Austin Clyde
Assignment 1

1. Normal v. Tumor
    a. In this part, I use a random forest classifier using entropy as the criterion along with 250 estimators. Random forest does generally benefit from feature selection, but it is not a requirement as it can independently compute feature importance and split on features of its own choosing. There is no data normalization as random forest with somewhat invariant to this. The result table is below, indicating it solved the problem quite well regardless of the dataset. Since one dataset is a superset of the other, this would be expected (with a possible boost from the coding subset as it would decrease the general noise form the larger dataset). I think results are possibly 0.5% off from the highest result I got a few years ago studying this problem closely. As one can see, that in general using coding features, a smaller set size, bodes well with classical machine learning methods.

Using all as input features.

	acc	confusion	f1	r2
0	0.964286	[[136 4] [6 134]]	0.964029	0.857143
1	0.967857	[[133 7] [2 138]]	0.968421	0.871429
2	0.967857	[[137 3] [6 134]]	0.967509	0.871429
3	0.967857	[[136 4] [5 135]]	0.967742	0.871429
4	0.975	[[136 4] [3 137]]	0.975089	0.9

Using coding as input features.

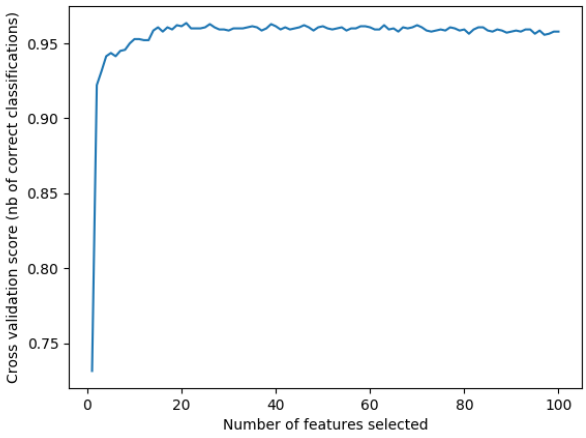
	acc	confusion	f1	r2
0	0.967857	[[135 5] [4 136]]	0.967972	0.871429
1	0.964286	[[136 4] [6 134]]	0.964029	0.857143
2	0.971429	[[137 3] [5 135]]	0.971223	0.885714
3	0.971429	[[136 4] [4 136]]	0.971429	0.885714
4	0.985714	[[137 3] [1 139]]	0.985816	0.942857

b. In this section, I run across multiple standard ML classifiers based on a list I found on scikit learn website. I use data normalization as some of the methods listed are sensitive to scale. I set a few hyperparameters to increase learning speed for a few of them, but in general use the default parameters. The table indicates that Linear SVM is the best, which I found quite interesting given AdaBoost or Random Forest would typically be my guess as the highest performers (of course, with optimizing the parameters a bit, we know from the previous part Random Forest can actually get much higher scores than the basic linear SVM, but in an out of box performance test, it perofmrs much better).

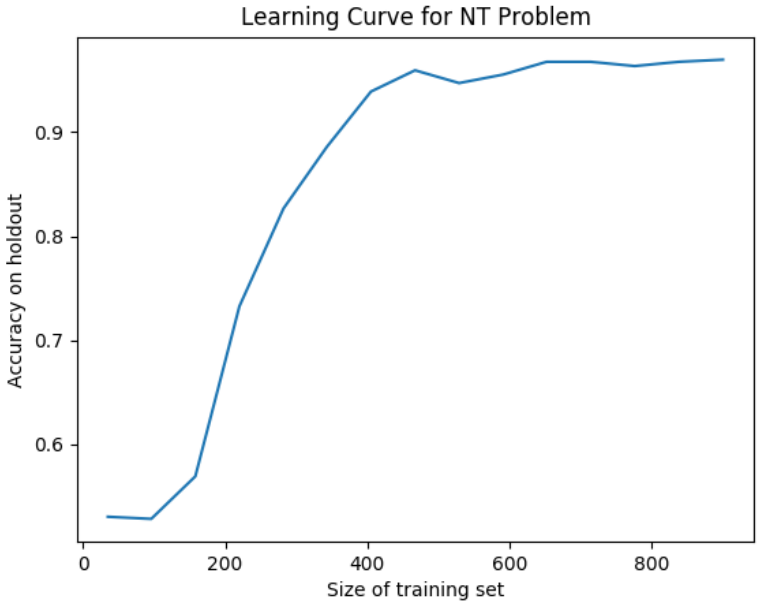
	acc	confusion	f1	r2
Gaussian Process	0.5	[[140 0] [140 0]]	0	-1
QDA	0.614286	[[90 50] [58 82]]	0.602941	-0.542857
Nearest Neighbors	0.792857	[[137 3] [55 85]]	0.745614	0.171429
Decision Tree	0.903571	[[125 15] [12 128]]	0.904594	0.614286
Naive Bayes	0.921429	[[130 10] [12 128]]	0.920863	0.685714
RBF SVM	0.942857	[[129 11] [5 135]]	0.944056	0.771429
Random Forest	0.95	[[136 4] [10 130]]	0.948905	0.8
Logistic Regression	0.953571	[[137 3] [10 130]]	0.952381	0.814286
AdaBoost	0.960714	[[137 3] [8 132]]	0.96	0.842857
Linear SVM	0.964286	[[136 4] [6 134]]	0.964029	0.857143

