CS205 Spring 2019
Project 2 Report:
Cancer prediction using NHANES Dataset

Michael Sun
505228712
UCLA CS

Abstract

Cancer as a whole is one of the most deadly diseases in the world, second only to cardiovascular disease. Every cancer is different, and there is much study into predicting and treating all forms of cancer. However, there is little research on predicting cancer as a whole. In the course of this research, I analyzed and used the NHANES dataset to predict cancer based on a variety of features and using a variety of strategies.

1 Introduction

The goal of this project is to predict whether someone has or is at rick for any kind of cancer. I selected and manipulated data and programmed statistical models to make these predictions. I consulted many resources online, including research papers, technical blog posts, and coding forums. Environment: I set up my notebook on an AWS Sagemaker server for ease of remote access and consistent computation.
Specs:

- ml.t2.medium instance
- 2 virtual CPUs
- 4 GIB RAM
- Python 3.6
- Pytorch 1.0

2 Methodology

2.1 Data Source

The data comes from NHANES, or the National Health and Nutrition Examination Survey. NHANES "is a program of studies designed to assess the health and nutritional status of adults and children in the United States. The survey is unique in that it combines interviews and physical examinations." In this study I focused on using data from the 2015-2016 data collection, but also included data from older surveys and records if they contained the same features. The NHANES.py script labels 0 as has cancer, and 1 as no cancer (healthy).

2.2 Baseline

Sample code was given for predicting arthritis - more specifically predicting the target "MCQ160 - Doctor ever said you had arthritis." It used about 30 features.

The test accuracies using various classification methods are as follows. This served as the baseline performance to compare my new model against.

Random Forest	0.756
Support Vector	0.758
Logistic Regression	0.767

```
accu_tst_RFC 0.714
             precision    recall  f1-score   support

          0       0.70      0.76      0.73       500
          1       0.74      0.67      0.70       500

avg / total       0.72      0.71      0.71      1000

[[381 119]
 [167 333]]
```

Figure: Base RFC stats

```
accu_tst_SVC 0.711
             precision    recall  f1-score   support

          0       0.68      0.78      0.73       500
          1       0.75      0.64      0.69       500

avg / total       0.72      0.71      0.71      1000

[[392 108]
 [181 319]]
```

Figure: Base SVC stats

```
accu_tst_LR 0.702
             precision    recall  f1-score   support

          0       0.67      0.78      0.72       500
          1       0.74      0.62      0.68       500

avg / total       0.71      0.70      0.70      1000

[[390 110]
 [188 312]]
```

Figure: Base LR stats

2.3 Data Preparation

I selectively loaded data by extracting the desired features via their feature code. This allowed me to flexibly change the features I used.

2.4 Feature Selection

2.4.1 Exhaustive search

The first method I attempted to select features was an exhaustive search to evaluate correlation. I initially hoped to automate the entire process of extracting features from the data set, applying preprocessing, and calculating mutual information. However, automating preprocessing proved unsuccessful due to the intricate nature of selecting the correct preprocessing procedure for each type of data. Different data such as binary, categorical, continuous, and sparse each need unique preprocessing procedures. I also wish to note that a classmate who did attempt to run an exhaustive mutual information analysis stated that all the mutual information he extracted were too low and similar to use. Thus I switched to manual feature selection.

2.4.2 Manual Selection

Because of the difficulty of exhaustive search and limitation of time, I tried instead to select many features by hand via intuition. This allowed me to appropriately curate preprocessing methods for each feature. The downside was that, due the multitude of features, it was impossible to analyze and preprocess every feature by hand, as that alone would have cost weeks. Thus I used my intuition, prior knowledge, and research, to manually select features I assumed to be the most predictive of cancer. Because most research focuses on specific types of cancer, I browsed much research of more specific cancer research, thus treating my project as a sort of ensemble research. I cross referenced the available NHANES features with existing research papers to find features that existed in prior research.

- Vaginal swab testing [1]
- Blood features [2] [3]
- BMI and body shape/composition [4] [5] [6]
- Age [7]
- Cultural background, socioeconomic status, and education [8] [9]
- Alcohol and drugs [10] [11]

2.5 Predictive Modeling - Standard

The model itself is arguably the most important part of the predictive modeling process. There are many models so I researched some of the most popular and well-performing models used today for our type of data, especially in the application of medical analysis. I also considered factors such as complexity and efficiency. [12] [13] [14]

- Random Forest Classifier [15] - This is an ensemble learning method. It runs many different simpler decision tree models and uses bagging and boosting to compile their predictive power together to form a super-model. Ensemble methods can be very powerful because they are not confined to a single scope of the data.
- Support Vector Classifier [16] - The SVM is a straightforward but still-popular model.
- Logistic Regression - a statistical model that computes probabilities for each output class.
- Linear Discriminant Analysis (LDA) - "attempts to express one dependent variable as a linear combination of other features or measurements." This is a commonly used model. [17]

2.6 Data Size

I ran into an interesting observation after including more data (from the baseline) and increasing the train and test set sizes. I increased the training set size from 5000 to 20000, and testing set from 2000 to 10000. After doing so, the precision and recall for each classifier became heavily skewed toward the positive class and achieved much lower precision and recall for the negative class while increasing for the positive class. The classifiers are seemingly weighted too. It can be that the

