UNIVERSITY OF CALIFORNIA,
IRVINE

# Data Augmentation Policies for Cancer Classification

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Biomedical Engineering

by

Aditya Vyas

Thesis Committee:
Associate Professor James Brody, Chair
Associate Professor Michelle Digman
Associate Professor Jered Haun
Associate Professor Elliot Hui

2020

# DEDICATION

To Professor James Brody for his guidance that has taught me to research and write independently with confidence,

To my mother and father, Mrs. Anita Vyas and Dr. Kamlesh Vyas, who taught me the value of education and critical thought.

This thesis is also dedicated to my family and friends who have supported me unconditionally, and to my mentor Dr. Nitin Sapre who encouraged me in pursuit of my interests.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# ACKNOWLEDGEMENTS

I would like to express the most profound appreciation to my thesis advisor, Professor James Brody. The door to Professor Brody's office was always open whenever I ran into a trouble spot or had a question about my research or writing. He has consistently allowed this thesis to be my own work but steered me in the right direction whenever he thought I needed it. Without his guidance and persistent help, this thesis would not have been possible.

I would like to thank the members of my thesis committee: Professor Michelle Digman, Professor Jered Haun and Professor Elliot Hui for generously offering their time, support, guidance, and goodwill throughout the preparation and review of this document.

I would like to thank the rest of the Biomedical Engineering department at UC Irvine for providing such a collaborative environment. I would like to thank the University of California, Irvine, for giving me a platform to grow holistically and professionally. Furthermore, I would like to thank my roommates and all my friends for their consistent support throughout my master's journey.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

# ABSTRACT OF THE THESIS

Data Augmentation Policies for Cancer Classification

By

Aditya Vyas

Master of Science in Biomedical Engineering

University of California, Irvine, 2020

Professor James Brody, Chair

Data augmentation is a beneficial technique to improve the performance of modern classifiers. However, current data augmentation implementations are mostly for image-related data. In this thesis, two data augmentation policies have been created to augment non-image limited data. The policies are modeled after the measurement errors found in scientific measurements and aims to increase the limited data by inducing those errors. The second purpose of this thesis is to study the role of Copy Number Variations (CNVs) in cancer manifestation. CNVs are structural variations, and they can be useful to identify who will develop particular cancers. The germ line data is collected from The Cancer Genome Atlas Program (TCGA), and contemporary machine learning models are being used to classify cancer types. However, these classification models perform poorly due to limited availability of data, thereby, the data augmentation policies are used to increase data diversity. The usage of these policies has significantly improved the classifier's performance, giving an improvement of ≈3-4%, which can be highly beneficial for future research. These policies can be used to find dominant chromosomal regions which are correlated with respective cancer type, thereby giving more insights to the medical community for further analysis.

# INTRODUCTION

Data augmentation is the process of expanding the volume and diversity of data. Instead of collecting new data, rather we transform the already existing data. Data augmentation adds value to base data by combining information derived from internal and external sources within an enterprise. Data augmentation is an indispensable process in deep learning, as in deep learning, we need large amounts of data, and in some cases, it is not feasible to collect thousands or millions of data samples, so data augmentation comes to the rescue.

Recent advancements in deep learning models are mainly due to the quantity and diversity of data gathered in recent years. For example, Google has reached state-of-the-art accuracy on datasets such as CIFAR-10 with 'AutoAugment,' a new automated data augmentation technique [1]. The usage of Data Augmentation is prominent for an Image-Based Neural Network model, and there has been much ongoing research on creating a better data augmentation policy. Krizhevsky et al. developed AlexNet CNN architecture for image classification by applying convolution networks to ImageNet (a sizeable visual database designed for use in visual object recognition software research) dataset and increased their data size by a magnitude of 2048 with the help of data augmentation [2].

We can apply multiple operations to augment our data; for example, in the case of image data, we can apply Rotation, Shearing, Zooming, Cropping, Flipping, and Changing the color characteristics etcetera. As mentioned earlier, data augmentation has been popular with image-based datasets, unlike standard numerical datasets. Therefore, this thesis research focuses on using data augmentation techniques on standard numerical data taken from The Cancer Genome Atlas Program (TCGA) [3].

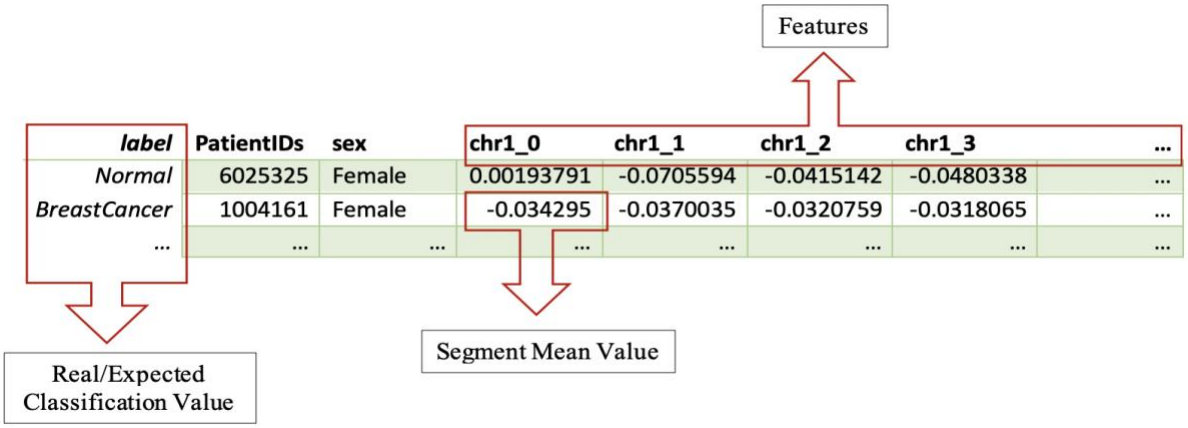
## 1.1. Data Description

Cancer genomics, a prominent research area that uses state of the art technology to study our full set of DNAs, and aims to find genomic alterations in DNA, i.e., cause of cancer. One of the significant DNA mutations that are associated with human cancer is DNA Copy Number Variants (CNV). CNVs are a type of structural variation which involves deletion or duplication of relatively vast stretches of DNA (that is, thousands of nucleotides [>1 kb], which may span many different genes), but can vary in size as well as prevalence [4].

The degree to which copy number variation contributes to human disease is not yet known. There has been ongoing research in the field to relate the significance of CNVs to specific cancer. Zhang et al. classified cancers based on copy number variation landscape with an accuracy of 0.75 by using only the CNVs of 19 genes based on the Dagging method in 10-fold cross-validation [5].

In this research, two different types of datasets obtained from TCGA are being used. TCGA translates noisy intensity measurements into chromosomal regions of equal copy number with the help of Affymetrix SNP 6.0 array data along with Circular Binary Segmentation (CBS) analysis. CBS, in general, is used to find the changepoints in sequential data, i.e., it is an algorithm that translates noisy intensity measurements into regions of equal copy number [6]. These copy number values are further transformed into segment values, which are equal to log2(copy number/2). Therefore, diploid regions, amplified regions, and deletions will correspond to zero, positive and negative segment mean value, respectively.