c. In this section I consider only the coding dataset and reduce the feature set to 100 solely based on mutual information score between each feature and the target. From there, I apply a recursive feature eliminator using a random forest with the same criteria from part a. As expected, there is no reduction in model performance as shown in the plot below. I would argue that we do not see a perofmorance boost in this case as the number of features is so small that are

informative that the initial random forest model probably learns a quite simple decision splitting pattern.



d. In this part I train a Keras deep neural network three hidden layers and a softmax output layer. I train using categorical crossentropy and SGD. The results are decent but could probably be made better with performing some hyperparameter optimization, especially along the model as I attempted to make the model as small as possible (mostly out of curiosity and not wanting to train on a GPU). To create the learning curve, I created a standard holdout set and then take varying sizes of the training set (using a stratify test split). I use a normalized version of the coding dataset.



	acc	confusion	f1	r2
0	0.530612	[[227 18] [212 33]]	0.222973	-0.877551
1	0.528571	[[21 224] [7 238]]	0.673267	-0.885714
2	0.569388	[[34 211]	0.699001	-0.722449

s

```
|    |          |  [  0 245]] |          |            |
|  3 | 0.732653 | [[126 119]  | 0.78057  | -0.0693878 |
|    |          |  [ 12 233]] |          |            |
|  4 | 0.826531 | [[164  81]  | 0.850088 |  0.306122  |
|    |          |  [  4 241]] |          |            |
|  5 | 0.885714 | [[194  51]  | 0.895522 |  0.542857  |
|    |          |  [  5 240]] |          |            |
|  6 | 0.938776 | [[222  23]  | 0.940711 |  0.755102  |
|    |          |  [  7 238]] |          |            |
|  7 | 0.959184 | [[236   9]  | 0.959016 |  0.836735  |
|    |          |  [ 11 234]] |          |            |
|  8 | 0.946939 | [[229  16]  | 0.947581 |  0.787755  |
|    |          |  [ 10 235]] |          |            |
|  9 | 0.955102 | [[231  14]  | 0.955645 |  0.820408  |
|    |          |  [  8 237]] |          |            |
| 10 | 0.967347 | [[237   8]  | 0.967347 |  0.869388  |
|    |          |  [  8 237]] |          |            |
| 11 | 0.967347 | [[238   7]  | 0.967213 |  0.869388  |
|    |          |  [  9 236]] |          |            |
| 12 | 0.963265 | [[237   8]  | 0.963115 |  0.853061  |
|    |          |  [ 10 235]] |          |            |
| 13 | 0.967347 | [[239   6]  | 0.967078 |  0.869388  |
|    |          |  [ 10 235]] |          |            |
| 14 | 0.969388 | [[237   8]  | 0.96945  |  0.877551  |
|    |          |  [  7 238]] |          |            |
+----+----------+-------------+----------+------------+
```

e. For the extra-credit, I implemented the automatic learning with TPOT, which returned the following model (I did not have time to run this for a decent amount of time. The code is written though…)

