Machine Learning algorithms for the detection and localization of surgically resectable cancers

Project mentor: Joshua Popp

Members: Chris Wilhelm cwilhel8@jh.edu, Jayden Kunwar jkunwar1@jhu.edu, Bryce Thalheimer bthalhe1@jh.edu, Arthur Beyer cbeyer4@jhu.edu

GitHub Repository:
https://github.com/ChrisWilhelm/MachineLearningFinalProject

Inspiration for Project:
https://www.science.org/doi/10.1126/science.aar3247?url_ver=Z39.88-2003&rfr_id=ori:rid:crossref.org&rfr_dat=cr_pub%20%200pubmed

Outline and Deliverables

Uncompleted Deliverables

1. "Expect to complete #3": Multi-Class Classification of Cancer Type
2. "Would like to complete #2": Additional Data

Completed Deliverables

1. "Must complete #1": We discuss our logistic regression in "Models" below.
2. "Must complete #2": We discuss training our neural network in "Models" below.
3. "Must complete #3": We discuss AdaBoost in "Models" below.
4. "Expect to complete #2": We discussed interpretability...
5. "Expect to complete #1": Data preprocesing
6. "Would like to complete #1": High Specificity

Additional Deliverables

(example) 1. We decided to add a second baseline using the published model from this paper. We discuss this in "Baselines" below.

1. ...

Preliminaries

What problem were you trying to solve or understand?

What are the real-world implications of this data and task?
Cancer is a highly prevalent and life-changing disease where early diagnosis and intervention/treatment can drastically alter the outcome of disease, so that is why attempting to classify someone as having or not having cancer has real-world implications. We are attempting to improve the prediction algorithms specified in the aforementioned paper CancerSEEK (more information on this in the project proposal) by building our own models and using an optimized subset of data from the dataset (including biomarkers such as circulating tumor DNA and various protein levels) to predict cancer status in healthy control individuals and patients with cancers.

How is this problem similar to others we've seen in lectures, breakouts, and homeworks? What makes this problem unique?
We've seen many classification problems in class, with a particular focus on binary classifications. Therefore, this was very similar to others we had already seen in class, but the aspect that makes this problem unique is that our calculations are being performed on novel data (with the exception of CancerSEEK having already performed classification). The model being developed would allow for the rapid identification of cancer with a mere multi-analyte blood test, which would be a much cheaper and more efficient way to detect the disease.

What ethical implications does this problem have?
The ethical implications of this are the false positive rate and false negative rate, as having tests like this that incorrectly classify you and prevent or cause treatment when unnecessary are harmful. Since we lacked abundant demographic data on features such as race, this could potentially misclassify people in different communities worse, which could be a harmful affect. Overall, models like this help significantly, but should not be the only source of diagnosis as it may overlook certain cancer patients or classify healthy patients as cancer patients, which have significant implications.

Dataset(s)

Describe the dataset(s) you used.

From the original CancerSeek research paper, we used their dataset of many biomarkers and demographic data in our models. We decided to create 4 main datasets. 1) All Biographic and Demographic Markers, 2) All Biographic Markers, 3) The biographic markers used in the student, and 4) All the Biographic Markers and Demographic markers for those with Colorectum cancer or no cancer. The starting unfiltered dataset has 41 proteins biomarkers, a circulating tumor mutant dna score, along with age and sex.

How were they collected?

They were collected in the original research paper from real patients and using a multi-analyte blood test.

Why did you choose them?

There was a robust dataset with a binary classification for if they had cancer or not.

How many examples in each?

There were 1817 total patients, of which 1005 had cancer and 812 were healthy. The classes were slightly imbalanced.

In [76]:
```
# Load your data and print 2-3 examples
import pandas as pd
dataset = pd.read_csv('dataset/Consolidated_CancerSEEK_Data.csv')
dataset.tail()
```

Out[76]:

	Patient ID #	Sample ID #	Age	Sex	Race	Tumor type	AJCC Stage	Ω score	AFP (pg/ml)	Angiopoietin-2 (pg/ml)	...	S
1812	INDI 512	INDI 512 PLS 1	47	0	Caucasian	Breast	II	0.47	822.14	957.01	...	1
1813	INDI 702	INDI 702 PLS 1	74	1	Caucasian	Lung	II	3.95	1349.07	1797.39	...	5
1814	INDI 048	INDI 048 PLS 1	79	0	Caucasian	Breast	II	0.96	781.39	2075.34	...	8
1815	LCR 592	LCR 592 PLS1	53	0	Caucasian	Normal	0	0.97	2782.15	759.62	...	3
1816	PANC 760	PANC 760 PLS 1	48	1	Caucasian	Pancreas	II	0.45	849.62	662.45	...	4

5 rows × 50 columns

Pre-processing

What features did you use or choose not to use? Why?

The CancerSeek article only used 8 potential protein biomarkers and ctDNA (circulating tumor mutant DNA), however we had access to more data than this. We decided to use all of the features to see if that would allow us to refine their model, and analyze different subsets of this based on type of data. This led to the three aforementioned datasets, as the bio/dem dataset included all of the features and the demographic data, the bio dataset included all of the features, and the replicated dataset only included the features used in the CancerSEEK model. This was done since we wanted to refine their model, and we wanted to compare our accuracy with all of the features to the ones CancerSEEK used. We assumed if some features weren't impactful, our models (such as neural networks) would ideally not assign weights to them as it would learn which features were the impactful ones.

How did you deal with missing data? What about outliers?

Luckily, we had a very robust dataset so there were very few N/A inputs in our data. Therefore, we assigned 0s to these values as we assumed it would be negligible.

What approach(es) did you use to pre-process your data? Why?

Once again, since we had a very robust dataset, very minimal pre-processing was needed, which was confirmed by Josh. We filled in the N/A values as stated above, converted gender to numerical values, and we standardized the data (setting each feature to have a mean of 0 and variance of 1) before feeding it to the neural network.

Are your features continuous or categorical? How do you treat these features differently?

We had almost entirely continuous data, since the values for protein biomarker expression or ctDNA are continuous numerical values. The only categorical data was gender in the demographic dataset, so we made that data binary (0,1 for male and female, respectively), and race. However, we didn't use race since we assumed it wouldn't impact our models and it wasn't very good data, as there were a disproportionate amount of caucasians in the dataset and several hundred patients whose race was unknown.

In [32]:
```
# For those same examples above, what do they look like after being pre-processed
```

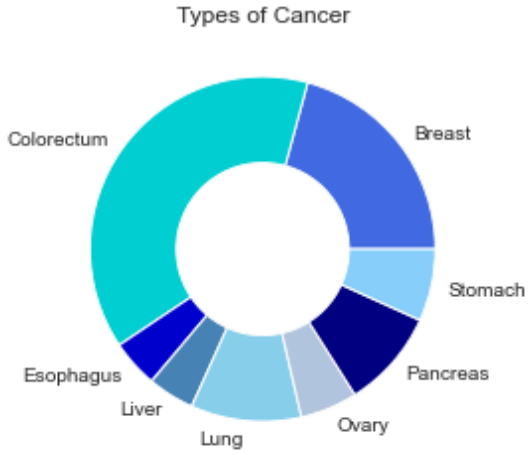
In [33]:
```
# Visualize the distribution of your data before and after pre-processing.
#   You may borrow from how we visualized data in the Lab homeworks.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scikitplot as skplt
from sklearn.metrics import confusion_matrix

df = pd.read_csv("dataset/Consolidated_CancerSEEK_Data.csv")
array = df.values
# np.unique(array[:, 5], return_index=False, return_inverse=False, return_counts=
cancerTypes = ['Breast', 'Colorectum', 'Esophagus', 'Liver', 'Lung', 'Normal',
       'Ovary', 'Pancreas', 'Stomach']
cancerTypeCases = []

# cases for each cancer type
for i in range(len(cancerTypes)):
    cancerTypeCases.append(array[:, 5][array[:, 5] == cancerTypes[i]].shape[0])
    if cancerTypes[i] == 'Normal':
        print("Normal cases: " + str(cancerTypeCases[i]))
    else:
        print(cancerTypes[i] + " cancer cases: " + str(cancerTypeCases[i]))
```

```
Breast cancer cases: 209
Colorectum cancer cases: 388
Esophagus cancer cases: 45
Liver cancer cases: 44
```

```
Lung cancer cases: 104
Normal cases: 812
Ovary cancer cases: 54
Pancreas cancer cases: 93
Stomach cancer cases: 68
```

In [34]:

```
names = ['Breast', 'Colorectum', 'Esophagus', 'Liver', 'Lung', 'Ovary', 'Pancreas
size = [209, 388, 45, 44, 104, 54, 93, 68]

my_circle = plt.Circle( (0,0), 0.5, color='white')

plt.pie(size, labels=names, colors=['royalblue', 'darkturquoise', 'mediumblue',
                                    'steelblue', 'skyblue', 'lightsteelblue', 'na

# pie chart of cancer types
p = plt.gcf()
p.gca().add_artist(my_circle)
plt.title("Types of Cancer")
plt.show()
```



In [35]:

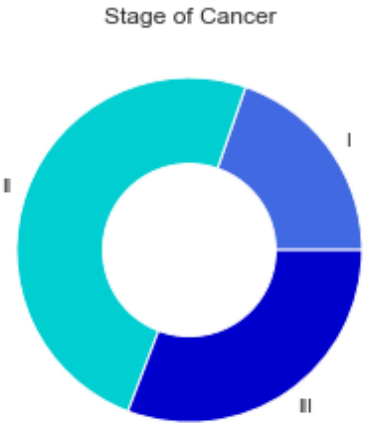
```
stageTypes = ['0', 'I', 'II', 'III']
stageTypeCases = []

for i in range(len(stageTypes)):
    stageTypeCases.append(array[1:, 6][array[1:, 6] == stageTypes[i]].shape[0])
    print("Stage " + stageTypes[i] + " cancer cases: " + str(stageTypeCases[i]))
```

```
Stage 0 cancer cases: 812
Stage I cancer cases: 199
Stage II cancer cases: 496
Stage III cancer cases: 309
```

In [36]:

```
names = ['I', 'II', 'III']
size = [198, 497, 309]

my_circle = plt.Circle( (0,0), 0.5, color='white')

plt.pie(size, labels=names, colors=['royalblue', 'darkturquoise', 'mediumblue'])

# pie charts of stage types
p = plt.gcf()
p.gca().add_artist(my_circle)
```

```
plt.title("Stage of Cancer")
plt.show()
```



In [37]:

```
# distribution of control and cancer patients
cancerPatient = 0
controlPatient = 0
totalCases = 1817.
for value in array[:, 49]:
    if value == 1:
        cancerPatient += 1
    else:
        controlPatient += 1

print("Total number of cancer patients: " + str(cancerPatient))
print("Total number of healthy patients: " + str(controlPatient))
```

```
Total number of cancer patients: 1005
Total number of healthy patients: 812
```

In [38]:

