Comparing Machine Learning and Deep Learning Classifiers on Gene Expression Data

December 2021

Abstract

Developing and testing four classifiers, two deep learning neural networks and two k-nearest neighbor classifiers on gene expression data. Comparing and discussing the developed feature selection algorithm and its effect on classifier success.

All source code written for this report and additional plots relating to the algorithms can be found on my GitHub.

Hannah Nissenbaum

Student Number: 20049077

Table of Contents

List of Tables and Figures

Section I: Introduction

The objective of this report is to construct a predictor that can accurately classify datapoints in one of two classes. These results will then be compared with a second predictor's results to determine which predictor can provide the most accurate classification of new datapoints.

The data used to train and test the two predictors contains 7129 features and 60 datapoints and was collected by (Pomeroy, et al., 2001). The features represent specific individual genes, which have all been collected from patients with medulloblastomas, a malignant brain tumor. The gene expressions are calculated and then a log operation is performed on them to normalize the data for processing.

For the purposes of this report, the first group, non-responders will henceforth be referred to as Class 0 and the second group, responders, as Class 1. The genes (rows in the data matrix) will be referred to as features, and the patients (each full set of gene information) will be referred to as datapoints.

Section II: Methodology

Data Pre-Processing

The first step is processing the data. With 60 datapoints, the data can be evenly split for a train set and a test set. There are 21 datapoints in the non-responders class (Class 0), and 39 in the responders class (Class 1). It is important to ensure that the split of classes over both datasets is relatively equal to avoid overfitting the data to a specific class. Therefore, the data will be split into two matrices, Class 0 datapoints and Class 1 datapoints. To ensure randomness and objectivity, all separation of the data will be randomized using MATLAB random integer generator functions. Due to the uneven values of datapoints in each set, a bias will be assigned with a randomly generated integer between 0 and 1. The value of the bias represents the dataset (training or testing) that will contain 11 datapoints in Class 0 and 19 datapoints in Class 1. The dataset not selected by the random integer will contain 10 datapoints in Class 0 and 20 datapoints in Class 1. To randomize the data, two strings of unique random integers will be generated, one of length 21 and one of length 39. These strings will serve as the indices for the Class 0 and Class 1 datapoints. Using the random integer strings as indices to select the next assigned datapoint, they will be alternately placed into the training set or the test set. Then, the bias will be consulted, and the selected set will receive the leftover datapoint in Class 0. The other set will receive the leftover Class 1 datapoint. Both sets will have 30 datapoints, and 7129 features per datapoint. Two 30x1 arrays will be created, one for the training set and one for the testing set, storing the class labels for each datapoint at the same indices as that datapoint is stored.

Report Decisions

Four predictors will be used for this report: two K-nearest neighbor (KNN) predictors and two trained neural networks. One of the neural networks and one of the KNN classifiers have been through a feature selection step and the other two will use all 7129 features. Comparing the two KNN predictors and two neural networks, the feature selection step will be evaluated for validity based on the classifiers that used it, and those that didn't. Additionally, the training types will be evaluated for effectiveness, as there will be a direct comparison between the machine learning and deep learning pairs, using the same number of features.

The accuracy of the classifiers will be measured as a function of the prediction errors over the total predictions made. It will therefore be a value normalized between 0 and 1. Class 0 errors have been defined as those where the predictor selects 1 as the class, however the true class is 0. Conversely, Class 1 errors have been defined as those where the predictor selects 0 as the class and the true class is 1. Class 0 errors are expected to be more common, as there are significantly more Class 1 datapoints in both the training set and the testing set as a function of the raw data.

These two predictors were chosen because KNN represents a classical machine learning technique commonly used for classifiers, and neural networks represent the new trend of deep learning which has been gaining in popularity since AlexNet in 2012. (Krizhevsky, Sutskever, & Hinton, 2012)

Method Description

Both neural networks that will used are from the MATLAB Machine Learning app, using the Classification Learner. They will both be trained on the same number of features as their respective KNN predictor pairs.

The KNN predictor was developed specifically for this report. It is written in Python and compiled using Visual Studio Code. The algorithm takes the randomized data from a csv file and uses the Skikit-learn Python library to fit a KNN model to the training set, cross validating the training accuracy and scoring each k value according to its mean accuracy achieved over 30 folds. Feature selection is also implemented as an additional parameter during training. The algorithm fits a KNN model using the default value of k=5, and stores accuracy values of KNN on the training set for models trained with 5-500 of the most correlated features to Class 0. It then looks for the model with maximum accuracy achieved in that set, and picks how many correlated features to use. The k parameter selection works similarly; the algorithm takes the training set and trains a model with k values 2-10, storing their accuracy. Then, evaluating the array of accuracies for different k values, the best k is selection. The model implementing both the best number of features, and the best k values is then passed the test set, where accuracy can be evaluated.

The feature selection is also implemented for one of the Bilayer Neural Networks, however, the neural network does not use a parameter selection for number of features to use. The exact same dataset must be used for both members in a predictor pair to truly evaluate the efficacy of different predictor methods. Therefore, the features used by the neural network is the feature array chosen by the Python feature selection algorithm.