2. Tumor Type
    a. In this part, I use a random forest classifier using entropy as the criterion along with 250 estimators. Random forest does generally benefit from feature selection, but it is not a requirement as it can independently compute feature importance and split on features of its own choosing. There is no data normalization as random forest with somewhat invariant to this. The result table is below, indicating it solved the problem quite well regardless of the dataset. Since one dataset is a superset of the other, this would be expected (with a possible boost from the coding subset as it would decrease the general noise form the larger dataset). In this case, it seems that some of the non-coding genes actually improved the prediction slightly. This may be because the RF model is simply learning the tissue type and non-coding genes will generally be more indicative of a tissue type due to epigenetic factors.

```
Using RandomForestClassifer with 500 estimators and 5-fold StratifiedKFold.
No normalization or scaling is applied (RF doesn't need it).
Using all as input features.
+---+----------+-----------------------------------------------------------+----------+----------+
|   |      acc | confusion                                                 |       f1 |       r2 |
|---+----------+-----------------------------------------------------------+----------+----------|
| 0 | 0.966667 | [[60  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]  | 0.966413 | 0.963536 |
|   |          |  [ 0 54  0  0  0  0  4  0  0  1  0  0  0  0  0  0  1  0]  |          |          |
|   |          |  [ 0  0 59  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0]  |          |          |
|   |          |  [ 0  0  2 57  0  0  0  0  0  0  1  0  0  0  0  0  0  0]  |          |          |
|   |          |  [ 0  0  0  0 60  0  0  0  0  0  0  0  0  0  0  0  0  0]  |          |          |
|   |          |  [ 0  0  0  0  0 56  2  0  0  0  1  0  0  1  0  0  0  0]  |          |          |
|   |          |  [ 0  2  0  0  1  8 48  0  0  0  1  0  0  0  0  0  0  0]  |          |          |
|   |          |  [ 0  0  0  0  0  0  0 60  0  0  0  0  0  0  0  0  0  0]  |          |          |
|   |          |  [ 0  0  0  0  0  0  0  0 60  0  0  0  0  0  0  0  0  0]  |          |          |
|   |          |  [ 0  1  0  0  0  0  0  0  0 59  0  0  0  0  0  0  0  0]  |          |          |
|   |          |  [ 0  0  0  0  0  0  0  0  0  0 57  0  0  1  0  0  2  0]  |          |          |
|   |          |  [ 1  1  0  0  0  0  0  0  0  0  0 58  0  0  0  0  0  0]  |          |          |
|   |          |  [ 0  0  0  0  0  0  0  0  0  0  0  0 60  0  0  0  0  0]  |          |          |
|   |          |  [ 0  0  0  0  0  0  0  0  0  0  0  0  0 60  0  0  0  0]  |          |          |
|   |          |  [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0 60  0  0  0]  |          |          |
|   |          |  [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 60  0  0]  |          |          |
|   |          |  [ 0  0  0  0  0  0  0  0  0  0  0  0  0  4  0  0 56  0]  |          |          |
```

		[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 60]]		
1	0.973148	[[60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	0.972835	0.943137
		[0 55 0 0 0 0 0 0 0 1 0 0 0 0 0 0 4 0]		
		[0 0 60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]		
		[0 0 0 59 0 0 0 0 0 0 1 0 0 0 0 0 0 0]		
		[0 0 0 0 60 0 0 0 0 0 0 0 0 0 0 0 0 0]		
		[0 0 0 0 0 59 1 0 0 0 0 0 0 0 0 0 0 0]		
		[0 0 0 0 0 11 48 0 0 0 0 0 0 0 0 1 0 0]		
		[0 0 0 0 0 0 0 59 0 0 0 1 0 0 0 0 0 0]		
		[0 0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0 0]		
		[0 0 0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0]		
		[1 4 0 0 0 0 1 0 0 1 53 0 0 0 0 0 0 0]		
		[0 0 0 0 0 0 0 0 0 0 0 60 0 0 0 0 0 0]		
		[0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0 0 0]		
		[0 0 0 0 1 0 0 0 0 0 0 0 0 59 0 0 0 0]		
		[0 0 0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0]		
		[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 60 0 0]		
		[0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 59 0]		
		[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 60]]		
2	0.962963	[[60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	0.962658	0.951187
		[0 54 0 0 0 0 1 0 0 0 0 0 0 0 0 0 5 0]		
		[0 0 57 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0]		
		[0 0 4 56 0 0 0 0 0 0 0 0 0 0 0 0 0 0]		
		[0 0 0 0 60 0 0 0 0 0 0 0 0 0 0 0 0 0]		
		[0 0 0 0 0 60 0 0 0 0 0 0 0 0 0 0 0 0]		
		[0 1 0 0 0 14 45 0 0 0 0 0 0 0 0 0 0 0]		
		[0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0 0 0]		
		[0 0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0 0]		
		[0 0 0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0]		
		[0 1 0 0 0 0 2 0 0 0 56 0 0 0 0 0 1 0]		
		[0 0 0 0 0 0 0 0 0 0 0 60 0 0 0 0 0 0]		
		[0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0 0 0]		
		[0 0 0 0 0 0 0 0 0 1 0 0 0 59 0 0 0 0]		
		[0 0 0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0]		
		[0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 58 1 0]		
		[0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 55 3]		
		[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 60]]		
3	0.968519	[[60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	0.968533	0.972136
		[0 59 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]		
		[0 0 57 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0]		
		[0 0 3 56 0 0 0 0 0 0 1 0 0 0 0 0 0 0]		
		[0 0 0 0 58 0 0 0 0 2 0 0 0 0 0 0 0 0]		
		[0 0 0 0 0 58 2 0 0 0 0 0 0 0 0 0 0 0]		
		[0 2 0 0 0 7 51 0 0 0 0 0 0 0 0 0 0 0]		
		[0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0 0 0]		
		[0 0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0 0]		
		[0 0 0 0 0 0 1 0 0 58 0 0 0 0 0 1 0 0]		
		[0 2 0 0 0 0 0 0 0 2 56 0 0 0 0 0 0 0]		
		[0 0 0 0 0 0 0 0 0 0 0 60 0 0 0 0 0 0]		
		[0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0 0 0]		
		[0 0 0 0 0 0 0 0 0 1 0 0 0 59 0 0 0 0]		
		[0 0 0 0 0 0 0 0 0 0 1 0 0 0 59 0 0 0]		
		[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 59 1 0]		
		[0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 57 0]		
		[0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 59]]		
4	0.963889	[[60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	0.963566	0.951703
		[0 55 0 0 0 0 1 0 0 0 1 0 0 1 0 2 0 0]		
		[0 0 56 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0]		
		[0 0 2 57 0 0 0 0 0 0 1 0 0 0 0 0 0 0]		
		[0 0 0 0 59 0 0 0 0 1 0 0 0 0 0 0 0 0]		
		[0 0 0 0 0 58 1 0 0 0 0 0 0 0 0 0 0 1]		
		[0 2 0 0 0 3 52 0 0 1 0 0 0 0 0 2 0 0]		
		[0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0 0 0]		
		[0 0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0 0]		
		[0 0 0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0]		
		[0 2 0 0 0 0 0 0 0 3 52 0 0 0 1 1 1 0]		
		[0 0 0 0 0 0 0 0 0 0 0 60 0 0 0 0 0 0]		
		[0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0 0 0]		
		[0 0 0 0 0 0 0 0 0 0 0 0 0 59 0 0 0 1]		
		[0 0 0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0]		
		[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 60 0 0]		
		[0 0 0 0 0 0 0 0 0 0 0 0 0 6 0 0 53 1]		
		[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 60]]		