[1] https://jamanetwork.com/journals/jama/article-abstract/192260
[2] https://www.sciencedirect.com/science/article/pii/S0140673696034307
[3] https://www.sciencedirect.com/science/article/abs/pii/S0168827806002972
[4] https://www.sciencedirect.com/science/article/pii/S014067360860269X
[5] https://www.nature.com/articles/0802606
[6] https://academic.oup.com/ajcn/article/80/4/1012/4690349
[7] https://www.nature.com/articles/ng0299_163
[8] https://onlinelibrary.wiley.com/doi/full/10.3322/canjclin.54.2.78
[9] https://onlinelibrary.wiley.com/doi/full/10.3322/canjclin.56.3.168
[10] https://www.sciencedirect.com/science/article/abs/pii/S1470204506705770
[11] https://onlinelibrary.wiley.com/doi/full/10.1002/cncr.27554
[12] https://www.sciencedirect.com/science/article/pii/S1532046401910044
[13] https://www.kaggle.com/jeffd23/10-classifier-showdown-in-scikit-learn
[14] https://towardsdatascience.com/which-machine-learning-model-to-use-db5fdf37f3dd
[15] https://www.researchgate.net/profile/Andy_Liaw/publication/228451484_Classification_and_Regression_by_RandomForest/links/53fb24c
and-Regression-by-RandomForest.pdf
[16]
[17] https://en.wikipedia.org/wiki/Linear_discriminant_analysis

importance of rejecting False Negatives (False cancer) starts outweighing the importance of detecting True Positives(True has-cancer). It seems that a tradeoff must be made between the two. Both have their pros and cons. With less data in the model, there is more of a chance someone with cancer being detected as such, but many non-cancer patients will be falsely diagnosed. With more data, fewer patients will be falsely told they have cancer, but fewer of those who have or are at risk for cancer will be told. For the rest of this project, I will use the larger dataset that is more representative of all the data. Because this model so far is not highly accurate and should only be used as a supplement to an expert's judgement, I would personally shy away from sowing worry into a healthy patient's mind with a wrong prediction, as this itself can be dramatically detrimental to their health and even contribute to unnecessary CAT scans.

```
Random Forest Classifier 0.6897572528123149
              precision    recall  f1-score   support

           0       0.59      0.76      0.67       689
           1       0.79      0.64      0.71      1000

   micro avg       0.69      0.69      0.69      1689
   macro avg       0.69      0.70      0.69      1689
weighted avg       0.71      0.69      0.69      1689

[[521 168]
 [356 644]]
Support Vector Classifier 0.6909413854351687
              precision    recall  f1-score   support

           0       0.59      0.79      0.68       689
           1       0.81      0.62      0.70      1000

   micro avg       0.69      0.69      0.69      1689
   macro avg       0.70      0.71      0.69      1689
weighted avg       0.72      0.69      0.69      1689

[[544 145]
 [377 623]]
Logistic Regression 0.6962699822380106
              precision    recall  f1-score   support

           0       0.60      0.77      0.68       689
           1       0.80      0.64      0.71      1000

   micro avg       0.70      0.70      0.70      1689
   macro avg       0.70      0.71      0.69      1689
weighted avg       0.72      0.70      0.70      1689

[[533 156]
 [357 643]]
Linear Discriminant Analysis 0.6998223801065719
              precision    recall  f1-score   support

           0       0.60      0.79      0.68       689
           1       0.81      0.64      0.72      1000

   micro avg       0.70      0.70      0.70      1689
   macro avg       0.71      0.71      0.70      1689
weighted avg       0.73      0.70      0.70      1689

[[541 148]
 [359 641]]
```

Figure: Classifiers with 5000/2000 train/test split

```
Random Forest Classifier 0.8582510578279267
              precision    recall  f1-score   support

           0       0.36      0.25      0.30       672
           1       0.90      0.94      0.92      5000

   micro avg       0.86      0.86      0.86      5672
   macro avg       0.63      0.60      0.61      5672
weighted avg       0.84      0.86      0.85      5672

[[ 171  501]
 [ 303 4697]]
Support Vector Classifier 0.6694287729196051
              precision    recall  f1-score   support

           0       0.23      0.74      0.35       672
           1       0.95      0.66      0.78      5000

   micro avg       0.67      0.67      0.67      5672
   macro avg       0.59      0.70      0.56      5672
weighted avg       0.86      0.67      0.73      5672

[[ 500  172]
 [1703 3297]]
Logistic Regression 0.6771861777150917
              precision    recall  f1-score   support

           0       0.23      0.74      0.35       672
           1       0.95      0.67      0.79      5000

   micro avg       0.68      0.68      0.68      5672
   macro avg       0.59      0.70      0.57      5672
weighted avg       0.87      0.68      0.73      5672

[[ 498  174]
 [1657 3343]]
Linear Discriminant Analysis 0.7928420310296191
              precision    recall  f1-score   support

           0       0.29      0.50      0.37       672
           1       0.93      0.83      0.88      5000

   micro avg       0.79      0.79      0.79      5672
   macro avg       0.61      0.67      0.62      5672
weighted avg       0.85      0.79      0.82      5672

[[ 338  334]
 [ 841 4159]]
```

Figure: Classifiers with 20000/10000 train/test split

2.7 Preprocessing and Feature Engineering

2.7.1 Principal Component Analysis

Principal component analysis (PCA) is a popular strategy for data analysis and decomposition. It reduces data complexity by reducing the data to fewer (important) dimensions. [18] I tested decomposing the data into different numbers of components. For Random Forest and LDA, the testing shows that the more components that are decomposed, the more the models skew toward predicting the major class, improving the weighted average at the cost of accuracy of the minor class. Beyond 10 components, the results do not change. It is difficult to definitively say whether this is better or worse, but should be assessed more before real use since the predictions are so skewed to one side. For SVC and LR PCA seems to simply reduce performance. test

