Project 2 Report - CMSC 409 - Artificial Intelligence

Steven Hernandez

You will notice for each scenario (on the following pages), there are 4 graphs. These graphs are described in the table below.

Final sep_line after learning	Graph of all sep_lines during learning
Graph of errors (blue: training set error, gray: testing set error)	Change of weights over time. (red: x_weight, green: y_weight, blue: bias)

Error for training set across each different scenario.

	25%	50%	75%
Hard	0.092	0.1025	0.153
Soft	0.089	0.095	0.0903333333333

Error for testing set across each different scenario.

	25%	50%	75%
Hard	0.095	0.1075	0.152
Soft	0.0956666666667	0.087	0.088

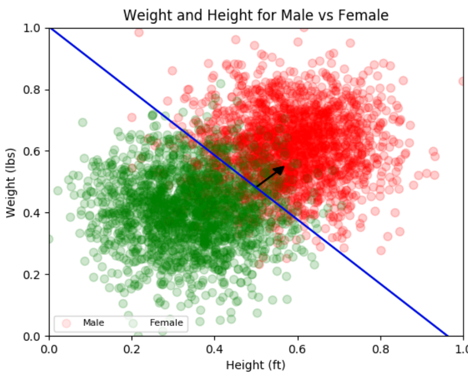
As we can see, **soft** activation results in the lowest error compared to **hard** activation. We can see that while soft activation with 75% training data results in the lowest error for the training set, soft activation with 50% training data actually does better for the testing set.

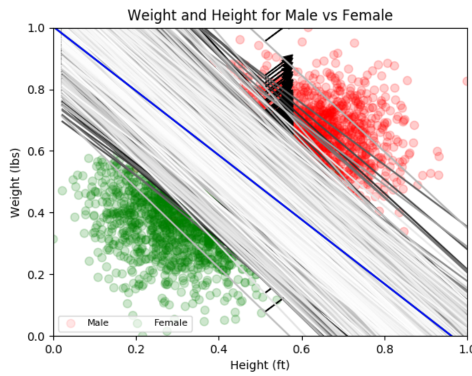
Based on the graphs for `% error over iterations`, we can see that error jumps around quite extremely for **hard** activation. As a result, it seems we do not actually end up with the best error. For example, you will see in the *Errors* table for Hard activation with sample size 75%, the final error was 0.152% while the best error had actually been 0.073% (which happened to have happened quite early on iteration 39). It might be the case that we need to lower alpha for these graphs. **Soft** activation on the other hand smoothly moves towards it's best value

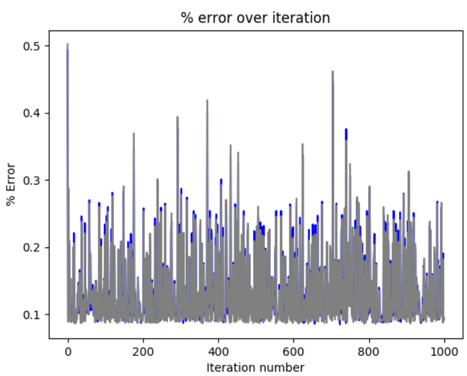
On that point, it seems to be the case that **soft** activation reaches just about its best accuracy after the first iteration (after going through each item in the training set once).

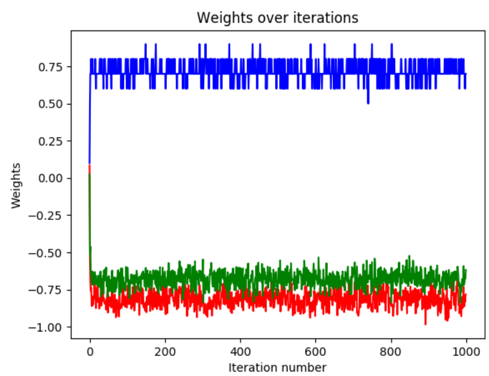
Surprisingly, Hard activation with 75% training results in the best overall error of 0.073%. Unfortunately, this error was not the final output from training and as such was lost.

Hard Activation With A Sample Size Of 25%









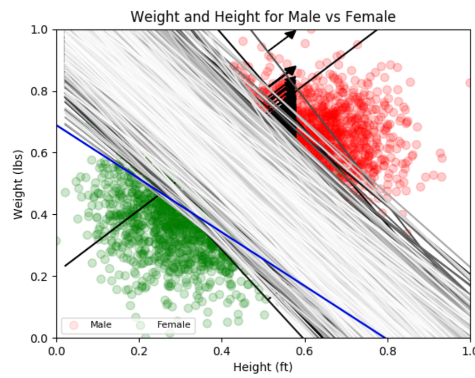
Errors

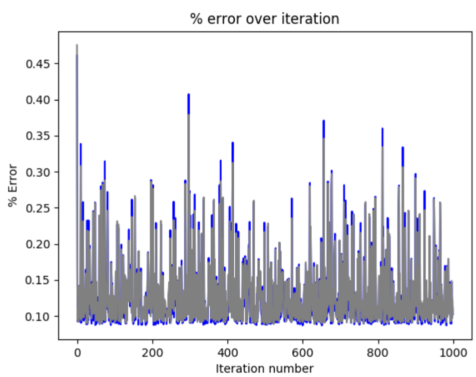
	Training Set Error	Test Set Error
Start	0.493	0.502333333333
End	0.092	0.095
Best	0.085	0.0856666666667

