HW03 Notebook

Complete the following notebook, as described in the PDF for Homework 03 (included in the download with the starter code). Submit the following:

1. This notebook file and `hw3.py`, along with your COLLABORATORS.txt file, to the Gradescope link for code.
2. A PDF of this notebook and all of its output, once it is completed, to the Gradescope link for the PDF.

NOTE: The purpose of this notebook is to demonstrate the functionality implemented in `hw3.py`. As part of this demo, all analysis (i.e., questions that prompt for a short answer) are to be added to the notebook. Keep the order of the problems as listed in the assignment description. Furthermore, cells are provided as placeholders for each response; however, cells can be added as needed.

Please report any questions to the class Piazza page.

Import required libraries.

In [ ]:
```
import os
import numpy as np
import pandas as pd

import warnings

import sklearn.linear_model
import sklearn.metrics
from hw3 import calc_confusion_matrix_for_threshold
from hw3 import calc_percent_cancer
from hw3 import calc_binary_metrics
from hw3 import predict_0_always_classifier
from hw3 import calc_accuracy
from hw3 import print_perf_metrics_for_threshold
from hw3 import calc_perf_metrics_for_threshold

from matplotlib import pyplot as plt
import seaborn as sns
%matplotlib inline
plt.style.use('seaborn-v0_8') # pretty matplotlib plots
```

In [ ]:
```
%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

1) Function to calculate TP, TN, FP, and FN.

The following four calls to the function `calc_binary_metrics` to test it. This way, the function can be tested for several edge cases. ***Don't modify this.***

In [ ]:
```
all0 = np.zeros(10)
all1 = np.ones(10)
TP, TN, FP, FN = calc_binary_metrics(all0, all1)
print(f"0 vs 1\n========\nTP: {TP}\nTN: {TN}\nFP: {FP}\nFN: {FN}")
```

```
0 vs 1
========
TP: 0.0
TN: 0.0
FP: 10.0
FN: 0.0
```

In [ ]:
```
TP, TN, FP, FN = calc_binary_metrics(all1, all0)
print(f"1 vs 0\n========\nTP: {TP}\nTN: {TN}\nFP: {FP}\nFN: {FN}")
```

```
1 vs 0
========
TP: 0.0
TN: 0.0
FP: 0.0
FN: 10.0
```

In [ ]:
```
TP, TN, FP, FN = calc_binary_metrics(all1, all1)
print(f"1 vs 1\n========\nTP: {TP}\nTN: {TN}\nFP: {FP}\nFN: {FN}")
```

```
1 vs 1
========
TP: 10.0
TN: 0.0
FP: 0.0
FN: 0.0
```

In [ ]:
```
TP, TN, FP, FN = calc_binary_metrics(all0, all0)
print(f"0 vs 0\n========\nTP: {TP}\nTN: {TN}\nFP: {FP}\nFN: {FN}")
```

```
0 vs 0
========
TP: 0.0
TN: 10.0
FP: 0.0
FN: 0.0
```

Load the dataset.

The following should ***not*** be modified.

After it runs, the various arrays it creates will contain the 2- or 3-feature input datasets.

In [ ]:
```
# Load the x-data and y-class arrays
x_train = np.loadtxt('./data/x_train.csv', delimiter=',', skiprows=1)
x_test = np.loadtxt('./data/x_test.csv', delimiter=',', skiprows=1)

y_train = np.loadtxt('./data/y_train.csv', delimiter=',', skiprows=1)
y_test = np.loadtxt('./data/y_test.csv', delimiter=',', skiprows=1)
```

Inspect Data. The following should ***not*** be modified.

In [ ]:
```
feat_names = np.loadtxt(f'data/x_train.csv', delimiter=',', dtype=str, max_rows=1)
print(f"features: {feat_names}\n")
target_name = np.loadtxt(f'data/x_test.csv', delimiter=',', dtype=str, max_rows=1)
df_sampled_data = pd.DataFrame(x_test, columns=feat_names)
df_sampled_data[str(target_name)] = y_test
df_sampled_data.sample(15)
```

```
features: ['age' 'famhistory' 'marker']
```

Out[ ]:

	age	famhistory	marker	['age' 'famhistory' 'marker']
81	67.11634	1.0	1.349243	0.0
117	68.24641	0.0	0.921616	0.0
162	53.74131	1.0	2.004676	0.0
23	63.12071	0.0	0.592084	0.0
41	58.86290	0.0	0.234737	0.0
114	60.32011	0.0	0.857177	0.0
56	57.48566	0.0	1.362557	0.0
159	64.84370	0.0	0.594441	0.0
100	71.86153	0.0	0.879809	0.0
169	64.29861	0.0	2.843979	0.0
154	65.63218	0.0	0.096532	0.0
123	59.45962	0.0	0.070890	0.0
102	61.60947	0.0	0.588344	0.0
8	59.21431	0.0	0.154820	0.0
118	62.65216	0.0	0.677188	0.0

2) Compute the fraction of patients with cancer.

Complete the following code. Your solution needs to ***compute*** these values from the training and testing sets (i.e., don't simply hand-count and print the values).

In [ ]:
```
#DONE: modify these prints
tr_percent = calc_percent_cancer(y_train)
te_percent = calc_percent_cancer(y_test)

print("Percent of data that has_cancer on TRAIN: %.3f" % tr_percent)
print("Percent of data that has_cancer on TEST : %.3f" % te_percent)
```

```
Percent of data that has_cancer on TRAIN: 14.035
Percent of data that has_cancer on TEST : 13.889
```

3) The predict-0-always baseline

(i) Compute the accuracy of the always-0 classifier.

Complete the functions to compute and calculate the accuracy of the always-0 classifier on validation and test outputs.

In [ ]:
```
#TODO: implement predict_0_always_classifer()
y_train_pred = predict_0_always_classifier(x_train)
y_test_pred = predict_0_always_classifier(x_test)

acc_train = calc_accuracy(*calc_binary_metrics(y_train, y_train_pred))
acc_test = calc_accuracy(*calc_binary_metrics(y_test, y_test_pred))

print("acc on TRAIN: %.3f" % (acc_train * 100)) #DONE: modify these values
print("acc on TEST : %.3f" % (acc_test * 100))
```

```
acc on TRAIN: 85.965
acc on TEST : 86.111
```

(ii) Print a confusion matrix for the always-0 classifier.

Add code below to generate a confusion matrix for the always-0 classifier on the test set.

In [ ]:
```
# DONE: call print(calc_confusion_matrix_for_threshold(...))
print(calc_confusion_matrix_for_threshold(y_test, predict_0_always_classifier(y_test))
```