These datasets mainly contain segment mean information at different chromosomes along with the patient's gender and the cancer type. The main aim of this research would be to classify the cancer type based on segment mean data. While neural network architectures have been investigated in-depth, more focus has been put into discovering durable data augmentation policies that capture data invariances.



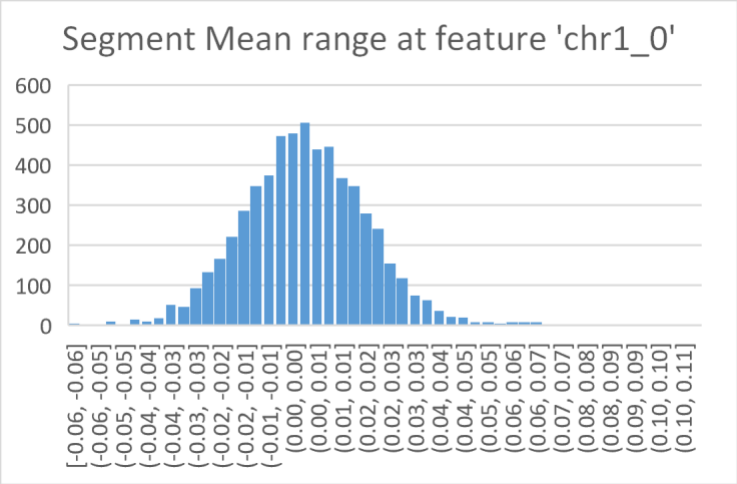
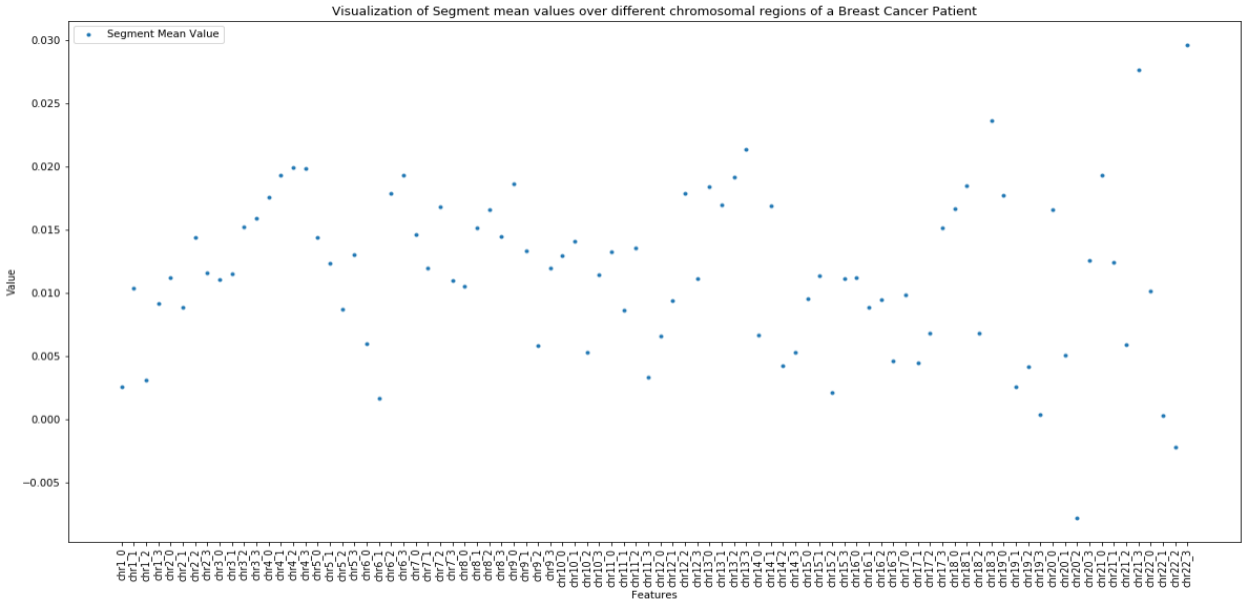




Figure 1: Visualization of a sample dataset obtained from TCGA (top), Segment Mean values over a single chromosomal region (left), Visualizing segment mean values over different chromosomal regions of a breast cancer patient(bottom).



### 1.2. Motivation

When dealing with segment mean data, we face the issue of missing data. To handle the missing data, we can either delete a specific sample or impute the missing value by inference. The process to use to handle the missing data is shown below:

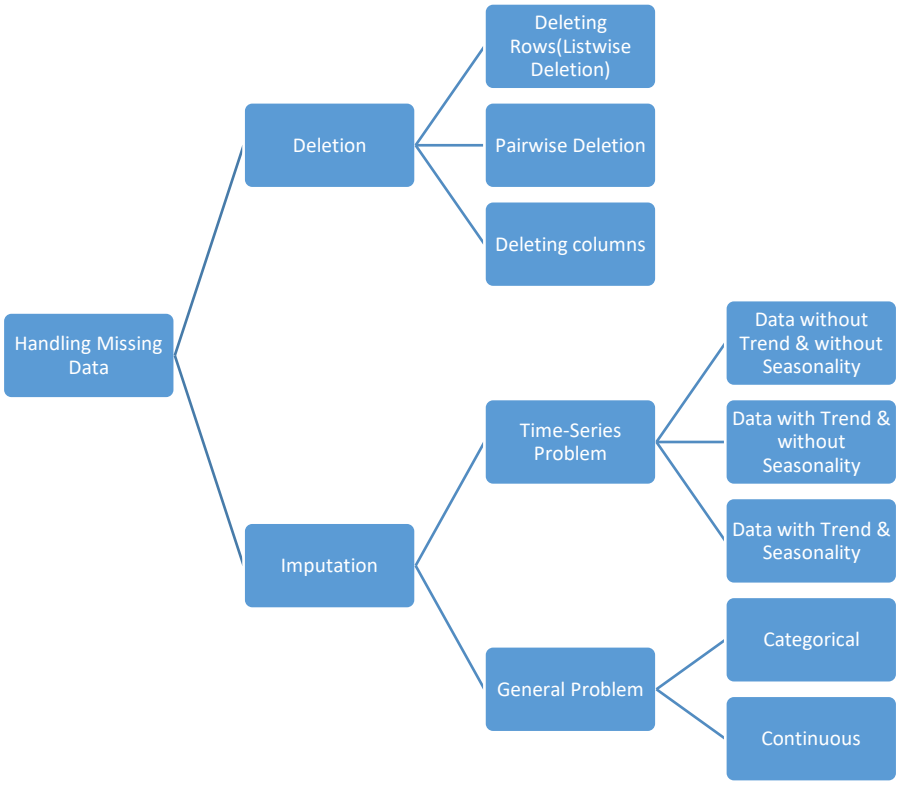



Figure 2: Handling Missing Data [7]

If we start with the deletion of the sample (a single patient's information) who have some missing values, then, data size would reduce by ≈ 95% as we might not have segment mean value for a specific chromosome. Therefore, the better choice would be to go with data imputation, where, we can statistically assign a value lying in the expected range of that parameter, for example, replacing the missing value by the mean value of its relevant set. There have been instances where Generative Adversarial Networks (GANs) to generate synthetic data from training data [8];

Therefore, another possible approach would be to use Generative Adversarial Network to create discrete synthetic samples based on characteristics on available training data.