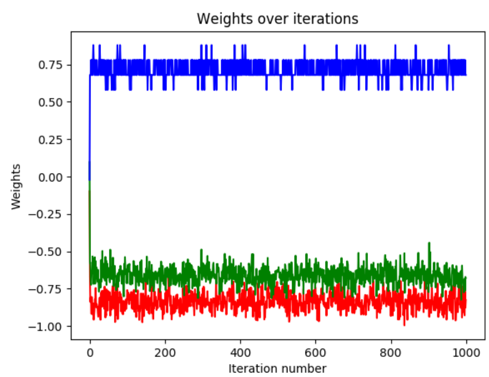
Weights

	x_weight	y_weight	bias
Random initial	0.0845839554527	0.0233985844388	0.0997600145608
Final	-0.781978073692	-0.617995540181	0.699760014561

Hard Activation With A Sample Size Of 50%







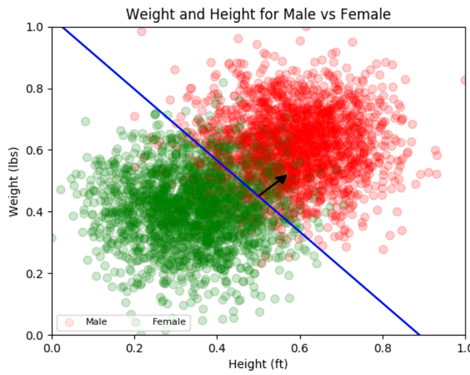
Errors

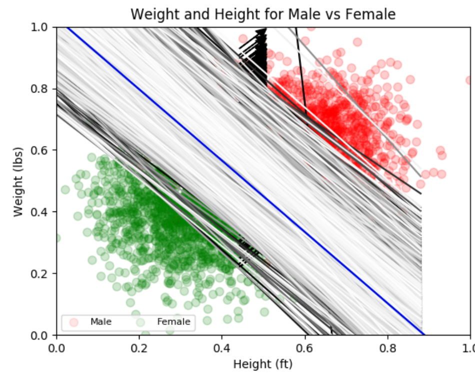
	Training Set Error	Test Set Error
Start	0.461	0.4755
End	0.1025	0.1075
Best	0.087	0.089

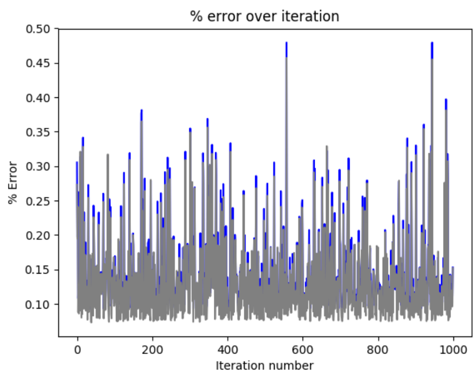
Weights

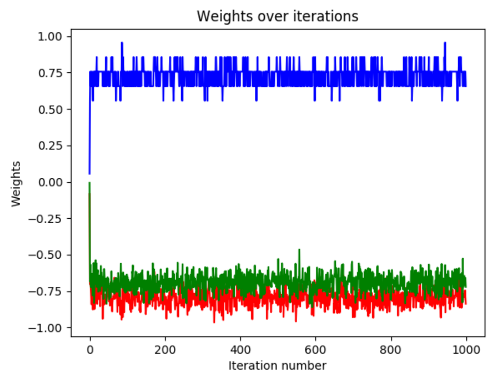
	x_weight	y_weight	bias
Random initial	-0.0985943520343	0.0971523720202	-0.0205306359243
Final	-0.825315759475	-0.678091052275	0.679469364076