```
Predicted    0  1
True
0          155  0
1           25  0
```

(iii) Reflect on the accuracy of the always-0 classifier.

Answer: Even though the always-0 classifier doesn't even look at the data, it is still relatively accurate as the amount of people without cancer far outweigh the amount of people that do. However, this also shows where accuracy can fail as a metric, as it doesn't do any detection at all. This basically says that most people don't have cancer.

(iv) Analyze the various costs of using the always-0 classifier.

Answer: It literally doesn't do anything when predicting whether or not someone actually has cancer. It just assumes that people don't because most people don't have cancer. It has a 0 percent accuracy when catching positive cases, which are the more important cases to worry about. It is more important to know if someone actually does have cancer (aka false negatives are more dangerous).

4: Basic Perceptron Models

(i) Normalize data

In [ ]:
```
#TODO
from hw3 import standardize_data
scaledTrainX, scaledTestX = standardize_data(x_train, x_test)
print(scaledTrainX)
print(scaledTestX)
```

```
[[0.53637473 0.         0.10816263]
 [0.43257395 0.         0.05418465]
 [0.72202778 1.         0.18904397]
 ...
 [0.45372565 0.         0.57842732]
 [0.33260801 0.         0.1189701 ]
 [0.62875077 0.         0.03734379]]
[[4.52932628e-01 0.00000000e+00 1.29363870e-01]
 [6.41709616e-01 1.00000000e+00 4.42667981e-02]
 [4.53226277e-01 0.00000000e+00 2.18670957e-01]
 [6.27107844e-01 0.00000000e+00 3.88584749e-01]
 [5.35089577e-01 0.00000000e+00 6.34107532e-02]
 [4.89981992e-01 0.00000000e+00 1.98314102e-01]
 [1.31167277e-01 1.00000000e+00 6.26043500e-02]
 [8.40709760e-01 0.00000000e+00 8.75066995e-02]
 [4.03642870e-01 0.00000000e+00 2.29611761e-02]
 [6.70801737e-01 0.00000000e+00 1.42722202e-01]
 [5.12940567e-01 0.00000000e+00 2.27504536e-01]
 [7.13664592e-01 0.00000000e+00 5.40049381e-02]
 [3.95619178e-01 0.00000000e+00 7.35783742e-01]
 [7.63991116e-01 1.00000000e+00 8.87774053e-03]
 [6.66562110e-01 0.00000000e+00 4.70254745e-02]
 [6.34915071e-01 0.00000000e+00 8.54837609e-01]
 [5.09380511e-01 1.00000000e+00 2.41484949e-01]
 [4.14368594e-01 0.00000000e+00 1.05263297e-01]
 [5.40531655e-01 0.00000000e+00 7.15893392e-02]
 [4.16190842e-01 0.00000000e+00 2.49895857e-02]
 [3.70672380e-01 1.00000000e+00 1.98627054e-02]
 [4.92084541e-01 0.00000000e+00 9.14690365e-03]
 [4.65470444e-01 0.00000000e+00 7.95394713e-02]
 [5.16993617e-01 0.00000000e+00 8.80220673e-02]
 [5.58276224e-01 0.00000000e+00 1.32340317e-02]
 [3.22050585e-01 0.00000000e+00 1.06597045e-01]
 [6.80313984e-01 1.00000000e+00 2.05977588e-01]
 [4.81126683e-01 0.00000000e+00 2.53537922e-01]
 [5.34326149e-01 1.00000000e+00 8.35379050e-02]
 [5.77372679e-01 0.00000000e+00 7.89488301e-02]
 [7.53373044e-01 0.00000000e+00 2.78943155e-02]
 [6.23175214e-01 0.00000000e+00 1.55315108e-01]
 [6.29529866e-01 1.00000000e+00 3.29923855e-01]
 [4.89464915e-01 0.00000000e+00 4.46330727e-01]
 [4.13106949e-01 0.00000000e+00 1.66368398e-02]
 [7.24815701e-01 0.00000000e+00 2.24844345e-02]
 [4.77297642e-01 0.00000000e+00 9.04762725e-02]
 [5.01123526e-01 0.00000000e+00 4.65908111e-02]
 [8.22131252e-01 0.00000000e+00 8.36052776e-03]
 [4.34389811e-01 0.00000000e+00 1.75850215e-01]
 [6.80839186e-01 0.00000000e+00 1.87835785e-01]
 [3.93446119e-01 0.00000000e+00 3.48520705e-02]
 [6.92414808e-01 0.00000000e+00 2.37944911e-01]
 [4.47722684e-01 0.00000000e+00 2.96809694e-02]
 [6.36395792e-01 0.00000000e+00 2.86893362e-02]
 [6.61245384e-01 0.00000000e+00 2.34499504e-01]
 [4.63647036e-01 0.00000000e+00 4.11304259e-01]
 [5.42290066e-01 0.00000000e+00 1.04760397e-02]
 [4.42042960e-01 0.00000000e+00 6.97854999e-02]
 [9.62303855e-01 0.00000000e+00 3.96653438e-02]
 [6.28451316e-01 0.00000000e+00 2.94511764e-02]
 [6.93623063e-01 0.00000000e+00 1.40858192e-01]
 [5.02536928e-01 0.00000000e+00 1.74694748e-02]
```