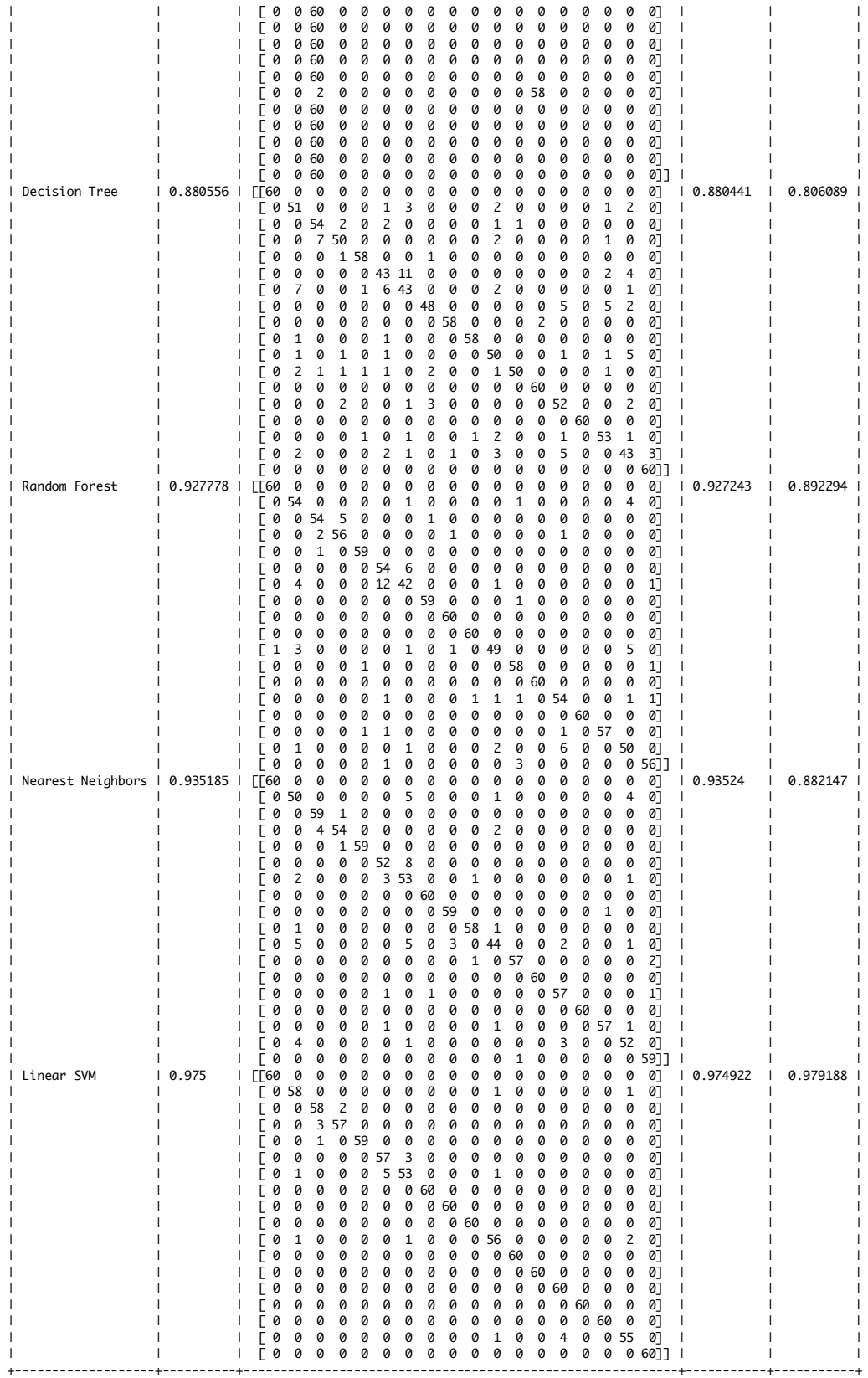
Using coding as input features.