Hard Activation With A Sample Size Of 75%









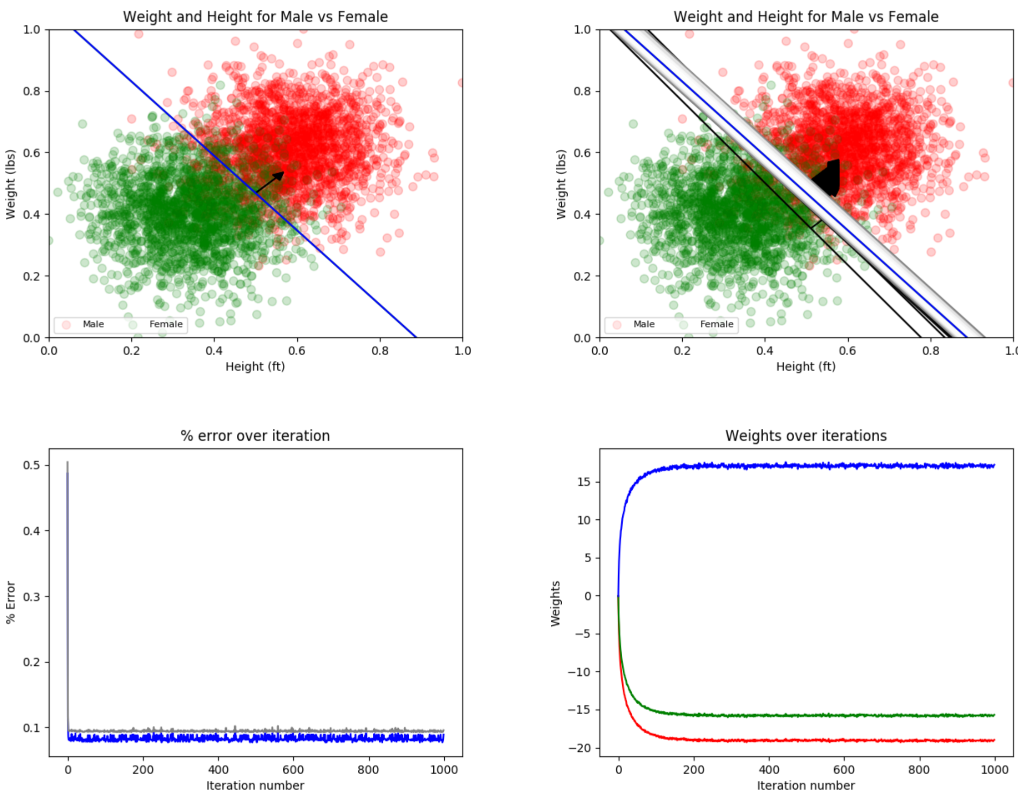
Errors

	Training Set Error	Test Set Error
Start	0.305333333333	0.273
End	0.153	0.152
Best	0.092	0.073

Weights

	x_weight	y_weight	bias
Random initial	-0.0839186901965	-0.00733468132758	0.0559203491984
Final	-0.836525679328	-0.721681753922	0.655920349198

Soft Activation With A Sample Size Of 25%



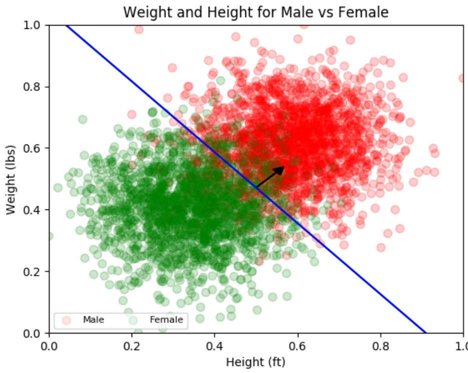
Errors

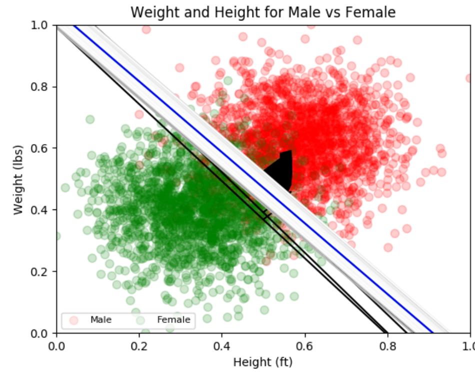
	Training Set Error	Test Set Error
Start	0.487	0.504333333333
End	0.089	0.0956666666667
Best	0.077	0.0926666666667

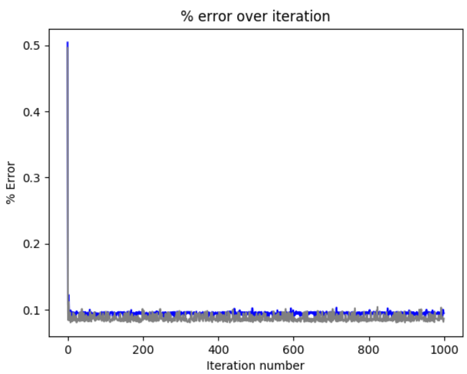
Weights

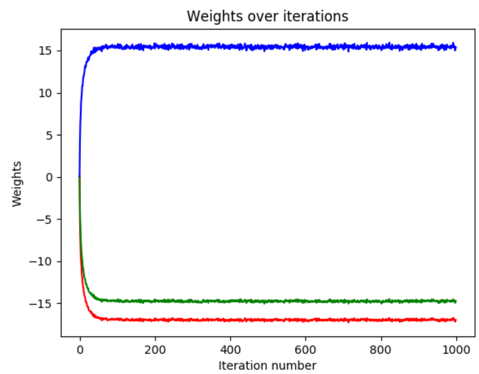
	x_weight	y_weight	bias
Random initial	-0.0775117844112	-0.058170674437	-0.0882074610986
Final	-18.9860892731	-15.6895262882	17.2005178237