Even after we have handled the missing values, we still have the problem of limited data, and that's where our data augmentation policy comes in. It has been assumed that the measurement of the segment means might have an error (let's say the measurement method has 98% accuracy; therefore, the error rate is 2%). This error can be caused either by technical errors in an experiment or by aberrant copy number in a region covering only a single marker. We will use this fact to create the data augmentation policy and later on analyze the efficiency of deep learning-based cancer classification model trained on both, augmented/non-augmented dataset.

In order to keep consistency in the results, it's been decided to use XGBoost, a popular applied machine learning algorithm for the classification model. XGBoost supports three primary forms of gradient boosting i.e., Gradient Boosting, Stochastic Gradient Boosting, and Regularized Gradient Boosting with L1/L2 Regularization. The main advantage of using XGBoost is that it is engineered for the efficiency of computing time and memory resources. Also, to measure the performance of a classification model, we will be using the essential evaluation metric i.e., AUC (Area under the curve) - ROC (Receiver Operating characteristics) curve. ROC is a probability curve, whereas AUC represents the degree or measure of separability. This represents the model's capability to distinguish between the classes. Prediction of Zero(s) to Zero and One(s) to One will be better with higher AUC. Thereby, with better AUC (≈1), then the classification model can distinguish patients having cancer or not. The ROC curve is plotted with True Positive Rate (TPR; y-axes) against the False Positive Rate (FPR; x-axes).

ROC-AUC curve will be derived from the confusion matrix, which is being created with the help of real values and the predicted values of our deep learning model. A deep learning model is

considered useful if the validation error continues to diminish with the training error. With the help of data augmentation, we can represent a more comprehensive set of available data points; thereby, we can reduce the distance between the training and validation set, as well as for any future testing set. Another metric that has also been included to evaluate a model is 'Accuracy.'

The accuracy is calculated from confusion matrix, i.e.

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{Total Count}}$$

The main aim of this research would be to increase the AUC of a classifier from a baseline model. AUC was chosen as a critical metric to track as it is scale-invariant, i.e., measuring how well predictions are ranked, rather than their absolute values and irrespective of what classification threshold is chosen. Another thing to avoid during this experiment is overfitting; for example, if accuracy is increasing while AUC decreases, this will indicate overfitting, i.e., in general words, the classifier will perform better over test data, but will give a poor performance on real-time data. Another reason to avoid accuracy as the key metric is that the provided data is highly unbalanced, i.e., we will get high accuracy by predicting that all observations belong to the majority class.

# EXPERIMENT AND RESULTS

## 2.1. EXPERIMENT 1:

### 2.1.1. Dataset Description:

This dataset consists of segment mean values over different chromosomal regions of patients suffering from particular type of cancer, with a total of 8826 observations and 152 attributes. Before applying the data augmentation, the first step is to preprocess the data. Therefore, observations with unknown cancer types were removed, decreasing the observations to 8704. This dataset consists of 32 different cancer types, with Breast Cancer (BRCA) having the highest number of observations; Thereby, instead of going for the multi-class classification model with higher complexity, the focus will be kept on a binary classifier to measure the effect of data augmentation.

As Breast cancer only occurs in females and rarely in males (≈1%), all the observations with male patients were removed, thereby decreasing our dataset by almost 50%, i.e. the number of observations dropped to 4622 with BRCA accounting for 970 observations. These observations are further divided into a training dataset (80%) and a test dataset (20%) randomly.

### 2.1.2. Handling Missing Data:

After preprocessing, the next step was to handle the missing values and see which method has better performance. At first, the data was kept as it is, to have a baseline performance. Later, the missing values were replaced with the mean and median of their corresponding attribute, i.e., ideal segment value at that specific chromosomal region. For the latter experiment, Gaussian noise was added with varying parameters to each value, and the performance was measured. The results are as follows:

| S. No. | Replacing missing values with | Accuracy | AUC | STD |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| 1 | Not Applicable (NA) | 0.81390 | **0.74460** | 0.008 |
| 2 | Zero | 0.81173 | 0.73945 | 0.008 |
| 3 | Mean of column | 0.81471 | 0.73819 | 0.0084 |
| 4 | Median of column | 0.81660 | 0.74165 | 0.0096 |
| 5 | Gaussian Noise | 0.81147 | 0.73378 | 0.0080 |
| 6 | Gaussian Noise on just training set | 0.80741 | 0.49139 | 0.0092 |

Table 1: Performance of XGBoost on the handled data

From the above observations (Table 1), it can be observed that the AUC is maximum when the missing values are kept as it is. XGBoost applies Sparsity-aware Split Finding to learn the best direction in a tree to deal with these missing values [9].

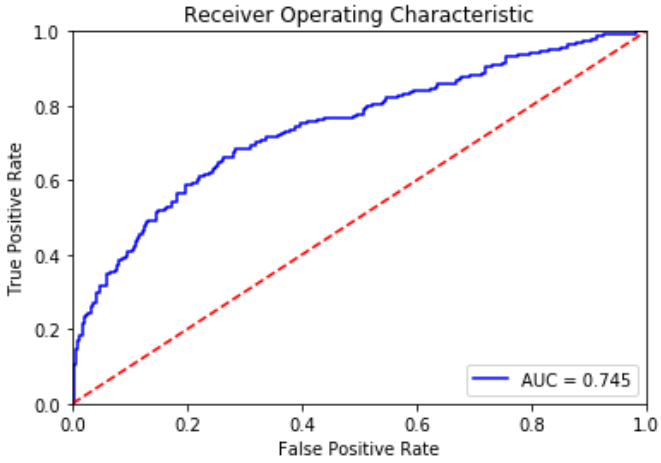



Figure 3: Receiver Operating Characteristic diagram of an XGBoost model after data pre-processing. (Refer code at Appendix 1.1)

### 2.1.3. Data Augmentation Policies

**Policy 1**

The aim of this policy to add a certain amount of noise (Gaussian noise in this case) to the given segment mean value and then use this value to create a new observation. The policy is described below:

1. Take the segment mean value and decide a standard deviation you want to have in the data

2. Based on this create a normal distribution of a specific size (say 1000 samples)
3. Select any random value from that 1000 samples
4. Replace the new value from old value and augment it into the data