[18] https://www.cs.cmu.edu/ elaw/papers/pca.pdf

```
Random Forest Classifier 0.6969430780042164
              precision    recall  f1-score   support

           0       0.14      0.29      0.19       692
           1       0.88      0.75      0.81      5000

   micro avg       0.70      0.70      0.70      5692
   macro avg       0.51      0.52      0.50      5692
weighted avg       0.79      0.70      0.74      5692

[[ 202  490]
 [1235 3765]]
Support Vector Classifier 0.5989107519325368
              precision    recall  f1-score   support

           0       0.18      0.64      0.28       692
           1       0.92      0.59      0.72      5000

   micro avg       0.60      0.60      0.60      5692
   macro avg       0.55      0.62      0.50      5692
weighted avg       0.83      0.60      0.67      5692

[[ 441  251]
 [2032 2968]]
Logistic Regression 0.6126141953619114
              precision    recall  f1-score   support

           0       0.18      0.61      0.28       692
           1       0.92      0.61      0.74      5000

   micro avg       0.61      0.61      0.61      5692
   macro avg       0.55      0.61      0.51      5692
weighted avg       0.83      0.61      0.68      5692

[[ 421  271]
 [1934 3066]]
Linear Discriminant Analysis 0.8691145467322557
              precision    recall  f1-score   support

           0       0.14      0.01      0.03       692
           1       0.88      0.99      0.93      5000

   micro avg       0.87      0.87      0.87      5692
   macro avg       0.51      0.50      0.48      5692
weighted avg       0.79      0.87      0.82      5692

[[  10  682]
 [  63 4937]]
```

Figure 1: PCA 1 Component Performance

```
Random Forest Classifier 0.8134615384615385
              precision    recall  f1-score   support

           0       0.14      0.09      0.11       720
           1       0.88      0.92      0.90      5000

   micro avg       0.81      0.81      0.81      5720
   macro avg       0.51      0.51      0.50      5720
weighted avg       0.78      0.81      0.80      5720

[[  67  653]
 [ 414 4586]]
Support Vector Classifier 0.5641608391608391
              precision    recall  f1-score   support

           0       0.13      0.42      0.20       720
           1       0.88      0.58      0.70      5000

   micro avg       0.56      0.56      0.56      5720
   macro avg       0.50      0.50      0.45      5720
weighted avg       0.78      0.56      0.64      5720

[[ 303  417]
 [2076 2924]]
Logistic Regression 0.5087412587412588
              precision    recall  f1-score   support

           0       0.12      0.46      0.19       720
           1       0.87      0.52      0.65      5000

   micro avg       0.51      0.51      0.51      5720
   macro avg       0.49      0.49      0.42      5720
weighted avg       0.77      0.51      0.59      5720

[[ 332  388]
 [2422 2578]]
Linear Discriminant Analysis 0.8741258741258742
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       720
           1       0.87      1.00      0.93      5000

   micro avg       0.87      0.87      0.87      5720
   macro avg       0.44      0.50      0.47      5720
weighted avg       0.76      0.87      0.82      5720

[[   0  720]
 [   0 5000]]
```

Figure 2: PCA 2 Components Performance

2.7.2 Singular Value Decomposition

SVD is a similar data decomposition method to PCA, and causes the models to behave similarly to PCA with the same numbers of components.

test

2.8 Predictive Modeling - Neural Network

I implemented a deep neural network using Pytorch 1.0 in python 3.6. This was my primary goal and point of analysis in this project.

After researching and experimenting [19] [20] I chose the following layers and hidden neurons for my network.

```
    self.fc1 = nn.Linear(75, 120)
    self.fc2 = nn.Linear(120, 400)
    self.fc3 = nn.Linear(400, 800)
    self.fc4 = nn.Linear(800, 300)
```

[19] https://www.researchgate.net/publication/258393467_Review_on_Methods_to_Fix_Number_of_Hidden_Neurons_in_Neural_Networks

[20] https://towardsdatascience.com/beginners-ask-how-many-hidden-layers-neurons-to-use-in-artificial-neural-networks-51466afa0d3e

```
Random Forest Classifier 0.8522270742358079
              precision    recall  f1-score   support

           0       0.11      0.02      0.04       725
           1       0.87      0.97      0.92      5000

   micro avg       0.85      0.85      0.85      5725
   macro avg       0.49      0.50      0.48      5725
weighted avg       0.78      0.85      0.81      5725

[[  18  707]
 [ 139 4861]]
Support Vector Classifier 0.514585152838428
              precision    recall  f1-score   support

           0       0.13      0.49      0.20       725
           1       0.88      0.52      0.65      5000

   micro avg       0.51      0.51      0.51      5725
   macro avg       0.50      0.51      0.43      5725
weighted avg       0.78      0.51      0.59      5725

[[ 357  368]
 [2411 2589]]
Logistic Regression 0.5093449781659388
              precision    recall  f1-score   support

           0       0.12      0.48      0.20       725
           1       0.87      0.51      0.65      5000

   micro avg       0.51      0.51      0.51      5725
   macro avg       0.50      0.50      0.42      5725
weighted avg       0.78      0.51      0.59      5725

[[ 346  379]
 [2430 2570]]
Linear Discriminant Analysis 0.8733624454148472
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       725
           1       0.87      1.00      0.93      5000

   micro avg       0.87      0.87      0.87      5725
   macro avg       0.44      0.50      0.47      5725
weighted avg       0.76      0.87      0.81      5725

[[   0  725]
 [   0 5000]]
```