Soft Activation With A Sample Size Of 50%









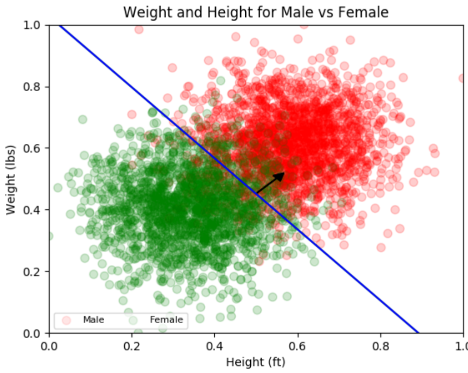
Errors

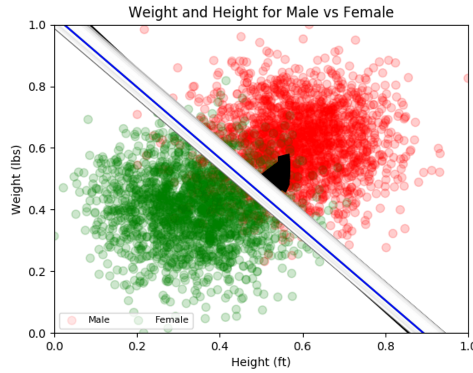
	Training Set Error	Test Set Error
Start	0.504	0.496
End	0.095	0.087
Best	0.0895	0.081

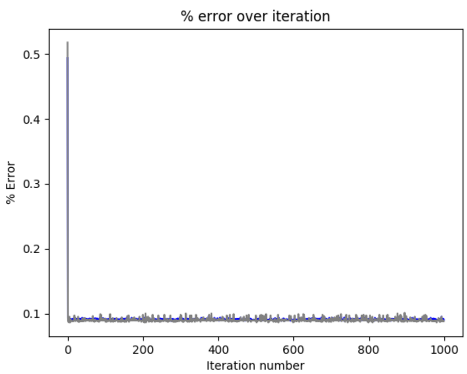
Weights

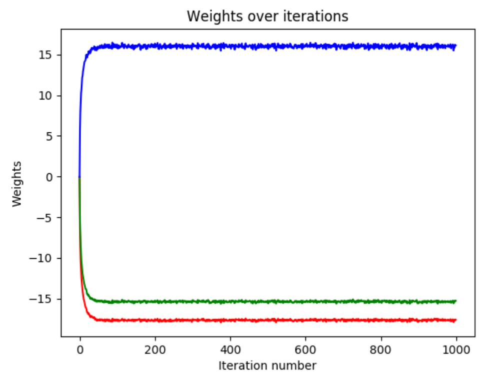
	x_weight	y_weight	bias
Random initial	0.0537412708628	0.073126432088	0.0895714708106
Final	-16.9690622612	-14.7168771707	15.3914435517

Soft Activation With A Sample Size Of 75%









Errors

	Training Set Error	Test Set Error
Start	0.494	0.518
End	0.0903333333333	0.088
Best	0.0883333333333	0.086

Weights

	x_weight	y_weight	bias
Random initial	-0.0144619777597	-0.0879317249207	-0.0261476237457
Final	-17.5950450411	-15.2551140412	16.1321462858