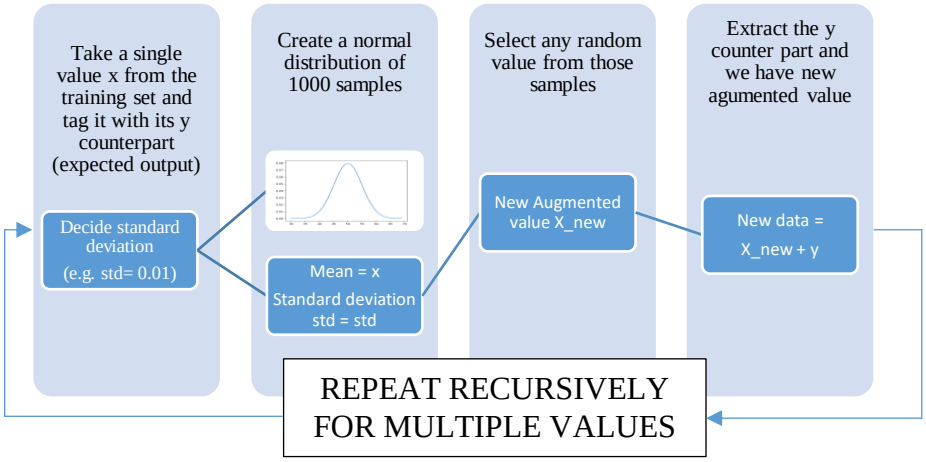



Figure 4: Semantic graph of Policy 1

```
def augmenter(X_train,y_train,sd,col):
    dataset = X_train.copy() #DeepCopy the data
    length= len(dataset) #Take the length of dataset
    new = pd.DataFrame() #Create a new DF to store data
    for i in range(length): #Number of times the loop has to run
        row = dataset.iloc[[i]] #Take the ith element
        row.insert(0, 'y',y_train.iloc[i]) #To avoid data mismatch - tag the row with its y_train[i]th element
        #Apply augmentation on that single full row
        ans = gf.augment_column_byvalue_single(row,col_index=col+1,times=5,std=sd,sample_size=1000,add_initial=False)
        new=new.append(ans) #Store the data
    X_train = new.iloc[:,1:] #Extract the new X_train
    y_train = new.iloc[:,0:1].values.ravel() #Extract the new y_train
    return X_train,y_train
```

Figure 5: Augmenter function that takes training/test data and returns augmented data. (Refer Appendix 1.2 for more information)

The above policy was used to augment the given dataset on all the segment values, and the performance of the classification model on the augmented data was measured (Check Table 2).

| Augmentation Times | Accuracy | AUC | Standard Deviation |
|---|---|---|---|
| 1 time | 0.80173 | **0.77749** | 0.00256 |
| 32 times | 0.82525 | 0.74580 | 0.0109 |

Table 2: Performance of XGBoost on Augmented Data

It can be observed from the results that AUC was increased by almost 3.3% when the data was augmented once. However, when the dataset was augmented by the factor of 32, AUC was decreased, and accuracy was increased, indicating overfitting. Another set of experiments was to apply Policy 1 feature-wise; therefore, data was increased by 5 times, but the AUC of the classifier was decreased by 1% (≈0.73). Therefore, it can be concluded from the previous set of results that we cannot augment each value in the dataset.

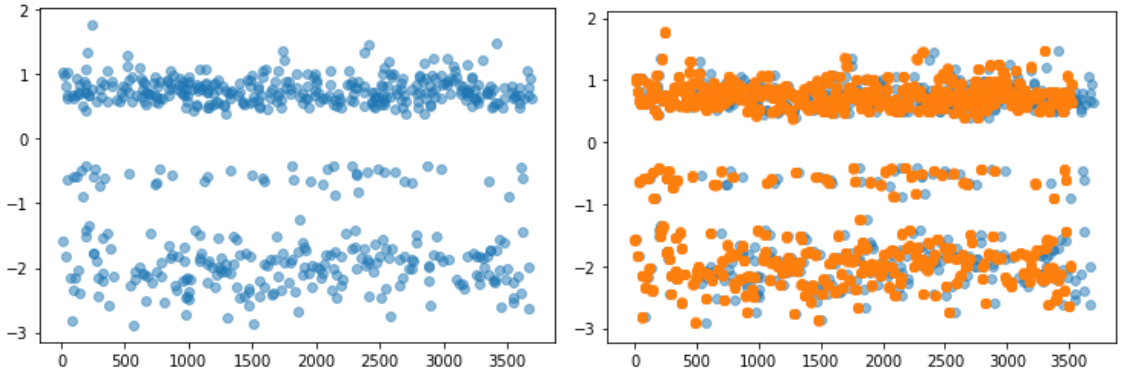



Figure 6: Augmented Data Visualization (Blue represents the original data and orange represent the augmented data.)

**Policy 2**

This policy was created to apply Policy 1 to the data in, either way, i.e., row-wise or column-wise. In this, the focus will be kept more on applying policy 1 on the columns (feature-wise). This policy was inspired by the Genetic Algorithm, where the best performing subjects are selected for the next round of gene transfer. The policy is described below:

1. Apply Policy 1 to all the features and record the classifier performance (one feature at a time).
2. Now find the top n features (say n=7) who performed better when augmented.
3. The total number of combinations possible for these features is 2n and Total possible combination of subset excluding blank and single digits 2n - n -1.

4. Now we augment all the possible combinations of top n features and record the classifier performance.
5. Repeat from step 2. The exit condition can be set up if the metrics are not approved.

This augmentation policy was applied to the given dataset, and n was set to be 7. The top-7 features were selected based on their respective trained classifier's performance. The top-7 features can be augmented in 120 ways, giving 120 classifier models. To have consistency in the results, the data was increased by a factor of 5 for each augmentation. Also, each augmentation was performed just on the given training dataset. Again top-7 models with their combinations were selected, and the whole process was repeated. One thing to note is that when this process is repeated for a long time, there might be instances when the possible combinations of features might have some common features; These common features can be ignored with the help of programming.

```
"""
data contains the names of features and their individual
performances associated with it.
"""
index_list = get_index_combo(data,num=7) #Get top 7 performing feature indexes and thei
r combinations
data1  = pd.DataFrame() #Create a new DF
for i in index_list:
    Xt,yt = X_train.copy(),y_train.copy()
    print("Currently augmenting {}".format(i))
    for j in i: #Loop through only selected features and augment those feature data onl
y
        Xtr,ytr=augmenter(X_train.copy(),y_train.copy(),sd=0.1,col=j) #creating augment
ed values
        Xt = Xt.append(Xtr)
        yt = yt.append(pd.Series(ytr))
    d = gf.run_xgboost(Xt,yt,X_test,y_test)
    d["combination"]=i
    data1= data1.append(d,ignore_index=True)
    print(d["accuracy"])
    print("-------")
```

Figure 7: Code to get indexes of top-7 performing features and their combinations, apply augmentation to only those features and store the results. This code can be used recursively to re-apply Policy 2. (Refer Appendix 1.3)

After multiple iterations, we got a set of features that profoundly affects the classifier's metrics. An interesting thing to note was that the maximum accuracy reached in this process was 0.981, giving us a ≈17% improvement from the baseline classifier; however, the AUC was still 0.74, indicating overfitting.

### 2.1.4. Testing with other Classification Models

Since there were many classifiers we created with better performances, this part aimed to combine those classifiers with the help of different classifier combination techniques. The results are as follows:

| Technique | AUC (NON-AUGMENTED) | AUC (AUGMENTED) |
|---|---|---|
| Voting | 0.71921 | 0.68284 |
| Average | 0.74695 | 0.71355 |
| Weighted Average | 0.74807 | 0.71810 |
| Stacking | 0.59798 | 0.59051 |
| Bagging meta estimator | 0.69035 | 0.6467 |
| AdaBoost | 0.68083 | 0.67991 |
| Gradient Boost | 0.70732 | 0.706104 |

Table 3: Performance of combined classification models on augmented and non-augmented data. (Refer Appendix 1.4)

## 2.2. EXPERIMENT 2:

### 2.2.1. Dataset Description

This dataset consists of segment mean information over different chromosomal regions of female patients, with a total of 5925 observations with 88 attributes/features. Before applying the data augmentation, the first step is to preprocess the data. Unwanted features were removed, and each observation is labeled as either 'normal' or 'breast cancer.' This dataset primarily focused on creating a binary classifier and measure the effects of data augmentation.