		[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 60]]		
1	0.965741	[[60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	0.965594	0.953698
		[0 54 0 0 0 0 0 0 0 1 0 0 1 0 2 2 0]		
		[0 0 58 2 0 0 0 0 0 0 0 0 0 0 0 0 0]		
		[0 0 2 56 0 0 0 0 0 0 2 0 0 0 0 0 0]		
		[0 0 0 1 58 0 0 0 0 1 0 0 0 0 0 0 0]		
		[0 0 0 0 0 58 2 0 0 0 0 0 0 0 0 0 0]		
		[0 2 0 0 0 8 47 0 0 1 0 0 0 0 2 0 0]		
		[0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0 0]		
		[0 0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0]		
		[0 2 0 0 0 0 0 0 0 57 0 0 0 0 1 0 0]		
		[0 5 0 0 0 0 1 0 0 1 50 0 0 0 0 3 0]		
		[0 0 0 0 0 0 0 0 0 0 0 60 0 0 0 0 0]		
		[0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0 0]		
		[0 0 0 0 0 0 0 0 0 1 1 0 0 58 0 0 0]		
		[0 0 0 0 0 0 0 0 0 0 1 0 0 0 59 0 0]		
		[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 59 1]		
		[0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 58]		
		[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 60]]		

		[0 60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]			
		[0 0 57 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0]			
		[0 0 1 57 0 0 0 0 0 0 2 0 0 0 0 0 0 0]			
		[0 0 0 0 59 0 0 0 0 1 0 0 0 0 0 0 0 0]			
		[0 0 0 0 0 56 1 0 0 0 0 0 0 1 0 1 0 1]			
		[0 1 0 0 0 7 52 0 0 0 0 0 0 0 0 0 0 0]			
		[0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0 0 0]			
		[0 0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0 0]			
		[0 0 0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0]			
		[0 1 0 0 0 0 0 0 0 1 56 0 0 0 0 0 2 0]			
		[0 2 0 0 0 0 0 0 0 1 0 57 0 0 0 0 0 0]			
		[0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0 0 0]			
		[0 0 0 0 0 0 0 1 0 1 0 0 0 58 0 0 0 0]			
		[0 0 0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0]			
		[0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 59 0 0]			
		[0 1 0 0 0 0 0 0 0 0 0 0 0 5 0 0 52 2]			
		[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 60]]			
2	0.968519	[[60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	0.968237	0.972962	
		[0 54 0 0 0 0 4 0 0 0 1 0 0 0 0 0 1 0]			
		[0 0 57 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0]			
		[0 0 1 59 0 0 0 0 0 0 0 0 0 0 0 0 0 0]			
		[0 0 0 0 60 0 0 0 0 0 0 0 0 0 0 0 0 0]			
		[0 0 0 0 0 59 1 0 0 0 0 0 0 0 0 0 0 0]			
		[0 2 0 0 0 7 49 0 0 0 1 0 0 0 0 1 0 0]			
		[0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0 0 0]			
		[0 0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0 0]			
		[0 0 0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0]			
		[0 1 0 0 0 0 1 0 0 0 56 1 0 0 1 0 0 0]			
		[0 0 0 0 0 0 1 0 0 0 0 59 0 0 0 0 0 0]			
		[0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0 0 0]			
		[0 0 0 0 0 0 0 0 0 0 0 0 0 59 0 0 1 0]			
		[0 0 0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0]			
		[0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 59 0 0]			
		[0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 55 1]			
		[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 60]]			