Figure 3: PCA 3 Components Performance

```
Random Forest Classifier 0.877906976744186
              precision    recall  f1-score   support

           0       0.05      0.00      0.00       676
           1       0.88      1.00      0.93      5000

   micro avg       0.88      0.88      0.88      5676
   macro avg       0.47      0.50      0.47      5676
weighted avg       0.78      0.88      0.82      5676

[[   1  675]
 [  18 4982]]
Support Vector Classifier 0.5785764622973926
              precision    recall  f1-score   support

           0       0.13      0.44      0.20       676
           1       0.89      0.60      0.71      5000

   micro avg       0.58      0.58      0.58      5676
   macro avg       0.51      0.52      0.46      5676
weighted avg       0.80      0.58      0.65      5676

[[ 297  379]
 [2013 2987]]
Logistic Regression 0.4714587737843552
              precision    recall  f1-score   support

           0       0.12      0.56      0.20       676
           1       0.89      0.46      0.60      5000

   micro avg       0.47      0.47      0.47      5676
   macro avg       0.50      0.51      0.40      5676
weighted avg       0.79      0.47      0.56      5676

[[ 379  297]
 [2703 2297]]
Linear Discriminant Analysis 0.8809020436927414
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       676
           1       0.88      1.00      0.94      5000

   micro avg       0.88      0.88      0.88      5676
   macro avg       0.44      0.50      0.47      5676
weighted avg       0.78      0.88      0.83      5676

[[   0  676]
 [   0 5000]]
```

Figure 4: PCA 10 Components Performance

```
    self.fc5 = nn.Linear(300, 2)
```

I used a Relu activation function between each layer and softmax (equivalent to sigmoid with this binary classification) after the final hidden layer. Most modern implementations I found utilized Relu to avoid the vanishing gradient problem. I tested multiple loss functions with multiple optimization algorithms.

I used 3 optimization algorithms - **Stochastic Gradient Descent with Momentum**, **Adagrad**, and **Adam**. [21]

- **SGD** - Vanilla SGD is the "traditional" method, but is not very effective compared to more modern methods and is rarely used anymore. Only the learning rate is tunable so it can suffer from either slow learning or overstepping the minimum. A common augmentation is to add momentum (exponential smoothing) so that the loss can converge faster in the beginning and slow down to approach the minimum.
- **Adagrad** - Adagrad is an algorithm that adaptively changes its learning rate automatically. I interpreted it as a smarter and more robust version of SGD with momentum. Footnote [3] explains that this makes it well suited for sparser data which could suit our dataset.

[21] https://arxiv.org/pdf/1609.04747.pdf

```
Random Forest Classifier 0.7082307421737601
              precision    recall  f1-score   support

           0       0.13      0.26      0.18       686
           1       0.88      0.77      0.82      5000

   micro avg       0.71      0.71      0.71      5686
   macro avg       0.51      0.52      0.50      5686
weighted avg       0.79      0.71      0.74      5686

[[ 179  507]
 [1152 3848]]
Support Vector Classifier 0.5939148786493141
              precision    recall  f1-score   support

           0       0.17      0.58      0.26       686
           1       0.91      0.60      0.72      5000

   micro avg       0.59      0.59      0.59      5686
   macro avg       0.54      0.59      0.49      5686
weighted avg       0.82      0.59      0.66      5686

[[ 400  286]
 [2023 2977]]
Logistic Regression 0.5963770664790714
              precision    recall  f1-score   support

           0       0.16      0.57      0.26       686
           1       0.91      0.60      0.72      5000

   micro avg       0.60      0.60      0.60      5686
   macro avg       0.54      0.59      0.49      5686
weighted avg       0.82      0.60      0.67      5686

[[ 393  293]
 [2002 2998]]
Linear Discriminant Analysis 0.8770664790714034
              precision    recall  f1-score   support

           0       0.26      0.01      0.02       686
           1       0.88      1.00      0.93      5000

   micro avg       0.88      0.88      0.88      5686
   macro avg       0.57      0.50      0.48      5686
weighted avg       0.81      0.88      0.82      5686

[[   7  679]
 [  20 4980]]
```

Figure 5: PCA 1 Component Performance

```
Random Forest Classifier 0.8133660761269953
              precision    recall  f1-score   support

           0       0.11      0.08      0.09       701
           1       0.88      0.92      0.90      5000

   micro avg       0.81      0.81      0.81      5701
   macro avg       0.49      0.50      0.49      5701
weighted avg       0.78      0.81      0.80      5701

[[  53  648]
 [ 416 4584]]
Support Vector Classifier 0.5300824416768988
              precision    recall  f1-score   support

           0       0.12      0.44      0.19       701
           1       0.87      0.54      0.67      5000

   micro avg       0.53      0.53      0.53      5701
   macro avg       0.50      0.49      0.43      5701
weighted avg       0.78      0.53      0.61      5701

[[ 309  392]
 [2287 2713]]
Logistic Regression 0.480617435537625
              precision    recall  f1-score   support

           0       0.12      0.51      0.20       701
           1       0.87      0.48      0.62      5000

   micro avg       0.48      0.48      0.48      5701
   macro avg       0.50      0.50      0.41      5701
weighted avg       0.78      0.48      0.56      5701

[[ 361  340]
 [2621 2379]]
Linear Discriminant Analysis 0.8770391159445712
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       701
           1       0.88      1.00      0.93      5000

   micro avg       0.88      0.88      0.88      5701
   macro avg       0.44      0.50      0.47      5701
weighted avg       0.77      0.88      0.82      5701

[[   0  701]
 [   0 5000]]
```