The feature selection algorithm will be implemented for one of the KNN predictors and Bilayer Neural Networks trained and follows (Golub, et al., 1999) suggestions on fair feature selection for gene expressions. The Feature Selection function (shown in Appendix B: Python Source Code) is written in Python and compiled through Visual Studio Code. The input is a training dataset array. It calculates the number of Class 0 and Class 1 datapoints, storing those values. Then, looping through every cell in the dataset, the algorithm calculates the average values in Class 0 and Class 1 for each feature. Then, the standard deviation of each cell is calculated from it’s respective average using Equation 1.

$$(1) \qquad \sigma(\mathrm{j},\mathrm{k}) = \sqrt{\frac{\sum(x_{i,j} + \mu_{j,k})^2}{N_k}}$$

$Where$:
$i \ = the\ datapoint\ array$
$j \ \ = the\ feature\ array$
$k \ \ = the\ class\ array$
$x_{i,j} = cell\ value, per\ feature, per\ datapoint$
$\mu_{j,k} \ \ = the\ previously\ calculated\ average, per\ feature, per\ class$
$N_k \ = the\ total\ datapoints\ in\ the\ respective\ class$

After calculating the standard deviation for each feature in Class 0 and Class 1, each feature is given a score, calculated using Equation 2, where ‘j’ is the feature array.

$$(2) \qquad P(j) = \frac{\mu_{j,k=0} - \mu_{j,k=1}}{\sigma_{j,k=0} + \sigma_{j,k=1}}$$

$Where:$
$j \quad = the\ feature\ array$
$k \quad = the\ class\ array$
$\mu_{j,k} \quad = the\ previously\ calculated\ average, per\ feature, per\ class$
$\sigma_{j,k} \ = the\ standard\ deviation\ per\ feature, per\ class$

The feature scores equation is such that large and positive values will be strongly correlated to Class 0, and large and negative values will be strongly correlated to Class 1. Those close to 0 provide little to no useful classification information. Therefore, the center values must be immediately filtered out. Furthermore, due to the datapoints being 60% Class 1, the features with the largest positive scores are chosen to represent the dataset.

Section III: Findings

Feature Selection

For feature selection, the largest positive feature scores were targeted. As can be seen in Figure 1, this results in features that have far fewer outliers and normalized averages over the 30 datapoints. Because feature scores prioritize low standard deviation values, the overall standard deviation for the selected features are either low, or there are high values on either side of zero, cancelling each other out. Therefore, the average values of most features selected will hover closer to zero then the rest of the datapoints.

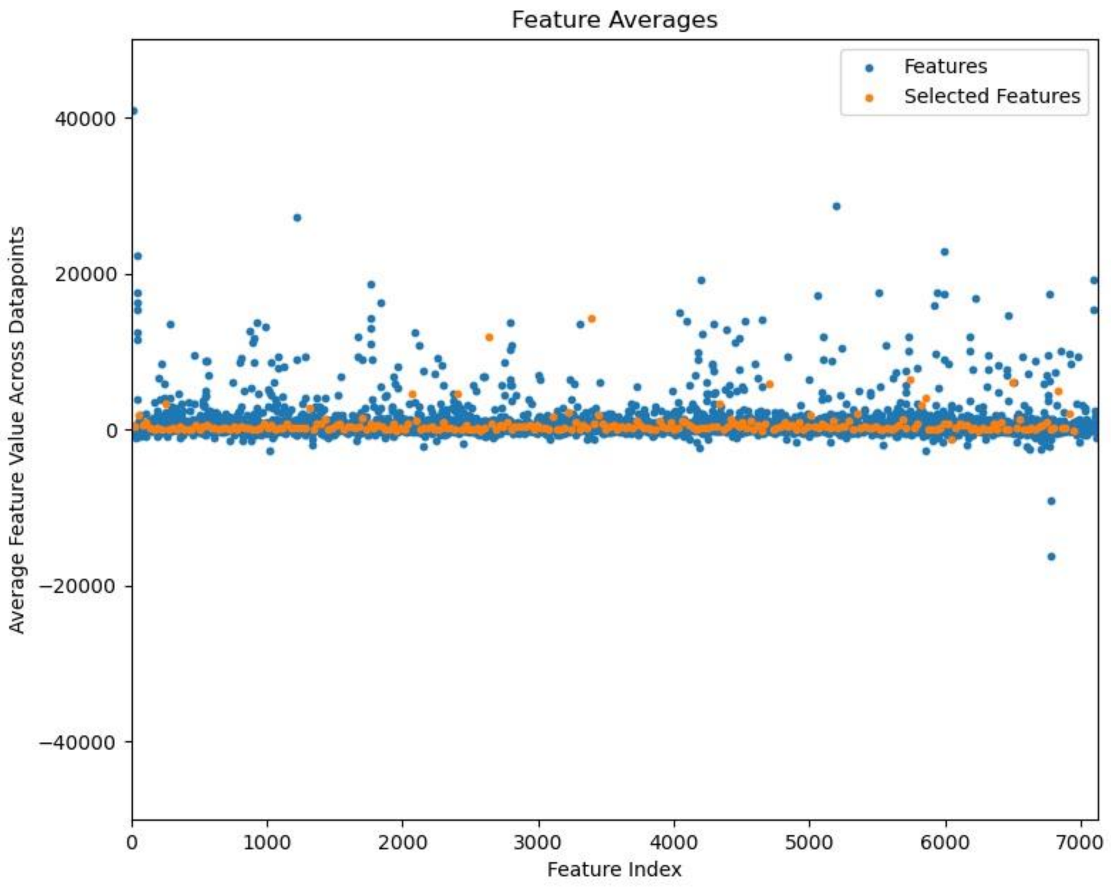

Figure 1: A Comparison of the Feature Averages across all datapoints for before and after feature selection.

K-Nearest Neighbor Classifier

The KNN classifier was run two times, first with the developed feature selection algorithm implemented to solely use the heavily Class 0 correlated features. Leave-one-out cross validation was also implemented when calculating the parameter k. The accuracy found for KNN With Feature Selection was 73.3%, with 4 Class 0 errors and 4 Class 1 errors. The accuracy found for KNN with All Features was 66.6% with 8 Class 0 errors and 2 Class 1 errors.