3	0.958333	[[60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	0.957992	0.953354	
		[0 53 0 0 0 0 3 0 0 0 0 0 0 0 0 1 3 0]			
		[0 0 57 2 0 0 0 0 0 1 0 0 0 0 0 0 0 0]			
		[0 0 4 56 0 0 0 0 0 0 0 0 0 0 0 0 0 0]			
		[0 0 0 0 60 0 0 0 0 0 0 0 0 0 0 0 0 0]			
		[0 0 0 0 0 57 2 0 0 0 0 1 0 0 0 0 0 0]			
		[0 1 0 0 0 13 45 0 0 0 1 0 0 0 0 0 0 0]			
		[0 0 0 0 0 0 0 59 0 0 0 1 0 0 0 0 0 0]			
		[0 0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0 0]			
		[0 0 0 0 0 0 1 0 0 59 0 0 0 0 0 0 0 0]			
		[0 1 0 0 0 0 0 0 0 3 55 0 0 1 0 0 0 0]			
		[0 1 0 0 0 0 0 0 0 0 0 59 0 0 0 0 0 0]			
		[0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0 0 0]			
		[0 0 0 0 0 0 0 0 0 0 1 0 0 58 0 0 1 0]			
		[0 0 0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0]			
		[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 60 0 0]			
		[0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 57 0]			
		[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 60]]			
4	0.969444	[[60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]	0.96922	0.96106	
		[0 57 0 0 0 0 1 0 0 0 0 0 0 0 0 0 2 0]			
		[0 0 57 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0]			
		[0 0 1 59 0 0 0 0 0 0 0 0 0 0 0 0 0 0]			
		[0 0 0 0 59 0 0 0 0 1 0 0 0 0 0 0 0 0]			
		[0 0 0 0 0 0 59 1 0 0 0 0 0 0 0 0 0 0]			
		[0 2 0 0 1 7 49 0 0 0 1 0 0 0 0 0 0 0]			
		[0 0 0 0 0 0 0 59 0 0 0 0 0 1 0 0 0 0]			
		[0 0 0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0]			
		[0 0 0 0 0 0 0 0 0 0 60 0 0 0 0 0 0 0]			
		[0 4 0 0 0 0 1 0 0 0 54 0 0 0 0 0 1 0]			
		[0 0 0 0 0 0 0 0 0 1 0 59 0 0 0 0 0 0]			
		[0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0 0 0]			
		[0 0 0 0 0 1 0 0 0 0 0 0 0 58 0 0 0 1]			
		[0 0 0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0]			
		[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 60 0 0]			
		[0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 58 0]			
		[0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 59]]			

b. In this section, I run across multiple standard ML classifiers based on a list I found on scikit learn website. I use data normalization as some of the methods listed are sensitive to scale. I set a few hyperparameters to increase learning speed for a few of them, but in general use the default parameters. In this case, a linear SVM performed the best, which was very surprising to me—espically given the large class size.

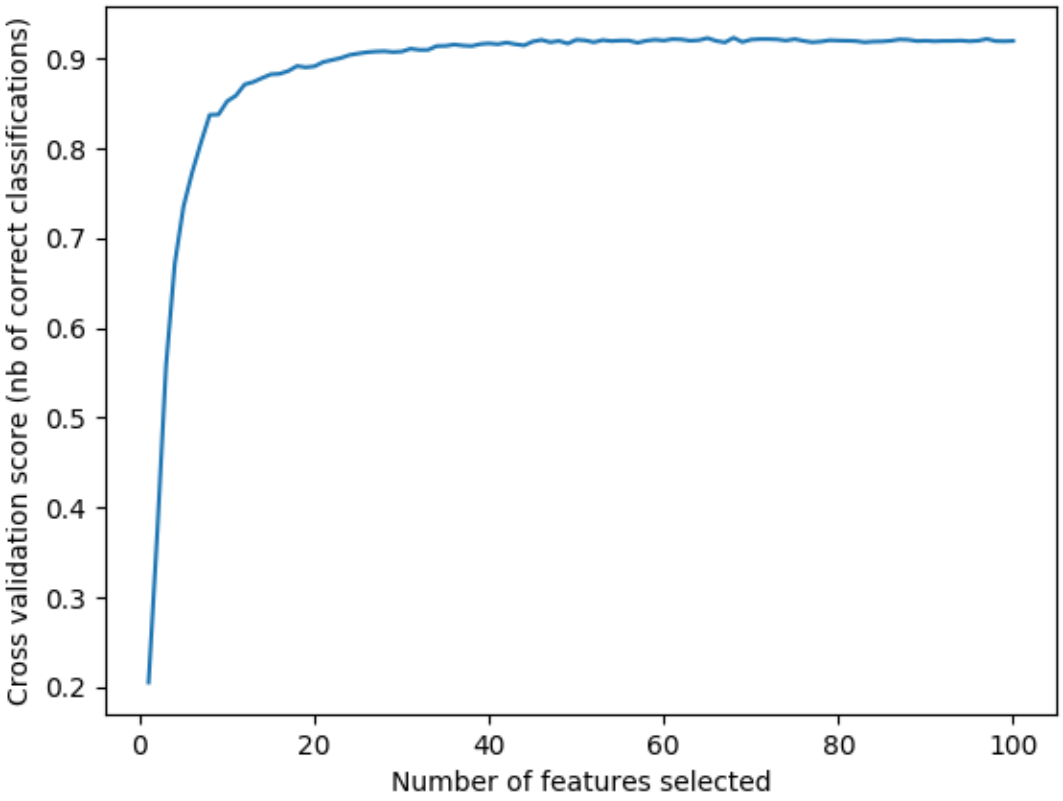
	acc	confusion	f1	r2
AdaBoost	0.109259	[[0 0 0 0 0 0 0 0 0 0 0 60 0 0 0 0 0 0]	0.0611371	-1.61163
		[0 0 60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]		
		[0 0 60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]		
		[0 0 60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]		
		[0 0 60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]		
		[0 0 60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]		
		[0 0 60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]		