### 2.2.2. Testing with other Classification Models

Before selecting XGBoost before-hand as the primary classifier, different types of classification models were used with their default parameters. The results can be observed in Table 4, where XGBoost has performed well as compared to its adversaries.

| S. No. | Classifier | Accuracy | AUC |
|---|---|---|---|
| 1 | XGBoost | **0.76160** | **0.68360** |
| 2 | Naïve Bayes Classifier | 0.67763 | 0.64212 |
| 3 | Decision Tree Classifier | 0.74936 | 0.63720 |
| 4 | K Nearest Neighbors | 0.74514 | 0.61838 |
| 5 | Bagging – Decision Tree | 0.75527 | 0.65859 |
| 6 | Ada Boost Classifier | **0.76202** | 0.64898 |
| 7 | Gradient Boosting Classifier | 0.75274 | 0.66287 |

Table 4: Performance of different classification model

### 2.2.3. Results

Data augmentation policy 1 was applied to the given dataset row-wise, and Grid Search has been used to have better hyperparameters for our primary classifier. The standard deviation was set to 0.001, and each value generates about 1000 samples from which one value is randomly selected.

```
#Augmenter function that takes training data and number of augmentation times
def augment(X_train,y_train,tim):
    if len(X_train) != len(y_train):
        print("Check your data")
        return False #Something wrong with data
    df = pd.concat([y_train,X_train],axis=1, sort =False) #Tag y_train to X_train to av
oid mismatch
    new_data = pd.DataFrame()
    for i in range(len(df)):
        row = df.iloc[[i]] #Access the specific row
        #Apply Augmentation to full row
        row2 = augment_row_byvalue_all(row,ignore_index=0,times=tim,std=0.001,sample_si
ze=1000,add_initial=True)
        new_data = new_data.append(row2,ignore_index=True)
    print("Previous data length {} ".format(len(df))) #Informative print
    print("New data length {} ".format(len(new_data))) #Informative print
    return new_data
```

Figure 8: Code to augment the data row-wise. (Refer Appendix 1.7)

The augmented data can be seen in Figure 3.

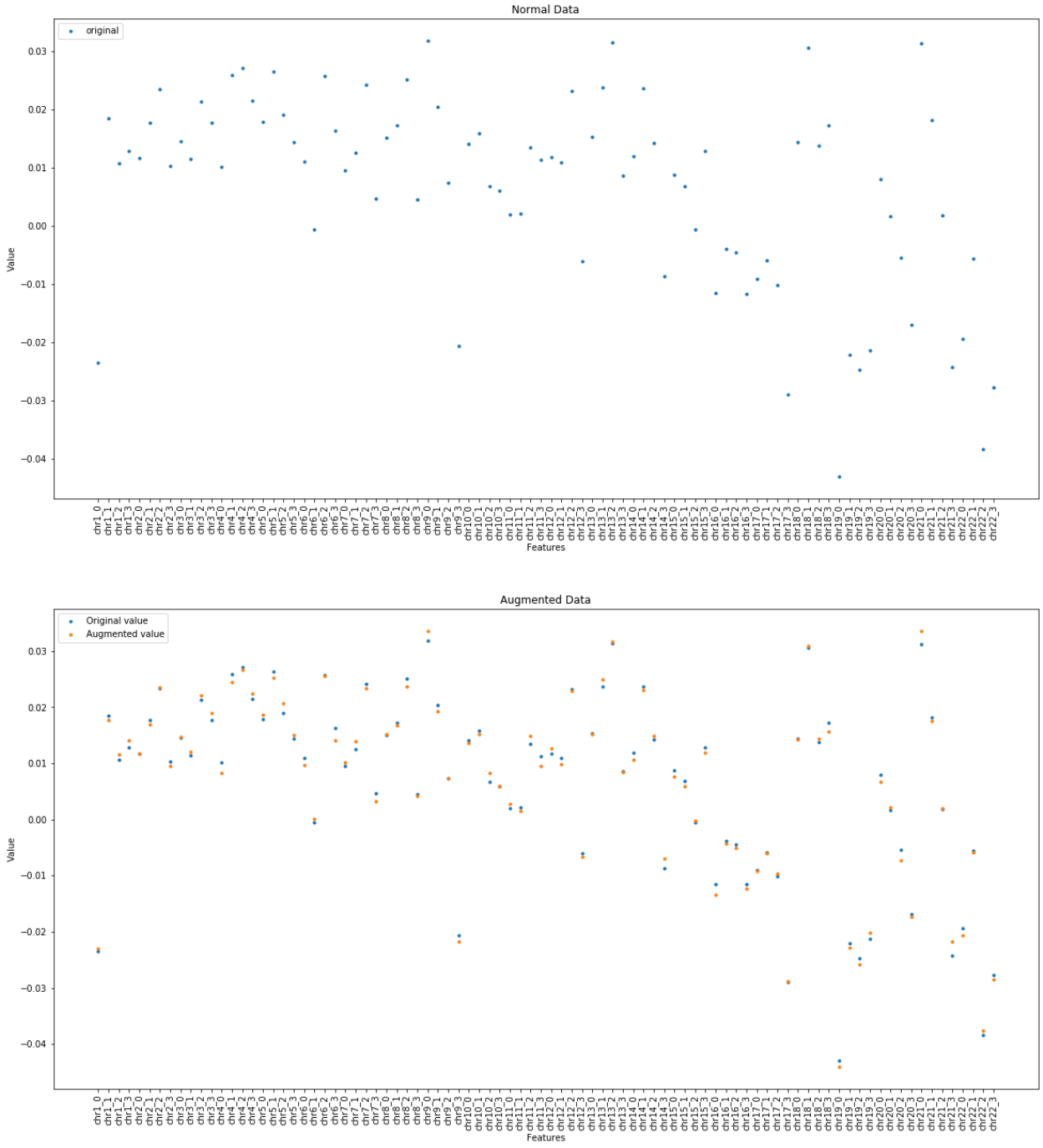



Figure 9: Augmented Data Visualization (Blue represents the original data and orange represent the augmented data.)