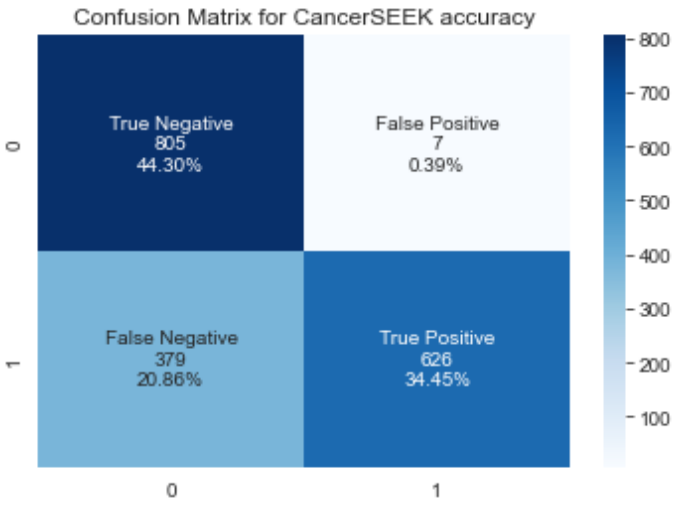
```
tpr = 0
tnr = 0
fpr = 0
fnr = 0

predAndActual = array[:, 48:]

# counting tpr, tnr, fpr, fnr cases
for i in range(len(predAndActual)):
    if predAndActual[i,0] == 0:
        if predAndActual[i,0] == predAndActual[i,1]:
            tnr += 1
        else:
            fnr += 1
    else:
        if predAndActual[i,0] == predAndActual[i,1]:
            tpr += 1
        else:
            fpr += 1
# creating confusion matrix
cf_matrix = np.zeros((2, 2))
cf_matrix[0,0] = 805
cf_matrix[0,1] = 7
cf_matrix[1,0] = 379
```

```
cf_matrix[1,1] = 626
print("Original Study:")
print(cf_matrix)
```

```
Original Study:
[[805.   7.]
 [379. 626.]]
```

In [39]:

```
# plotting confusion matrix
group_names = ['True Negative','False Positive','False Negative','True Positive']
group_counts = ["{0:0.0f}".format(value) for value in cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in cf_matrix.flatten()/np.

labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(group_names,group_counts,grou
labels = np.asarray(labels).reshape(2,2)
ax = plt.axes()
sns.heatmap(cf_matrix, annot=labels, fmt = '', cmap='Blues', ax = ax)
ax.set_title('Confusion Matrix for CancerSEEK accuracy')
plt.show()
```



In [40]:

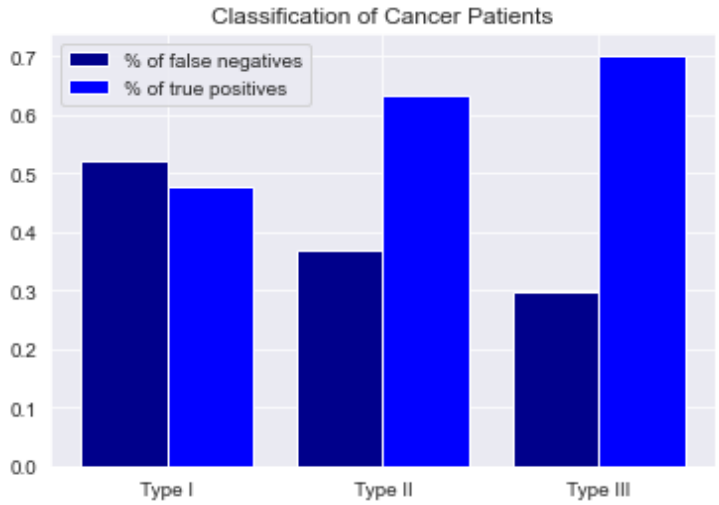
```
# initializing variables to count fnr or tpr / stage types
tpr = 0
tnr = 0
fpr = 0
fnr = 0
count11 = 0
count12 = 0
count13 = 0
count21 = 0
count22 = 0
count23 = 0

predAndActual = array[:, 48:]

for i in range(len(predAndActual)):
    if predAndActual[i,0] == 0:
        if predAndActual[i,0] == predAndActual[i,1]:
            tnr += 1
        else:
            fnr += 1
            if array[i,6] == 'I':
                count11 += 1
```

```
            elif array[i,6] == 'II':
                count12 += 1
            elif array[i,6] == 'III':
                count13 += 1


    else:
        if predAndActual[i,0] == predAndActual[i,1]:
            tpr += 1
            if array[i,6] == 'I':
                count21 += 1
            elif array[i,6] == 'II':
                count22 += 1
            elif array[i,6] == 'III':
                count23 += 1
        else:
            fpr += 1

print("False negatives")
print("Stage 1: " + str(count11))
print("Stage 2: " + str(count12))
print("Stage 3: " + str(count13))
print("True positives")
print("Stage 1: " + str(count21))
print("Stage 2: " + str(count22))
print("Stage 3: " + str(count23))
```

```
False negatives
Stage 1: 104
Stage 2: 183
Stage 3: 92
True positives
Stage 1: 95
Stage 2: 314
Stage 3: 217
```

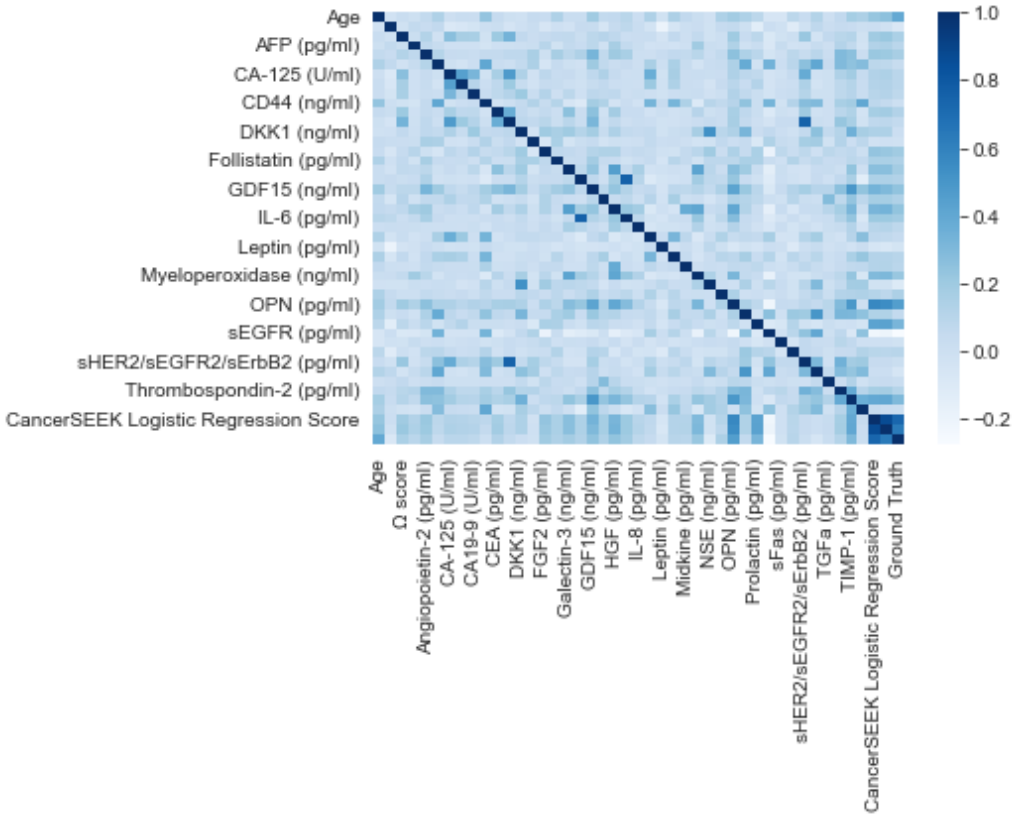
In [41]:

```
X = ['Type I','Type II','Type III']
fnr = [104/(104+95), 183/(183+314), 92/(92+217)]
tpr = [95/(104+95), 314/(183+314), 217/(92+217)]

X_axis = np.arange(len(X))

plt.bar(X_axis - 0.2, fnr, 0.4, label = '% of false negatives', color = 'darkblue
plt.bar(X_axis + 0.2, tpr, 0.4, label = '% of true positives', color = 'blue')

plt.xticks(X_axis, X)
plt.title("Classification of Cancer Patients")
plt.legend()
plt.show()
```



In [42]:
```
sns.heatmap(df.corr(), cmap="Blues")
```

Out[42]: <AxesSubplot:>



Models and Evaluation

Experimental Setup

How did you evaluate your methods? Why is that a reasonable evaluation metric for the task?

Our main source of evaluation for our methods was comparison to the original study. In the original paper, they only performed a logistic regression to determine if the individuals had cancer or not. We believed that with more advanced model types we could 1) get a model that performs with higher accuracy and 2) have a model with higher interpretability. The evaluation metrics we used for this model were overall accuracy, specificity, and sensitivity, which are common metrics for diagnostic purposes. These are reasonable evaluation metrics for this task due to it aligning with industry standard for evaluation of diagnostic tools. We also incorporated an ROC curve as well.

What did you use for your loss function to train your models? Did you try multiple loss functions? Why or why not?

Neural Network - Binary Cross Entropy Logistic Regression - Binary Cross Entropy Random Forest - N/A We did not try multiple loss functions, as BCE was appropriate for the classification task.

How did you split your data into train and test sets? Why?

We began by shuffling our data set so that the order of the data would be random. We used a test dataset of size 200, and then with the remaining 1600 samples, 80% were used for trainig and 20% for dev, so we had enough data to train our model.

Baselines

Baselines came from the original paper and the algorithm results in CancerSEEK. Baseline results are discussed in the results section.

Methods

What methods did you choose? Why did you choose them?

How did you train these methods, and how did you evaluate them? Why?

Which methods were easy/difficult to implement and train? Why?

For each method, what hyperparameters did you evaluate? How sensitive was your model's performance to different hyperparameter settings?

In [1]:
```
# Code for training models, or link to your Git repository
```

In [2]:
```
# Show plots of how these models performed during training.
#  For example, plot train loss and train accuracy (or other evaluation metric) o
#  with number of iterations or number of examples on the x-axis.
```

Random Forest

https://github.com/ChrisWilhelm/MachineLearningFinalProject/blob/main/RandomForest.py

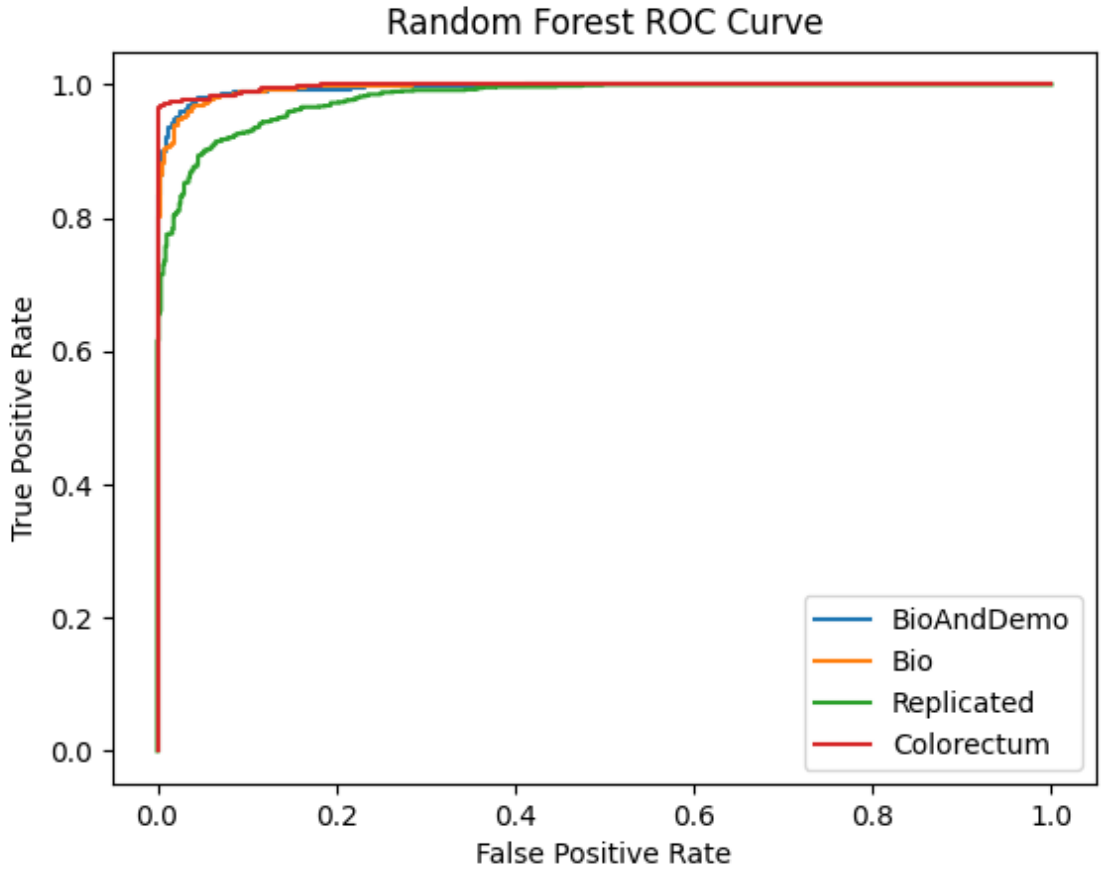
One method we chose was a Random Forest (link to file above). We initially chose this as we believed it would have the highest interpretability which would lead to better knowledge of which of the proteins actually signal cancer is present in the patient.