Figure 6: PCA 2 Components Performance

- **Adam** - Adam is a third popularly used algorithm. It stands for adaptive moment estimation. It is similar to Adagrad and adds in another element of momentum, this time by storing an exponentially decreasing weight of the history of gradients. [22]

I started with **MSE** loss. MSE is commonly used in regression problems but it is very popular so I gave it a chance.

- MSE with SGD loss performed surprisingly well, though not perfectly. It shows notable average improvement over the baseline. The loss function was interesting because it "bumped up" in the beginning instead of descending rapidly as usually happens.

- MSE with Adagrad performed the best within MSE. It converged fairly quickly at 150 epochs and obtained a high average f1-score of 0.83. However, the f1-score of the 0 class (has cancer) was low because of the unbalanced classes. The downside of MSELoss is that there is no quick way to balance the classes.

- MSE with Adam performed very poorly, predicting nearly 100% no cancer, the dominant class. Perhaps Adam has a tendency to bias toward the dominant class since it takes into account a history of previous iteration performance.

[22] https://towardsdatascience.com/the-3-best-optimization-methods-in-neural-networks-40879c887873

```
Random Forest Classifier 0.8585305545743553
              precision    recall  f1-score   support

           0       0.14      0.04      0.06       662
           1       0.88      0.97      0.92      5000

   micro avg       0.86      0.86      0.86      5662
   macro avg       0.51      0.50      0.49      5662
weighted avg       0.80      0.86      0.82      5662

[[  26  636]
 [ 165 4835]]
Support Vector Classifier 0.5506888025432709
              precision    recall  f1-score   support

           0       0.11      0.41      0.18       662
           1       0.88      0.57      0.69      5000

   micro avg       0.55      0.55      0.55      5662
   macro avg       0.50      0.49      0.43      5662
weighted avg       0.79      0.55      0.63      5662

[[ 272  390]
 [2154 2846]]
Logistic Regression 0.5162486753797245
              precision    recall  f1-score   support

           0       0.12      0.48      0.19       662
           1       0.88      0.52      0.66      5000

   micro avg       0.52      0.52      0.52      5662
   macro avg       0.50      0.50      0.42      5662
weighted avg       0.79      0.52      0.60      5662

[[ 319  343]
 [2396 2604]]
Linear Discriminant Analysis 0.8830801836806782
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       662
           1       0.88      1.00      0.94      5000

   micro avg       0.88      0.88      0.88      5662
   macro avg       0.44      0.50      0.47      5662
weighted avg       0.78      0.88      0.83      5662

[[   0  662]
 [   0 5000]]
```

Figure 7: PCA 3 Components Performance

```
Random Forest Classifier 0.8826232985681457
              precision    recall  f1-score   support

           0       0.23      0.00      0.01       657
           1       0.88      1.00      0.94      5000

   micro avg       0.88      0.88      0.88      5657
   macro avg       0.56      0.50      0.47      5657
weighted avg       0.81      0.88      0.83      5657

[[   3  654]
 [  10 4990]]
Support Vector Classifier 0.592893759943433
              precision    recall  f1-score   support

           0       0.12      0.39      0.18       657
           1       0.89      0.62      0.73      5000

   micro avg       0.59      0.59      0.59      5657
   macro avg       0.50      0.50      0.46      5657
weighted avg       0.80      0.59      0.67      5657

[[ 255  402]
 [1901 3099]]
Logistic Regression 0.5059218667138059
              precision    recall  f1-score   support

           0       0.12      0.50      0.19       657
           1       0.89      0.51      0.64      5000

   micro avg       0.51      0.51      0.51      5657
   macro avg       0.50      0.51      0.42      5657
weighted avg       0.80      0.51      0.59      5657

[[ 331  326]
 [2469 2531]]
Linear Discriminant Analysis 0.8838607035531201
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       657
           1       0.88      1.00      0.94      5000

   micro avg       0.88      0.88      0.88      5657
   macro avg       0.44      0.50      0.47      5657
weighted avg       0.78      0.88      0.83      5657

[[   0  657]
 [   0 5000]]
```

Figure 8: PCA 10 Components Performance

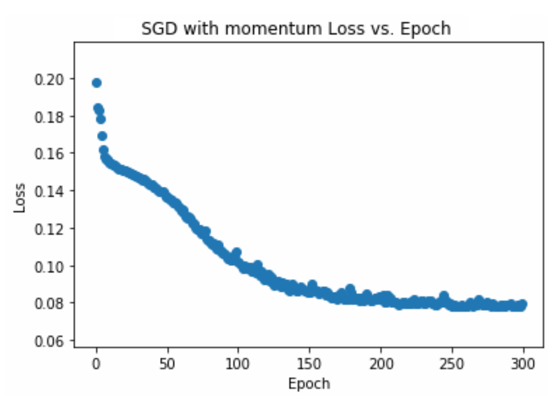

Figure: **MSE** with **SGD** Loss

	precision	recall	f1-score	support
0	0.23	0.40	0.29	653
1	0.91	0.83	0.87	5000
avg / total	0.83	0.78	0.80	5653