In figure 3, we can see that the augmented data is pretty close to the original data. Thereby, we will be testing the effect of augmentation on the performance of the classifier by increasing the

augmentation factor. Grid Search will be finding a better classifier at each iteration with the best hyperparameters by comparing 'ROC-AUC' metrics with the 10 Fold Cross Validation method. The results are as follows:

| Augmentation Times | Accuracy | AUC |
|---|---|---|
| 0 | 0.76160 | 0.68360 |
| 1 | 0.77468 | 0.70629 |
| 2 | 0.77215 | 0.71018 |
| 3 | 0.77046 | 0.71660 |
| 4 | 0.76962 | 0.69242 |
| 5 | 0.77468 | 0.70509 |
| 6 | 0.77215 | 0.71980 |
| 7 | 0.77721 | 0.71868 |
| **8** | **0.77299** | **0.72377** |
| 9 | 0.77383 | 0.713357 |
| 20 | 0.76624 | 0.71084 |
| 30 | 0.77383 | 0.718096 |
| 50 | 0.76624 | 0.70714 |

Table 5: Performance of XGBoost classifier on augmented data

From Table 5, it can be observed that the AUC gets higher until iteration 8, i.e., the classification performs best when this data was increased by a factor of 8. However, later we can see that there is a decrease in the AUC with a higher augmentation factor because our classifier reaches overfitting. Thereby, it can be concluded that the augmentation policy does help to increase the performance of the classifier by 4%. Overdoing the augmentation will lead to overfitting; thereby, augmentation can help improve the model up to an extent.

# DISCUSSION

Our genome consists of DNA sequences, and its variations contribute to our uniqueness. These variations can influence most characteristics, including susceptive to disease. A better comprehension of genetic variation can help us to know more about our prehistory but also helps to find the changes like the growth of cancer.

The study of cancer genomics is constructive to uncover the changes that occur in genes and how cancer growth happens. With dedicated research along with precision medicine, the growth of cancer can be prevented. For example, Lines of evidence have shown that CNVs of specific genes are involved in the growth of numerous cancers through the changes of their gene expression levels on an individual or several cancer types. However, it is not entirely clear whether the correlation will be a general phenomenon across multiple cancer types.

This thesis aimed to use CNV information to create a binary cancer classifier; meanwhile, studying the role of data augmentation policy for a numerical non-correlated limited data. From the above experiments, it can be said that data augmentation policies have given us a hand in increasing the classifier's performance. This significant improvement with the help of augmentation policy can help out relative researches dealing with a limited set of data. On the downside, there is always a risk of overfitting the model with a higher data augmentation factor.

Another problem that can be faced is computation power, i.e., with high data augmentation factor, the data will grow substantially such that it will increase the training time. For example, in Experiment 2, when the data augmentation factor was set to 50, it took almost 20 hours, even with the support of two GPUs, to find the optimal classifier. Therefore, there is only a limited range of augmentation that can be applied to the dataset to improve classifier performance.

The two data augmentation policy can be applied to any numerical data and has multiple possibilities of how it can be applied. For example, policy 1 and 2, both can be applied to certain features or specific observations. The possibilities of applying these policies can be increased based on the noise we are applying. In our experiments, Gaussian noise was applied with a standard deviation of 0.01, and 0.001 but these values can be varied.

# FUTURE WORKS AND LIMITATIONS

Research in the field of cancer and its relation to CNVs is in its infancy period but is maturing quickly. There are dozens of questions that can be addressed with more research in this field. For example, what are the exact roles of CNVs in cancer manifestation, and how can we use this information to cure it? How are CNVs related to neoplastic growth? These are questions that can be answered when using CNVs as potential biomarkers of cancer proliferation. This is an expanding field where insights from cancer CNVs can be used for new drug development.

The next phase of this thesis can be extended with the usage of Generative Adversarial Networks (GANs). GANs can be used to generate new data that can be augmented in the model to increase classifier performance. However, there is a limitation when dealing with generated data, i.e., it can induce bias and probably decrease the classifier's accuracy. Therefore, one has to ensure that the generated data has high covariance.

Though these policies are meant to be designed for non-image limited data, it would be interesting to see the role of these policies in digital image processing. Multiple noise models can be used along with the augmentation strategies to increase the study power. For example, we can try modeling the noise that occurs during image acquisition and transmission, then we can combine image denoising with data augmentation.

Another possible direction to explore is the use of time-series CNV data where the patient's history of CNVs would be studied concerning time. This approach can indicate the changes in the CNVs over time, and the patterns can be observed and mimicked by a machine learning model. Therefore, the more data we have on the patients over time, then, the dataset generated would be instrumental for future studies.

# BIBLIOGRAPHY

[1] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "AutoAugment: Learning Augmentation Strategies From Data," 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019.
[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," Communications of the ACM, vol. 60, no. 6, pp. 84–90, 2017.
[3] "The Cancer Genome Atlas Program," National Cancer Institute. [Online]. Available: https://www.cancer.gov/about-nci/organization/ccg/research/structural-genomics/tcga.
[4] A. Thapar and M. Cooper, "Copy Number Variation: What Is It and What Has It Told Us About Child Psychiatric Disorders?," Journal of the American Academy of Child & Adolescent Psychiatry, vol. 52, no. 8, pp. 772–774, 2013.
[5] N. Zhang, M. Wang, P. Zhang, and T. Huang, "Classification of cancers based on copy number variation landscapes," Biochimica et Biophysica Acta (BBA)-General Subjects, vol. 1860, no. 11, pp. 2750–2755, 2016.
[6] A. B. Olshen, E. S. Venkatraman, R. Lucito, and M. Wigler, "Circular binary segmentation for the analysis of array-based DNA copy number data," Biostatistics, vol. 5, no. 4, pp. 557–572, Jan. 2004.
[7] A. Swalin, "How to Handle Missing Data," Medium, 19-Mar-2018. [Online]. Available: https://towardsdatascience.com/how-to-handle-missing-data-8646b18db0d4.
[8] C. Bowles, L. Chen, R. Guerrero, P. Bentley, R. Gunn, A. Hammers, D. A. Dickie, M. V. Herna´ndez, J. Wardlaw, and D. Rueckert, "Gan augmentation: Augmenting training data using generative adversarial networks," arXiv preprint arXiv:1810.10863, 2018.
[9] T. Chen and C. Guestrin, "XGBoost," Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016.

# APPENDIX 1

## 1.1. Code Snippet 1

Description: This is the basic code that takes the 'TCGA data', pre-process it and train a XGBoost classifier.

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#Importing the dataset
dataset = pd.read_csv('data/TCGA_data.csv')

#Data Cleaning
dataset= dataset.drop(columns="Unnamed: 0")
dataset= dataset.drop(columns="sample_barcode")
dataset.dropna(subset=['project_name'], how='all', inplace = True)
dataset.drop(dataset.loc[dataset['gender']=="MALE"].index, inplace=True)

#Dividing the dataset
X = dataset.iloc[:, 2:]
y = dataset.iloc[:, 0]

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state
    = 0)

# Fitting XGBoost to the Training set
from xgboost import XGBClassifier
classifier = XGBClassifier()
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

# Applying k-Fold Cross Validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10)

print(accuracies.mean())
accuracies.std()

# Calculating Performance Metrics
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs =  classifier.predict_proba(X_test)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)
```

```
48.
49. # Plotting ROC-AUC curve
50. import matplotlib.pyplot as plt
51. plt.title('Receiver Operating Characteristic')
52. plt.plot(fpr, tpr, 'b', label = 'AUC = %0.3f' % roc_auc)
53. plt.legend(loc = 'lower right')
54. plt.plot([0, 1], [0, 1],'r--')
55. plt.xlim([0, 1])
56. plt.ylim([0, 1])
57. plt.ylabel('True Positive Rate')
58. plt.xlabel('False Positive Rate')
59. plt.show()
```

## 1.2. Code Snippet 2

Description: These are the functions that were used to implement Policy 1