For training our models for Random Forests, we used Grid Search Cross Validation to evaluate a number of parameters such as the number of estimators, max depth, min sample split, and criterion. Although the models for random forests are on different datasets, we found that the hyperparameters between the models were very similar for which performed best, so we decided to keep them consistent for the sake of comparison. The random forest specifically was very sensative to hyperparameter change at low number of estimators and low max depth. This makes sense as when there are a low number of estimators and not a very deep tree, this will lead to under fitting as not all the data will be able to be examined by the tree and can lead to a very high bias. Once the hyperparameters were selected, we used cross fold validation to choose the best model with the highest training accuracy.

The Random Forest was a slightly harder than average method to implement, as we had not implemented one previously. We found that once we understood the hyperparameters and what they were referencing, it was not complicated to implement. One aspect of the model that was harder than anticipated was the interpretability with producing the trees shown below and gathering information on the false positives and where there prediction went wrong. In the python file above, we have commented out the code to create new PNGs and printing the path taken by the trees on traversal. This took a while to learn how to generate the trees and also a while to learn how to find the path of traversal.

In [47]:
```
from PIL import Image
rf_roc = Image.open('graphs/rf_roc.png')
rf_roc
```

Out[47]:

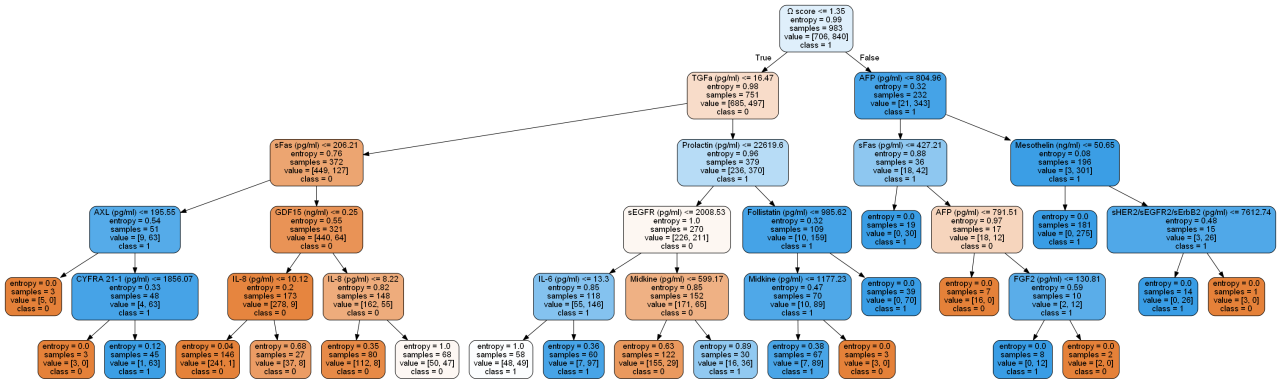


In [48]:

```
Bio_tree = Image.open('RandomForestVisual/Bio/bio_tree5.png')
print("Biomarker Data Tree(Tree #5)")
print("Blue = Cancer Positive, Orange = Negative")
Bio_tree
```

```
Biomarker Data Tree(Tree #5)
Blue = Cancer Positive, Orange = Negative
```

Out[48]:

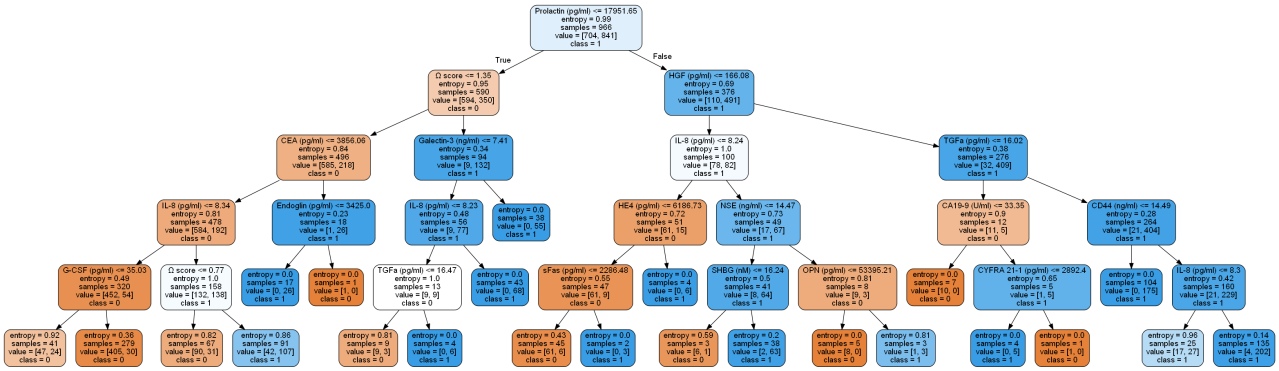


In [49]:

```
Dem_tree = Image.open('RandomForestVisual/BioAndDemo/dem_tree11.png')
print("Demographic and Biomarker Data Tree(Tree #11)")
print("Blue = Cancer Positive, Orange = Negative")
Dem_tree
```

```
Demographic and Biomarker Data Tree(Tree #11)
Blue = Cancer Positive, Orange = Negative
```

Out[49]:

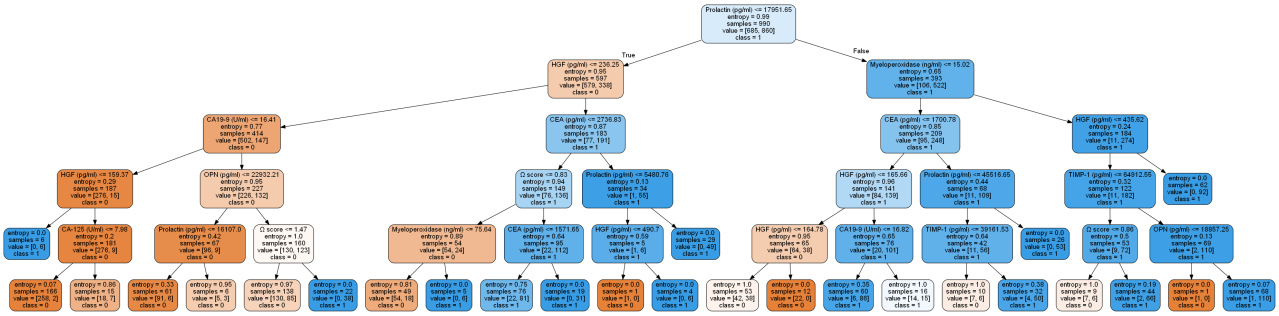


In [50]:

```
Rep_tree = Image.open('RandomForestVisual/Replicated/rep_tree0.png')
print("Replicated Data Tree(Tree #0)")
print("Blue = Cancer Positive, Orange = Negative")
Rep_tree
```

```
Replicated Data Tree(Tree #0)
Blue = Cancer Positive, Orange = Negative
```

Out[50]:

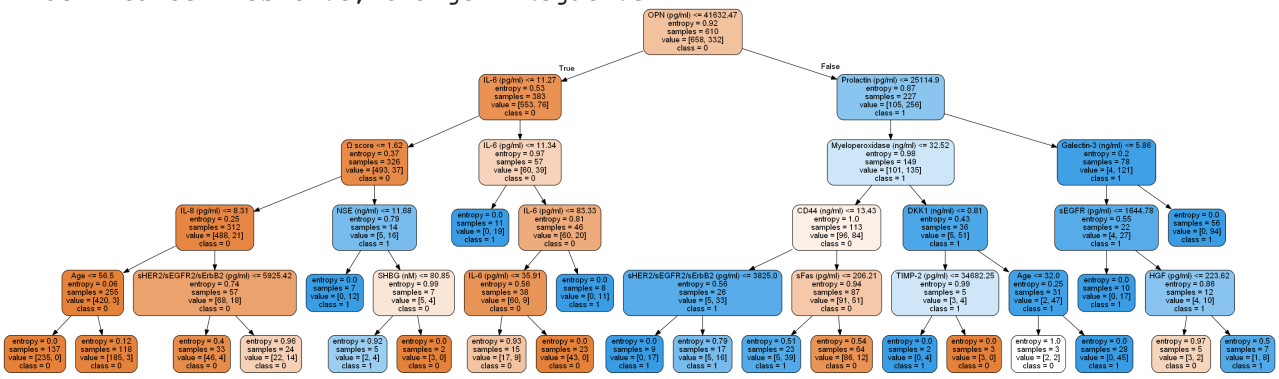


In [51]:

```
col_tree = Image.open('RandomForestVisual/Colorectum/colorectum_tree8.png')
print("Biomarker Data Tree(Tree #8)")
print("Blue = Cancer Positive, Orange = Negative")
col_tree
```

```
Biomarker Data Tree(Tree #8)
Blue = Cancer Positive, Orange = Negative
```

Out[51]:



In [52]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# importing file with random forest paths for false positive individuals and wher
# NaN means they started off as 1 and never improved (so there were no incorrect
df = pd.read_csv("RandomForestVisual/RandomForestPaths.csv")
array = df.values
```

```
for i in range(1,15):
    print(array[:4,2*i]) #dev bio dataset
    #print(array[6:9, 2*i]) #bio dataset
    #print(array[11:, 2*i]) #replicated dataset
```

```
[nan nan nan 'TBD']
['Age' 'Age' 'GDF15' nan]
['IL-8' nan 'CA19-9' 'CA19-9']
['CYFRA 21-1' nan nan nan]
[nan nan nan nan]
[nan nan nan nan]
['Age/OPN' nan nan nan]
['IL-8' nan nan nan]
['OPN' nan 'OPN' 'Prolactin']
['TIMP-1' nan nan 'HGF']
[nan nan 'TGFa' nan]
[nan nan nan nan]
['IL-8' 'IL-8' 'IL-6' nan]
[nan nan 'Age/IL-6/Thrombospondin' nan]
```

Age: 2

Age/OPN: 2

GDF15: 1

CA19-9: 2

CYFRA 21-1: 1

Prolactin: 1

Age/IL-6/Thrombospondin: 1

IL-8: 4

IL-6: 1

TGFa: 1

TIMP-1: 1

HGF: 1

OPN: 2

Total: 20

In [53]:

```
for i in range(1,15):
    print(array[6:9, 2*i]) #bio dataset
```

```
['sEGFR' nan nan]
[nan nan nan]
['sEGFR' 'HGF' nan]
['HE4' nan nan]
[nan nan nan]
[nan nan nan]
[nan nan nan]
[nan nan nan]
['Angiopoietin-2' nan nan]
['CYFRA 21-1' nan nan]
[nan nan nan]
[nan 'CA19-9' nan]
['Midkine' 'OPN' nan]
['OPN' nan nan]
```

sEGFR: 2

HGF: 1

HE4: 1

Angiopoietin-2: 1

CYFRA 21-1: 1

Midkine: 1

OPN: 2

Total: 9

In [54]:
```
for i in range(1,15):
     print(array[11:, 2*i]) #replicated dataset
```

```
[nan nan nan nan nan nan nan nan nan nan 'CEA/Prolactin']
[nan nan 'CEA' nan nan nan nan nan 'CEA' nan 'CEA']
[nan nan nan nan nan nan nan 'HGF' nan nan nan]
[nan nan 'CEA' nan nan nan nan nan 'CEA' 'Omega' 'CEA']
[nan nan nan 'OPN' nan nan nan 'CA-125' 'OPN' 'OPN' nan]
[nan 'Myeloperoxidase' nan nan nan nan nan 'Myeloperoxidase' nan nan nan]
['CA19-9' 'CA19-9' nan 'CA19-9' nan nan 'CA-125' nan nan 'CA19-9' 'CA19-9']
[nan 'CA-125' 'HGF' nan nan nan nan nan 'CA19-9' nan 'CA-125']
['CA-125' nan nan 'CA-125' nan nan 'CA-125' nan nan 'CA-125' nan]
[nan 'CA19-9' nan 'CA19-9' nan nan nan nan nan 'CA19-9' nan]
[nan nan 'TIMP-1' nan nan 'CPN' nan 'CA-125' nan nan nan]
[nan nan nan nan nan nan 'TIMP-1' 'TIMP-1' 'TIMP-1' 'TIMP-1' 'TIMP-1']
[nan nan nan nan nan 'Omega' nan nan nan nan nan]
[nan nan nan 'TIMP-1' nan 'Omega' nan nan 'TIMP-1' nan nan]
```

CEA: 6

CEA/Prolactin: 1

HGF: 2

CA-125: 9

CA19-9: 9

TIMP-1: 8

Omega: 3

CPN: 1

Myeloperoxidase: 2

Total: 41

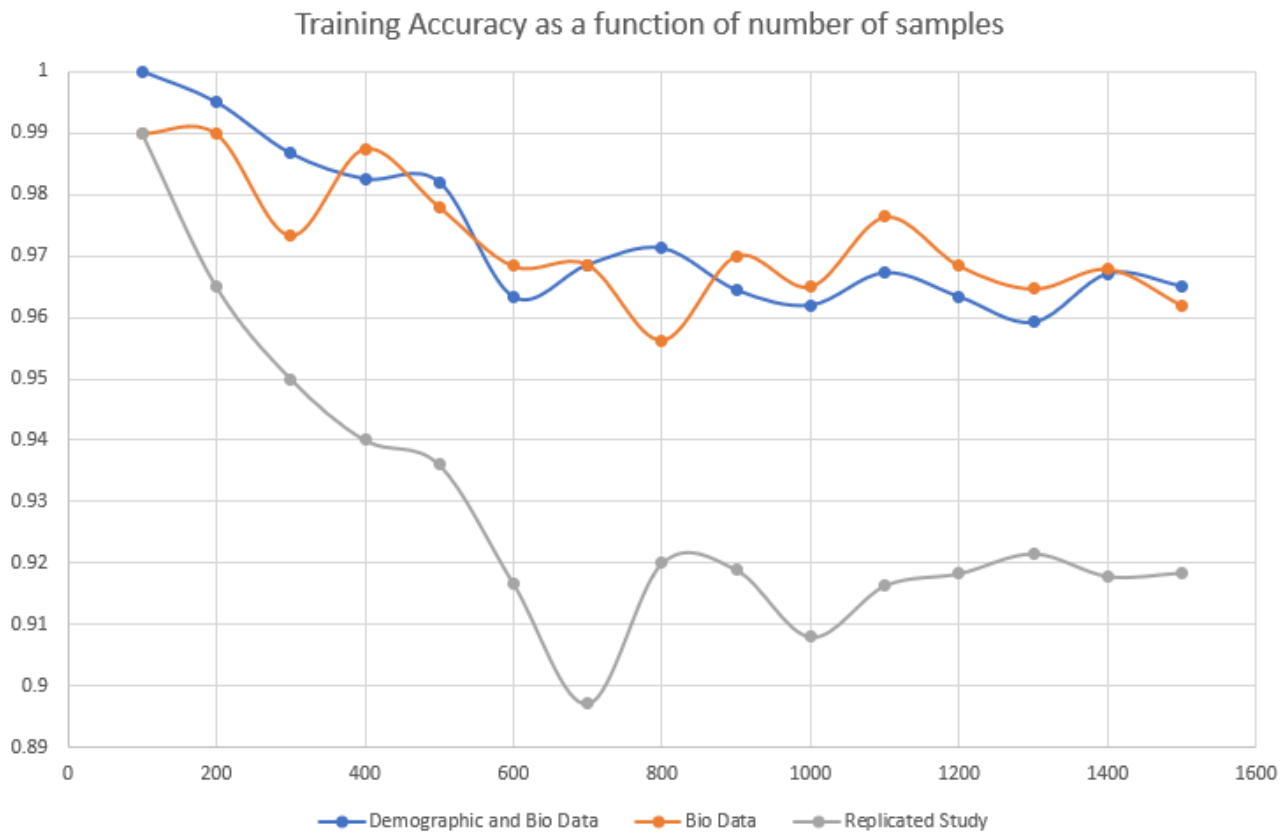
After generating the random forests and finding the specificity on our different datasets (94.8% for dem/bio, 95.2% for bio, 92.8% for replicated), we searched through all of the false positive cases to see if there was a common denominator in all of the trees which incorrectly classified the patients. For the dem/bio dataset, age was the cause of incorrect classifications in 25% of the cases, IL-8 was the cause of 20% of the cases, and for the replicated dataset, CA-125 and CA19-9 were the cause of 21% of the cases.

However, since there were so many decisions in the trees and these features were approximately uniform over all of them, we believe that the reason that some features were present in more of the incorrectly classified cases were simply due to variance. Similarly, due to the limited dataset we do have present, we aren't able to check any additional cases of incorrect classification, so

this is the extent that we are able to analyze this issue. Thus, we cannot conclude that any features are causing a disproportional amount of false positive classifications.

In [43]:
```
rf_acc = Image.open('graphs/RandomForestAccuracyPlot.png')
rf_acc
```

Out[43]:



Neural Network

https://github.com/ChrisWilhelm/MachineLearningFinalProject/blob/main/NeuralNetwork.py