Decision Tree	0.880556	[[60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0] [0 51 0 0 0 1 3 0 0 0 2 0 0 0 0 1 2 0] [0 0 54 2 0 2 0 0 0 0 1 1 0 0 0 0 0 0] [0 0 7 50 0 0 0 0 0 0 2 0 0 0 0 1 0 0] [0 0 0 1 58 0 0 1 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 43 11 0 0 0 0 0 0 0 0 2 4 0] [0 7 0 0 1 6 43 0 0 0 2 0 0 0 0 0 1 0] [0 0 0 0 0 0 0 48 0 0 0 0 0 5 0 5 2 0] [0 0 0 0 0 0 0 0 58 0 0 0 2 0 0 0 0 0] [0 1 0 0 0 1 0 0 0 58 0 0 0 0 0 0 0 0] [0 1 0 1 0 1 0 0 0 0 50 0 0 1 0 1 5 0] [0 2 1 1 1 1 0 2 0 0 1 50 0 0 0 1 0 0] [0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0 0 0] [0 0 0 2 0 0 1 3 0 0 0 0 0 52 0 0 2 0] [0 0 0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0] [0 0 0 0 1 0 1 0 0 1 2 0 0 1 0 53 1 0] [0 2 0 0 0 2 1 0 1 0 3 0 0 5 0 0 43 3] [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 60]]	0.880441	0.806089
Random Forest	0.927778	[[60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0] [0 54 0 0 0 0 1 0 0 0 0 1 0 0 0 0 4 0] [0 0 54 5 0 0 0 1 0 0 0 0 0 0 0 0 0 0] [0 0 2 56 0 0 0 0 1 0 0 0 0 1 0 0 0 0] [0 0 1 0 59 0 0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 54 6 0 0 0 0 0 0 0 0 0 0 0] [0 4 0 0 0 12 42 0 0 0 1 0 0 0 0 0 0 1] [0 0 0 0 0 0 0 59 0 0 0 1 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0] [1 3 0 0 0 0 1 0 1 0 49 0 0 0 0 0 5 0] [0 0 0 0 1 0 0 0 0 0 0 58 0 0 0 0 0 1] [0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0 0 0] [0 0 0 0 0 1 0 0 0 1 1 1 0 54 0 0 1 1] [0 0 0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0] [0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 57 0 0] [0 1 0 0 0 0 1 0 0 0 2 0 0 6 0 0 50 0] [0 0 0 0 0 1 0 0 0 0 3 0 0 0 0 0 0 56]]	0.927243	0.892294
Nearest Neighbors	0.935185	[[60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0] [0 50 0 0 0 0 5 0 0 0 1 0 0 0 0 0 4 0] [0 0 59 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0] [0 0 4 54 0 0 0 0 0 0 2 0 0 0 0 0 0 0] [0 0 0 1 59 0 0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 52 8 0 0 0 0 0 0 0 0 0 0 0] [0 2 0 0 0 3 53 0 0 1 0 0 0 0 0 0 1 0] [0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 59 0 0 0 0 0 0 1 0 0] [0 1 0 0 0 0 0 0 0 58 1 0 0 0 0 0 0 0] [0 5 0 0 0 0 5 0 3 0 44 0 0 2 0 0 1 0] [0 0 0 0 0 0 0 0 0 1 0 57 0 0 0 0 0 2] [0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0 0 0] [0 0 0 0 0 1 0 1 0 0 0 0 0 57 0 0 0 1] [0 0 0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0] [0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 57 1 0] [0 4 0 0 0 0 1 0 0 0 0 0 0 3 0 0 52 0] [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 59]]	0.93524	0.882147
Linear SVM	0.975	[[60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0] [0 58 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0] [0 0 58 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0] [0 0 3 57 0 0 0 0 0 0 0 0 0 0 0 0 0 0] [0 0 1 0 59 0 0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 57 3 0 0 0 0 0 0 0 0 0 0 0] [0 1 0 0 0 5 53 0 0 0 1 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 60 0 0 0 0 0 0 0 0] [0 1 0 0 0 0 1 0 0 0 56 0 0 0 0 0 2 0] [0 0 0 0 0 0 0 0 0 0 0 60 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0 0 0 60 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 60 0 0] [0 0 0 0 0 0 0 0 0 0 1 0 0 4 0 0 55 0] [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 60]]	0.974922	0.979188