```
2.  def gaussiansample(mu,sigma,vals):
3.      s = np.random.normal(mu, sigma, 1000) #Use Numpy library function to apply gaussian
     sampling
4.      return s
5.
6.  def choose_random(lis):
7.      return random.choice(lis) #Choose a random value from a list
8.
9.  def augment_row_byvalue_single(data,ignore_index,times,std,sample_size,add_initial=Fals
    e):
10.     a = data.copy() #Copy the original data
11.     i,ans = 0,0 #Variables
12.     for i in range(len(data.columns)): #Number of Features
13.         if i>ignore_index: #Ignore those columns with ignore_index variable
14.             if not np.isnan(data.iat[0,i]): #If data not empty
15.                 ans+=1
16.                 temp2 = 0 #Variable to run the while loop
17.                 s = gaussiansample(data.iat[0,i],std,sample_size) #Create samples
18.                 while temp2<times: #times will run the code multiple times
19.                     temp = a.copy()
20.                     val = choose_random(s) #Choose a random value
21.                     temp.iat[0,i]=val
22.                     data=data.append(temp) #Store the value
23.                     temp2+=1
24.                     #break
25.     print("Found {} values in the row and augmented data {} times".format(ans,ans*times
    ))
26.     #Include the first row i.e. the original data
27.     if add_initial:
28.         return data
29.     else:
30.         return data.iloc[1:]
31.
32. #Similarly the code can be created for columns
33. def augment_column_byvalue_single(data,col_index,times,std,sample_size,add_initial=Fals
    e):
34.     new = pd.DataFrame()
35.     if add_initial:
36.         new = new.append(data,ignore_index=True)
37.     if not np.isnan(data.iat[0,col_index]):
```

```
            s = gaussiansample(data.iat[0,col_index],std,sample_size)
            for i in range(times):
                temp = data.copy()
                val = choose_random(s)
                temp.iat[0,col_index]=val
                new = new.append(temp,ignore_index=True)
        return new
```

```
#Augment_row_byvalue_single function can be extended to augment all the value in the ro
ws
#The code is as follows:
def augment_row_byvalue_all(data,ignore_index,times,std,sample_size,add_initial=False):

    a = data.copy()
    i = 0
    ans=0
    temp = a.copy()
    for i in range(len(data.columns)):
        if i>ignore_index:
            if not np.isnan(data.iat[0,i]):
                ans+=1
                s = gaussiansample(data.iat[0,i],std,sample_size)
                val = choose_random(s)
                temp.iat[0,i]=val
    data=data.append(temp)

    print("Found {} values in the row and augmented {} values".format(ans,ans*times))
    #Include the first row i.e. the original data
    if add_initial:
        return data
    else:
        return data.iloc[1:]
```

## 1.3. Code Snippet 3

Description: These are the functions that were used to implement Policy 2.

```
#Return top n indexs based on value
def get_top_n(lis,num):
    return sorted(range(len(lis)), key=lambda i: lis[i], reverse=True)[:num]

#Return all the subsets of list except the blank one and itself
def subsets(lis):
    from itertools import chain, combinations
    def all_subsets(ss):
        return chain(*map(lambda x: combinations(ss, x), range(0, len(ss)+1)))
    comb = []
    for subset in all_subsets(lis):
        if len(subset)!=0 and len(subset)!=1:
            comb.append(subset)
    return comb
"""
Takes the list of features as data and extracts top n feature names.
Returns all the possible subset combinations indexes.
"""
def get_index_combo(data,num):
    lis=gf.get_top_n(data["accuracy"],num)
```

```
    temp = gf.subsets(lis)
    return temp
```

## 1.4. Code Snippet 4

Description: This code represents how multiple predictions can be used to create ensemble models.

```
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

classifier1 = RandomForestClassifier(n_estimators = 800, criterion = 'entropy', random_state = 0)
classifier1.fit(Xtr, ytr)
classifier2 = XGBClassifier()
classifier2.fit(Xtr, ytr)
classifier3 = DecisionTreeClassifier(criterion = 'entropy', random_state = 100,max_depth=1000,
                                     min_samples_leaf=60,min_impurity_decrease=0.0009)
classifier3.fit(Xtr, ytr)
pred1 = classifier1.predict_proba(Xte)[:,1] #Random Forest
pred2 = classifier2.predict_proba(Xte)[:,1] #XGBOOST
pred3 = classifier3.predict_proba(Xte)[:,1] #Decision Tree Classifier

#Applying Averaging
final_pred = np.array([])
for i in range(0,len(Xte)):
    final_pred = np.append(final_pred, np.mean([pred1[i], pred2[i], pred3[i]]))

#Weighted Average
final_pred = np.array([])
for i in range(0,len(Xte)):
    t = (pred1[i]*0.3)+(pred3[i]*0.3)+(pred2[i]*0.4)
    final_pred = np.append(final_pred, t )
#Similarly We can apply other ensemble models
#Refer Table 3 to get the names of the models that can be applied through this code
```

## 1.5. Code Snippet 5

Description: This code shows how dataset 2 was gathered, cleaned, and applied to different classifiers.

```
#Importing Libraries
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

#Method to get the dataset 2
def get_dataset():
    dataset = pd.read_csv('trainingdataBC_UKB.csv')
    #Data Cleaning
```

```
    dataset= dataset.drop(columns="Unnamed: 0")
    dataset= dataset.drop(columns="PatientIDs")
    #dataset['sex'].describe()
    dataset= dataset.drop(columns="sex") #removing sex as all are females
    dataset["label"] = np.where(dataset["label"]=="Normal",0,1)
    dataset['cancer_selfreported'].unique()
    return dataset

#Method to divide the dataset into training/test data.
def divide(dataset):
     #Dividing the dataset
    X = dataset.iloc[:, 5:]
    y = dataset.iloc[:, 0]
    # Splitting the dataset into the Training set and Test set
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_s
tate = 1000)
    return [X_train, X_test, y_train, y_test]

#Method that takes confusion matrix and returns the accuracy
def accuracy(confusion_matrix):
    diagonal_sum = confusion_matrix.trace()
    sum_of_all_elements = confusion_matrix.sum()
    return diagonal_sum / sum_of_all_elements

def run_xgboost(X_train,y_train,X_test,y_test,plot=True):
    # Fitting XGBoost to the Training set
    from xgboost import XGBClassifier
    classifier = XGBClassifier()
    classifier.fit(X_train, y_train)
    # Predicting the Test set results
    y_pred = classifier.predict(X_test)
    # Applying k-Fold Cross Validation
    from sklearn.model_selection import cross_val_score
    accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv =
 10)

    #Getting AUC-ROC
    import sklearn.metrics as metrics
    # calculate the fpr and tpr for all thresholds of the classification
    probs =  classifier.predict_proba(X_test)
    preds = probs[:,1]
    fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
    roc_auc = metrics.auc(fpr, tpr)

    # Plotting ROC Curve
    if plot:
        import matplotlib.pyplot as plt
        plt.title('Receiver Operating Characteristic')
        plt.plot(fpr, tpr, 'b', label = 'AUC = %0.3f' % roc_auc)
        plt.legend(loc = 'lower right')
        plt.plot([0, 1], [0, 1],'r--')
        plt.xlim([0, 1])
        plt.ylim([0, 1])
        plt.ylabel('True Positive Rate')
        plt.xlabel('False Positive Rate')
        plt.show()

    return {"accuracy":accuracies.mean(),"std":accuracies.std(),"AUC":roc_auc}
```

## 1.6. Code Snippet 6

Description: This code shows implementation of different classifiers used in Table 4.