Another method we implemented was a Neural Network. We chose this model as well for its robust ability to represent non-linear data, and we suspected that the data came from a non-linear source and had a nonlinear relationship with cancer diagnoses.

Our model structure was a Neural Network with 2 hidden layers, each with a width of 100 nodes. We utilized dropout on the first hidden layer, with a probability of 0.2. Our activation functions were ReLu, PReLu, and then the Sigmoid function to result in a binary classification algorithm. OOur loss function used BCELoss, which is appropriate for the task of binary classification. We also used SGD for our optimization method.

To train our models, for each dataset type (demographic + biomarker, biomarker, replicated), we fed it through our neural network with a learning rate of .005 and epochs of 400. Hyperparameters learning rate, epochs, dropout rate, and hidden layer width were primarily evaluated when optimizing our model. The model had its best accuracy with our implemented learning rate, with model performance rapidly decreasing after .01, potentially due to vanishing

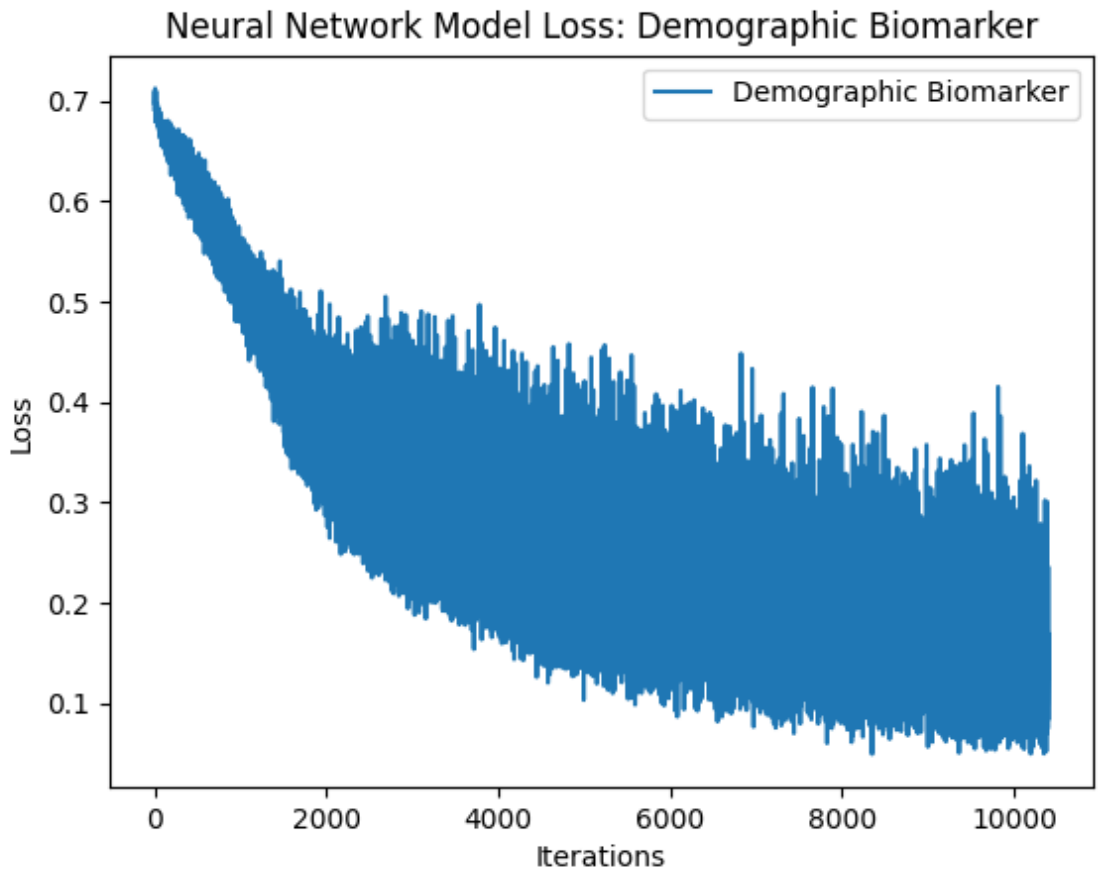
gradients. Dropout rate was also optimized at around 0.2-0.3, with the model worsening its fit to the training data past that. At 400 epochs, the loss of the model began to plateau, signaling that it had fit the training data.

One aspect of implementing the model that was difficult was around hyperparameter tuning and manipulating prediction scores to favor improved specificity.

Below are the training loss for our models on the train data, and the ROC curves for the dev data. A key note here is that, for our model loss, we did not average the individual losses, hence the noise in the loss plot is observable. However, we can see that the general average and direction of the loss begins to approach and plateau at the end of our training, so we can see the model is fitting the training data.

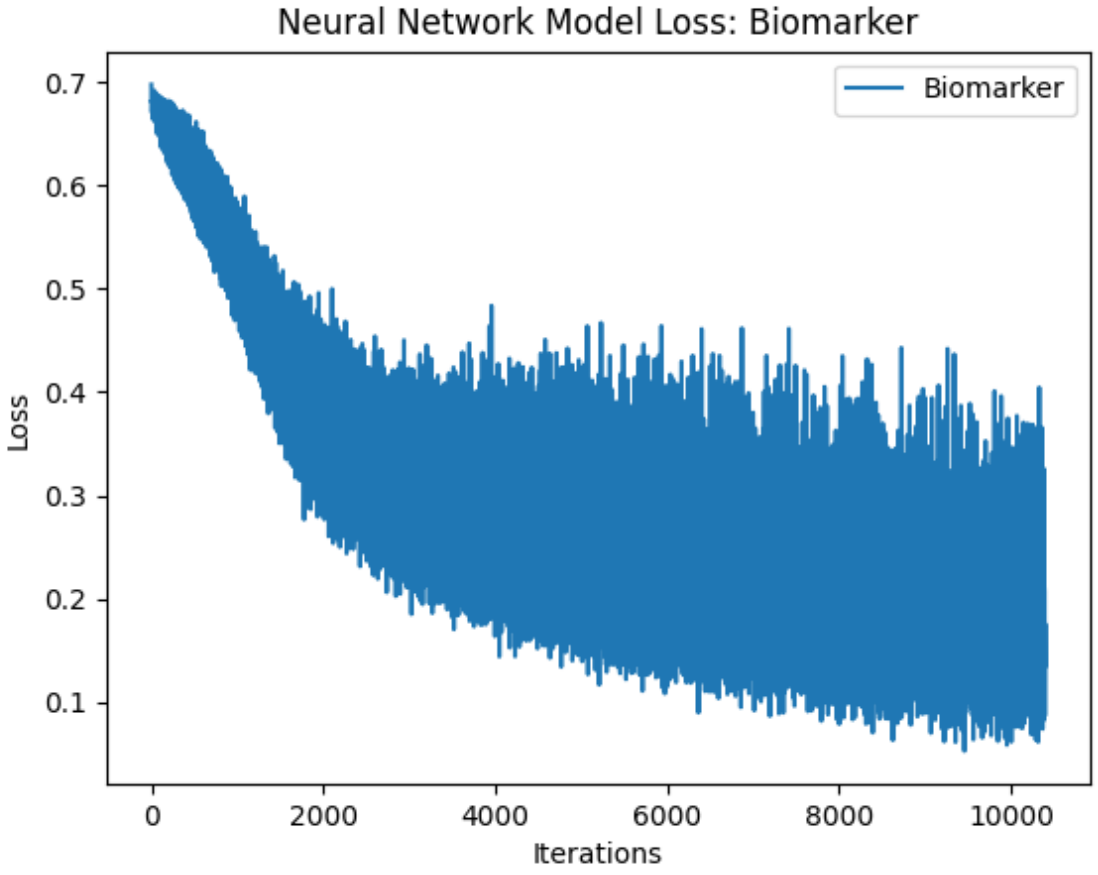
In [44]:
```
from PIL import Image
nn_loss_train_demographic = Image.open('graphs/nn_loss_train_Demographic Biomarke
nn_loss_train_demographic
```

Out[44]:



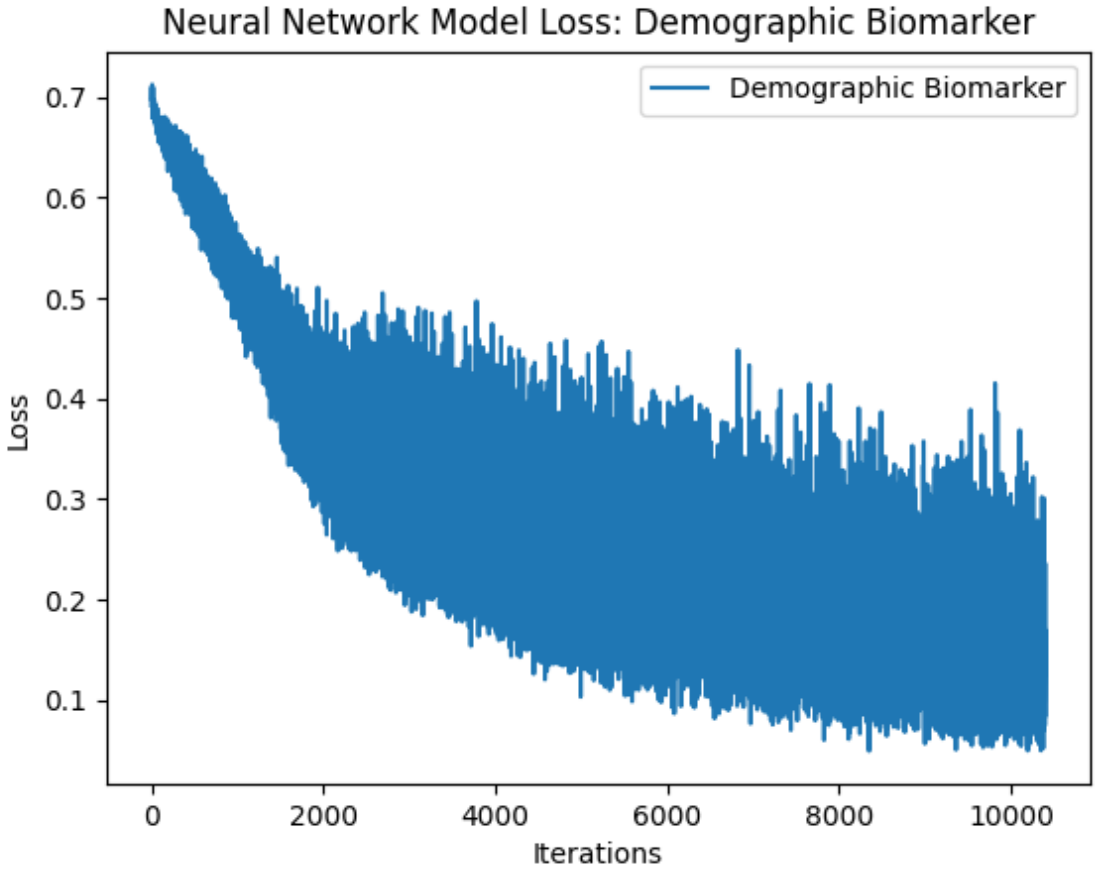
In [45]:
```
nn_loss_train_biomarker = Image.open('graphs/nn_loss_train_Biomarker.png')
nn_loss_train_biomarker
```

Out[45]:



In [46]:
```
nn_loss_train_replicated = Image.open('graphs/nn_loss_train_Demographic Biomarker
nn_loss_train_replicated
```

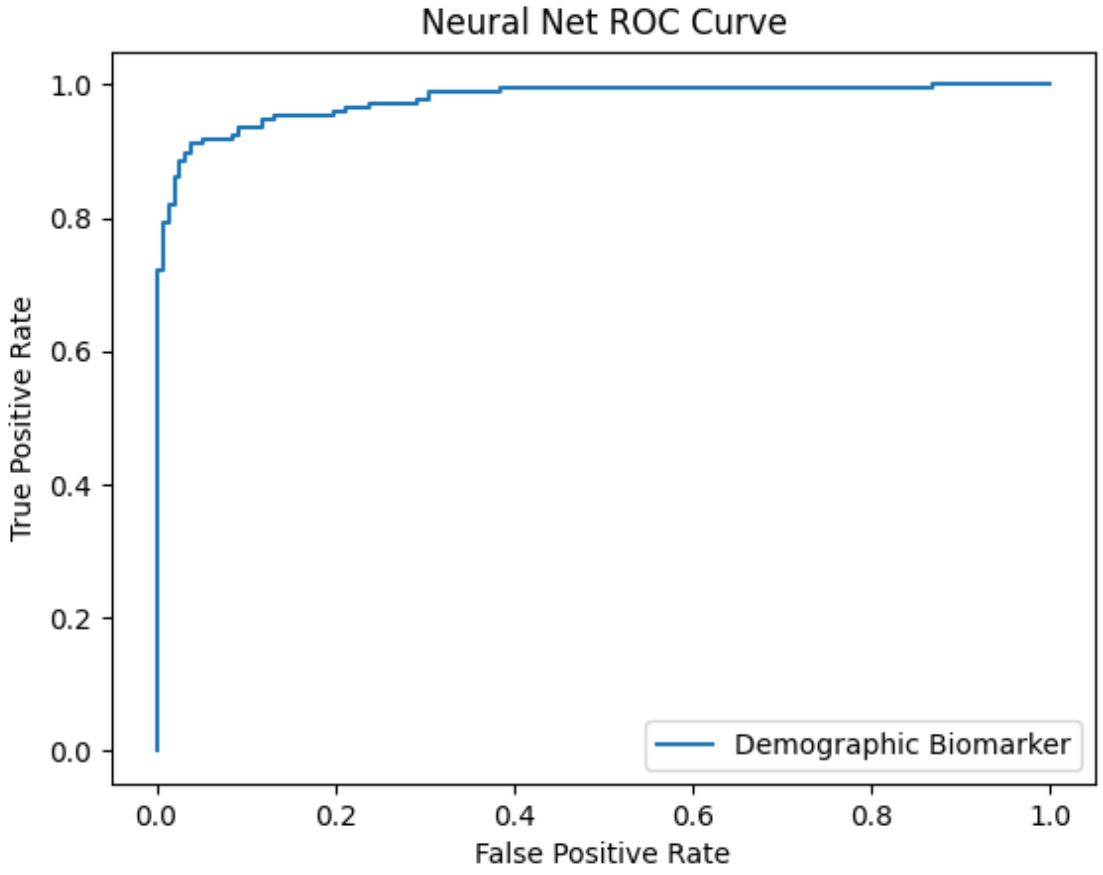
Out[46]:



In [47]:

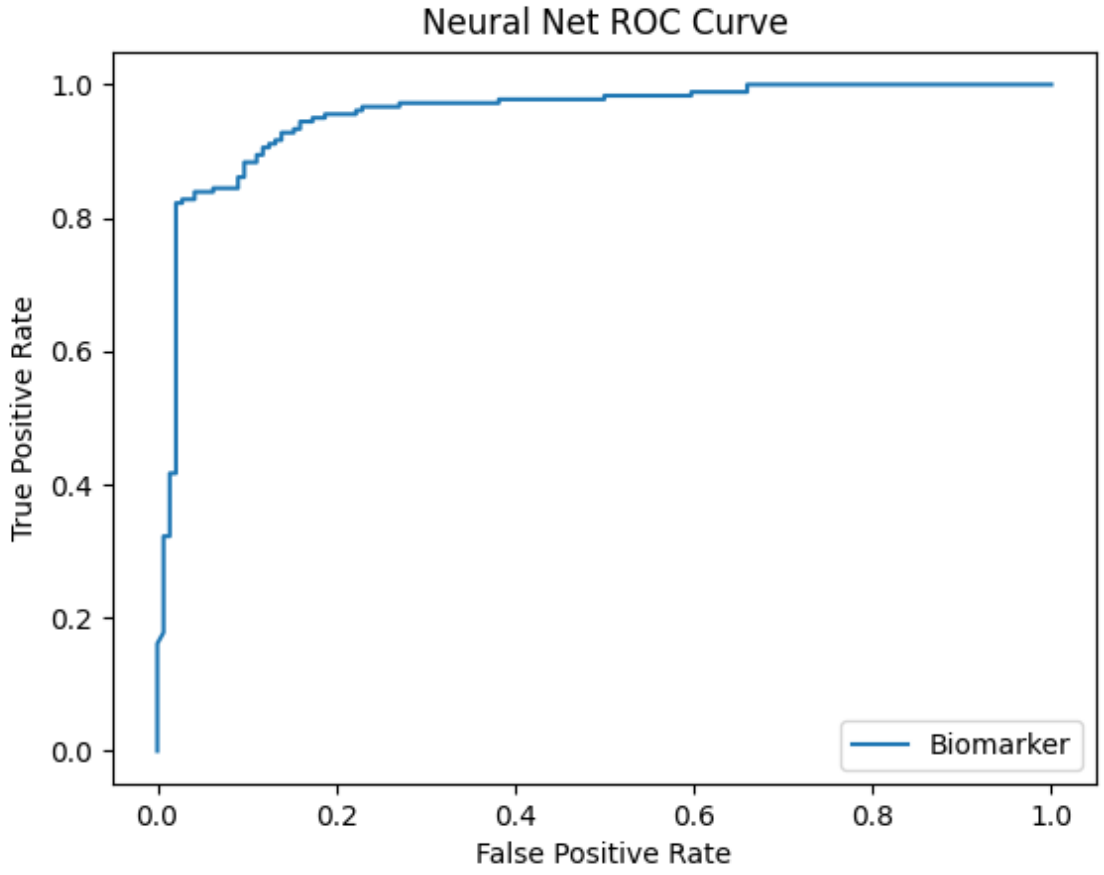
```
nn_roc_dev_replicated = Image.open('graphs/nn_roc_dev_Replicated.png')
nn_roc_dev_biomarker = Image.open('graphs/nn_roc_dev_Biomarker.png')
nn_roc_dev_demographic = Image.open('graphs/nn_roc_dev_Demographic Biomarker.png'