```
[6.27624631e-01 1.00000000e+00 2.61331451e-01]
[6.13795283e-01 0.00000000e+00 5.02104524e-02]
[8.64843510e-01 0.00000000e+00 1.93399382e-01]
[3.53483191e-01 0.00000000e+00 2.02661629e-01]
[8.71920388e-01 0.00000000e+00 8.64182479e-02]
[5.56485314e-01 0.00000000e+00 2.01996726e-02]
[4.78006810e-01 1.00000000e+00 2.39653182e-01]
[5.30711541e-01 0.00000000e+00 4.38698190e-01]
[4.22428847e-01 0.00000000e+00 6.31616471e-02]
[5.49019904e-01 0.00000000e+00 3.32008418e-01]
[5.51372576e-01 0.00000000e+00 7.22573517e-02]
[7.28748911e-01 0.00000000e+00 4.66917167e-01]
[5.47114379e-01 0.00000000e+00 7.15455797e-02]
[6.99618199e-01 0.00000000e+00 8.34033487e-01]
[4.54940002e-01 0.00000000e+00 2.45338088e-02]
[6.58875302e-01 0.00000000e+00 2.79644854e-02]
[4.20494305e-01 0.00000000e+00 7.20834447e-02]
[7.84715407e-01 0.00000000e+00 3.79074617e-01]
[7.62344130e-01 0.00000000e+00 5.78527357e-02]
[4.33210864e-01 0.00000000e+00 1.07161038e-01]
[7.00917856e-01 0.00000000e+00 1.65931949e-01]
[7.01887883e-01 0.00000000e+00 9.66271183e-02]
[5.19717993e-01 0.00000000e+00 4.77914512e-02]
[5.16973886e-01 1.00000000e+00 7.50841894e-02]
[7.92493618e-01 0.00000000e+00 2.42512694e-02]
[7.41910005e-01 0.00000000e+00 3.65570529e-02]
[7.37277783e-01 0.00000000e+00 4.71307443e-02]
[5.52118014e-01 0.00000000e+00 7.26872539e-02]
[6.32933522e-01 1.00000000e+00 2.00680624e-01]
[3.97034902e-01 0.00000000e+00 1.18157727e-01]
[7.81497458e-01 1.00000000e+00 3.21378930e-01]
[4.44266218e-01 1.00000000e+00 8.97068736e-02]
[3.08551737e-01 1.00000000e+00 9.03547101e-02]
[5.94252841e-01 0.00000000e+00 1.05635646e-01]
[6.07476031e-01 0.00000000e+00 1.23630681e-01]
[9.10790548e-01 0.00000000e+00 5.55211939e-02]
[6.41639396e-01 0.00000000e+00 8.87763936e-02]
[4.79338095e-01 0.00000000e+00 2.34280483e-01]
[4.52245804e-01 0.00000000e+00 2.72960815e-01]
[5.85055715e-01 0.00000000e+00 2.40117157e-02]
[7.11003762e-01 0.00000000e+00 6.71718204e-02]
[3.43522056e-01 1.00000000e+00 2.05976695e-01]
[4.76988034e-01 0.00000000e+00 4.80208082e-01]
[5.36523001e-01 0.00000000e+00 6.57465504e-02]
[5.50176799e-01 0.00000000e+00 4.98858795e-02]
[5.07716115e-01 0.00000000e+00 9.13852671e-03]
[6.35571718e-01 0.00000000e+00 1.24584521e-01]
[7.70623168e-01 0.00000000e+00 1.30833034e-01]
[7.52694924e-01 0.00000000e+00 1.93031571e-01]
[4.73142454e-01 0.00000000e+00 8.74655588e-02]
[4.37642000e-01 0.00000000e+00 5.77648745e-03]
[4.94074504e-01 0.00000000e+00 7.24644347e-01]
[5.85476167e-01 0.00000000e+00 6.90269628e-02]
[8.30869334e-01 0.00000000e+00 2.87724315e-01]
[4.98456893e-01 1.00000000e+00 3.53314010e-02]
[8.37584663e-01 0.00000000e+00 7.47596462e-02]
[4.51186405e-01 0.00000000e+00 2.07769181e-01]
[6.57424758e-01 0.00000000e+00 6.91241234e-02]
[4.83148275e-01 0.00000000e+00 3.75672701e-02]
[4.48715635e-01 0.00000000e+00 1.18025571e-02]
```

```
[6.87609357e-01 1.00000000e+00 5.57988232e-02]
[4.35729511e-01 0.00000000e+00 1.27465549e-01]
[6.25668617e-01 0.00000000e+00 9.73519393e-03]
[4.89779456e-01 0.00000000e+00 8.18694805e-03]
[6.65724398e-01 0.00000000e+00 1.37053603e-01]
[5.03397853e-01 0.00000000e+00 1.00684816e-01]
[4.49114034e-01 0.00000000e+00 9.38694090e-02]
[7.08153164e-01 0.00000000e+00 9.35014635e-02]
[8.83784726e-01 1.00000000e+00 6.91562474e-02]
[6.05023251e-01 0.00000000e+00 1.46910077e-01]
[4.10760951e-01 0.00000000e+00 1.04730639e-02]
[3.44491503e-01 0.00000000e+00 1.04652955e-01]
[5.14323502e-01 0.00000000e+00 9.05245105e-04]
[5.17681312e-01 0.00000000e+00 2.50895882e-02]
[2.98815481e-01 0.00000000e+00 1.87913901e-01]
[3.90295779e-01 0.00000000e+00 1.14743537e-01]
[4.02253261e-01 1.00000000e+00 2.47203929e-02]
[5.11474935e-01 0.00000000e+00 1.13982470e-02]
[7.31191245e-01 0.00000000e+00 2.72180897e-02]
[8.30454976e-01 1.00000000e+00 1.09025837e-01]
[3.26610846e-01 1.00000000e+00 1.47909313e-01]
[9.44332956e-01 0.00000000e+00 2.20195367e-02]
[4.62545854e-01 0.00000000e+00 9.06851305e-02]
[6.83427763e-01 0.00000000e+00 1.19452255e-01]
[4.24412717e-01 0.00000000e+00 6.25075911e-03]
[4.80038267e-01 0.00000000e+00 6.50852334e-02]
[5.27083586e-01 0.00000000e+00 4.42606188e-01]
[8.34596816e-01 0.00000000e+00 1.54614793e-03]
[3.26355790e-01 0.00000000e+00 3.37534817e-01]
[5.66532919e-01 0.00000000e+00 2.72535912e-02]
[4.71805365e-01 0.00000000e+00 6.42944384e-02]
[6.51962369e-01 0.00000000e+00 3.01551964e-02]
[7.33014943e-01 0.00000000e+00 3.42555476e-01]
[4.03587158e-01 0.00000000e+00 7.94720838e-02]
[3.38846599e-01 1.00000000e+00 5.09192635e-02]
[5.62494378e-01 0.00000000e+00 6.19726871e-03]
[5.88871698e-01 0.00000000e+00 5.63811128e-01]
[7.23746146e-01 0.00000000e+00 8.61599019e-02]
[6.07931883e-01 0.00000000e+00 2.21087771e-01]
[6.29594283e-01 0.00000000e+00 1.42397346e-01]
[6.59610585e-01 1.00000000e+00 1.00476657e-01]
[5.89868131e-01 0.00000000e+00 1.42883953e-02]
[4.64457762e-01 0.00000000e+00 1.37911369e-01]
[4.94690238e-01 0.00000000e+00 5.07616788e-02]
[7.71303029e-01 0.00000000e+00 6.92244384e-02]
[5.17284074e-01 0.00000000e+00 4.52027939e-01]
[5.66989061e-01 0.00000000e+00 8.83728572e-02]
[4.66715840e-01 0.00000000e+00 6.52745999e-02]
[8.46748130e-01 0.00000000e+00 5.90269059e-02]
[2.44834596e-01 1.00000000e+00 2.98203227e-01]
[8.08255477e-01 0.00000000e+00 3.59994582e-02]
[6.47058550e-01 0.00000000e+00 1.75248206e-01]
[5.95840924e-01 0.00000000e+00 5.89083342e-03]
[5.28030981e-01 0.00000000e+00 4.36991154e-02]
[5.79727963e-01 0.00000000e+00 1.87357868e-01]
[6.00351276e-01 1.00000000e+00 1.58560882e-02]
[5.51172361e-01 0.00000000e+00 4.23084051e-01]
[6.17138874e-01 0.00000000e+00 4.21906667e-01]
[9.43857663e-01 0.00000000e+00 2.29369231e-02]
[6.06631355e-01 1.00000000e+00 1.20784962e-01]
```