Class Predictions vs True Classes

Table 1: Results Achieved for KNN Predictors

Results Achieved with KNN and Feature Selection – 210 Genes Used			Results Achieved with KNN, without Feature Selection – 7129 Genes		
Data Point Index	Predicted Class	True Class	Data Point Index	Predicted Class	True Class
1	1	0	1	1	0
2	1	0	2	1	0
3	0	0	3	0	0
4	0	0	4	1	0
5	0	0	5	1	0
6	1	0	6	1	0
7	0	0	7	0	0
8	0	0	8	1	0
9	1	0	9	1	0
10	0	0	10	0	0
11	0	0	11	1	0
12	1	1	12	1	1
13	1	1	13	1	1
14	1	1	14	1	1
15	1	1	15	0	1
16	1	1	16	1	1
17	1	1	17	1	1
18	1	1	18	1	1
19	1	1	19	1	1
20	1	1	20	1	1
21	1	1	21	0	1
22	0	1	22	1	1
23	1	1	23	1	1
24	1	1	24	1	1
25	1	1	25	1	1
26	0	1	26	1	1
27	0	1	27	1	1
28	1	1	28	1	1
29	1	1	29	1	1
30	0	1	30	1	1
Accuracy over All Data Points: 73.3%			**Accuracy over All Data Points: 66.6%**		

Contingency Tables

Table 2: Results Achieved with KNN and Feature Selection – 250 Genes Used

Contingency Table		***Predicted Classes***		*Marginal Totals*
		Class 0	*Class 1*	
True Classes	*Class 0*	7	4	11
	Class 1	4	15	19
	Marginal Totals	11	19	30

Table 3: Results Achieved with KNN, without Feature Selection – 7129 Genes

Contingency Table		***Predicted Classes***		*Marginal Totals*
		Class 0	*Class 1*	
True Classes	*Class 0*	3	8	11
	Class 1	2	17	19
	Marginal Totals	5	25	30

Neural Networks

The accuracy found for Bilayer Neural Network with Feature Selection was 76.7%, with 4 Class 0 errors and 3 Class 1 errors. The accuracy found for Bilayer Neural Network with All Features was 70.0% with 6 Class 0 errors and 3 Class 1 errors.

Class Predictions vs True Classes

Table 4: Results Achieved for Neural Networks

Results Achieved on Bilayer Neural Network with Feature Selection – 210 Genes Used			Results Achieved on Bilayer Neural Network Trained on All Features – 7129 Genes Used		
Data Point Index	Predicted Class	True Class	Data Point Index	Predicted Class	True Class
1	0	0	1	0	0
2	1	0	2	1	0
3	0	0	3	1	0
4	0	0	4	0	0
5	0	0	5	1	0
6	1	0	6	1	0
7	0	0	7	0	0
8	0	0	8	0	0
9	1	0	9	1	0
10	0	0	10	0	0
11	1	0	11	1	0
12	1	1	12	1	1
13	0	1	13	0	1
14	1	1	14	1	1
15	1	1	15	1	1
16	1	1	16	1	1
17	1	1	17	1	1
18	1	1	18	1	1
19	0	1	19	1	1
20	1	1	20	1	1
21	1	1	21	0	1
22	1	1	22	1	1
23	1	1	23	1	1
24	1	1	24	1	1
25	1	1	25	1	1
26	0	1	26	1	1
27	1	1	27	1	1
28	1	1	28	0	1
29	1	1	29	1	1
30	1	1	30	1	1
Accuracy over All Data Points: 76.7%			**Accuracy over All Data Points: 70.0%**		

Contingency Tables

Table 5: Results Achieved on Bilayer Neural Network with Feature Selection – 210 Genes Used

Contingency Table		***Predicted Classes***		*Marginal Totals*
		Class 0	*Class 1*	
True Classes	*Class 0*	7	4	11
	Class 1	3	16	19
Marginal Totals		10	20	30

Table 6: Results Achieved on Bilayer Neural Network Trained on All Features – 7129 Genes Used

Contingency Table		***Predicted Classes***		*Marginal Totals*
		Class 0	*Class 1*	
True Classes	*Class 0*	5	6	11
	Class 1	3	16	19
Marginal Totals		8	22	30

The findings in this report are all the result of a single randomized split in the data. However, other randomized iterations may have produced different results. In this dataset, the training set was randomly assigned to have 10 Class 0 datapoints and 20 Class 1 datapoints. After receiving the results, the training set should have prioritized as even a split between Class 0 and Class 1 as possible, therefore the training set should have been forcibly biased to have 11 Class 0 and 19 Class 1 datapoints. The test set for this report had 11 Class 0 and 19 Class 1 datapoints. In a more extensive study, the data could have been randomized for a large number of iterations, conducting training for the 4 predictors for every iteration to produce an average of results.

Section IV: Discussion

Though the accuracy value for both the KNN with and without feature selection is similar, the quality of predictor varies due to the biases associated with predictors. Feature selection attempts to minimize the bias between classes due to the classification split of the datapoints. Given that approximately 60% of the datapoints are classified into Class 1, a predictor that is 100% biased to Class 1 will automatically get 60% accuracy. Therefore, a more accurate way to evaluate predictors is using Fisher's exact probability, which considers the correct values, and whether the errors are Class 0 or Class 1. This filters out biased predictors, because the calculated Matthew's Correlation Coefficient favors evenly distributed marginal totals over skewed ones.

K-Nearest Neighbor Classifier

A KNN algorithm was developed using the Skikit-learn python library and implementing Leave-One-Out cross validation to determine the best k value. It was trained on the randomized set representing the training data as described above. Two versions of the KNN algorithm are being evaluated in this section, one that performed feature selection, and one that uses all 7129 features from the original data.