nn_roc_dev_demographic
```

Out[47]:



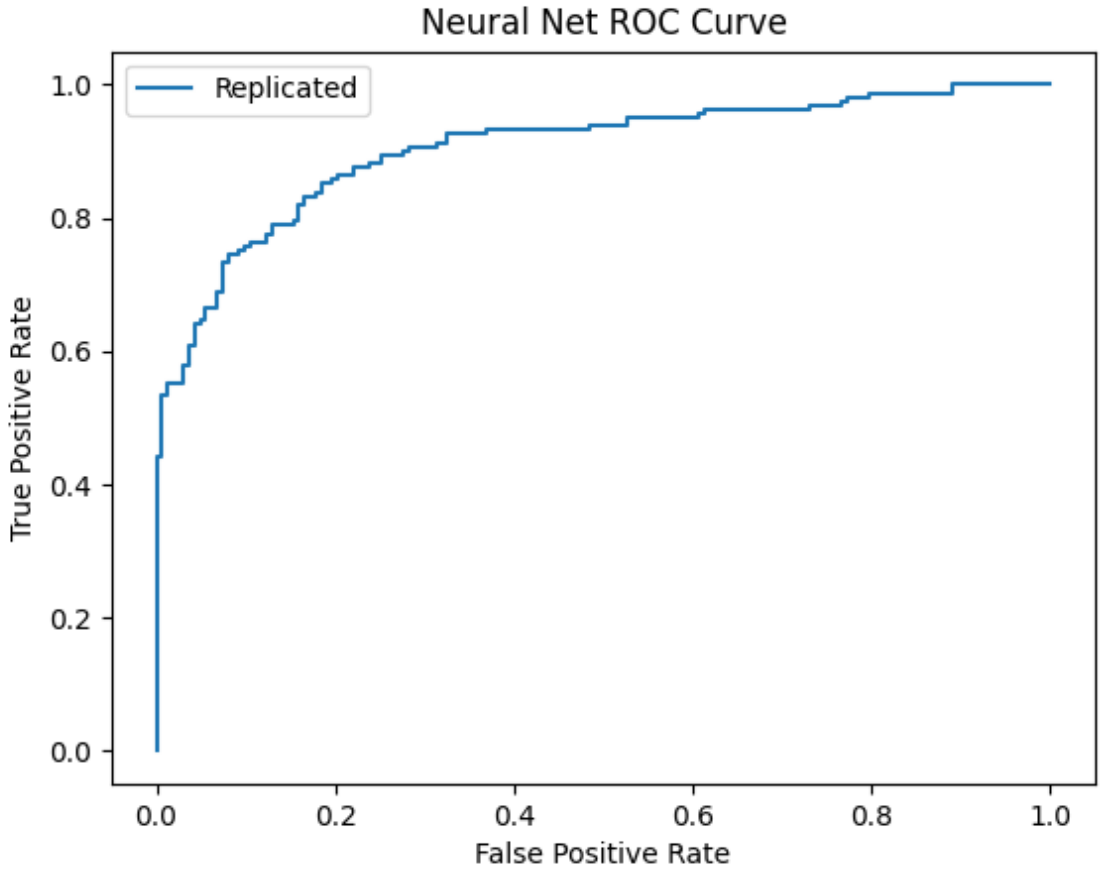
In [48]:
```
nn_roc_dev_biomarker
```

Out[48]:



In [49]:
```
nn_roc_dev_replicated
```

Out[49]:



Logistic Regression

https://github.com/ChrisWilhelm/MachineLearningFinalProject/blob/main/LogisticRegression.py

Another method we chose was Logistic Regression (link to file above). We initially chose this because we wanted to reproduce the original paper's CancerSEEK model. Additionally, logistic regression has a very high level of interpretability which is important for ML models when deployed in the medical field because patients like to know why they are or are not being diagnosed with a disease

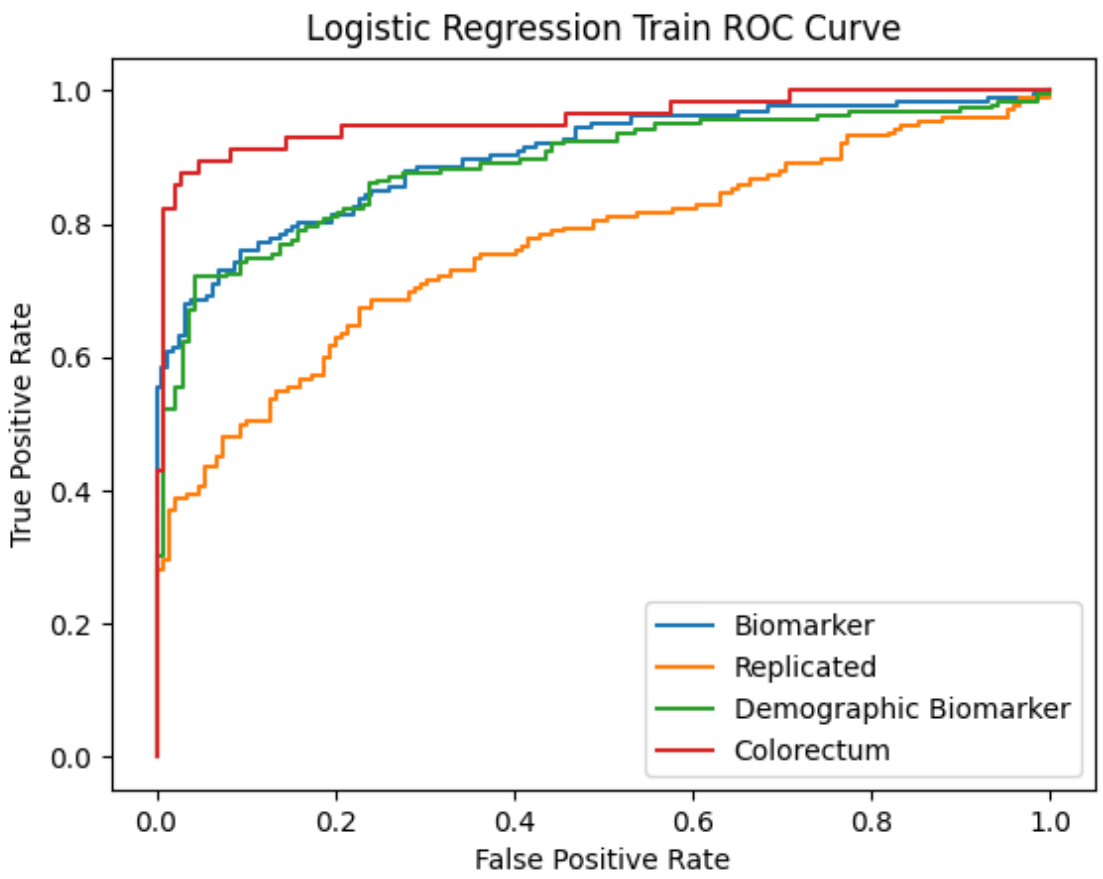
For training our models for Logistic Regression, we used 10-Fold Cross Validation to get a more comprehensive view of the accuracy of this model. We can see from the ROC plot below that the accuracy, sensitivity, and specificity of the model are all hugely dependent on the threshold we choose to classify. If we wanted a truly unbiased classifier, we would choose a threshold of 0.5. However, in the medical field it is often preferred to give false negatives instead of false positives so we might prefer to set the threshold higher (maybe 0.7 or 0.8) for this specific application. We settled with a threshold of 0.5 for the purposes of this analysis.

The logisitc regression model was fairly straightforward to implement. The underlying math is clear to us since we implemented one in one of the previous homeworks so it was a simple matter of using an sklearn package.

In [53]:
```
lr_roc_train = Image.open('graphs/lr_roc_train.png')
```

