-16 20:03:21] Features: 41/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  41 out of  41 | elapsed:    2.9s finished

[2022-12-16 20:03:24] Features: 40/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed:    2.7s finished

[2022-12-16 20:03:27] Features: 39/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  39 out of  39 | elapsed:    2.6s finished

[2022-12-16 20:03:29] Features: 38/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  38 out of  38 | elapsed:    2.6s finished

[2022-12-16 20:03:32] Features: 37/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  37 out of  37 | elapsed:    2.4s finished

[2022-12-16 20:03:35] Features: 36/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  36 out of  36 | elapsed:    2.3s finished

[2022-12-16 20:03:37] Features: 35/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  35 out of  35 | elapsed:    2.2s finished

[2022-12-16 20:03:39] Features: 34/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  34 out of  34 | elapsed:    2.2s finished

[2022-12-16 20:03:42] Features: 33/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  33 out of  33 | elapsed:    2.0s finished

[2022-12-16 20:03:44] Features: 32/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  32 out of  32 | elapsed:    2.1s finished

[2022-12-16 20:03:46] Features: 31/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  31 out of  31 | elapsed:    2.0s finished

[2022-12-16 20:03:48] Features: 30/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  30 out of  30 | elapsed:    1.9s finished

[2022-12-16 20:03:50] Features: 29/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  29 out of  29 | elapsed:    1.9s finished

[2022-12-16 20:03:52] Features: 28/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  28 out of  28 | elapsed:    1.7s finished

[2022-12-16 20:03:54] Features: 27/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  27 out of  27 | elapsed:    1.7s finished

[2022-12-16 20:03:56] Features: 26/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  26 out of  26 | elapsed:    1.6s finished

[2022-12-16 20:03:57] Features: 25/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  25 out of  25 | elapsed:    1.5s finished

[2022-12-16 20:03:59] Features: 24/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  24 out of  24 | elapsed:    1.5s finished

[2022-12-16 20:04:01] Features: 23/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  23 out of  23 | elapsed:    1.4s finished

[2022-12-16 20:04:02] Features: 22/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  22 out of  22 | elapsed:    1.4s finished

[2022-12-16 20:04:04] Features: 21/20 -- score: 1.0[Parallel(n_jobs=1)]: Using bac
kend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  21 out of  21 | elapsed:    1.2s finished

[2022-12-16 20:04:05] Features: 20/20 -- score: 1.0
```

In [45]:
```
sfs.k_feature_idx_
```

Out[45]:
```
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20)
```

Therefore, using the forward and backward selection methods we found the following 20 protein expressions that show heavy weightage in predicting the behavior of the mice.

- DYRK1A_N
- ITSN1_N
- BDNF_N
- NR1_N
- NR2A_N
- pAKT_N
- pBRAF_N
- pCAMKII_N
- pCREB_N

- pELK_N
- pERK_N
- pJNK_N
- PKCA_N
- pMEK_N
- pNR1_N
- pNR2A_N
- pNR2B_N
- pPKCAB_N
- pRSK_N
- BRAF_N
- SOD1_N

4.6 LINEAR MODEL SELECTION AND REGULARIZATION

We now fit the ridge regressioin model to predict the behavior using the above best selected protein expressions and check how accurate it is.

Before fitting our regression model, we will first select the alpha value using trial and error method. We randomly choose alphas as 0.001. 0.01, 0.1, 1, 10, 100 and find the error for each alpha. Finally we choose the alpha value that gives least error.

In [46]:
```
feature_selected_df = refined_mice_df.iloc[:,[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
predictors = numericColumns_mice_df.columns
target = "Behavior_Code"

train, test = train_test_split(mice_df, test_size=0.2,  random_state=1)

def ridge_fit(train, predictors, target, alpha):
    X = train[predictors].copy()
    y = train[[target]].copy()

    x_mean = X.mean()
    x_std = X.std()

    X = (X - x_mean) / x_std
    X["intercept"] = 1
    X = X[["intercept"] + predictors]

    penalty = alpha * np.identity(X.shape[1])
    penalty[0][0] = 0

    B = np.linalg.inv(X.T @ X + penalty) @ X.T @ y
    B.index = np.append(["intercept"], predictors)
#["intercept",  "DYRK1A_N","ITSN1_N","BDNF_N","NR1_N","NR2A_N","pAKT_N","pBRAF_N","p
    return B, x_mean, x_std

def ridge_predict(test, predictors, x_mean, x_std, B):
    test_X = test[predictors]
    test_X = (test_X - x_mean) / x_std
    test_X["intercept"] = 1
    test_X = test_X[["intercept"] + predictors]

    predictions = test_X @ B
    return predictions


predictors = feature_selected_df.columns.tolist()
errors = []
alphas = [10**i for i in range(-3,3)]
```