Fisher Exact Probability Test

Developed KNN with feature selection

Table 7: 2x2 Contingency Table of True Values Predicted by developed KNN with feature selection

Contingency Table		*Predicted Classes*		*Marginal Totals*
		Class 0	*Class 1*	
True Classes	*Class 0*	7	4	11
	Class 1	4	15	19
Marginal Totals		11	19	30

Calculating Matthew's Correlation Coefficient:

$$\frac{(7*15)-(4*4)}{\sqrt{11*19*11*19}} = 0.4258$$

Calculating Fisher's Probability:

$$\frac{(7+4)!\,(4+15)!\,(7+4)!\,(4+15)!}{30!\,4!\,7!\,4!\,15!} = 0.0234$$

Table 8: Contingency table keeping marginal totals and changing values to maximize Matthew's Correlation Coefficient for KNN

Contingency Table		*Predicted Classes*		*Marginal Totals*
		Class 0	*Class 1*	
True Classes	*Class 0*	11	0	11
	Class 1	0	19	19
Marginal Totals		11	19	30

Calculating Matthew's Correlation Coefficient:

$$\frac{(11*19)-(0*0)}{\sqrt{11*19*11*19}} = 1$$

Calculating Fisher's Probability:

$$\frac{(11+0)!\,(0+19)!\,(11+0)!\,(19)!}{30!\,11!\,0!\,0!\,19!} = 1.8306*10^{-8}$$

Table 9: Contingency table keeping marginal and changing values to minimize Matthew's Correlation Coefficient for KNN

Contingency Table		*Predicted Classes*		*Marginal Totals*
		Class 0	*Class 1*	
True Classes	*Class 0*	0	11	11
	Class 1	11	8	19
Marginal Totals		11	19	30

Calculating Matthew's Correlation Coefficient:

$$\frac{(8*0)-(11*11)}{\sqrt{11*19*11*19}} = -0.5789$$

Calculating Fisher's Probability:

$$\frac{(8+11)!\,(11+11)!\,(0+11)!\,(0+11)!}{30!\,8!\,0!\,11!\,11!} = 0.001384$$

Table 10: Contingency tables and Fisher's Probability (FP) for all other possibilities of Matthew's Correlation Coefficient for these marginal totals

	0	1	
0	11	0	11
1	0	19	19
	11	19	

FP: 1.8031*10^ (-8) (Max MCC)

	0	1	
0	10	1	11
1	1	18	19
	11	19	

FP: 3.8259*10^ (-6)

	0	1	
0	9	2	11
1	2	17	19
	11	19	

FP: 1.7217*10^ (-4)

	0	1	
0	8	3	11
1	3	16	19
	11	19	

FP: 0.0029268

	0	1	
0	7	4	11
1	4	15	19
	11	19	

FP: 0.0234 (Actual Values)

	0	1	
0	6	5	11
1	5	14	19
	11	19	

FP: 0.09834 (not included)

	0	1	
0	5	6	11
1	6	13	19
	11	19	

FP: 0.2295 (not included)

	0	1	
0	4	7	11
1	7	12	19
	11	19	

FP: 0.3044 (not included)

	0	1	
0	3	8	11
1	8	11	19
	11	19	

FP: 0.2283 (not included)

	0	1	
0	2	9	11
1	9	10	19
	11	19	

FP: 0.09300 (not included)

	0	1	
0	1	10	11
1	10	9	19
	11	19	

FP: 0.0186

	0	1	
0	0	11	11
1	11	8	19
	11	19	

FP: 0.001384 (Min MCC)

$$knn\ two\ tailed\ probability\ = 1.8031 * 10^{-8} + 3.8259 * 10^{-6} + 1.7217 * 10^{-4} + 0.0029268 + 0.0234 + 0.0186 + 0.001384 = \mathbf{0.0465}$$

Developed KNN without feature selection

Table 11: 2x2 Contingency Table of True Values Predicted by developed KNN without feature selection

Contingency Table		***Predicted Classes***		*Marginal Totals*
		Class 0	*Class 1*	
True Classes	*Class 0*	3	8	11
	Class 1	2	17	19
	Marginal Totals	5	25	30

Calculating Matthew's Correlation Coefficient:

$$\frac{(3*17)-(8*2)}{\sqrt{11*19*5*25}} = 0.2165$$

Calculating Fisher's Probability:

$$\frac{(3+8)!\,(2+17)!\,(3+2)!\,(8+17)!}{30!\,3!\,2!\,8!\,17!} = 0.1980$$

Table 12: Contingency table keeping marginal totals and changing values to maximize Matthew's Correlation Coefficient for KNN

Contingency Table		***Predicted Classes***		*Marginal Totals*
		Class 0	*Class 1*	
True Classes	*Class 0*	5	6	11
	Class 1	0	19	19
	Marginal Totals	5	25	30

Calculating Matthew's Correlation Coefficient:

$$\frac{(5*19)-(6*0)}{\sqrt{11*19*5*25}} = 0.5878$$

Calculating Fisher's Probability:

$$\frac{5+6!\,(0+19)!\,(5+0)!\,(6+19)!}{30!\,5!\,0!\,6!\,19!} = 0.003241$$

Table 13: Contingency table keeping marginal and changing values to minimize Matthew's Correlation Coefficient for KNN

Contingency Table		***Predicted Classes***		*Marginal Totals*
		Class 0	*Class 1*	
True Classes	*Class 0*	0	11	11
	Class 1	5	14	19
Marginal Totals		5	25	30