```
lr_roc_train
```

Out[53]:



Results

Show tables comparing your methods to the baselines.

What about these results surprised you? Why?

Did your models over- or under-fit? How can you tell? What did you do to address these issues?

What does the evaluation of your trained models tell you about your data? How do you expect these models might behave differently on different data?

Baseline Evaluation Metrics (CancerSEEK)

The performance of the CancerSEEK algorithm is shown below, along with its respective ROC curve for test data.

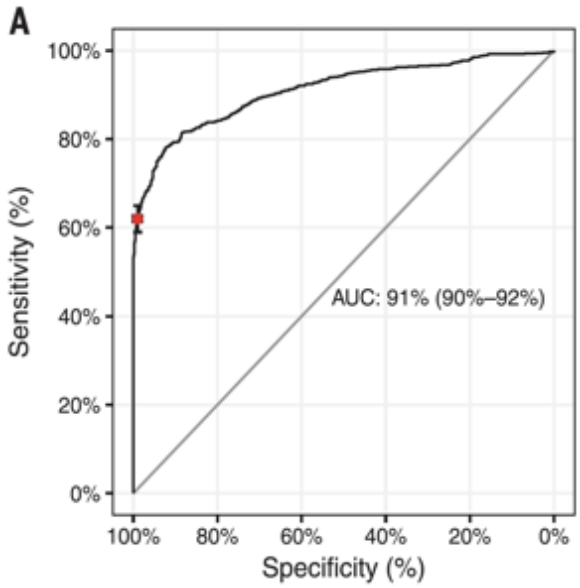
Metrics	CancerSEEK Data Set
ROC, AUC Score	0.91
Training Accuracy	N/A
Test Accuracy	0.788

Metrics	CancerSEEK Data Set
Specificity	0.996
Sensitivity	0.791

We can see here that, for the test data, the model significantly favored specificity over sensitivity. It is possible that the model could optimize better for overall accuracy, however due to the functionality of the test, a desirable property was high specificity.

In [41]:
```
cancerseek_ROC = (Image.open('graphs/CancerSEEK_ROC.png')).resize((300,300))
cancerseek_ROC
```

Out[41]:



Random Forest Evaluation Metrics

The performance of the Random Forest algorithm is shown below.

	Bio And Demographic	Bio	Replicated	Colorectum
Training ROC, AUC Score	0.994650294	0.994717508	0.9793735	0.99737583
Training Accuracy	0.953488372	0.964912281	0.9302326	1
Test Accuracy	0.91	0.9	0.81	0.97
Specificity	0.88372093	0.837209302	0.8604651	0.98484848
Sensitivity	0.929824561	0.947368421	0.7719298	0.94117647

We can see here that the Random Forest worked very well with our dataset. For the replicated dataset, comparing it to the CancerSEEK algorithm, overall accuracy on the test data was better with the RF implementation than the CancerSEEK implementation. All metrics were substantially improved with the larger datasets and the colorectum only dataset. It was surprising to see how well the RF worked compared to the logistic regression, likely showing that the data is nonlinear. Because generally speaking the training accuracy was substantially better than the test accuracy, it is possible that the model was slightly overfitting the data.

Neural Network Evaluation Metrics

The performance of the Random Forest algorithm is shown below.

	Bio And Demographic	Bio	Replicated	Colorectum
Training ROC, AUC Score	0.97645	0.95484	0.897708	0.96439
Training Accuracy	0.919753	0.942604	0.897708	0.9
Test Accuracy	0.86	0.835	0.775	0.915
Specificity	0.95	0.925	0.95	0.96323
Sensitivity	0.8	0.775	0.658333	0.8125

We can see here that the Neural Network worked well with our dataset. For the replicated dataset, comparing it to the CancerSEEK algorithm, overall accuracy on the test data was very similar with the Neural Network implementation than the CancerSEEK implementation. All metrics were substantially improved with the larger datasets and the colorectum only dataset. It was surprising to see that the neural network did not perform substantially better than the baseline logistic regression used in CancerSEEK, given that the RF suggests the data is nonlinear. It is possible that our model could be tuned better to fit this data. Because generally speaking the training accuracy was substantially better than the test accuracy, it is possible that the model was slightly overfitting the data.

Logistic Regression Evaluation Metrics

The performance of the Logistic Regression algorithm is shown below.

	Bio And Demographic	Bio	Replicated	Colorectum
Training ROC, AUC Score	0.97645	0.95484	0.897708	0.97550
Training Accuracy	0.919753	0.942604	0.897708	0.961562
Test Accuracy	0.86	0.835	0.775	0.86
Specificity	0.95	0.925	0.95	0.95
Sensitivity	0.8	0.775	0.658333	0.8

After creating the logistic regression models we found the finding the specificity on our different datasets (85.1% for dem/bio, 84.4% for bio, 35.9% for replicated). We were shocked by how poorly the model performed on the dataset replicated in the paper. Barely above 65% accuracy with a specificity of 35.9% is abysmal. We do not know how the original authors of the paper achieved a signficanty high accuracy than us with the same data and, according to them, the same logistic regression model. Our biomarker and demographic data outperforms their model, though.

We expect to be underfitting the data to some degree with this model. This is because we are using a simple logistic regression model even though the underlying data is almost certainly non-linear. However, this problem is inherent to logistic regression and we cannot add more complexity to the model to allow it to fit better.

If nothing else, logistic regression provides us with a good baseline. If any model we choose in the future cannot even outperform logistic regression, it is likely not worth considering as the final model because of the simplicity of logistic regression.

Confusion Matrices: Cancer Type

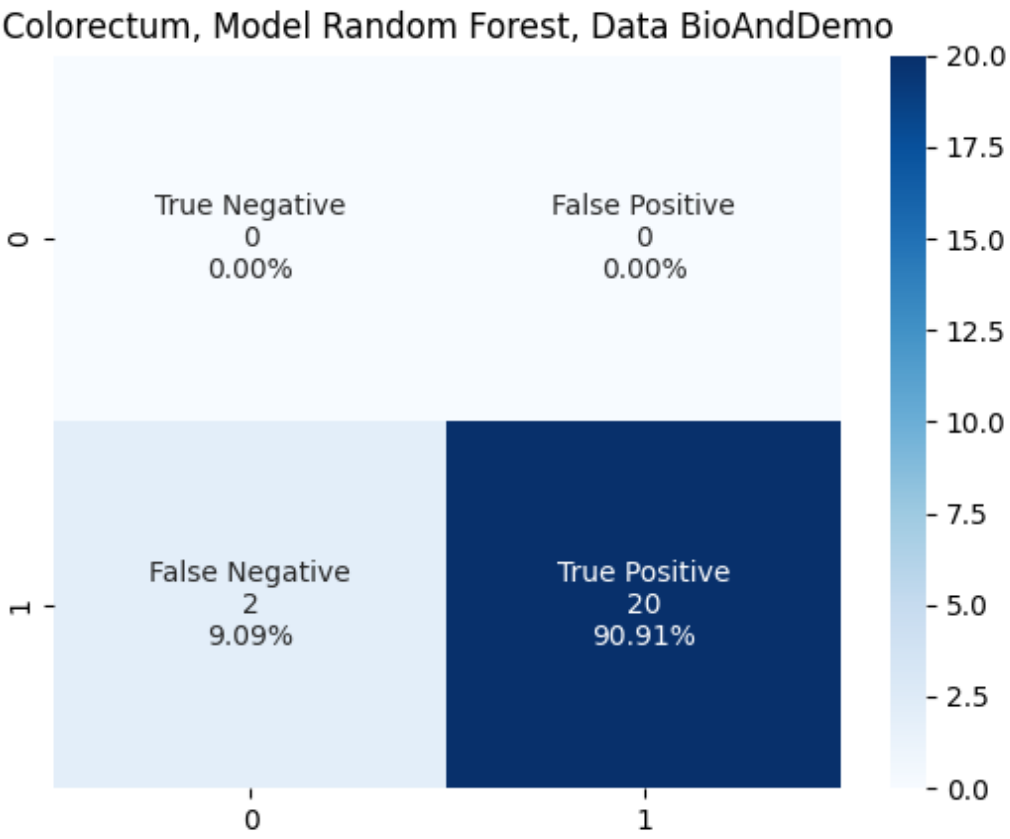
For each model, we created various confusion matrices to understand the evaluation of our algorithms on test data, on a per-cancer basis. Below are some examples of confusion matrices from our Random Forest Algorithm. We are only including a handful of examples from the RF algorithm for concision of the report and given the RF was our best performing model, and that Normal and Colorectum typically had the highest frequency in our test dataset. A comprehensive analysis of our performance and confusion matrices for each cancer per dataset for the RF, NN, and RL are available in the GitHub repo, in the graphs folder.

Random Forest: Confusion Matrices

In [72]:
```
rf_confusion_colorectum_demographic = Image.open('graphs/Random Forest_cfmatrix_C
rf_confusion_colorectum_bio = Image.open('graphs/Random Forest_cfmatrix_Colorectu
rf_confusion_colorectum_replicated = Image.open('graphs/Random Forest_cfmatrix_Co