This project uses code from `project1.py` from last time as well as new code from `project2.py`

```
# project2.py
import os
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import random
import markdown as md
import project1

directory = "Project1_data/"
dataFileName = directory + "data.txt"
reportFileName = "report2.md"

area = 50
alpha = 0.1
epsilon = 0.00005
ni = 1000


def plot_xy_sep_line(sep_line, data_frame, color="0.18"):
    x_weight = sep_line[0]
    y_weight = sep_line[1]
    bias = sep_line[2]

    min = data_frame[0].min()
    max = data_frame[0].max()
    mid = (min + ((max - min) / 2))

    # formula is y_weight(y) = x_weight(x) + bias(1)
    # or y = (x_weight/a)y_weight + (bias/y_weight)
    y1 = -(((x_weight * min) / y_weight) + (bias / y_weight))
    y2 = -(((x_weight * max) / y_weight) + (bias / y_weight))
    y_mid = -(((x_weight * mid) / y_weight) + (bias / y_weight))

    ax = plt.axes()
   ax.arrow(mid, y_mid, 0.05, 0.05, head_width=0.025, head_length=0.025, fc='k', ec='k', color="b")

    plt.plot([min, max], [y1, y2], color=color)

    return plt


def build_height_plot(data_frame):
    return project1.plot_male_and_females(data_frame, remove_y_axis=True)


def build_height_weight_plot(data_frame):
    return project1.plot_male_and_females(data_frame)


# Now, we normalize the data down to unipolar.
```

```
# From 0 to 1
def normalize_data_frame(dataframe):
    ndf = dataframe.copy()

    min_height = dataframe[0].min()
    max_height = dataframe[0].max()
    min_weight = dataframe[1].min()
    max_weight = dataframe[1].max()

    ndf[0] = (dataframe[0] - min_height) / (max_height - min_height)
    ndf[1] = (dataframe[1] - min_weight) / (max_weight - min_weight)
    return ndf


def calculate_weight_after_delta_d(current_weight, current_pattern, hard_activation=True, alpha=alpha, k=
    net = (current_weight[0] * current_pattern[0] +
           current_weight[1] * current_pattern[1] +
           current_weight[2])

    if hard_activation:
        output = 1 if net > 0 else 0
        delta_d = alpha * (current_pattern[2] - output)
    else:
        output = (np.tanh(net * k) + 1) / 2
        delta_d = alpha * (current_pattern[2] - output)

    current_pattern[0] *= delta_d
    current_pattern[1] *= delta_d
    current_pattern[2] = delta_d

    current_weight[0] += current_pattern[0]
    current_weight[1] += current_pattern[1]
    current_weight[2] += current_pattern[2]

    return current_weight


errors = []
total_errors = []
weights = [[], [], []]


def calculate_error(data_frame, sep_line):
    error_matrix = project1.get_confusion_matrix(data_frame, sep_line)
    return 1 - ((error_matrix[1] + error_matrix[0]) / (data_frame[0].count()))


def learn(train_df, test_df, sep_line, number_of_iterations, hard):
    final_sep_line = None

    for i in range(0, number_of_iterations):
        print("iteration", i)

        total_error = calculate_error(test_df, sep_line)
```

```
        total_errors.append(total_error)

        plot_xy_sep_line(sep_line, test_df, color=str(i / number_of_iterations))

        err = calculate_error(train_df, sep_line)
        errors.append(err)

        weights[0].append(sep_line[0])
        weights[1].append(sep_line[1])
        weights[2].append(sep_line[2])

        if epsilon > total_error:
            break

        # mix up the test data frame so that we learn in different ways(?)
        train_df = train_df.sample(frac=1)
        # For each element in the data_frame `train_df`
        for index, row in train_df.iterrows():
         new_weights = calculate_weight_after_delta_d(sep_line, row, hard_activation=hard)

            sep_line[0] = new_weights[0]
            sep_line[1] = new_weights[1]
            sep_line[2] = new_weights[2]

        final_sep_line = sep_line

    return final_sep_line


def build_graphs(full_data_frame, folder_destination, original_sep_line, final_sep_line):
    plt.figure(2)
    build_height_weight_plot(full_data_frame)
    plot_xy_sep_line(original_sep_line, full_data_frame, color="g")
    plot_xy_sep_line(final_sep_line, full_data_frame, color="b")

    plt.figure(1)
    plt.axis((0, 1, 0, 1))
    plt.savefig(folder_destination + "all_sep_lines")
    plt.gcf().clear()
    plt.figure(2)
    plt.axis((0, 1, 0, 1))
    plt.savefig(folder_destination + "start_end_lines")
    plt.gcf().clear()

    plt.figure(3)
    plt.plot(np.arange(len(errors)), errors, color="b")
    plt.plot(np.arange(len(total_errors)), total_errors, color="gray")
    plt.title("% error over iteration")
    plt.xlabel("Iteration number")
    plt.ylabel("% Error")
    plt.savefig(folder_destination + "error")
    plt.gcf().clear()

    plt.figure(4)
```

```
    plt.plot(np.arange(len(weights[0])), weights[0], color='r')
    plt.plot(np.arange(len(weights[1])), weights[1], color='g')
    plt.plot(np.arange(len(weights[2])), weights[2], color='b')
    plt.title("Weights over iterations")
    plt.xlabel("Iteration number")
    plt.ylabel("Weights")
    plt.savefig(folder_destination + "weights")
    plt.gcf().clear()


def build_report_file(filename, arr):
    file = open(filename, 'w')

    for item in arr:
        file.write("%s\n" % item)


def build_report_file_from_weights_arrays(filename, arr):
    file = open(filename, 'w')

    for i in range(0, len(arr[0])):
        file.write("%s,%s,%s\n" % (arr[0][i], arr[1][i], arr[2][i]))


def build_all_report_files(filepath):
    build_report_file(filepath + "errors.txt", errors)
    build_report_file(filepath + "total_errors.txt", total_errors)
    build_report_file_from_weights_arrays(filepath + "weights.txt", weights)


def reset_global_values():
    errors.clear()