Calculating Matthew's Correlation Coefficient:

$$\frac{(14*0)-(5*11)}{\sqrt{11*19*11*19}} = -0.3403$$

Calculating Fisher's Probability:

$$\frac{(0+11)!\,(5+14)!\,(0+5)!\,(11+14)!}{30!\,5!\,0!\,14!\,11!} = 0.08160$$

Table 14: Contingency tables and Fisher's Probability (FP) for all other possibilities of Matthew's Correlation Coefficient for these marginal totals

	0	1	
0	5	6	11
1	0	19	19
	5	25	

FP: 0.003241 (Max MCC)

	0	1	
0	4	7	11
1	1	18	19
	5	25	

FP: 0.0440

	0	1	
0	3	8	11
1	2	17	19
	5	25	

FP: 0.19799 (Actual Values)

	0	1	
0	2	9	11
1	3	16	19
	5	25	

FP: 0.3740 (not included)

	0	1	
0	1	10	11
1	4	15	19
	5	25	

FP: 0.2992 (not included)

	0	1	
0	0	11	11
1	5	14	19
	5	25	

FP: 0.08160

$$developed\ knn\ two\ tailed\ probability = 0.003241 + 0.0440 + 0.19799 + 0.08160 = \mathbf{0.3268}$$

Neural Networks

Using the available MATLAB software to train neural networks with the same randomized training sets used for the KNN algorithm testing.

Fisher Exact Probability Test

Bilayer Neural Network Trained on All Features

Table 15: 2x2 Contingency Table of True Values Predicted by Bilayer Neural Network Trained on All Features

Contingency Table		*Predicted Classes*		*Marginal Totals*
		Class 0	*Class 1*	
True Classes	*Class 0*	5	6	11
	Class 1	3	16	19
	Marginal Totals	8	22	30

Calculating Matthew's Correlation Coefficient:

$$\frac{(5*16)-(3*6)}{\sqrt{11*19*8*22}} = 0.3233$$

Calculating Fisher's Probability:

$$\frac{(5+6)!\,(3+16)!\,(5+3)!\,(6+16)!}{30!\,5!\,6!\,3!\,16!} = 0.0765$$

Table 16: Contingency table keeping marginal totals and changing values to maximize Matthew's Correlation Coefficient for Bilayer Neural Network Trained on All Features

Contingency Table		*Predicted Classes*		*Marginal Totals*
		Class 0	*Class 1*	
True Classes	*Class 0*	8	3	11
	Class 1	0	19	19
	Marginal Totals	8	22	30

Calculating Matthew's Correlation Coefficient:

$$\frac{(8*19)-(3*0)}{\sqrt{11*19*8*22}} = 0.7925$$

Calculating Fisher's Probability:

$$\frac{(8+0)!\,(3+19)!\,(8+3)!\,(0+19)!}{30!\,8!\,0!\,3!\,19!} = 0.00002819$$

Table 17: Contingency table keeping marginal totals and changing values to minimize Matthew's Correlation Coefficient for Bilayer Neural Network Trained on All Features

Contingency Table		*Predicted Classes*		*Marginal Totals*
		Class 0	*Class 1*	
True Classes	*Class 0*	0	11	11
	Class 1	8	11	19
Marginal Totals		8	22	30

Calculating Matthew's Correlation Coefficient:

$$\frac{(0*11)-(8*11)}{\sqrt{11*19*8*22}} = -0.4588$$

Calculating Fisher's Probability:

$$\frac{(8+0)!\,(11+11)!\,(0+11)!\,(8+11)!}{30!\,8!\,0!\,11!\,11!} = 0.0129$$

Table 18: Contingency tables and Fisher's Probability (FP) for all other possibilities of Matthew's Correlation Coefficient for these marginal totals for Bilayer Neural Network Trained on All Features

	0	1	
0	1	10	11
1	7	12	19
	8	22	

FP: 0.0947 (not included)

	0	1	
0	2	9	11
1	6	13	19
	8	22	

FP: 0.2550 (not included)

	0	1	
0	3	8	11
1	5	14	19
	8	22	

FP: 0.3278 (not included)

	0	1	
0	4	7	11
1	4	15	19
	8	22	

FP: 0.2185 (not included)

	0	1	
0	5	6	11
1	3	16	19
	8	22	

FP: 0.0765 (True MCC)

	0	1	
0	6	5	11
1	2	17	19
	8	22	

FP: 0.0135

	0	1	
0	7	4	11
1	1	18	19
	8	22	

FP: 0.001073

	0	1	
0	8	3	11
1	0	19	19
	8	22	

FP: 0.00002819 (Biggest MCC)

	0	1	
0	0	11	11
1	8	11	19
	8	22	

FP: 0.0129 (Smallest MCC)

$$\text{bilayered neural network trained on all features two tailed probability} = 0.0765 + 0.0135 + 0.001073 + 0.00002819 + 0.0129 = \mathbf{0.104}$$

Bilayer Neural Network Trained with Feature Selection

A neural network was trained using only the feature selected by the feature selection process. It is therefore a direct comparison to the KNN predictor trained on the same set of datapoints and features.