Accuracy of the network on the test data: 77 %

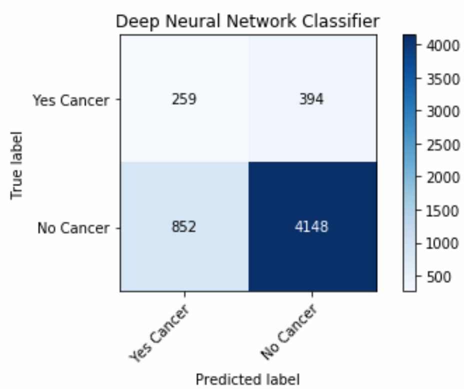

Figure: **MSE** with **SGD** Stats

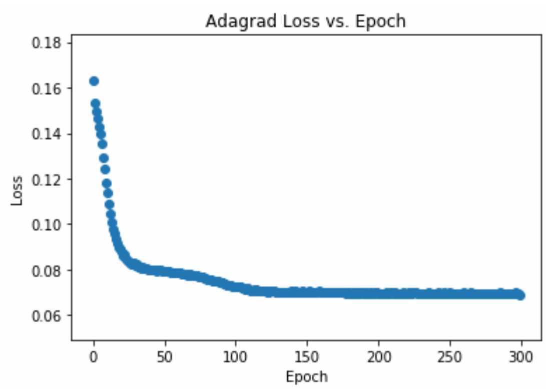

Figure: **MSE** with **Adagrad** Loss

	precision	recall	f1-score	support
0	0.26	0.18	0.21	653
1	0.90	0.93	0.91	5000
avg / total	0.82	0.85	0.83	5653

Accuracy of the network on the test data: 84 %

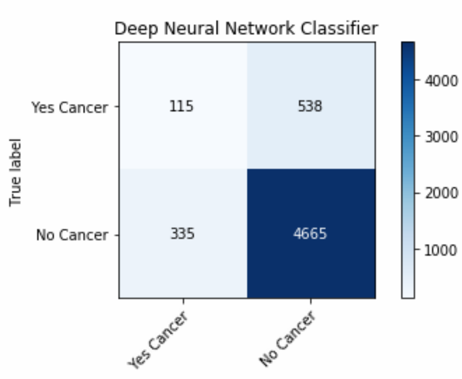

Figure: **MSE** with **Adagrad** Stats

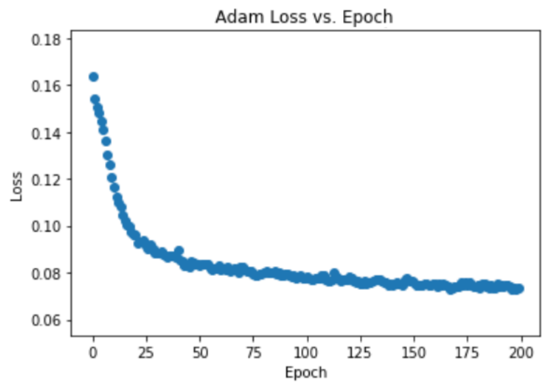

Figure: **MSE** with **Adam** Loss

```
             precision    recall  f1-score   support

          0       0.00      0.00      0.00       653
          1       0.88      1.00      0.94      5000

avg / total       0.78      0.88      0.83      5653

Accuracy of the network on the test data: 88 %
```

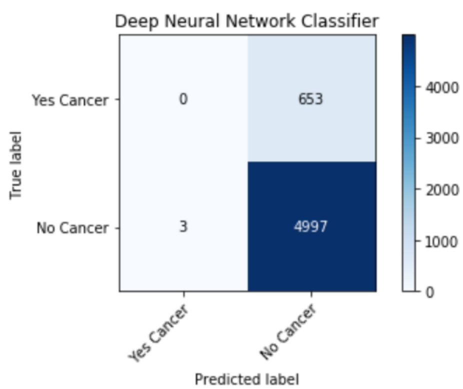

Figure: **MSE** with **Adam** Stats

I then tested **Binary Cross Entropy** as my loss function. It surprisingly performed quite similarly to MSE. I believe this could be due to the data not being well-structured enough for BCE to make as much a difference.

- The model had an interesting issue with Adam and SGD. The model would train for a certain number of epochs, but suddenly the loss would spike and stay constant. This destroyed any predictive power of the model.

- After that issue I tried limiting the number of epochs so the model would not experience the spike. This prevented the spike but the resulting model was still severely undertrained.

- The model trained terribly with SGD as well. It ran into the same "sudden loss increase issue."

- BCE with Adagrad was the best model and had the most balanced results. It still had trouble with class balancing though. I tried using the object's class weight modification parameter and manually balancing weights for classes, but could not achieve a better balance during the tests.

