Recommendation for Cancer Treatment Using Machine Learning(2023)

Jon Pendon, *NJIT*

[[1]](#footnote-1)

***Abstract*—Healthcare treatment has had significant advances in the machine learning field. Using machine learning and past instances of cancer patients, the ability to tell if a person requires surgery can be modeled. In this paper, models give near perfect accuracy on whether or not a patient needs treatment. Future work can be done with this paper in regards to multi class learning for treatment. In this paper, a multi-layer perceptron neural network, random forest classifier, and an AdaBoost classifier are all used to classify patient data. They are able to determine the class with 99.9% accuracy.**

***Index Terms*—AdaBoost, Classifiers, Healthcare, Machine Learning, Neural Networks, Random Forest**

# I. INTRODUCTION

T

HIS project aims to develop a recommendation system that suggests personalized healthcare treatment plans based on a patient's medical history, symptoms, and demographic information. Medical history includes items like their diagnosed disease, amount of tumors, treatment recommendation (surgery, chemotherapy, etc.), and disease stage information. Symptoms include site specific items and the extent of the disease. Demographic information includes age, race, sex, and date specific items.

This project is important because it helps doctors find a speedy treatment for individuals that have life threatening diseases. By using past data, doctors can use this program to find a quick and simple direction to take their treatment of a patient with ease. Instead of spending months to years of diagnosis time, doctors can quickly take new information and devise a simple plan to treat patients. Obviously, every new patient has new upcoming diseases, but with a simple plan of surgery, chemotherapy, or other treatment plan, it is easy for doctors to devise a strategic point to move forward with a patient's disease.

# II. Background and Related Work

Neurons counts used in this paper are referenced from two equations. The first equation was from Masters, Timothy. Practical neural network recipes in C++. Morgan Kaufmann, 1993. This equation detailed the number of neurons in one layer. The total of neurons in the single layer was two-thirds of the input plus the output. The second equation was a variation the Tamura and Tateishi method.

# II. Methodology

## A. Data

The data being used will be supplied by the National Cancer Institute: Surveillance, Epidemiology, and End Results Program (SEER). In this data, there are many databases to be looked at, but the main database will be the Incidence - SEER Research Data, 8 Registries, Nov 2022 Sub (2000-2020). This data includes demographics, year of death, year diagnosis, months to treatment, treatment type (therapy and surgery), a list of their treatment, and an extensive list of what their disease is. There are a total of 8,216,169 records in this database. The amount of variables that can be looked at are over 200, but the plan is to trim this down to the important ones. This data needs to be checked to see which variables are not useful towards the program. In addition, the variables need to be scoped out to a smaller amount to fulfill the time requirements of this project.

In order to generate a model, the variables need to be trimmed to the most important ones. Since the data is mainly categorical data, the chi-square metric is best used. The Chi-square metric tells us if the features are different or not. The higher the value, the higher the probability of a difference in features. Thus, the selection of features is done by recording all feature chi square values and using an eyeball approach to figure out the best and applying it to multiple sheets. As this data is very large, I split the data into 82 individual text files and generate all the chi square values using that information for each individual file. Data is split by going through each line in a hash-like format. The data is preprocessed using OneHotEncoder for all the X values and a LabelEncoder for the y target values. I preprocess the data before doing the feature selection on all files. The feature selection returns the chi-square values in text format and we use this format to generate the plots.

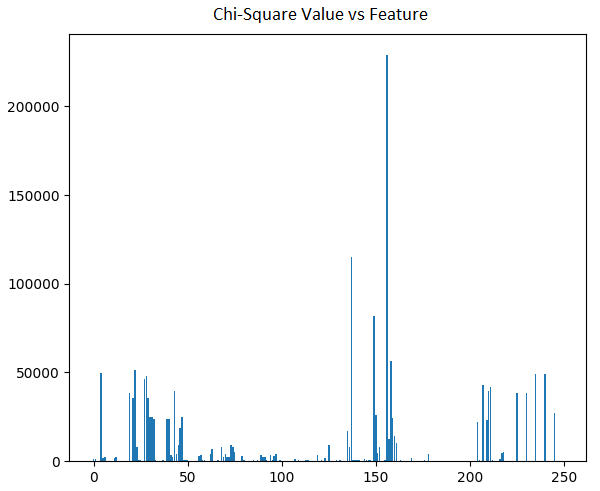
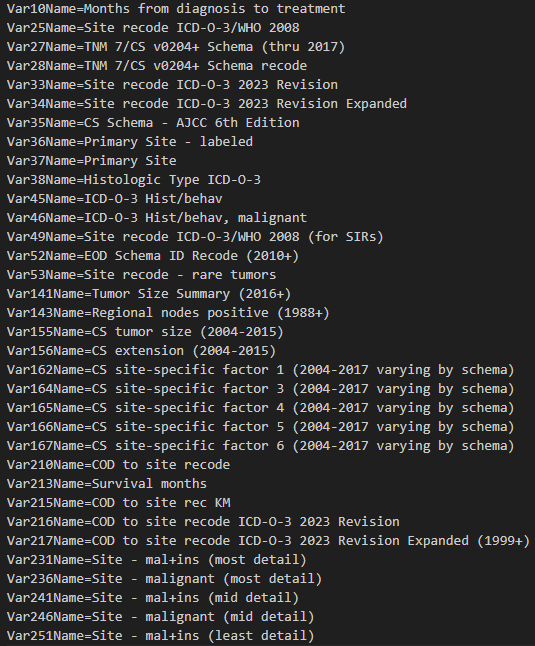
 **Fig. 1.** Chi Square Values.