```
 [6.19673481e-01 0.00000000e+00 2.14635445e-01]
 [7.41934379e-01 0.00000000e+00 5.40507063e-02]
 [5.15594433e-01 1.00000000e+00 1.20954108e-01]
 [5.72976072e-01 0.00000000e+00 1.24840204e-02]
 [4.66174099e-01 0.00000000e+00 8.85094772e-02]
 [3.97281544e-01 0.00000000e+00 4.60337029e-01]
 [2.20143435e-01 0.00000000e+00 3.13507032e-02]]
```

(ii) Create a basic `Perceptron` classifier

Fit a perceptron to the training data. Print out accuracy on this data, as well as on testing data. Print out a confusion matrix on the testing data.

In [ ]:
```
#TODO: train a basic perceptron model using default parameter values, and modify these
from hw3 import perceptron_classifier
from sklearn.metrics import accuracy_score

trainPredict, testPredict = perceptron_classifier(scaledTrainX, y_train, scaledTestX,

print("acc on TRAIN: %.3f" % (accuracy_score(y_train, trainPredict) * 100))
print("acc on TEST : %.3f" % (accuracy_score(y_test, testPredict) * 100))


print("")
print("Confusion matrix for TEST:")
# TODO: call print(calc_confusion_matrix_for_threshold(...))
print(calc_confusion_matrix_for_threshold(y_test, testPredict))
```

```
acc on TRAIN: 24.912
acc on TEST : 27.222

Confusion matrix for TEST:
Predicted   0    1
True
0          24  131
1           0   25
```

(iii) Compare the `Perceptron` to the always-0 classifier.

Answer:
The Perceptron did significantly worse than the always-0, as it way over estimated how many people had cancer. In other words, it picked up on a feature in the positive cases that did not generalize. This si seenas it always got things that were positive cased for the confusion matrix, but it would also get false positives. However, I would consider this better than the always-0 classifier even though the accuracy was lower as it was able to actually detect cancer, and this is a case that is better safe then sorry.

(iv) Generate a series of regularized perceptron models

Each model will use a different `alpha` value, multiplying that by the L2 penalty. You will record and plot the accuracy of each model on both training and test data.

In [ ]:
```
train_accuracy_list = list()
test_accuracy_list = list()
```

```
from hw3 import series_of_preceptrons

# TODO: create, fit models here and record accuracy of each (Implement functions neede
alphas = np.logspace(-5, 5, base=10, num=100)
train_accuracy_list, test_accuracy_list = series_of_preceptrons(scaledTrainX, y_train,
print("Alphas:", alphas)
print("Train Accuracy:", train_accuracy_list)
print("Test Accuracy:", test_accuracy_list)
```

```
Alphas: [1.00000000e-05 1.26185688e-05 1.59228279e-05 2.00923300e-05
 2.53536449e-05 3.19926714e-05 4.03701726e-05 5.09413801e-05
 6.42807312e-05 8.11130831e-05 1.02353102e-04 1.29154967e-04
 1.62975083e-04 2.05651231e-04 2.59502421e-04 3.27454916e-04
 4.13201240e-04 5.21400829e-04 6.57933225e-04 8.30217568e-04
 1.04761575e-03 1.32194115e-03 1.66810054e-03 2.10490414e-03
 2.65608778e-03 3.35160265e-03 4.22924287e-03 5.33669923e-03
 6.73415066e-03 8.49753436e-03 1.07226722e-02 1.35304777e-02
 1.70735265e-02 2.15443469e-02 2.71858824e-02 3.43046929e-02
 4.32876128e-02 5.46227722e-02 6.89261210e-02 8.69749003e-02
 1.09749877e-01 1.38488637e-01 1.74752840e-01 2.20513074e-01
 2.78255940e-01 3.51119173e-01 4.43062146e-01 5.59081018e-01
 7.05480231e-01 8.90215085e-01 1.12332403e+00 1.41747416e+00
 1.78864953e+00 2.25701972e+00 2.84803587e+00 3.59381366e+00
 4.53487851e+00 5.72236766e+00 7.22080902e+00 9.11162756e+00
 1.14975700e+01 1.45082878e+01 1.83073828e+01 2.31012970e+01
 2.91505306e+01 3.67837977e+01 4.64158883e+01 5.85702082e+01
 7.39072203e+01 9.32603347e+01 1.17681195e+02 1.48496826e+02
 1.87381742e+02 2.36448941e+02 2.98364724e+02 3.76493581e+02
 4.75081016e+02 5.99484250e+02 7.56463328e+02 9.54548457e+02
 1.20450354e+03 1.51991108e+03 1.91791026e+03 2.42012826e+03
 3.05385551e+03 3.85352859e+03 4.86260158e+03 6.13590727e+03
 7.74263683e+03 9.77009957e+03 1.23284674e+04 1.55567614e+04
 1.96304065e+04 2.47707636e+04 3.12571585e+04 3.94420606e+04
 4.97702356e+04 6.28029144e+04 7.92482898e+04 1.00000000e+05]
Train Accuracy: [0.8631578947368421, 0.8333333333333334, 0.6, 0.7842105263157895, 0.3
56140350877193, 0.8736842105263158, 0.7912280701754386, 0.7157894736842105, 0.3596491
2280701755, 0.3736842105263158, 0.37543859649122807, 0.8701754385964913, 0.4508771929
8245615, 0.8701754385964913, 0.5526315789473685, 0.8614035087719298, 0.81403508771929
82, 0.8614035087719298, 0.5754385964912281, 0.8105263157894737, 0.8596491228070176,
0.3385964912280702, 0.5754385964912281, 0.23157894736842105, 0.8263157894736842, 0.85
96491228070176, 0.8596491228070176, 0.8596491228070176, 0.8771929824561403, 0.2017543
8596491227, 0.8614035087719298, 0.14035087719298245, 0.8614035087719298, 0.8596491228
070176, 0.8596491228070176, 0.8596491228070176, 0.8596491228070176, 0.140350877192982
45, 0.14035087719298245, 0.8596491228070176, 0.8596491228070176, 0.8596491228070176,
0.8596491228070176, 0.8596491228070176, 0.8596491228070176, 0.8596491228070176, 0.859
6491228070176, 0.8596491228070176, 0.8596491228070176, 0.8596491228070176, 0.85964912
28070176, 0.8596491228070176, 0.8596491228070176, 0.8596491228070176, 0.8596491228070
176, 0.8596491228070176, 0.8596491228070176, 0.8596491228070176, 0.8596491228070176,
0.8596491228070176, 0.8596491228070176, 0.8596491228070176, 0.8596491228070176, 0.859
6491228070176, 0.8596491228070176, 0.8596491228070176, 0.8596491228070176, 0.85964912
28070176, 0.8596491228070176, 0.8596491228070176, 0.8596491228070176, 0.8596491228070
176, 0.8596491228070176, 0.8596491228070176, 0.8596491228070176, 0.8596491228070176,
0.8596491228070176, 0.8596491228070176, 0.8596491228070176, 0.8596491228070176, 0.859
6491228070176, 0.8596491228070176, 0.8596491228070176, 0.8596491228070176, 0.85964912
28070176, 0.8596491228070176, 0.8596491228070176, 0.8596491228070176, 0.8596491228070
176, 0.8596491228070176, 0.8596491228070176, 0.8596491228070176, 0.8596491228070176,
0.8596491228070176, 0.8596491228070176, 0.8596491228070176, 0.8596491228070176, 0.859
6491228070176, 0.8596491228070176, 0.8596491228070176]
Test Accuracy: [0.8666666666666667, 0.8277777777777777, 0.6444444444444445, 0.7611111
111111111, 0.43333333333333335, 0.8888888888888888, 0.7666666666666667, 0.75555555555
55555, 0.4388888888888889, 0.45555555555555555, 0.4722222222222222, 0.877777777777777
8, 0.5333333333333333, 0.8833333333333333, 0.6555555555555556, 0.8611111111111112, 0.
8222222222222222, 0.8611111111111112, 0.6722222222222223, 0.8111111111111111, 0.86111
11111111112, 0.35, 0.6444444444444445, 0.23333333333333334, 0.8333333333333334, 0.861
1111111111112, 0.8611111111111112, 0.8666666666666667, 0.9111111111111111, 0.20555555
555555555, 0.8722222222222222, 0.1388888888888889, 0.8611111111111112, 0.861111111111
1112, 0.8611111111111112, 0.8611111111111112, 0.8611111111111112, 0.1388888888888889,
0.1388888888888889, 0.8611111111111112, 0.8611111111111112, 0.8611111111111112, 0.861
1111111111112, 0.8611111111111112, 0.8611111111111112, 0.8611111111111112, 0.86111111
```