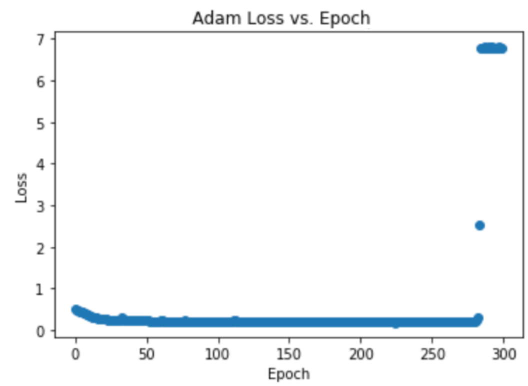

Figure: **BCE** with **Adam** Loss (bad)

	precision	recall	f1-score	support
0	0.12	1.00	0.21	653
1	0.00	0.00	0.00	5000
avg / total	0.01	0.12	0.02	5653

Accuracy of the network on the test data: 11 %

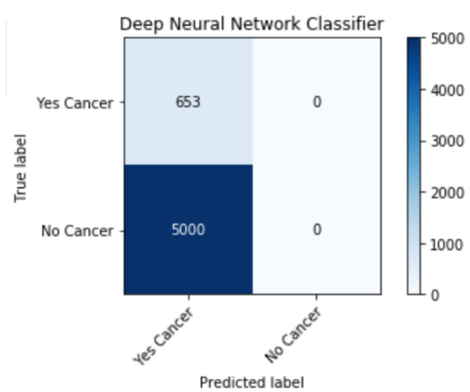

Figure: **BCE** with **Adam** Stats (bad)

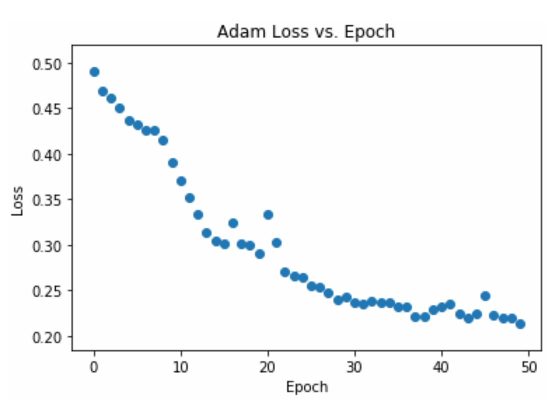

Figure: **BCE** with **Adam** Loss

```
tensor([1, 1, 1,  ..., 1, 1, 1])
             precision    recall  f1-score   support

          0       0.00      0.00      0.00       695
          1       0.88      1.00      0.93      5000

avg / total       0.77      0.88      0.82      5695
Accuracy of the network on the test data: 87 %
```

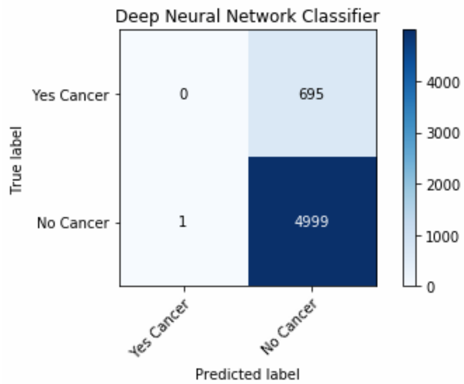

Figure: **BCE** with **Adam** Stats

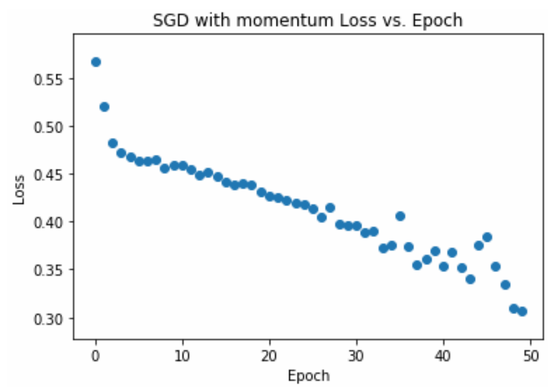

Figure: **BCE** with **SGD** Loss

```
             precision    recall  f1-score   support

          0       0.00      0.00      0.00       695
          1       0.88      1.00      0.93      5000

avg / total       0.77      0.88      0.82      5695
Accuracy of the network on the test data: 87 %
```

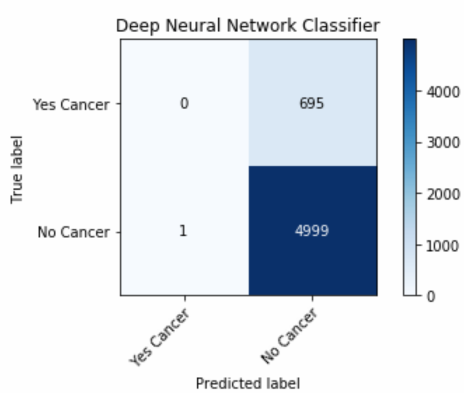

Figure: **BCE** with **SGD** Stats

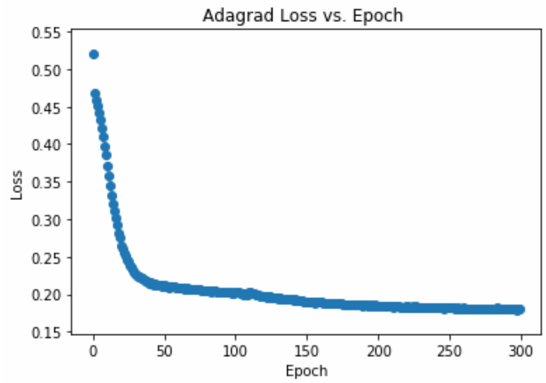

Figure: **BCE** with **Adagrad** Loss

```
             precision    recall  f1-score   support

          0       0.26      0.21      0.23       653
          1       0.90      0.92      0.91      5000

avg / total       0.82      0.84      0.83      5653
Accuracy of the network on the test data: 83 %
```