Fig. 1 plots all chi square values for all features. As you can see a lot of features have very low chi square values which means they have little differences to each other. However, there are about 30 peaks above 10000 chi square value listed there. In order to retrieve these features, the dictionary file retrieved from the SEER data base is recoded into variables and then compared to the features that have been retrieved. I generate an array called arrayFeature to hold the number of instances that a feature over 10000 occurs in the 82 text files. Afterwards, using that array, I call each variable name and print the names of it in a new file. Fig 2. Shows all of those variables that have had over 60 occurrences in all of the split files.



**Fig. 2.** Chi Square Values.

Some variables are still not necessary. These have been removed as they are redundant and provide the exact same information. Age, sex, and race have been provided as demographic data as well.

After retrieving the variables, I redownload the SEER database file. However, because the data is too large, the setup becomes extremely imbalanced in the target values. To help accommodate this imbalance, I download all occurrences of 2020 data and all occurrences for the minority class in the entire database and put it into a csv called 2020-extra\_removed.csv. This file is our new data. This file contains 434,900 cases. The distribution is as follows:

* 2106 Intraoperative rad with other rad before/after surgery
* 5730 Intraoperative radiation
* 350698 No radiation and/or cancer-directed surgery
* 55549 Radiation after surgery
* 10128 Radiation before and after surgery
* 4410 Radiation prior to surgery
* 4989 Sequence unknown, but both were given
* 1290 Surgery both before and after radiation

As you can see this data is incredibly imbalanced toward the “No radiation and/or cancer-directed surgery,” “Radiation after surgery,” and “Radiation prior to surgery” values. These values had their set limited to only 2020 while the others contained all data from the database.

## B. Preprocessing – Multi Class

Data is preprocessed using libraries from scikit learn. Data is pulled into python using the pandas library. The two main libraries are LabelEncoder and OneHotEncoder. The target values are processed using LabelEncoder because they are categorical labels. I pull the target column, “RX Summ—Surg/Rad Seq,” out of the pandas dataframe and create a new one for the LabelEncoder. This then turns into a new pandas dataframe that can be exported using the .to\_csv() function. After the target dataframe is created, I drop the target data column and any other excess variable to generate the input features dataframe. These targets are put into a targets csv file.

In the final chosen input features, a large majority of them were encoded using OneHotEncoder as they were categorical. For every OneHotEncoder used, the pandas dataframe needs to be updated. I encode the feature first using a OneHotEncoder function. These columns are then put into a new pandas dataframe. The original data and the encoded data frame are then merged and the original data column is dropped along the column. This is done for every One Hot Encoded feature. These features are:

* Race and origin recode (NHW, NHB, NHAIAN, NHAPI, Hispanic)
* TNM 7/CS v0204+ Schema recode
* Site recode ICD-O-3 2023
* Site recode ICD-O-3 2023 Revision Expanded
* CS Schema - AJCC 6th Edition
* Primary Site – labeled
* ICD-O-3 Hist/behav
* ICD-O-3 Hist/behav, malignant
* Site recode ICD-O-3/WHO 2008 (for SIRs)
* EOD Schema ID Recode (2010+)
* Site recode - rare tumors
* Tumor Size Summary (2016+)
* Regional nodes positive (1988+)
* Site - mal+ins (most detail)
* Site - malignant (most detail)

Three other values had other methods of preprocessing. In these, pandas supports inplace transformation of data using the .replace() function. “Age recode with <1 years olds” was processed to have each value equal to the lowest value of the age range. For instance if the value was “01-04 years”, the value would turn into 1. If it was 85+ the value would be 85. “Sex” has been changed so that all “Female” cases are replaced with 0 and all “Male” cases are replaced with 1. “Months from diagnosis to treatment” has blanks. These blanks have been changed to a -1 value as all values in there are integers. After all input variables are encoded, the dataframe is saved to a csv file. In total, the One Hot Encoder makes the input have 2943 features, each corresponding to a specific categorical value.