Table 19: 2x2 Contingency Table of True Values Predicted by Bilayer Neural Network Trained with Selected Features

Contingency Table		*Predicted Classes*		*Marginal Totals*
		Class 0	*Class 1*	
True Classes	*Class 0*	7	4	11
	Class 1	3	16	19
Marginal Totals		10	20	30

Calculating Matthew's Correlation Coefficient:

$$\frac{(7*16)-(3*4)}{\sqrt{11*19*10*20}} = 0.4891$$

Calculating Fisher's Probability:

$$\frac{(4+7)!\,(3+16)!\,(7+3)!\,(4+16)!}{30!\,4!\,7!\,3!\,16!} = 0.0106$$

$$\text{Bilayer neural network two tailed probability} = \mathbf{0.01469}$$

Table 20: Summary of Accuracy and Fisher's Two Tailed Probability for all classifiers

Overall Results Achieved with 4 Classifiers	
Results Achieved with KNN Trained with Selected Features	Results Achieved with KNN without Feature Selection
Accuracy over All Data Points: 73.3%	Accuracy over All Data Points: 66.7%
Fisher's Exact Two Tailed Probability: 0.0465	Fisher's Exact Two Tailed Probability: 0.3268
Results Achieved on Bilayer Neural Network Trained with Selected Features	Results Achieved on Bilayer Neural Network without Feature Selection
Accuracy over All Data Points: 76.6%	Accuracy over All Data Points: 70.0%
Fisher's Exact Two Tailed Probability: 0.01469	Fisher's Exact Two Tailed Probability: 0.104

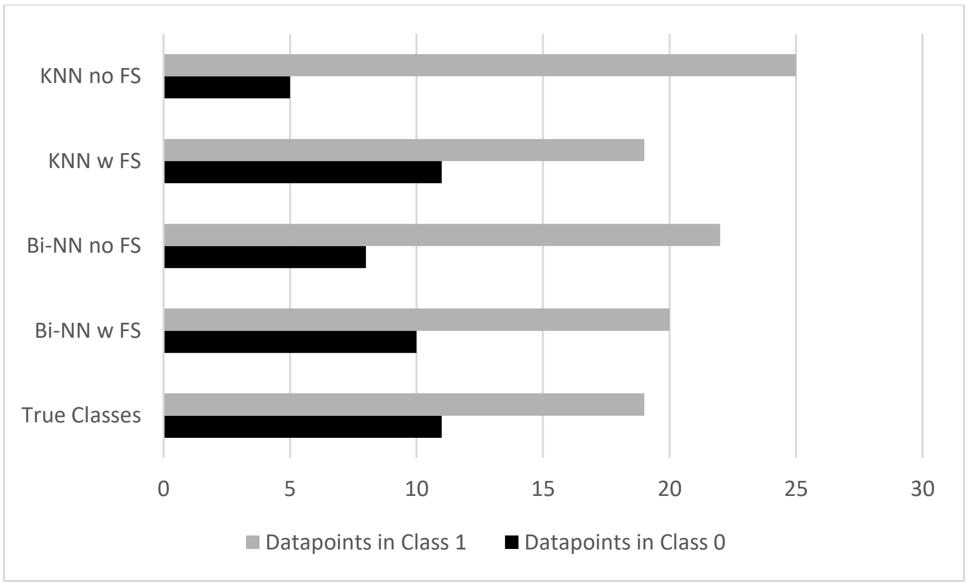

Figure 2: Comparison of the Class 1 and Class 0 Prediction Numbers of Each Predictor Compared to the True Predictions

Though accuracy tended above 50% in the four classifiers, the results aren't as promising as they seem. This is because both the training set and the test set had a strong bias towards Class 1 (66% of the datapoints in each set were Class 1, whereas only 33% percent were Class 0. Though the predictor training resulted in a strong Class 1 bias, this was not reflected in the accuracy scores because much of that bias was interpreted as correct predictions. If a predictor had predicted 100% Class 1 datapoints, showing complete Class 1 bias, the test set accuracy score would have still resulted in a 66.6%. However, the negative impacts of this bias on the predictors are represented in the Fisher's exact probabilities. Fisher's probability is a measure of independence for a predictor's predictions by looking at the ratio of predictions made for each class. The independence of predictions can be defined as the absence of reliance on the total number of predictions per class. The more biased a predictor is, the more predictions it will make for the favored class, minimizing the predictions made in the unfavored class. This makes for uneven predicted marginal totals, differing significantly from the true marginal totals which severely effects the success of Fisher's exact probability. The predictor with the highest accuracy score – the KNN algorithm using feature selection, showed the lowest two tailed probability at only 0.0465.

Analyzing the results found from all tested predictors, none of the four were able to classify the patients with reliable accuracy while being unbiased. However, bringing attention to the number of total Class 0 datapoints predicted by each prediction, the KNN predictor that used the results from feature selection predicted by far the largest number of Class 0 datapoints. Furthermore, the KNN predictor using feature selection had an accuracy of 70% predicting the Class 0 datapoints, significantly more then any of the other predictors tested. Because the feature selection focused solely on features that were strongly correlated to Class 0 in the training set, it follows that the predictors using it would be more inclined to predict Class 0. This could mean that the reason behind the inaccuracy and bias of the predictors tested in this report are due to the inequalities in the dataset, not in the inability of classifiers or neural networks to find patterns in the feature data.

Section V: Conclusions and Recommendations

This report does not clearly answer the question of whether machine learning or deep learning is a better method to determine the likelihood of whether a medulloblastoma will have a positive outcome for their treatment. However, there is enough evidence that, given a larger, more evenly distributed dataset, either method would be successful at predicting patients' outcomes. Given more time, additional feature selection algorithms would be beneficial, to learn information about which genes can be infallible markers for each class. The specific genes, and their meaning in the contribution to predicting each class is the most meaningful data to be extracted from this type of analysis. Ranking genes and associating them with the implications of cancer and cancer treatment could be instrumental in the development of an early detection system in the future.

This analysis shows that further than the additional presence of Class 1 datapoints in the data, the gene expression values correlate significantly more with Class 1 then Class 0. Therefore, finding genes that are strongly correlated with Class 0 and classing all other datapoints into Class 1 was the best approach to attempt as unbiased a predictor as possible.