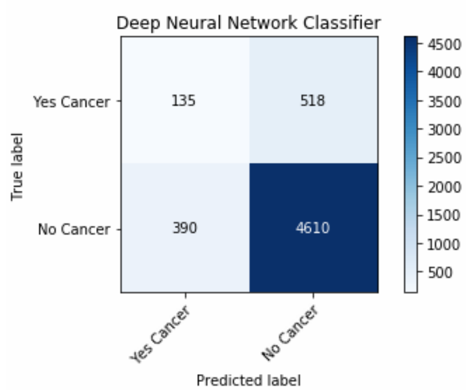

Figure: **BCE** with **Adagrad** Stats

Finally, while browsing the Pytorch forums I came across recommendations for the BCEWithLogitsLoss optimization function. This supposedly was more stable than using the original BCE. It also offered a simpler class weight parameter so I hoped it would solve my balancing issue. Unfortunately the results were subpar. It could not reduce its loss and would only predict the positive class. I may have used it wrong since it is a newer contribution to Pytorch, or it is just less stable.

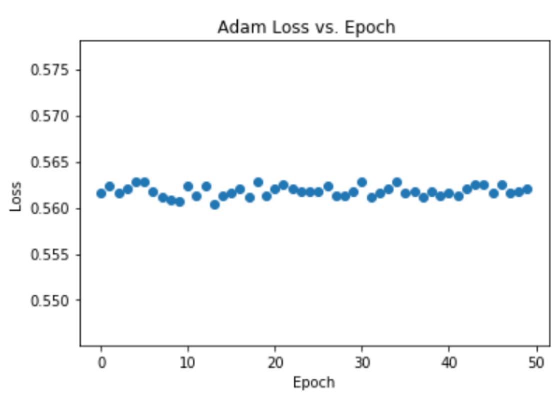

Figure: **BCEWithLogitsLoss** with **Adam** Loss

```
             precision    recall  f1-score   support

          0       0.00      0.00      0.00       653
          1       0.88      1.00      0.94      5000

avg / total       0.78      0.88      0.83      5653

Accuracy of the network on the test data: 88 %
```

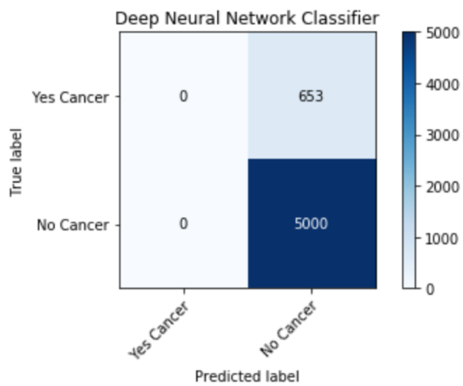

Figure: **BCEWithLogitsLoss** with **Adam** Stats

I additionally tested the neural network again with a more limited data size of 5000 train and 2000 test. This is to see if it improves the class balance issue as seen earlier in the paper, when increasing the data size also increased the gap between the classes. However, none of the stats improved and the overall performance simply decreased. This is likely due to the fact that deep neural networks can do so much more the more data they have, and is already complex enough to cause the class imbalances to appear with less data.

```
              precision    recall  f1-score   support

           0       0.61      0.14      0.22       667
           1       0.62      0.94      0.75      1000

   micro avg       0.62      0.62      0.62      1667
   macro avg       0.61      0.54      0.49      1667
weighted avg       0.61      0.62      0.54      1667

Confusion matrix, without normalization
[[ 91 576]
 [ 59 941]]
Accuracy of the network on the test data: 61 %
```

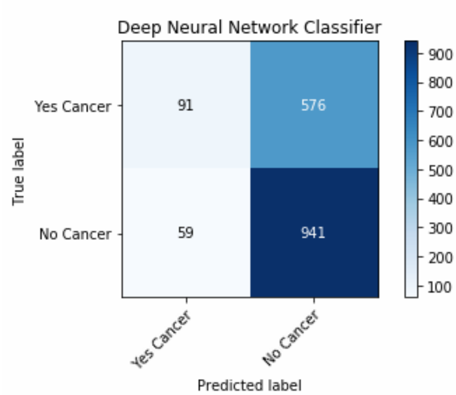

Figure: Smaller train/test sets **BCE** with **Adagrad** Stats

3 Results

The non-deep methods of Random Forest, SVC, Logistic Regression, and LDA performed decently but were not so accurate or robust.

In the end, the deep neural network performed the best. The best settings were Binary Cross Entropy with Adagrad loss. This obtained an f1-score for “has cancer” of 0.23 and an f1-score fr “healthy” of 0.91. The weighted average was 0.91.

This project shows definite predictive power of machine learning models, especially for DNNs, but I do not believe is is yet discriminatory enough to be used for real patients.

4 Afterthought and Further Potential

If I re-approached this project I would do more research and testing into the deep neural network. I believe more work can be done in terms of tuning the network architecture and weighting schemes. Less conventional (for this data) methods can be explored such as introducing convolution to extract more predictive power from the data.

References

Please find all references in the footnotes throughout the paper.

An extra thank you to Dr. Majid Sarrafzadeh and our TA Orpaz in leading the class, presenting this project to us, and continued guidance throughout.