```
11111112, 0.8611111111111112, 0.8611111111111112, 0.8611111111111112, 0.8611111111111
112, 0.8611111111111112, 0.8611111111111112, 0.8611111111111112, 0.8611111111111112,
0.8611111111111112, 0.8611111111111112, 0.8611111111111112, 0.8611111111111112, 0.861
1111111111112, 0.8611111111111112, 0.8611111111111112, 0.8611111111111112, 0.86111111
11111112, 0.8611111111111112, 0.8611111111111112, 0.8611111111111112, 0.8611111111111
112, 0.8611111111111112, 0.8611111111111112, 0.8611111111111112, 0.8611111111111112,
0.8611111111111112, 0.8611111111111112, 0.8611111111111112, 0.8611111111111112, 0.861
1111111111112, 0.8611111111111112, 0.8611111111111112, 0.8611111111111112, 0.86111111
11111112, 0.8611111111111112, 0.8611111111111112, 0.8611111111111112, 0.8611111111111
112, 0.8611111111111112, 0.8611111111111112, 0.8611111111111112, 0.8611111111111112,
0.8611111111111112, 0.8611111111111112, 0.8611111111111112, 0.8611111111111112, 0.861
1111111111112, 0.8611111111111112, 0.8611111111111112, 0.8611111111111112, 0.86111111
11111112, 0.8611111111111112, 0.8611111111111112]
```

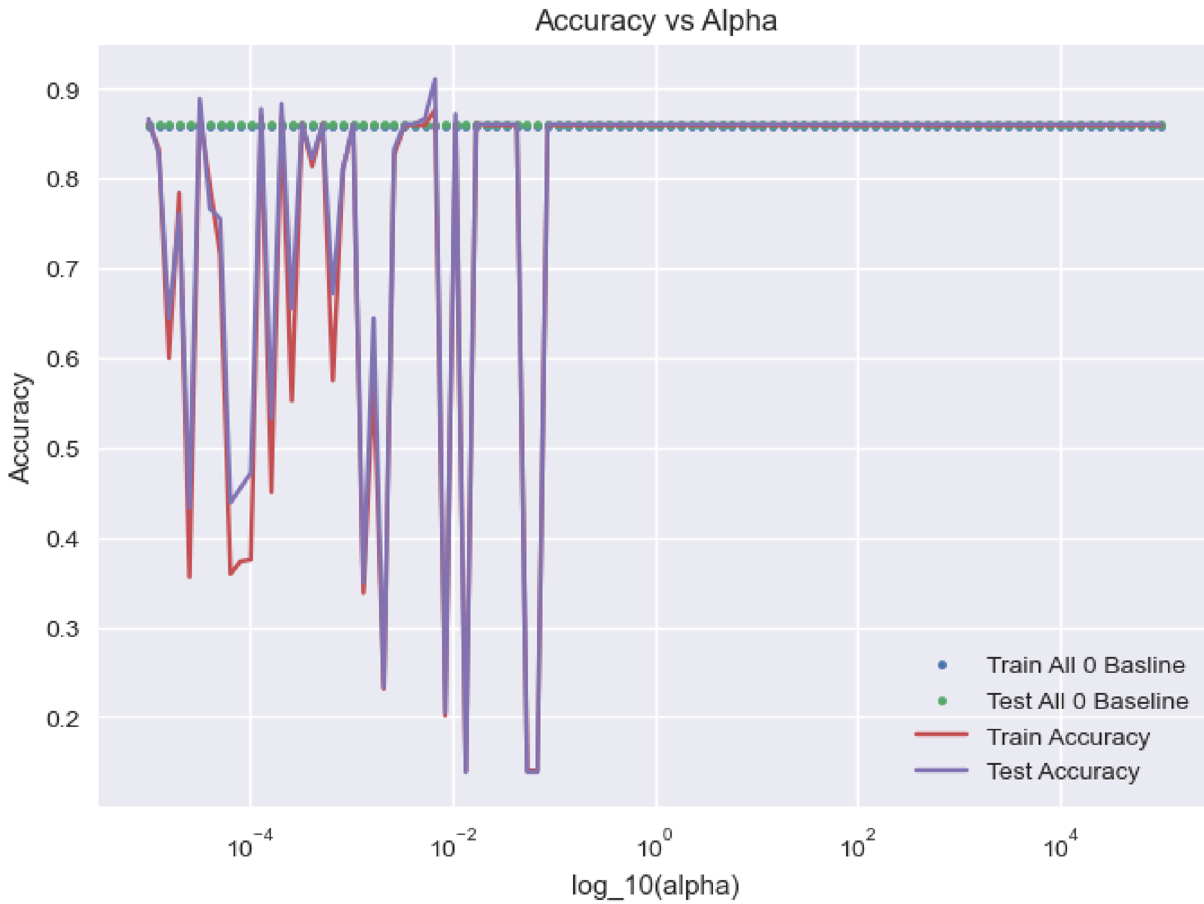
Plot accuracy on train/test data across the different alpha values plotted on a logarithmic scale. Make sure to show title, legends, and axis labels.

In [ ]:

```
# TODO make plot
plt.xlabel('log_10(alpha)');
plt.ylabel('Accuracy');
plt.xscale("log")

allZeroTrain, allZeroTest = [acc_train] * len(alphas) , [acc_test] * len(alphas)
plt.plot(alphas, allZeroTrain, '.', label = "Train All 0 Basline")
plt.plot(alphas, allZeroTest, '.', label = "Test All 0 Baseline")

plt.plot(alphas, train_accuracy_list, label = "Train Accuracy")
plt.plot(alphas, test_accuracy_list, label = "Test Accuracy")
plt.title("Accuracy vs Alpha")


# TODO add legend, titles, etc. set x-scale appropriately
# plt.legend(...);
plt.legend()
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x1ce0d8aab00>
```



(iv) Discuss what the plot is showing you.

Answer: The plot is show that as the alphas get closer to 1, it gets less accurate, then after passing 1, it is around ~80% accurate (which is no more accurate than just guessing all 0s). Note that the accuracy is not stable, as sometimes it jumps up to being as accurate as guessing all zeros. There are some points where alpha is low that it spikes up, but for the most part, it is either as good as guessing all zeros, or worse.

5: Decision functions and probabilistic predictions

(a) Create two new sets of predictions

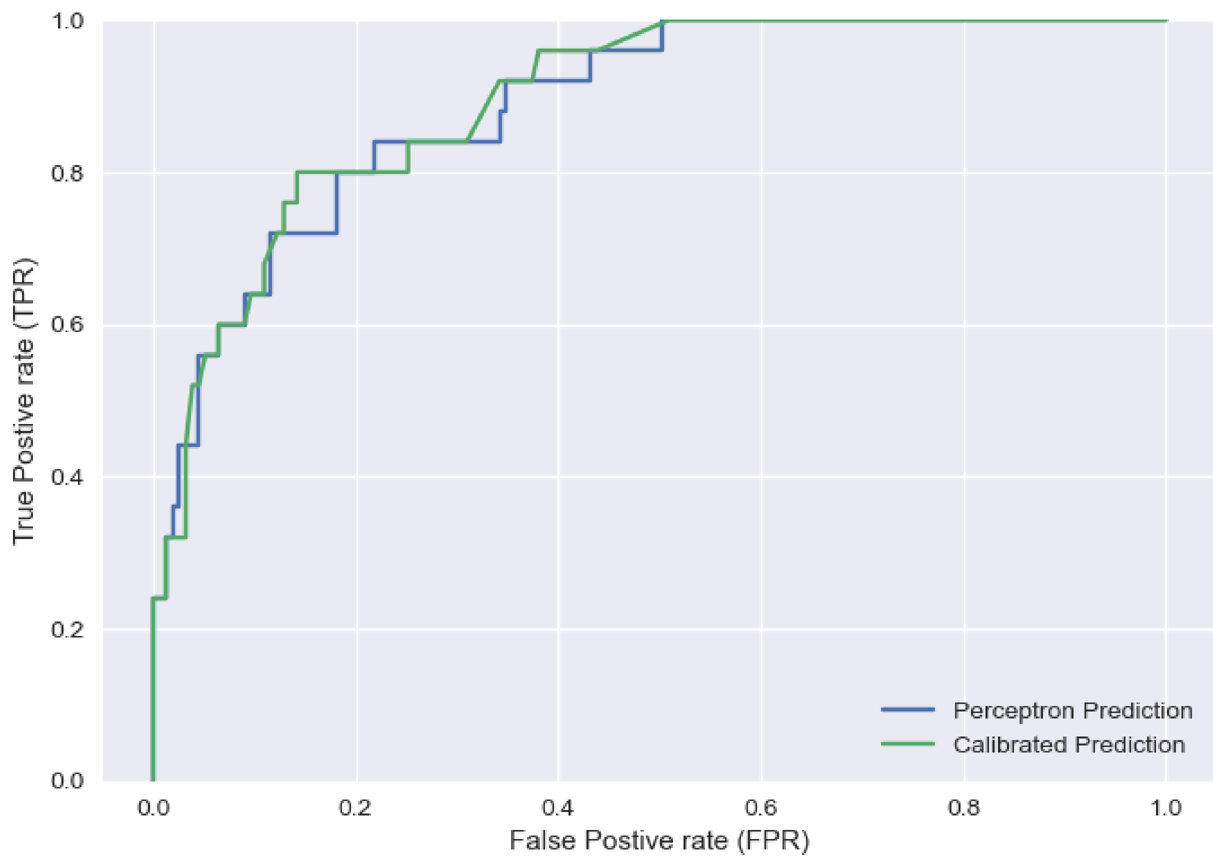
Fit `Perceptron` and `CalibratedClassifierCV` models to the data. Use their predictions to generate ROC curves.

In [ ]:
```
# TODO: fit a Perceptron and generate its decision_function() over the test data.