## C. Models – Multi Class

After preprocessing, model selection begins. For this code, the plan is to use a neural network to solve the problem. Other models that could work are SVM and balanced random forest classifiers from imblearn. Imblearn libraries also provide EasyEnsembleClassifier.

The problematic ones are the EasyEnsembleClassifier, BalancedRandomForest, and SVM. These models cannot support the large data set that is being provided. The two models from imblearn are unable to generate the necessary arrays for this dataset. SVM also has a similar issue. I have tried lowering the time complexity by using a bagging classifier to split the SVM load time into a couple of classifiers. Unfortunately, the time complexity for SVM is too large as it is calculated as O(n\_features \* n\_samples^2) when using scikit learn’s SVM library. Our sample size is large (434,900) and our feature size input is also large (2943).

Thus the obvious choice is to rely on a neural network. Neural networks can solve many problems. Scikit learn provides a multi layer preceptron classifier, MLPClassifier, that can help classify multi class problems. MLPClassifier trains using backpropagation. The loss function used is cross entropy. In order to support multi class, the output layer’s activation function remains as softmax. The input layer contains each feature. In our case, there are 2943 input neurons.

The problem of tuning the parameters of the MLPClassifier depends on how complex the problem is. The number of neurons, the number of layers, and the number of iterations have to be configured. For each layer, the ReLU activation function will be used to avoid the vanishing gradient problem. Adam is our solver which is the stochastic gradient based optimizer proposed by Kingma, Diederik, and Jimmy Ba. Alpha refers to the L2 Regularization term. This term will be used to prevent overfitting by penalizing weights in each layer. The default for MLPClassifier is 0.0001.

## D. Parameters– Multi Class

The issue with hypertuning hidden layers is the amount of time that the program will take to fit and train. The time complexity of this system is O(n\*m\*h\*iterations) where n is the input, m is the neuron count, and h is the amount of layers.

To start, we need to have a base layer set. The chosen layer set is a simple symmetric (8,3,8) hidden layer. This hidden layer setup reported a score of 0.81. Unfortunately, this is because the classifier was able to determine the majority class and used that as its base accuracy. The provided score did not truly recall the other minor classes that well.

The second hidden layer group to test was a single layer with 1963 neurons.This classifier setup was also the same, but it took 17x longer than the (8, 3, 8) setup.

The final set was (841, 241, 69, 20, 6, 2). This had the same problems as the others.

## E. Change of Plans – Binary Class

Based on this data, it is extremely difficult to generate a MLPClassifier for this data. Thus, the idea is to reimagine the problem. The data can be split into a binary problem. One Target is no radiation or surgery and the other target is surgery or radiation occurred. From this, the problem will have a distribution of:

* 84202 Surgery or radiation occurred
* 350698 No radiation and/or cancer-directed surgery

Obviously, the data is still imbalanced. However, the classification should be easier. Other methods of classification can also be applied.

## E. Preprocessing – Binary Class

The data can be easily preprocessed. In addition to the target LabelEncoder, the values in that dataframe were changed to be binary. By simply changing all non “No radiation and/or cancer-directed surgery” values to 0 and everything else to 1, the target data has been changed to a binary problem.

## F. Models – Binary Class

Models that can be applied are the older used MLPClassifier and classifiers used in Imbalanced Learn Library. In the Imbalanced Learn Library, models are designed to balance the classes automatically by applying weights. They can also add overfitting and underfitting methods if needed. The model used in this paper from there will be the EasyEnsembleClassifier. This classifier is uses AdaBoosting as its main bagging classifier. The last classifier will be a BalancedRandomForestClassifier.

## G. Parameters – Binary Class

The first tests completed were done using MLPClassifier. The first method tested was with 3 layers of (8, 3, 8) neurons. The second was a single layer of (1963) neurons, and the third was 5 layers with (841, 241, 69, 20, 6, 2) neurons. The MLPClassifier held the same problems as the previous setup. The single layer and five layer network took too long to fit. The three layer network did not take as long to fit. In addition, the single and five layer network were unable to differentiate the two classes. However, the three layer network was able to classify the test data with a score of 0.9998.

The second tests were done with EasyEnsembleClassifier. This classifier did not take as long as the MLPClassifier, but it was able to return an accuracy score of 1.0. More work would need to be done in understanding if the EasyEnsembleClassifier provides a good model. However, due to the size and length of the data provided for both test and train, it is likely that this model provides a good understanding of the data without overfitting.

The last test was done with the BalancedRandomForestClassifier. The classifier used 100 estimators. This classifier took about the same time as the MLPClassifier with (8, 3, 8) layers. It returned an accuracy score of 0.9993. Based on only accuracy, it is difficult to know if the model returned is not overfitted.

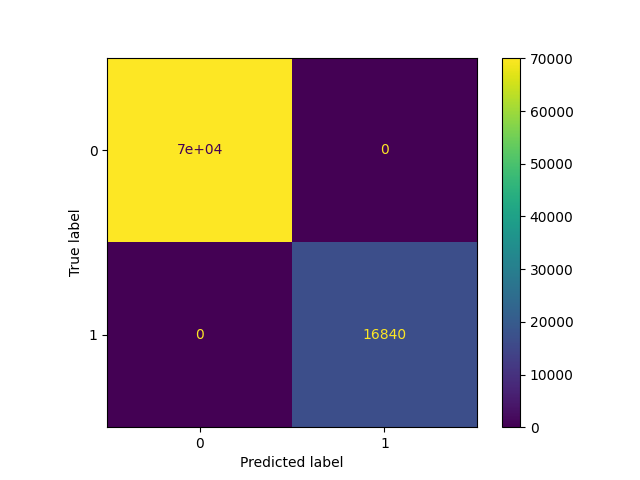
# IV. Results and Evaluations

## A. MLPClassifier

The MLPClassifier produced the worst initial results. All 3 classifiers in the multi-class version were unable to classify non majority classes. As a result, learning this model will take a lot more time. There is potential in future research for figuring out the number of layers and neurons needed for that. When swapped to a binary problem, the imbalance was alleviated a little bit and one of the classifiers was able to classify data with good accuracy of 0.9999. Unfortunately, the other two were unable to classify.

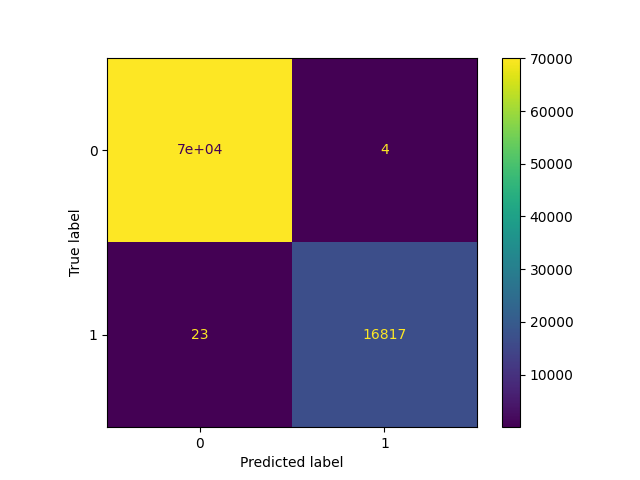
## B. Easy Ensemble Classifier

This classifier did extremely well. The AdaBoostClassifier is used as its base classifier. It accurately predicted each instance in the training set with a score of 1.0.



## C. Balanced Random Forest Classifier

This classifier also did extremely well. The DecisionTree is used as its base classifier with under sampling occurring in each boostrap. It accurately predicted each instance in the training set with a couple of errors with a score of 0.9993. Due to the large number of data sets, The model retrieved accurately labeled the data.



# V. Conclusion

Moving from multi class to a binary classification made this project much easier. The use of binary allows for quicker analysis and eases the imbalance. Here, the paper shows that determining whether or not a patient should receive treatment can be found with accuracy up to 99.9%. Future work can be done by taking the new class that is labeled as “treatment done” and learning a new model on that data to classify which type of treatment should be done.

# VI. References

[1] Heaton, J. (2005). Introduction to neural networks with Java. Heaton Research.

[2] Tamura & Tateishi (1997) Tamura S, Tateishi M. Capabilities of a four-layered feedforward neural network: four layers versus three. IEEE Transactions on Neural Networks. 1997;8(2):251–255. doi: 10.1109/72.557662

1. [↑](#footnote-ref-1)