    total_errors.clear()
    weights[0].clear()
    weights[1].clear()
    weights[2].clear()
    for i in range(1, 5):
        plt.figure(i)
        plt.gcf().clear()


def render(plt, hard=True, sampleFraction=0.25):
    df = pd.read_csv(dataFileName, header=None)

    df = normalize_data_frame(df)

    # smaller amount of random items
    train_df = df.sample(frac=sampleFraction)

    test_df = df[~df.isin(train_df)]

    plt.figure(1)
    plt = build_height_weight_plot(df)
    plt.figure(2)
```

```
    plt = build_height_weight_plot(df)

    rand_x = 0.1
   sep_line = [random.uniform(-rand_x, rand_x), random.uniform(-rand_x, rand_x), random.uniform(-rand_x, r
    original_sep_line = sep_line

    plt.figure(1)
    final_sep_line = learn(train_df, test_df, sep_line, ni, hard)

    plt.figure(1)
    build_height_weight_plot(df)
    plot_xy_sep_line(original_sep_line, df, color="g")
    plot_xy_sep_line(sep_line, df, color="b")

    # Store graph images
    folder_destination = "images/project2/"
    folder_destination += "hard/" if hard else "soft/"
    folder_destination += str(int(100 * sampleFraction)) + "_"
    build_graphs(df, folder_destination, original_sep_line, final_sep_line)

    # Save graphed values to a reports file so that we don't lose data
    folder_destination = "data/project2/"
    folder_destination += "hard/" if hard else "soft/"
    folder_destination += str(int(100 * sampleFraction)) + "_"
    build_all_report_files(folder_destination)

    # reset global values
    reset_global_values()


def build_report():
    file = open(reportFileName, "w")
    project1.save_markdown_report(file, [
     md.meta_data("Project 2 Report - CMSC 409 - Artificial Intelligence", "Steven Hernandez"),
        md.p(
        "You will notice for each scenario (on the following pages), there are 4 graphs. These graphs are des
        md.table([
            ["Final sep_line after learning", "Graph of all sep_lines during learning"],
            ["Graph of errors (blue: training set error, gray: testing set error)",
             "Change of weights over time. (red: x_weight, green: y_weight, blue: bias)"]
        ], width=40),
    ])

    final_training_errors = {
        "hard": [],
        "soft": [],
    }

    final_testing_errors = {
        "hard": [],
        "soft": [],
    }

    for activation_type in ("hard", "soft"):
```

```
    for sample_size in ("25", "50", "75"):
        # Calculate errors
    train_error_df = pd.read_csv("./data/project2/" + activation_type + "/" + sample_size + "_errors.tx
                                     header=None)
    test_error_df = pd.read_csv("./data/project2/" + activation_type + "/" + sample_size + "_total_erro
                                    header=None)

        final_training_errors[activation_type].append(str(train_error_df[0].iloc[-1]))
        final_testing_errors[activation_type].append(str(test_error_df[0].iloc[-1]))

project1.save_markdown_report(file, [
    md.h4("Error for training set across each different scenario."),
    md.table([
        ["", "25%", "50%", "75%"],
        ["Hard", final_training_errors["hard"][0], final_training_errors["hard"][1],
         final_training_errors["hard"][2]],
        ["Soft", final_training_errors["soft"][0], final_training_errors["soft"][1],
         final_training_errors["soft"][2]],
    ], width=15),
    md.h4("Error for testing set across each different scenario."),
    md.table([
        ["", "25%", "50%", "75%"],
    ["Hard", final_testing_errors["hard"][0], final_testing_errors["hard"][1], final_testing_errors[
    ["Soft", final_testing_errors["soft"][0], final_testing_errors["soft"][1], final_testing_errors[
    ], width=15),
  md.p("""As we can see, **soft** activation results in the lowest error compared to **hard** activation.
     We can see that while soft activation with 75% training data results in the lowest error for the trai
     soft activation with 50% training data actually does better for the testing set."""),
    md.p("""Based on the graphs for `% error over iterations`,
         we can see that error jumps around quite extremely for **hard** activation.
         As a result, it seems we do not actually end up with the best error.
     For example, you will see in the *Errors* table for Hard activation with sample size 75%,
         the final error was 0.152% while the best error had actually been 0.073%
         (which happened to have happened quite early on iteration 39).
         It might be the case that we need to lower alpha for these graphs.
       **Soft** activation on the other hand smoothly moves towards it's best value"""),
  md.p("""On that point, it seems to be the case that **soft** activation reaches just about its best accu
     after the first iteration (after going through each item in the training set once)."""),
  md.p("""Surprisingly, Hard activation with 75% training results in the best overall error of 0.073%.
     Unfortunately, this error was not the final output from training and as such was lost.
         """),
    md.page_break(),
])

for activation_type in ("hard", "soft"):
    for sample_size in ("25", "50", "75"):
        # Calculate errors
    train_error_df = pd.read_csv("./data/project2/" + activation_type + "/" + sample_size + "_errors.tx
                                     header=None)
    test_error_df = pd.read_csv("./data/project2/" + activation_type + "/" + sample_size + "_total_erro
                                    header=None)
    weights_df = pd.read_csv("./data/project2/" + activation_type + "/" + sample_size + "_weights.txt",
                                 header=None)
```

```