```
"""
Different Classifiers code that were used in Table 4
"""
#Naive Bayes Classifier
def naivebayesclassifier(Xtr,ytr,Xte,yte):
    # Fitting Naive Bayes to the Training set
    from sklearn.naive_bayes import GaussianNB
    classifier = GaussianNB()
    classifier.fit(Xtr, ytr)
    # Predicting the Test set results
    y_pred = classifier.predict(Xte)
    # Making the Confusion Matrix
    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(yte, y_pred)
    #CALCULATING AUC
    import sklearn.metrics as metrics
    p = classifier.predict_proba(Xte)
    preds = p[:,1]
    fpr, tpr, threshold = metrics.roc_curve(yte, preds)
    roc_auc = metrics.auc(fpr, tpr)
    return {"Accuracy":accuracy(cm),"AUC":roc_auc}
print(naivebayesclassifier(X_train,y_train,X_test,y_test))

#KNN Classifier
def knnclassifier(Xtr,ytr,Xte,yte):
    from sklearn.neighbors import KNeighborsClassifier
    classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
    classifier.fit(Xtr, ytr)
    # Predicting the Test set results
    y_pred = classifier.predict(Xte)
    # Making the Confusion Matrix
    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(yte, y_pred)
    #CALCULATING AUC
    import sklearn.metrics as metrics
    p = classifier.predict_proba(Xte)
    preds = p[:,1]
    fpr, tpr, threshold = metrics.roc_curve(yte, preds)
    roc_auc = metrics.auc(fpr, tpr)
    return {"Accuracy":accuracy(cm),"AUC":roc_auc}
print(knnclassifier(X_train,y_train,X_test,y_test))

#Decision Tree Classifier
def decisiontreeclassifier(Xtr,ytr,Xte,yte):
    # Fitting Decision Tree Classification to the Training set
    from sklearn.tree import DecisionTreeClassifier
    classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 100,max_d
epth=1000,
                                        min_samples_leaf=60,min_impurity_decrease=0.000
9)
    classifier.fit(Xtr, ytr)
    # Predicting the Test set results
    y_pred = classifier.predict(Xte)
```

```
    # Making the Confusion Matrix
    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(yte, y_pred)
    #CALCULATING AUC
    import sklearn.metrics as metrics
    p = classifier.predict_proba(Xte)
    preds = p[:,1]
    fpr, tpr, threshold = metrics.roc_curve(yte, preds)
    roc_auc = metrics.auc(fpr, tpr)
    return {"Accuracy":accuracy(cm),"AUC":roc_auc}
print(decisiontreeclassifier(X_train,y_train,X_test,y_test))

#Bagging Classifier
from sklearn.ensemble import BaggingClassifier
from sklearn import tree
model = BaggingClassifier(tree.DecisionTreeClassifier(random_state=1),n_jobs=-1)
model.fit(Xtr, ytr)
print(model.score(Xte,yte))

#ADA Boost Classifier
from sklearn.ensemble import AdaBoostClassifier
model = AdaBoostClassifier(random_state=1)
model.fit(Xtr, ytr)
print(model.score(Xte,yte))

#Gradient Boosting Classifier
from sklearn.ensemble import GradientBoostingClassifier
model= GradientBoostingClassifier(learning_rate=0.01,random_state=1)
model.fit(Xtr, ytr)
print(model.score(Xte,yte))
```

## 1.7. Code Snippet 7

Description: This code implements a similar augmentation algorithm code to augment the data.

```
#This is similar to code discussed in appendix 1.2.
#A new update is that I have added verbosity as a new parameter.
def augment_row_byvalue_all(data,ignore_index,times,std,sample_size,add_initial=False,verbose=0):
    a = data.copy()
    temp = a.copy()
    for _ in range(times):
        ans=0
        for i in range(len(data.columns)):
            if i>ignore_index:
                if not np.isnan(data.iat[0,i]):
                    ans+=1
                    s = gaussiansample(data.iat[0,i],std,sample_size)
                    val = choose_random(s)
                    temp.iat[0,i]=val
        data=data.append(temp)
    if verbose!=0:
        print("Found {} values in the row and augmented {} values".format(ans,ans*times))
    #Include the first row i.e. the original data
    if add_initial:
        return data
```

```
    else:
        return data.iloc[1:]
#Visualizing the data
row2 = augment_row_byvalue_all(X_train.iloc[[0]],ignore_index=-
1,times=1,std=0.001,sample_size=1000,add_initial=True)
fig = plt.figure()
ax1 = fig.add_subplot(111)
ax1.plot(row2.iloc[0],'.' , label='original')
plt.legend(loc='upper left');
plt.title('Normal Data')
plt.ylabel('Value')
plt.xlabel('Features')
plt.show()

fig = plt.figure()
ax1 = fig.add_subplot(111)
ax1.plot(row2.iloc[0],'.' , label='original')
ax1.plot(row2.iloc[1],'.', label='first')
plt.legend(loc='upper left');
plt.title('Augmented Data')
plt.ylabel('Value')
plt.xlabel('Features')
plt.show()
```

## 1.8. Code Snippet 8

Description: This code shows how grid search was implemented.

```
#Applying Grid Search
from sklearn.model_selection import GridSearchCV
import xgboost as xgb
from xgboost.sklearn import XGBClassifier

estimator = XGBClassifier(objective= 'binary:logistic', seed=42)
parameters = {
    'max_depth': range (10, 15, 1),
    'n_estimators': range(160, 260, 40),
    'learning_rate': [0.1]
}
grid_search = GridSearchCV(
    estimator=estimator,
    param_grid=parameters,
    scoring = 'roc_auc',
    n_jobs = 10,
    cv = 10,
    verbose=True
)
grid_search.fit(X, y)
grid_search.best_estimator_

#Method that takes test ip/op and the classifier and return it's metrics
def accuracy(y_test,x_test,cls):
    y_pred = cls.predict(x_test)
    # Making the Confusion Matrix
    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(y_test, y_pred)
    diagonal_sum = cm.trace()
```

```
    sum_of_all_elements = cm.sum()
    acc= diagonal_sum / sum_of_all_elements
    #CALCULATING AUC
    import sklearn.metrics as metrics
    p = cls.predict_proba(x_test)
    preds = p[:,1]
    fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
    roc_auc = metrics.auc(fpr, tpr)
    return {"Accuracy":acc,"AUC":roc_auc}

accuracy(y_test,X_test,grid_search)
```