Section VI: References

Golub, T., Slonim, D., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J., . . . Lander, E. (1999). Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science, 286*(5439), 531-537. doi:https://doi.org/10.1126/science.286.5439.531

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classificaiton with Deep Convolutional Nerual Networks. *Advances in Neural Information Processing Systems, 25*.

Lowry, R. (1998-2021). *2x2 Contingency Table*. Retrieved from Vassar Stats: http://vassarstats.net/tab2x2.html

Pedregosa, F., Varoquaux, G., Gramfort, A., Thirion, B., Grisel, O., Blondel, M., . . . Duchesnay, E. (2011). Skikit-learn Machine Learning in Python. *Journal of Machine Learning Research, 12*, 2825-2830. Retrieved from Ski-Kit Learn.

Pomeroy, S. L., Tamayo, P., Gaasenbeek, M., Sturla, L., Angelo, M., Mclaughlin, M. E., . . . Golub, T. (2001). Prediction of central nervous system embryonal tumour outcome based on gene expression. *Nature*(415), 436-442. doi:https://doi.org/10.1038/415436a

Thompson, C., & Shure, L. (1995-2021). *Machine Learning App - Mathworks*. (Mathworks, Producer) Retrieved 2021, from MATLAB R2021a.

Appendix A: MATLAB Code for Data Randomization

```
function [trainingdata, testingdata,bias,trainingdat,testingdat,training_label,↙
testing_label] = data_original(class1,class2) %#ok<*FNDEF>
[r1,~] = size(class1); %rows, columns
[r2,~] = size(class2);

trainingdata = zeros(max(r1,r2),20,2);
testingdata = zeros(max(r1,r2),20,2);
training_label = blanks(30);
testing_label = blanks(30);
trainingdat = zeros(7129,30);
testingdat = zeros(7129,30);


class1_idx = randperm(21,21);
class2_idx = randperm(39,39);
for i = 1:10
    trainingdata(:,i,1) = class1(:,class1_idx(i));
    testingdata(:,i,1) = class1(:,class1_idx(21-i));
    training_label(i) = 'b';
    testing_label(i) = 'b';
    trainingdat(:,i) = class1(:,class1_idx(i));
    testingdat(:,i) = class1(:,class1_idx(21-i));

end


bias = randi(2);
if bias==1
    trainingdata(:,11,1) = class1(:,class1_idx(21));
    trainingdata_indexing = 0;
    testingdata_indexing = 1;
    training_label(11) = 'b';
    testing_label(11) = 'g';
    trainingdat(:,11) = class1(:,class1_idx(21));
    testingdat(:,11) = class2(:,class2_idx(39));
    testingdata(:,1,2) = class2(:,class2_idx(39));
else
    testingdata(:,11,1) = class1(:,class1_idx(21));
    trainingdata_indexing = 1;
    testingdata_indexing = 0;
    training_label(11) = 'g';
    testing_label(11) = 'b';
    testingdat(:,11) = class1(:,class1_idx(21));
    trainingdat(:,11) = class2(:,class2_idx(39));
    trainingdata(:,1,2) = class2(:,class2_idx(39));

end


for i = 1:19
    trainingdata(:,trainingdata_indexing+i,2) = class2(:,class2_idx(i));
    testingdata(:,testingdata_indexing+i,2) = class2(:,class2_idx(39-i));
    training_label(i+11) = 'g';


    testing_label(i+11) = 'g';
    trainingdat(:,i+11) = class2(:,class2_idx(i));
    testingdat(:,i+11) = class2(:,class2_idx(39-i));
end
fprintf("Data loaded!");
return
```

Appendix B: Python Source Code

A complete inventory of source code discussed, written, and included in trial runs for these classifiers can be found on my GitHub – Link Here.

Feature Selection Function

```
### Author - Hannah Nissenbaum (Student #: 20049077)
import numpy as np
from sklearn import neighbors,model_selection,metrics
import matplotlib.pyplot as plt
import random

def feature_selection(train_set,test_set,train_labels,gene_num):
    train_set = train_set.transpose()
    test_set = test_set.transpose()
    # print(train_set.shape)

    features = len(train_set[:,1])
    data_pts = len(train_set[1,:])
    num1 = np.count_nonzero(train_labels)
    num0 = 30-num1
    #calculating the average of each feature through every datapoint
    features_avg0 = np.zeros((features))
    features_avg1 = np.zeros((features))
    data_sum0 = np.zeros((features))
    data_sum1 = np.zeros((features))
    for i in range(features): #through features
        for j in range(data_pts): #through points
            if train_labels[j] == 1:
                data_sum1[i] += train_set[i,j] #if corresponding to class 1,
they'll be negative.
            else: #if half the feature points are classified as class1, half as
class 2, the feature score will be close to zero (weak correlation)
                data_sum0[i] += train_set[i,j] #if half the feature points are
classified as class1, half as class 2, the feature score will be close to zero
        features_avg0[i] = data_sum0[i]/num0
        features_avg1[i] = data_sum1[i]/num1


    #signal noise ratio - how far each feature value is from the average of the
points in the data
    sig_noise_ratio0 = np.zeros((features))
    sig_noise_ratio1 = np.zeros((features))
    for i in range(features): #through features
        for j in range(data_pts): #through points
```

```
            if train_labels[j] == 1:
                sig_noise_ratio0[i] += (train_set[i,j] - features_avg0[i])**2
            else:
                sig_noise_ratio1[i] += (train_set[i,j] - features_avg1[i])**2
        sig_noise_ratio0[i] = np.sqrt(sig_noise_ratio0[i]/num0)
        sig_noise_ratio1[i] = np.sqrt(sig_noise_ratio1[i]/num1)

    #finding the features most indicative of each class through the signal noise
ratio - signal(class0-class1)/noise(class0+class1)
    #with this equation the scores that most strongly correspond to class 1 will
be large and negative (small signal noise ratios - few outliers)
    feature_score = np.zeros((features))
    for i in range(features): #through features
            feature_score[i] = (features_avg0[i]-
features_avg1[i])/(sig_noise_ratio0[i]+sig_noise_ratio1[i])

    #Finding the feature scores that are the largest/smallest (therefore
demostrating strong correlations with class 1 or 2)
    sort_idx = np.argsort(feature_score) #best and worst scores in order
    selected_features = np.zeros((gene_num,data_pts))
    test_selected_features = np.zeros((gene_num,data_pts))
    selected_scores = np.zeros((features))
    distributed_nfeatures = np.zeros((features,data_pts))
    feat_idx = []
    for i,feat in enumerate(sort_idx[0:((gene_num))]): #top 500 genes
corresponding to the non responders
        feat_idx.append(feat)
        selected_features[i,:] = (train_set[feat,:])
        test_selected_features[i,:] = (test_set[feat,:])
    # start = 7129-(int(gene_num/3))

    # for i,feat in enumerate(sort_idx[start:7129]): #top 500 genes corresponding
to the non responders
    #     feat_idx.append(feat)
    #     selected_features[i+(int(gene_num/2)),:] = (train_set[feat,:])
    #     test_selected_features[i+(int(gene_num/2)),:] = (test_set[feat,:])

    import csv
    header = ['Data Point Index']
    with open('selected_features.csv', 'w', encoding='UTF8', newline='') as f:
        writer = csv.writer(f)
        writer.writerow(header)
        writer.writerow(feat_idx)

    return selected_features, test_selected_features
```