            project1.save_markdown_report(file, [
           md.h3(str.title(activation_type + " activation with a sample size of " + sample_size + "%")),
                md.images([
             ["./images/project2/" + activation_type + "/" + sample_size + "_start_end_lines.png", ""],
             ["./images/project2/" + activation_type + "/" + sample_size + "_all_sep_lines.png", ""],
                ]),
                md.images([
             ["./images/project2/" + activation_type + "/" + sample_size + "_error.png", "errors"],
             ["./images/project2/" + activation_type + "/" + sample_size + "_weights.png", "weights"],
                ]),
                md.h4("Errors"),
                md.table([
                    ["", "Training Set Error", "Test Set Error"],
                 ["Start", str(train_error_df[0].iloc[0]), str(test_error_df[0].iloc[0])],
                 ["End", str(train_error_df[0].iloc[-1]), str(test_error_df[0].iloc[-1])],
                    ["Best", str(train_error_df[0].min()), str(test_error_df[0].min())],
                ], width=15),
                md.h4("Weights"),
                md.table([
                    ["", "x_weight", "y_weight", "bias"],
                 ["Random initial", str(weights_df[0].iloc[0]), str(weights_df[1].iloc[0]),
                     str(weights_df[2].iloc[0])],
             ["Final", str(weights_df[0].iloc[-1]), str(weights_df[1].iloc[-1]), str(weights_df[2].iloc[
                ], width=10),
                md.page_break(),
            ])

    project1.save_markdown_report(file, [
     md.h3("This project uses code from `project1.py` from last time as well as new code from `project2.py`"
        md.code(file="project2.py"),
        md.code(file="project1.py"),
    ])

    file.close()

   os.system("pandoc --latex-engine=xelatex -V geometry=margin=1in -s -o FINAL_REPORT_2.pdf report2.md")
    print("Report created")


def main(plt):
    # render(plt, hard=True, sampleFraction=0.25)
    # render(plt, hard=True, sampleFraction=0.5)
    # render(plt, hard=True, sampleFraction=0.75)

    # render(plt, hard=False, sampleFraction=0.25)
    # render(plt, hard=False, sampleFraction=0.5)
    # render(plt, hard=False, sampleFraction=0.75)

    build_report()


if __name__ == "__main__":
    main(plt)
```

```
# project1.py
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import markdown as md

directory = "Project1_data/"
dataFileName = directory + "data.txt"
sepLineAFileName = directory + "sep_line_a.txt"
sepLineBFileName = directory + "sep_line_b.txt"
reportFileName = "report.md"

area = 50
alpha = 0.1


def generate_random_data():
    data_file = open(dataFileName, "w")

    for gender in range(0, 2):
        height_mean = 70 / 12 if gender == 0 else 65 / 12
        weight_mean = 200 if gender == 0 else 165

        for i in range(0, 2000):
            # generate random heights and weights in a `normalized` way
            height = np.random.normal(height_mean, 0.2)
            weight = np.random.normal(weight_mean, 20)

            data_file.write(str(height) + "," + str(weight) + "," + str(gender) + "\n")

    data_file.close()


def separate_males_and_females(data_frame, remove_y_axis=False):
    # returns: (males, females)
    return data_frame[data_frame[2] == 0], data_frame[data_frame[2] == 1]


def plot_male_and_females(data_frame, remove_y_axis=False):
    males, females = separate_males_and_females(data_frame)

    male_x = males[0]
    male_y = np.full(males[0].shape, -0.001) if remove_y_axis else males[1]

    female_x = females[0]
    female_y = np.full(females[0].shape, 0.001) if remove_y_axis else females[1]

    male_plot = plt.scatter(male_x, male_y, s=area, c=np.full(males[2].shape, 'r'), alpha=alpha)
    female_plot = plt.scatter(female_x, female_y, s=area, c=np.full(females[2].shape, 'g'), alpha=alpha)

    plt.legend((male_plot, female_plot),
               ('Male', 'Female'),
               scatterpoints=1,
```

```
               loc='lower left',
               ncol=3,
               fontsize=8)

    if remove_y_axis:
        plt.title("Height for Male vs Female")
        plt.xlabel("Height (ft)")
    else:
        plt.title("Weight and Height for Male vs Female")
        plt.xlabel("Height (ft)")
        plt.ylabel("Weight (lbs)")

    return plt


def build_height_plot(data_frame, sep_line):
    plot_male_and_females(data_frame, remove_y_axis=True)

    # Plot a vertical line at `x`
    x = sep_line[0][1] / sep_line[0][0]
    plt.plot([x, x], [-0.1, 0.1])

    frame1 = plt.gca()
    frame1.axes.get_yaxis().set_visible(False)

    return plt


def build_height_weight_plot(data_frame, sep_line):
    plot_male_and_females(data_frame)

    # Plot separation line
    x_weight = sep_line[0][0]
    y_weight = sep_line[0][1]
    bias = sep_line[0][2]

    # So that this separation line covers the entire of the plotted data
    # we specify the minimum x and the maximum y for the line.
    x1 = data_frame[0].min()
    x2 = data_frame[0].max()

    # formula is y_weight(y) = x_weight(x) + bias(1)
    # or y = (x_weight/a)y_weight + (bias/y_weight)
    y1 = ((x_weight * x1) / y_weight) + (bias / y_weight)
    y2 = ((x_weight * x2) / y_weight) + (bias / y_weight)

    plt.plot([x1, x2], [y1, y2])

    return plt


def eq(formula, x_range):
    return formula(x_range)
```

```
# returns 0 for female, 1 for male
def get_output_for_row(current_pattern, current_weight):
    net = (current_weight[0] * current_pattern[0] +
           current_weight[1] * current_pattern[1] +
           current_weight[2])

    return 1 if net > 0 else 0


def get_soft_output_for_row(current_pattern, current_weight, k=0.3):