# TODO: Build a CalibratedClassifierCV, using a Perceptron as its base_estimator,
#       and generate its probabilistic predictions over the test data.
from hw3 import calibrated_perceptron_classifier
predictTest, calibratedTest = calibrated_perceptron_classifier(scaledTrainX, y_train,
print(y_test.shape, predictTest.shape, calibratedTest.shape)
print(predictTest)
print(calibratedTest)
```

```
(180,) (180,) (180,)
[ 0.34330422  2.05724554  0.78159876  2.65288979  0.51266198  0.90209
 -0.90890292  2.45969294 -0.4718141   1.71262687  1.18218962  1.53552292
  2.96446907  2.61616192  1.21948136  4.97868329  2.22921825 -0.00532034
  0.58521209 -0.38679493  0.31569996 -0.00998302  0.1748038   0.52465219
  0.40617655 -0.55135807  3.07875626  1.11902417  1.60647497  0.84169265
  1.64556271  1.48911934  3.38064946  2.11131123 -0.44608174  1.4481928
  0.29905348  0.22714608  1.9616238   0.45954594  1.99322186 -0.47472218
  2.30744181 -0.17513359  0.94929703  2.10404007  1.78557084  0.29701235
 -0.01309613  2.95362757  0.90547034  1.84010952  0.09325945  3.03396326
  0.91922085  3.12175036  0.1063455   2.6411796   0.42950566  2.03248137
  2.32087924 -0.16287147  1.90895723  0.65336426  3.64414115  0.62439821
  5.26426379 -0.15709484  1.08030205 -0.13084028  3.54974293  1.84569602
  0.1167338   2.00633315  1.67337408  0.34430957  1.46129318  1.86190633
  1.61929606  1.64325517  0.65992736  2.76927502 -0.0460403   4.24846328
  1.09758743  0.28845313  1.07317294  1.2402791   2.72280561  1.27439009
  1.01418762  1.04110147  0.61914342  1.58395718  1.06292864  2.20222651
  0.53265902  0.53685436  0.08353674  1.41310451  2.25197999  2.44870204
  0.25946666 -0.35231634  3.49931046  0.84169648  3.37946685  1.15614882
  2.37867993  0.71610102  1.27281012  0.07545015 -0.25658088  2.38834137
  0.23105817  0.79244228 -0.0284721   1.65452878  0.50517263  0.14695005
  1.69559575  3.6278146   1.33938902 -0.49025221 -0.42654374  0.08283974
  0.22115159 -0.29294834 -0.10306514  0.52846771  0.11708027  1.50949108
  3.50350078  0.67787294  2.75981167  0.21177952  1.67445375 -0.42918019
  0.19134462  2.31826706  2.00292579  0.6032441   0.52412412  0.13820224
  1.0496339   3.06178953 -0.19591977  0.2770168   0.39702775  3.28054663
  1.75303959  1.71938206  1.46439724  2.43914565  0.60041948  0.45406692
  0.20902788  1.95490369  2.30566732  0.82560795  0.11253064  2.35662432
  0.92305531  2.0136725   1.72950373  0.59512119  0.37406234  1.38569801
  1.6708279   2.36702216  2.75610084  2.76145109  2.22131341  1.75812058
  1.70495169  1.67725082  0.49049445  0.22286121  1.62802334 -1.52911716]
[0.00851064 0.17547124 0.04732545 0.3956044  0.02332545 0.05565879
 0.         0.30558608 0.         0.11268182 0.06815879 0.10468182
 0.38012821 0.3281044  0.1009439  0.85       0.17421577 0.
 0.02332545 0.         0.03881481 0.00851064 0.00851064 0.02332545
 0.00851064 0.         0.55801664 0.05565879 0.06702579 0.05565879
 0.12815801 0.10468182 0.59285714 0.187      0.         0.10468182
 0.00851064 0.00851064 0.177933   0.00851064 0.2002667  0.
 0.24469921 0.         0.05565879 0.20247619 0.12416865 0.00851064
 0.         0.62145022 0.05565879 0.17280087 0.00851064 0.49109404
 0.05565879 0.52130032 0.         0.43235803 0.00851064 0.14386177
 0.22438095 0.         0.15732468 0.02332545 0.65       0.02332545
 0.85       0.         0.08048148 0.         0.64944474 0.17280087
 0.00851064 0.20223087 0.11268182 0.00851064 0.06702579 0.17280087
 0.12815801 0.12815801 0.02332545 0.34040688 0.         0.65
 0.06702579 0.03562686 0.05565879 0.09535143 0.45846154 0.09754132
 0.05565879 0.05565879 0.02332545 0.11268182 0.05851515 0.19271429
 0.02332545 0.02332545 0.00851064 0.10468182 0.2447333  0.29606227
 0.00851064 0.         0.56430486 0.05565879 0.5820812  0.06702579
 0.27604762 0.04732545 0.10468182 0.00851064 0.         0.31128635
 0.00851064 0.05565879 0.00851064 0.11268182 0.02332545 0.00851064
 0.11268182 0.65       0.09634848 0.         0.         0.00851064
 0.00851064 0.         0.         0.04714815 0.00851064 0.10468182
 0.60238095 0.04714815 0.45846154 0.00851064 0.11268182 0.
 0.00851064 0.22438095 0.21128571 0.01764409 0.02332545 0.00851064
 0.08048148 0.50786307 0.         0.01481481 0.00851064 0.54528177
 0.12815801 0.11268182 0.10468182 0.31262821 0.02332545 0.00851064
 0.00851064 0.17580279 0.22438095 0.05565879 0.00851064 0.27604762
 0.04714815 0.21397845 0.11268182 0.02332545 0.00851064 0.09634848
```

```
 0.06702579 0.25391941 0.3956044  0.45846154 0.17421577 0.11268182
 0.12815801 0.06702579 0.02332545 0.00851064 0.07952579 0.        ]
```

In [ ]:
```
# TODO something like: fpr, tpr, thr = sklearn.metrics.roc_curve(...)

predictFpr, predictTpr, predictThr = sklearn.metrics.roc_curve(y_test, predictTest)
calibratedFpr, calibratedTpr, calibratedThr = sklearn.metrics.roc_curve(y_test, calibr
plt.plot(predictFpr, predictTpr, label = "Perceptron Prediction")
plt.plot(calibratedFpr, calibratedTpr, label = "Calibrated Prediction")

plt.ylim([0, 1]);
plt.legend(loc='lower right')
plt.xlabel("False Postive rate (FPR)");
plt.ylabel("True Postive rate (TPR)");
```



In [ ]:
```
print("AUC on TEST for Perceptron: %.3f" % sklearn.metrics.roc_auc_score(y_test, predi
print("AUC on TEST for probabilistic model: %.3f" % sklearn.metrics.roc_auc_score(y_te
```

```
AUC on TEST for Perceptron: 0.887
AUC on TEST for probabilistic model: 0.894
```

(b) Discuss the results above

Answer: The model did better than just flipping a random coin for the positive outcomes. Since the ROC curve is closer to vertical for the most part, that means it does a decent job at predicting positive outcomes. The curves are very similar, but the probabilisitic model does slightly better when looking at the area under the curve (as a higher area is better). In the future I would probably use a probabilisitc model, as the output is easier to understand than that of a confidence model, as the probability can be read off as a percent confidence, whereas the confidence is not constrained to the range [0, 1].

(c) Compute model metrics for different probabilistic thresholds

Complete `calc_perf_metrics_for_threshold` that takes in a set of correct outputs, a matching set of probabilities generated by a classifier, and a threshold at which to set the positive decision probability, and returns a set of metrics if we use that threshold.

(d) Compare the probabilistic classifier across multiple decision thresholds

Try a range of thresholds for classifying data into the positive class (1). For each threshold, compute the true positive rate (TPR) and positive predictive value (PPV). Record the best value of each metric, along with the threshold that achieves it, and the *other* metric at that threshold.

In [ ]:

```
# TODO: test different thresholds to compute these values
from hw3 import find_best_thresholds
best_TPR = 0
best_PPV_for_best_TPR = 0
best_TPR_threshold = 0

best_PPV = 0
best_TPR_for_best_PPV = 0
best_PPV_threshold = 0

best_TPR, best_PPV_for_best_TPR, best_TPR_threshold, best_PPV, best_TPR_for_best_PPV,
print("Best TPR:", best_TPR)
print("Corresponding PPV:", best_PPV_for_best_TPR)
print("Threshold:", best_TPR_threshold)
print()
print("Best PPV:", best_PPV)
print("Best Corresponding TPR:", best_TPR_for_best_PPV)
print("Threshold:", best_PPV_threshold)
```

```
Best TPR: 1.0
Corresponding PPV: 0.22935779816513763
Threshold: 0.040039999999999999

Best PPV: 1.0
Best Corresponding TPR: 0.24
Threshold: 0.6406399999999999
```

In [ ]:

```
print("Best TPR threshold: %.4f => TPR: %.4f; PPV: %.4f" % (best_TPR_threshold, best_T
print("Best PPV threshold: %.4f => PPV: %.4f; TPR: %.4f" % (best_PPV_threshold, best_P
```

```
Best TPR threshold: 0.0400 => TPR: 1.0000; PPV: 0.2294
Best PPV threshold: 0.6406 => PPV: 1.0000; TPR: 0.2400
```

(e) Exploring different thresholds

(i) Using default 0.5 threshold.

Generate confusion matrix and metrics for probabilistic classifier, using threshold 0.5.

In [ ]:
```
best_thr = 0.5
print("ON THE TEST SET:")
print("Chosen best threshold = %.4f" % best_thr)
print("")
# TODO: print(calc_confusion_matrix_for_threshold(...))
print(calc_confusion_matrix_for_threshold(y_test, calibratedTest, thresh = best_thr))
print("")
# TODO: print_perf_metrics_for_threshold(...)
print_perf_metrics_for_threshold(y_test, calibratedTest, best_thr)
```

```
ON THE TEST SET:
Chosen best threshold = 0.5000

Predicted    0   1
True
0          150   5
1           15  10

0.889 ACC
0.400 TPR
0.968 TNR
0.667 PPV
0.909 NPV
```

(ii) Using threshold with highest TPR.

Generate confusion matrix and metrics for probabilistic classifier, using threshold that maximizes TPR.

In [ ]:
```
best_thr = best_TPR_threshold
print("ON THE TEST SET:")
print("Chosen best threshold = %.4f" % best_thr)
print("")
# TODO: print(calc_confusion_matrix_for_threshold(...))
print(calc_confusion_matrix_for_threshold(y_test, calibratedTest, thresh = best_thr))
print("")
# TODO: print_perf_metrics_for_threshold(...)
print_perf_metrics_for_threshold(y_test, calibratedTest, best_thr)
```

```
ON THE TEST SET:
Chosen best threshold = 0.0400

Predicted   0   1
True
0          71  84
1           0  25

0.533 ACC
1.000 TPR
0.458 TNR
0.229 PPV
1.000 NPV
```

(iii) Using threshold with highest PPV.

Generate confusion matrix and metrics for probabilistic classifier, using threshold that maximizes PPV.

In [ ]:
```
best_thr = best_PPV_threshold
print("ON THE TEST SET:")
print("Chosen best threshold = %.4f" % best_thr)
print("")
# TODO: print(calc_confusion_matrix_for_threshold(...))
print(calc_confusion_matrix_for_threshold(y_test, calibratedTest, thresh = best_thr))
print("")
# TODO: print_perf_metrics_for_threshold(...)
print_perf_metrics_for_threshold(y_test, calibratedTest, best_thr)
```

```
ON THE TEST SET:
Chosen best threshold = 0.6406

Predicted    0  1
True
0          155  0
1           19  6

0.894 ACC
0.240 TPR
1.000 TNR
1.000 PPV
0.891 NPV
```

(iv) Compare the confusion matrices from (a)–(c) to analyze the different thresholds.

Answer: (a) has the highest accuracy of all of the values, as it it able to guess about there not being any cancer, although actually detecting cancer is still close to 50-50, with it being on the less accurate side. However, that is better than the other thresholds, as (b) just says that they have cancer to everything (aka it is just an always-1 model), so it is not actually very useful in detection. (c) underdetects, as even though it gets all of the negative cases, it also thinks that some cancer is just negative.

These scores probably occur because of what they are based off of. (a) is arbitrarily chosen in the middle, so it will generally get an even spread, whereas (b) and (c) both focus on the true positive. In (b)'s case it focuses only on getting the positive cases right, while not caring at all for the negative cases (because FP is never in the formula when deciding the best TPR). This leads it to always saying it is positive, because there is no counterbalancing factor. On the other hand, (b) focuses on getting the ratio of positives correct. This means that it is way more careful in assigning positives. However, since it only cares about what was assigned positive, it does not account for any positives that (b) misses. This means that it would rather give a false negative than actually guess the correct value. This means that the function would probably approach the all-0 predictor.

Notice that (a) does a good job because the threshold happened to be chosen in the range of the thresholds between the threshold of (b) and (c), which makes sense as it is trying to find something in between the all-0 and all-1, which will probably have a better well rounded approach. It would be best in the future to also account for the assigned negatives and how accurate those are.