Main KNN Function – with Feature Selection Currently Implemented

```
### Author - Hannah Nissenbaum (Student #: 20049077)
import numpy as np
from sklearn import neighbors,model_selection,metrics
import matplotlib.pyplot as plt
import random
from feature_selection import feature_selection
score_metric = metrics.make_scorer(metrics.accuracy_score)

#Importing the data
file = open("TrainSet - All Features.csv"); trains_set = np.loadtxt(file,
delimiter=",")
file2 = open("TestSet - All Features.csv"); tests_set = np.loadtxt(file2,
delimiter=",")

train_labels = [0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
test_labels = [0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]


#Defining the weighted voting scheme for cross validation
def weighting(scores,k_val):
    #The more accurate k values get a larger weight in the vote then the less
accurate ones
    kval = np.average(scores) #find the average accuracy score for this value of
k
    return kval,k_val

#Decide second parameter - number of features
faccuracy_scores = []
gene_nums = range(200,300)
for gene_num in gene_nums:
    print(gene_num)
    ftrain_set,_ = feature_selection(trains_set,tests_set,train_labels,gene_num)
    ftrain_set = ftrain_set.transpose()
    # re-fit the model with best-k
    knn = neighbors.KNeighborsClassifier(n_neighbors=5) #choosing random k - k
will be determined later
    knn.fit(ftrain_set, train_labels)
    train_results = knn.predict(ftrain_set)
    accuracy_train = metrics.accuracy_score(train_labels, train_results)
    faccuracy_scores.append(accuracy_train)

best_gene_num_idx = np.argmax(faccuracy_scores)
best_gene_num = gene_nums[best_gene_num_idx]
print("Best Feature Number Found: %d" %best_gene_num)
```

```
#re-import data with best gene number implemented
train_set,test_set =
feature_selection(trains_set,tests_set,train_labels,best_gene_num)
train_set = train_set.transpose()
test_set = test_set.transpose()

ks = range(2,10)
k_score = []
for k in ks:
    # Train the model with the whole training set and the current k
    knn2 = neighbors.KNeighborsClassifier(n_neighbors=k)
    knn2.fit(train_set, train_labels)

    #Setting up cross validation
    loo = model_selection.LeaveOneOut()
    cv = model_selection.cross_val_score(knn2, train_set, train_labels,
scoring=score_metric, cv=model_selection.KFold(n_splits=30))

    #Finding the best accuracy found in cross validation
    scoring = cv
    score,k = weighting(scoring,k)
    k_score.append(score)

# Fitting Cross Validation Results and Evaluating
ind = np.argmax(k_score) #the maximum average accuracy score is the k winner -
determined by the indices of the table.
best_k = ks[ind]
print(f"Best K Determined to be: {best_k}")


# Training new model with Best K and New Feature Number!
knn3 = neighbors.KNeighborsClassifier(n_neighbors=best_k)
knn3.fit(train_set, train_labels)
test_results = knn3.predict(test_set)


accuracy_test = metrics.accuracy_score(test_labels, test_results)


import csv
c1 = np.zeros((30,3), dtype='int')
for i in range(1,31):
    print(f"Data Point Index: {i}      |       Predicted Class: {test_results[i-
1]}       |       True Class: {test_labels[i-1]}")
```

```
    c1[i-1] = [i, test_results[i-1], test_labels[i-1]]

header = ['Data Point Index', 'Predicted Class', 'True Class']
with open('finalresults.csv', 'w', encoding='UTF8', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(header)
    writer.writerows(c1)

with open('trainset.csv', 'w', encoding='UTF8', newline='') as f:
    writer = csv.writer(f)
    writer.writerows(train_set)

with open('testset.csv', 'w', encoding='UTF8', newline='') as f:
    writer = csv.writer(f)
    writer.writerows(test_set)

print("Total Accuracy on Test Set: %2.1f %%" %(accuracy_test*100))
```