```
for alpha in alphas:
    B, x_mean, x_std = ridge_fit(train, predictors, target, alpha)
    predictions = ridge_predict(test, predictors, x_mean, x_std, B)

    errors.append(mean_absolute_error(test[target], predictions))

errors
```

Out[46]:
```
[0.1300603245018846,
 0.13005890178189722,
 0.13004469469432814,
 0.12990482719239888,
 0.12955017348362707,
 0.14212486274219302]
```

In [47]:
```
alphas
```

Out[47]:
```
[0.001, 0.01, 0.1, 1, 10, 100]
```

Therefore, we fit the ridge regression model by fixing alpha = 10.

In [63]:
```
#Ridge Regression

alpha = 10
x_train, x_test, y_train, y_test = train_test_split(feature_selected_df, refined_m

ridge_reg_model = Ridge(alpha=alpha)
ridge_reg_model.fit(x_train, y_train)
print("R-Squared: ", ridge_reg_model.score(x_train, y_train))
y_pred= ridge_reg_model.predict(x_test)
print("Mean Absolute Error:",mae)
print("Mean Squared Error:",mse)
print("Root Mean Squared Error :",rmse)
```

```
R-Squared:  0.8530355868266501
Mean Absolute Error: 0.0
Mean Squared Error: 0.0
Root Mean Squared Error : 0.0
```

Since the R squared = 0.853 and the errors MAE, MSE, RMSE are 0, we have enough evidence to say that our model fits the data with 85.3% accuracy.

We see that there is a negligable difference between the test values and the predicted values in the below output.

In [51]:
```
test["Behavior_Code"] - predictions["Behavior_Code"]
```

Out[51]:
```
754    -0.043310
532     0.155864
185     0.070240
522    -0.022932
1045    0.357692
          ...
827     0.176605
867    -0.139750
92     -0.105450
760    -0.010715
940    -0.215333
Name: Behavior_Code, Length: 111, dtype: float64
```

4.7 MOVING BEYOND LINEARITY

We now fit the polynomial regressioin model to predict the behavior and check how accurate it is.

In [64]:
```
#Polynomial Regression

x_train, x_test, y_train, y_test = train_test_split(feature_selected_df, refined_m
poly = PolynomialFeatures(degree=3, include_bias=True)
x_train_trans = poly.fit_transform(x_train)
x_test_trans = poly.transform(x_test)
poly_reg_model = LinearRegression()
poly_reg_model.fit(x_train_trans, y_train)
print("R-Squared: ", poly_reg_model.score(x_train_trans, y_train))
y_pred = poly_reg_model.predict(x_test_trans)
mae = mean_absolute_error(y_true=y_test,y_pred=y_pred)
mse = mean_squared_error(y_true=y_test,y_pred=y_pred) #default=True
rmse = mean_squared_error(y_true=y_test,y_pred=y_pred,squared=False)

print("Mean Absolute Error:",mae)
print("Mean Squared Error:",mse)
print("Root Mean Squared Error :",rmse)
```

```
R-Squared:  1.0
Mean Absolute Error: 0.23609575320861298
Mean Squared Error: 2.7181384260980197
Root Mean Squared Error : 1.6486777811622317
```

Since the R squared = 0.853 and the errors MAE, MSE, RMSE are negligably small, we have enough evidence to say that our model fits the data accurately.

We see that there is a negligable difference between the test values and the predicted values in the below output.

In [53]:
```
y_test-y_pred
```

Out[53]:
```
235   -0.082158
955   -0.051553
261   -0.037722
317    0.215271
623    0.120169
         ...
826    0.117512
911    0.023458
226    0.040381
999   -0.104863
825    0.124874
Name: Behavior_Code, Length: 111, dtype: float64
```

CONCLUSION

In the above statistical analysis we have used the techniques of comparing two samples, to find if the protein group and behavior follows same distribution or not using Mann-Whitney test. Later we performed ANOVA, to check if the protein groups and behavior code have same or different means using Kruskal-Walis test and identified the specific pairs that are different using Nemenyi's test. Later we used chi squared test of independence to observe that the behavior of mice is independent of the treatment given. We then filtered out the best 20 protein groups that would help us predict the behavior of mice. Ultimately, using those 20 protein groups we fitted the logistic, Ridge and polynomial regression models to predict the behavior of the mice. The fit of the model was tested on the test set, and we managed to get an accuracy of 100% in logistic and polynomial regressions and 85.3% in ridge regression model.

REFERENCES

1. James, G., Witten, D., Hastie, T., & Tibshirani, R. (2015). An Introduction to Statistical Learning with Applications in R, Edn. 6th.
2. Rice, J. A. (2006). Mathematical statistics and data analysis. Cengage Learning.
3. https://www.kaggle.com/datasets/ruslankl/mice-protein-expression
4. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4521889/
5. https://www.youtube.com/watch?v=mpuKSovz9xM
6. https://github.com/dataquestio/project-walkthroughs/blob/master/ridge_regression/ridge_regression.ipynb
7. https://www.real-statistics.com/multiple-regression/ridge-and-lasso-regression/estimating-ridge-regression-lambda/
8. https://machinelearningmastery.com/ridge-regression-with-python/
9. https://sit.instructure.com/courses/60138/modules
10. https://www.w3schools.com/python/module_statistics.asp
11. https://docs.python.org/3/library/statistics.html
12. https://scipy-lectures.org/packages/statistics/index.html