    net = (current_weight[0] * current_pattern[0] +
           current_weight[1] * current_pattern[1] +
           current_weight[2])

    return np.tanh(net * 0.5)


def get_confusion_matrix(data_frame, sep_line):
    true_positive = 0
    true_negative = 0
    false_positive = 0
    false_negative = 0

    for row in data_frame.iterrows():
        r = row[1]

        if len(sep_line) == 3:
            gender = r[2]

            if get_output_for_row(r, sep_line):
                if gender == 1:
                    true_positive += 1
                else:
                    false_positive += 1
            else:
                if gender == 0:
                    true_negative += 1
                else:
                    false_negative += 1
        else:
            height = r[0]
            weight = r[1]
            gender = r[2]
            x_weight = sep_line[0]
            bias = sep_line[1]

            # 0 <= bx - c
            net = x_weight * height - bias * 1

            if net < 0:
                if gender == 1:
                    true_positive += 1
                else:
```

```
                    false_positive += 1
            else:
                if gender == 0:
                    true_negative += 1
                else:
                    false_negative += 1

    return (true_positive,
            true_negative,
            false_positive,
            false_negative)


def save_markdown_report(file, arr):
    for block in arr:
        file.write(block)


def main():
    # Data has been generated, so we don't want to regenerate the data.
    # generate_random_data()

    df = pd.read_csv(dataFileName, header=None)
    sepLineA = pd.read_csv(sepLineAFileName, header=None)
    sepLineB = pd.read_csv(sepLineBFileName, header=None)
    #
    errorMatrix1 = get_confusion_matrix(df, sepLineA)
    errorMatrix2 = get_confusion_matrix(df, sepLineB)

    myPlt = build_height_plot(df, sepLineA)
    myPlt.savefig("images/1d")
    myPlt.gcf().clear()

    myPlt = build_height_weight_plot(df, sepLineB)
    myPlt.savefig("images/2d")
    myPlt.gcf().clear()

    file = open(reportFileName, "w")

    save_markdown_report(file, [
        md.h1("Project 1 Report"),
        md.h2("CMSC 409 - Artificial Intelligence"),
        md.h2("Steven Hernandez"),

        md.p("Fully generated data can be found in `./Project1_data/data.txt"),

        md.h3("*Scenerio 1:* using only height."),
        md.table([
            ["", "Weights"],
            ["x", sepLineA[0][0]],
            ["bias", sepLineA[0][1]]
        ]),
        md.p("Assuming the following"),
        md.image("./images/net.png"),
```

```
    md.p("Or in this situation: "),
    md.p("1 if 0 <= -a(Height) + bias, otherwise 0"),
    md.p("where *a* is some weight and *1* is male and *0* is female."),
md.p("In this situation a=" + str(sepLineA[0][0]) + " and bias=" + str(sepLineA[0][1])),
    md.image("./images/1d.png"),
    md.table([
        ["", "Predicted Male", "Predicted Female"],
        ["Actual Male", errorMatrix1[1], errorMatrix1[2]],
        ["Actual Female", errorMatrix1[3], errorMatrix1[0]]
    ]),
    md.p("**Confusion Matrix**"),
    md.table([
        ["", ""],
        ["Error", 1 - ((errorMatrix1[1] + errorMatrix1[0]) / 4000)],
        ["Accuracy", (errorMatrix1[1] + errorMatrix1[0]) / 4000],
        ["True Positive Rate", errorMatrix1[1] / 2000],
        ["True Negative Rate", errorMatrix1[0] / 2000],
        ["False Positive Rate", errorMatrix1[3] / 2000],
        ["False Negative Rate", errorMatrix1[2] / 2000],
    ]),

    md.h3("*Scenerio 2:* heights and weights."),
    md.table([
        ["", "Weights"],
        ["x", sepLineB[0][0]],
        ["y", sepLineB[0][1]],
        ["bias", sepLineB[0][2]]
    ]),
    md.p("Assuming the following"),
    md.image("./images/net.png"),
    md.p("Or in this situation:"),
    md.p("1 if 0 <= a(Height) - b(Weight) + bias, otherwise 0"),
    md.p("where *a* and *b* are some weights and *1* is male and *0* is female."),
md.p("In this situation a=" + str(sepLineB[0][0]) + " and b=" + str(sepLineB[0][1]) + " and bias=" + str(
        sepLineB[0][2])),
    md.image("./images/2d.png"),
    md.p("Notice, Male and Female are on slightly different levels in this graph"
         "so that one does not completely cover up the other."),
    md.p("**Confusion Matrix**"),
    md.table([
        ["", "Predicted Male", "Predicted Female"],
        ["Actual Male", errorMatrix2[1], errorMatrix2[2]],
        ["Actual Female", errorMatrix2[3], errorMatrix2[0]]
    ]),
    md.table([
        ["", ""],
        ["Error", 1 - ((errorMatrix2[1] + errorMatrix2[0]) / 4000)],
        ["Accuracy", (errorMatrix2[1] + errorMatrix2[0]) / 4000],
        ["True Positive Rate", errorMatrix2[1] / 2000],
        ["True Negative Rate", errorMatrix2[0] / 2000],
        ["False Positive Rate", errorMatrix2[3] / 2000],
        ["False Negative Rate", errorMatrix2[2] / 2000],
    ]),
```

```
        md.h3("Libraries Used"),
        md.p("matplotlib, numpy, pandas, pandoc"),

        md.h3("Selected Code Functions"),
        md.p("Functions used to generate this data and calculations."),
        md.p("The full code can be found in `./project1.py`"),
        md.code(function=generate_random_data),
        md.code(function=plot_male_and_females),
        md.code(function=plot_male_and_females),
        md.code(function=get_confusion_matrix),
    ])

    file.close()

    print("Markdown Report generated in ./report.md")
    print("Convert Markdown file to PDF with ")
    print("`pandoc --latex-engine=xelatex -V geometry=margin=1in -s -o FINAL_REPORT.pdf report.md`")


if __name__ == "__main__":
    main()
```