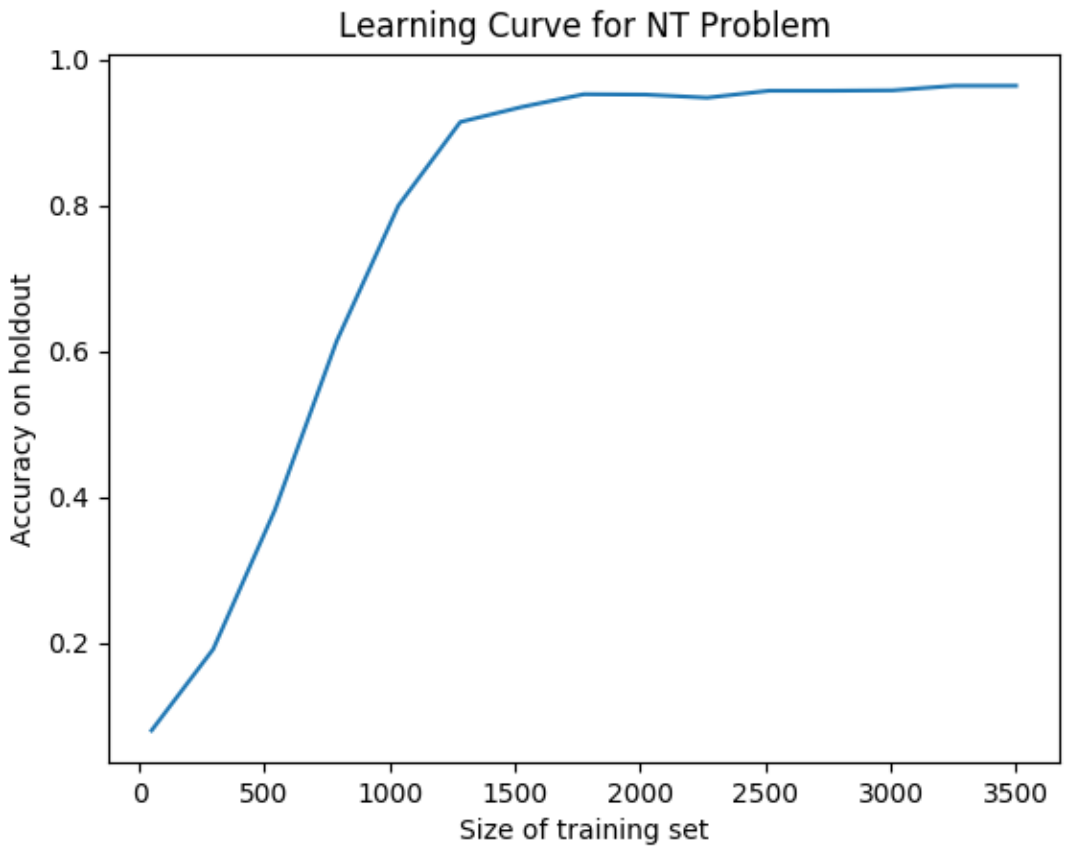
c. This performed well <20 features. This is the same method as in part 1 c. In the past I used a fancy joint mutual information maximalization method to increase

accuracy to 96% with five features. This is accomplished by creating a set of features that have a high mutual information with the target but I(X_1, X_2; Y) is low (aka the features generally share only new information about the target versus redundant information. Redudnant information often shows up with these RFE or select some k best based on a metric as the top features are typically calculated indepent of each other and just with the target).



d. I again use categorical cross entropy with a standard holdout set and varying sizes of the test set using the normalized coding dataset as it will be faster to train on the CPU. The results are expected, requiring a larger training set size given the greater number of classes from the previous part of the homework. I think to improve the overall training accuracy, I should have designed the network to have more hidden layers and added some depth to those hidden

layers.


Learning Curve for NT Problem
Accuracy on holdout
1.0
0.8
0.6
0.4
0.2
0
500
1000
1500
2000
2500
3000
3500
Size of training set