rf_confusion_normal_demographic = Image.open('graphs/Random Forest_cfmatrix_Norma
rf_confusion_normal_bio = Image.open('graphs/Random Forest_cfmatrix_Normal_datase
rf_confusion_normal_replicated = Image.open('graphs/Random Forest_cfmatrix_Normal
```

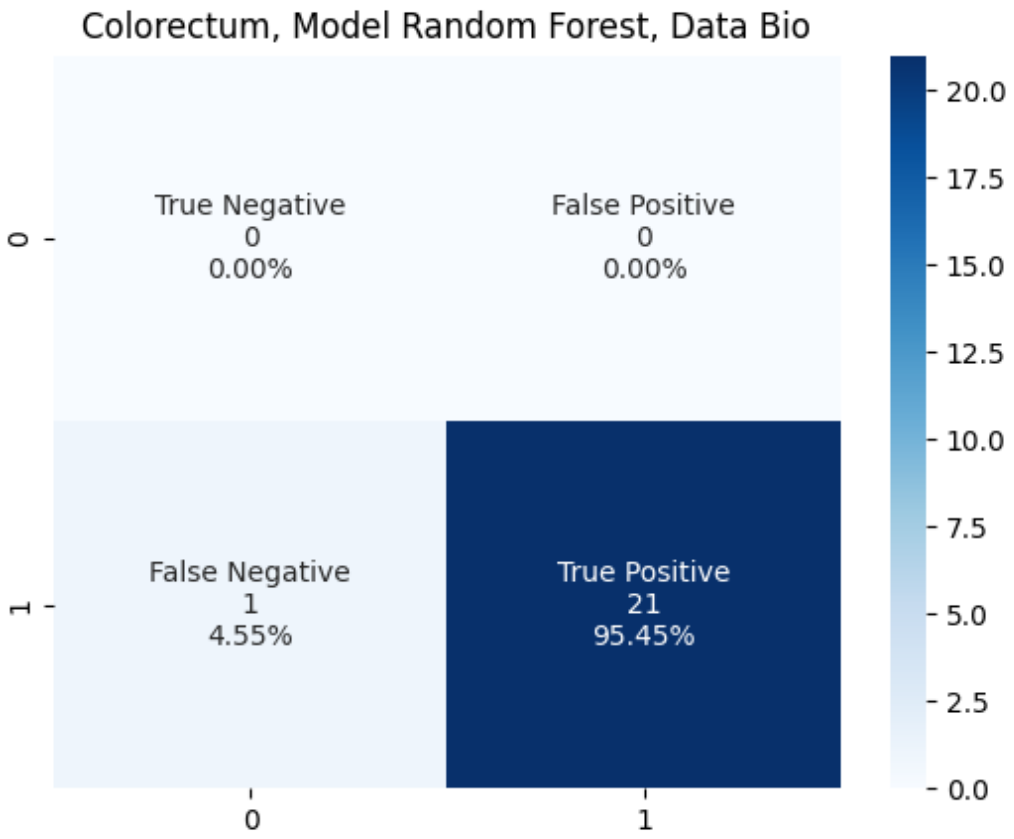
In [73]:
```
rf_confusion_colorectum_demographic
```

Out[73]:



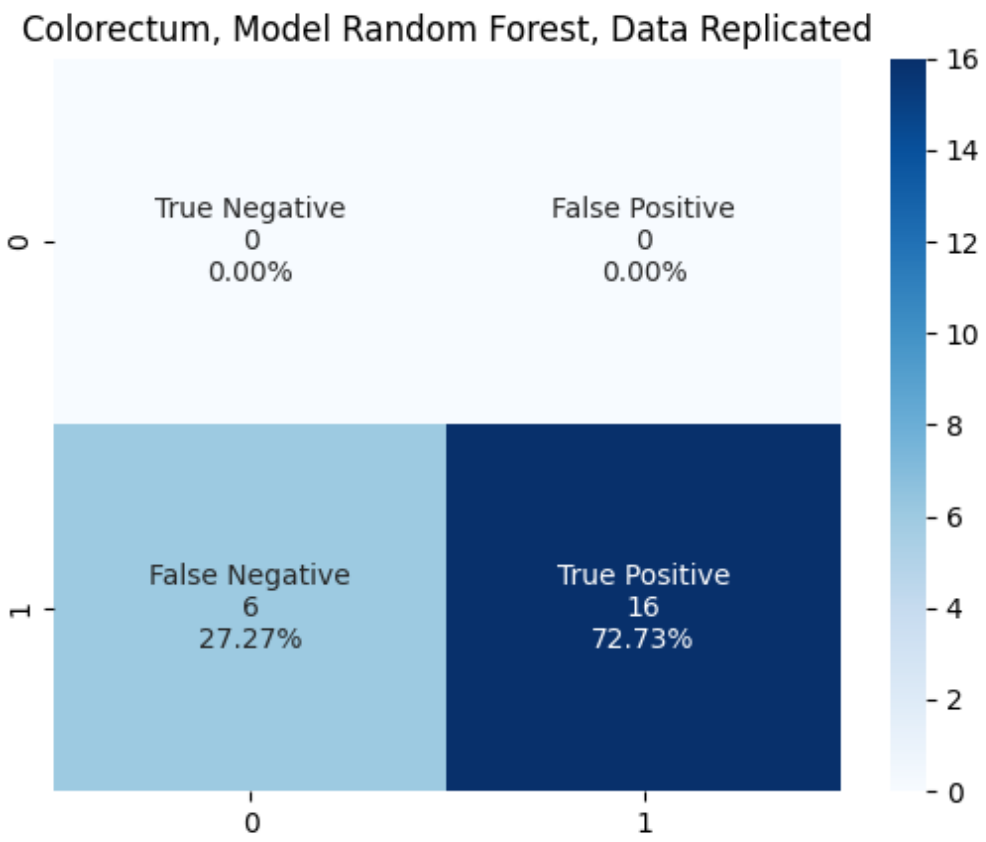
In [74]:
```
rf_confusion_colorectum_bio
```

Out[74]:



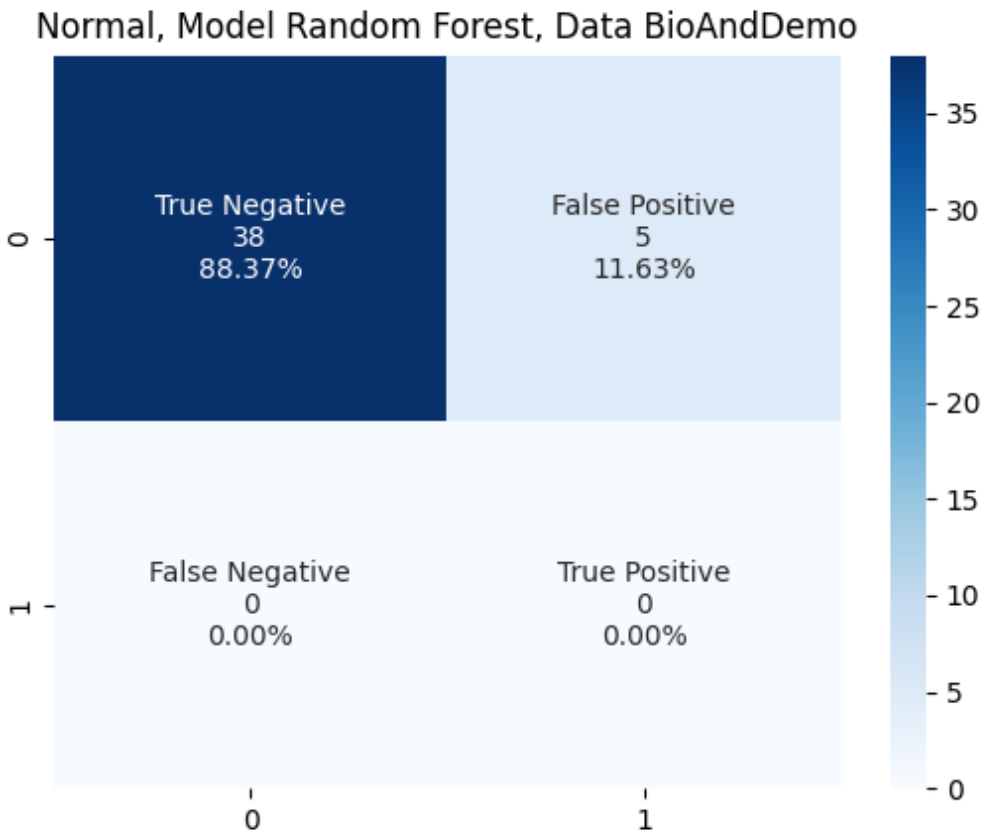
In [75]:
```
rf_confusion_colorectum_replicated
```

Out[75]:



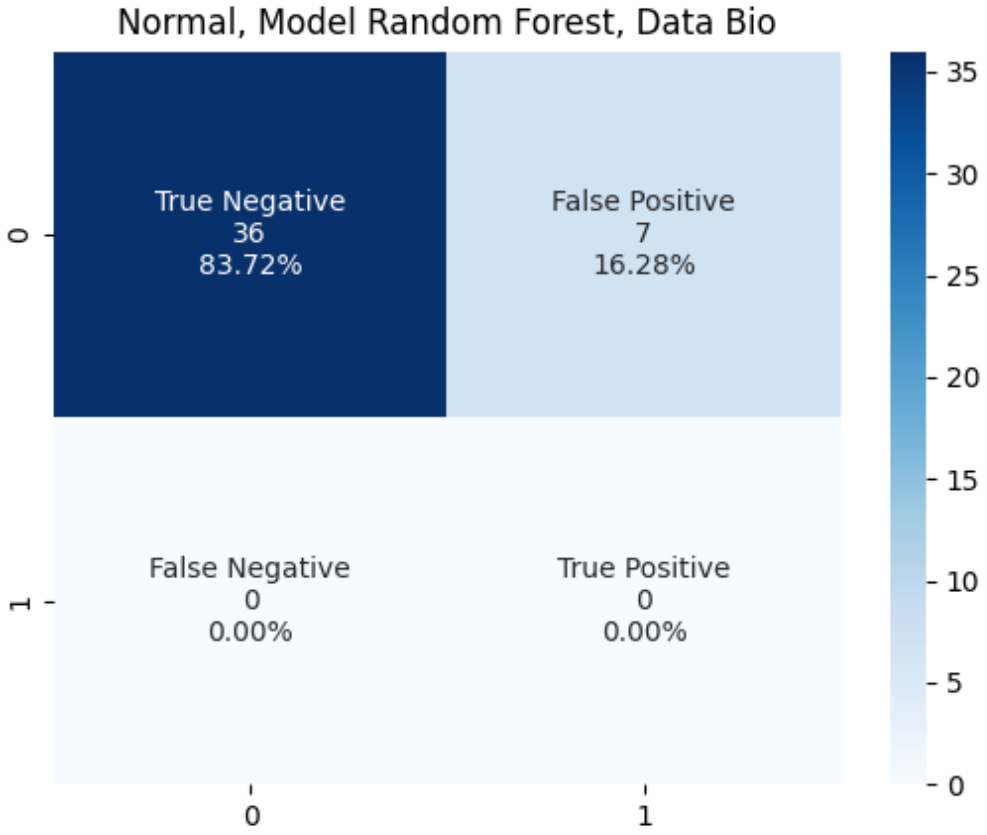
In [68]:
```
rf_confusion_normal_demographic
```

Out[68]:



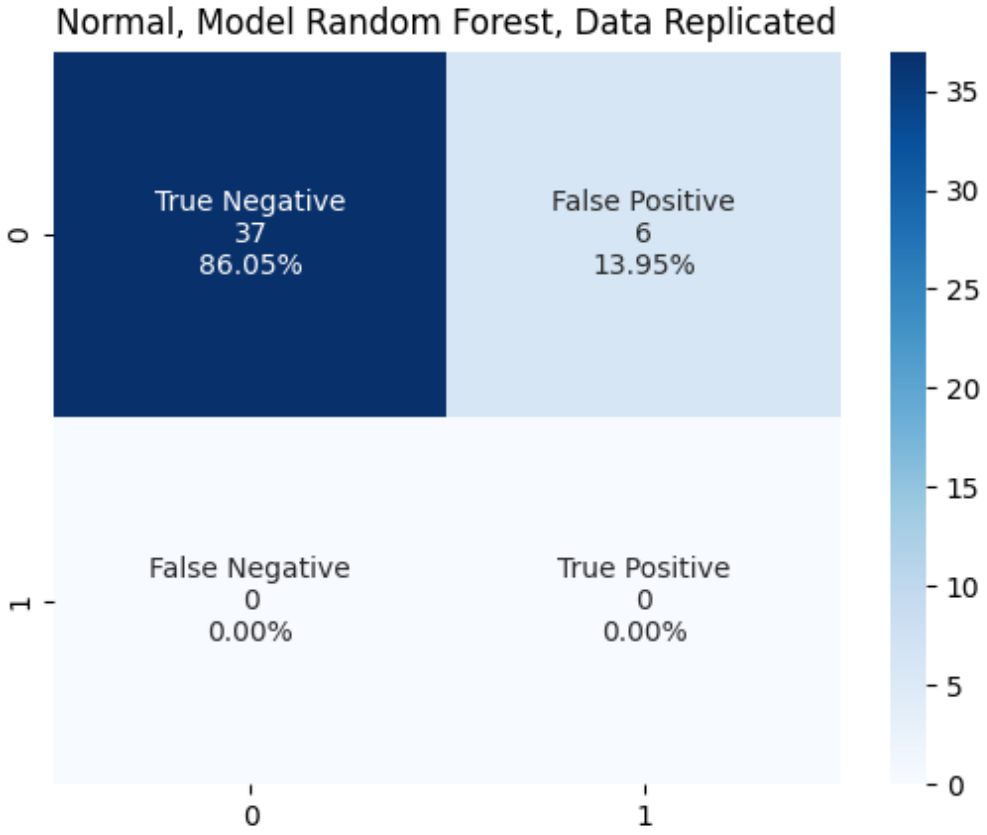
In [69]:
```
rf_confusion_normal_bio
```

Out[69]:



In [70]:
```
rf_confusion_normal_replicated
```

Out[70]:



Discussion

What you've learned

What concepts from lecture/breakout were most relevant to your project? How so?
There were several class concepts that were extremely relevant to our project. Firstly, the lectures on binary classification were useful as our main objective was creating a binary-classifier for if a patient had or did not have cancer. Similarly, the lectures on neural networks, decision trees, and logistic regression were useful because those were the binary-classifiers used. In addition, our familiarity with python libraries from section and homework were useful when creating our models, especially since getting used to using these types of libraries will be helpful going forward. Finally, the first lecture and discussion on FATE were some of our favorite parts of class since they gave us the insight, inspiration, and methodology needed to tackle a problem as complicated as cancer detection. We thoroughly discussed how this was a well-defined problem, had abundant data, and very meaningful/clear results before tackling the problem, and issues such as accountability and transparency guided the application of our models. For instance, we made sure to keep specificity high since false positives for issues like this have massive practical implications.

What aspects of your project did you find most surprising? What lessons did you take from this project that you want to remember for the next ML project you work on? Do you think those lessons would transfer to other datasets and/or models? Why or why not?
One of the most surprising results and lessons that we learned was that using different types of models (such as neural networks, decision trees, and logistic regression) are extremely useful since it is difficult to predict which model will perform the best. We unanimously assumed neural networks would yield the best results before we started, but it turned out that decision trees were superior. Even though it was very taxing to implement several robust models, this allowed us to discover which one performed the best. In practice, doing something similar to this would be time consuming in the short-term, but finding the best model and refining that one would be more helpful in the long-term. Therefore, this was a surprising result, but a lesson to be learned.

If you had two more weeks to work on this project, what would you do next? Why?
Even though we already discussed how the dataset for cancer type was disproportionately colorectal cancer, we still would definitely still work on creating a multivariate classification for cancer type if we had additional time to work on this project. There is a relatively limited dataset, but there might be some protein biomarkers that occur much more frequently in different types of cancer, so it still was a possibility to create a robust model with high specificity.

In [56]:

In [ ]: