

COMP6630: Mini Project One

By Colin Vande Vijvere

Coding the KNN algorithm

I chose to code my KNN algorithm, in the following way. I coded 2 different algorithms, one for Manhattan distance, and one for Euclidean distance. Then I coded a “KNN_tests” file, where I manipulated the test data and called my algorithms so they could perform their calculations. The code, you can see below.

KNN using Euclidean distance:

```

1  import numpy as np
2  from collections import Counter
3
4  #Euclidian function
5  def euclidian_distance(x1,x2):
6      return np.sqrt(np.sum((x1-x2)**2))
7
8  class KNNeuclidean:
9      # neirest neighbors = 3
10     def __init__(self, k=3):
11         self.k = k
12
13     # store training samples
14     # X = training samples, y = training lables
15     def fit(self, X, y):
16         self.x_train = X
17         self.y_train = y
18
19
20     #this will predict
21     def predict(self, X):
22         #call predict for all samples in X
23         predicted_labels = [self._predict(x) for x in X]
24         return np.array(predicted_labels)
25
26     def _predict(self, x):
27         # compute distances
28         distances = [euclidian_distance(x, x_train) for x_train in self.x_train]
29
30         # get k neirest samples and labels,
31         # sort array by distances and pick the ones from 0 -> k
32         k_indices = np.argsort(distances)[:self.k]
33
34         #get the labels of k-neirest neighbors
35         k_neirest_labels = [self.y_train[i] for i in k_indices]
36
37         # majority vote, mostcommon class label wins
38         most_common = Counter(k_neirest_labels).most_common(1)
39         return most_common[0][0]

```

KNN using Manhattan distance:

```

1  import numpy as np
2  from collections import Counter
3
4  #Euclidian function
5  def manhattan_distance(x1,x2):
6      return np.abs(np.sum((x1-x2)))
7
8  class KNNmanhattan:
9      # neirest neighbors = 3
10     def __init__(self, k=3):
11         self.k = k
12
13     # store training samples
14     # X = training samples, y = training lables
15     def fit(self, X, y):
16         self.x_train = X
17         self.y_train = y
18
19
20     #this will predict
21     def predict(self, X):
22         #call predict for all samples in X
23         predicted_labels = [self._predict(x) for x in X]
24         return np.array(predicted_labels)
25
26     def _predict(self, x):
27         # compute distances
28         distances = [manhattan_distance(x, x_train) for x_train in self.x_train]
29
30         # get k neirest samples and labels,
31         # sort array by distances and pick the ones from 0 -> k
32         k_indices = np.argsort(distances)[:self.k]
33
34         #get the labels of k-neirest neighbors
35         k_neirest_labels = [self.y_train[i] for i in k_indices]
36
37         # majority vote, mostcommon class label wins
38         most_common = Counter(k_neirest_labels).most_common(1)
39         return most_common[0][0]
40

```

Knn_tests.py:

```

1  from knnManhattan import KNNmanhattan
2  from knnEuclidean import KNNeuclidean
3  import numpy as np
4  import matplotlib.pyplot as plt
5  from matplotlib.colors import ListedColormap
6  from libsvm.svmutil import svm_read_problem
7
8  cmap = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
9  k = 7
10
11  # -----
12  # function to calculate accuracy of knn
13  # -----
14  def accuracy(y_true, y_pred):
15      accuracy = np.sum(y_true == y_pred) / len(y_true)
16      return accuracy
17
18  # -----
19  # Function To Shuffle and Split the Data
20  # -----
21  def shuffle_split_data(X, y, ratio):
22      split = np.random.rand(X.shape[0]) < ratio
23
24      X_Train = X[split]
25      y_Train = y[split]
26      X_Test = X[~split]
27      y_Test = y[~split]
28
29      return X_Train, y_Train, X_Test, y_Test
30
31
32  # -----
33  # formatting iris.scale data
34  # -----
35  y_raw, x_raw = svm_read_problem('iris.scale')
36  length_y = len(y_raw)
37  y = np.array(y_raw)
38  length_x = len(x_raw)
39  x = np.zeros((len(y_raw), 4))
40  for a in range(length_y):
41      for b, c in x_raw[a].items():
42          x[a][b-1] = c
43
44  x_train, y_train, x_test, y_test = shuffle_split_data(x, y, 0.6)
45  length_y_test = len(y_test)

```



```

47 # -----
48 # RUNNING KNN ON iris.scale USING EUCLIDEAN DISTANCE
49 # -----
50 clf = KNNeuclidean(k=k)
51 clf.fit(x_train, y_train)
52 predictions = clf.predict(x_test)
53 print("\n")
54 print("==== KNN classification (euclidean distance) ====", sep='')
55 print("dataset: iris.scale", sep='')
56 print("# nearest neighbors (k): ", k, sep='')
57 print("total predicted: ", len(y_test), sep='')
58 print("correct predictions: ", np.sum(y_test == predictions), sep='')
59 print("accuracy: ", accuracy(y_test, predictions), sep=' ')
60 print("\n")
61
62
63 # -----
64 # RUNNING KNN ON iris.scale USING MANHATTAN DISTANCE
65 # -----
66 clf = KNNmanhattan(k=k)
67 clf.fit(x_train, y_train)
68 predictions = clf.predict(x_test)
69 print("==== KNN classification (manhattan distance) ====", sep='')
70 print("dataset: iris.scale", sep='')
71 print("# nearest neighbors (k): ", k, sep='')
72 print("total predicted: ", len(y_test), sep='')
73 print("correct predictions: ", np.sum(y_test == predictions), sep='')
74 print("accuracy: ", accuracy(y_test, predictions), sep=' ')
75 print("\n")
76
77 Plotting graph
78 plt.figure()
79 plt.scatter(x[:, 0], x[:, 1], c=y, cmap=cmap, edgecolor='k', s=20)
80 plt.show()

```

```

83 # -----
84 # FORMATTING a4a DATA + figuring out value for k
85 # -----
86 y_raw, x_raw = svm_read_problem('a4aSubset')
87 length_y = len(y_raw)
88 y = np.array(y_raw)
89 length_x = len(x_raw)
90 x = np.zeros((len(y_raw), 400))
91 for a in range(length_y):
92     for b, c in x_raw[a].items():
93         x[a][b-1] = c
94
95
96 x_train, y_train, x_test, y_test = shuffle_split_data(x, y, 0.6)
97 length_y_test = len(y_test)
98
99
100 # -----
101 # RUNNING KNN ON iris.scale USING EUCLIDEAN DISTANCE
102 # -----
103 clf = KNNeuclidean(k=k)
104 clf.fit(x_train, y_train)
105 predictions = clf.predict(x_test)
106 print("\n")
107 print("==== KNN classification (euclidean distance) ====", sep='')
108 print("dataset: subset of a4a", sep='')
109 print("# nearest neighbors (k): ", k, sep='')
110 print("total predicted: ", len(y_test), sep='')
111 print("correct predictions: ", np.sum(y_test == predictions), sep='')
112 print("accuracy: ", accuracy(y_test, predictions), sep=' ')
113 print("\n")

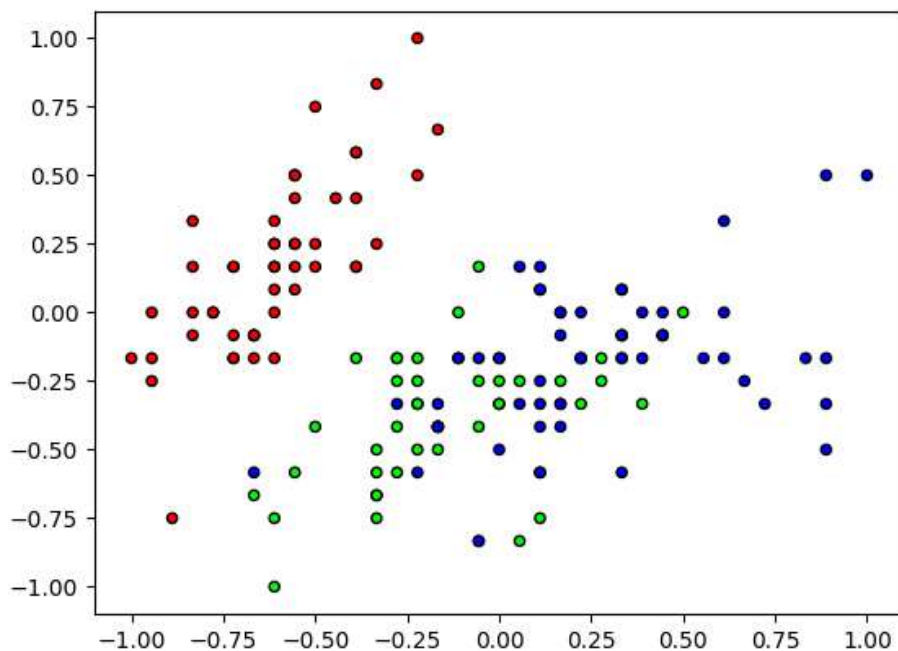
```

```

116 # -----
117 # RUNNING KNN ON iris.scale USING MANHATTAN DISTANCE
118 # -----
119 clf = KNNmanhattan(k=k)
120 clf.fit(x_train, y_train)
121 predictions = clf.predict(x_test)
122 print("==== KNN classification (manhattan distance) ====", sep='')
123 print("dataset: subset of a4a", sep='')
124 print("# nearest neighbors (k): ", k, sep='')
125 print("total predicted: ", len(y_test), sep='')
126 print("correct predictions: ", np.sum(y_test == predictions), sep='')
127 print("accuracy: ", accuracy(y_test, predictions), sep=' ')
128 print("\n")
129
130 #Plotting graph
131 plt.figure()
132 plt.scatter(x[:, 0], x[:, 1], c=y, cmap=cmap, edgecolor='k', s=20)
133 plt.show()
134

```

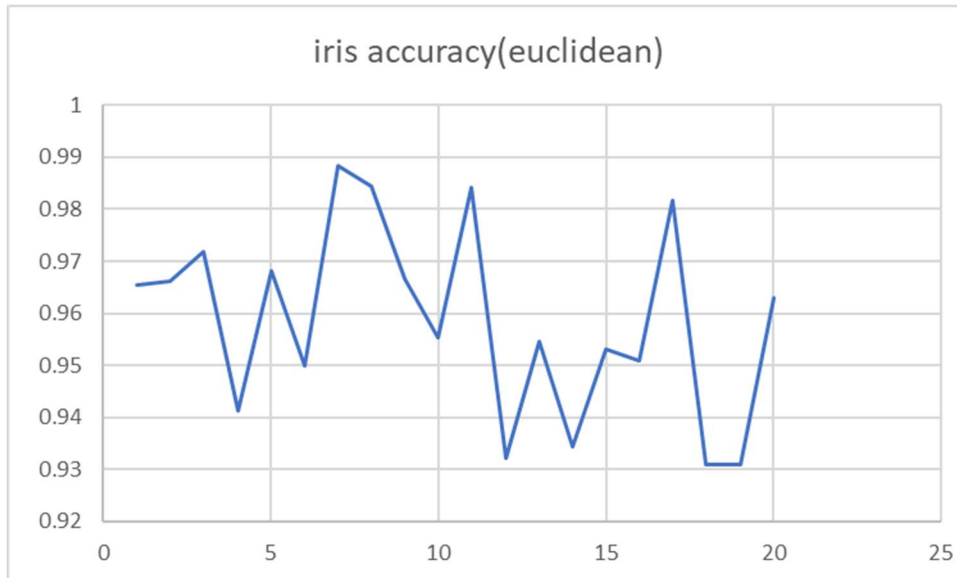
Note: if you've look closely, you can see that I've used matplotlib to visualize my data, here is the graph it created for the iris dataset. (I chose to exclude the a4a graph since there is not much to see)



Accuracy of KNN in relation with the k value

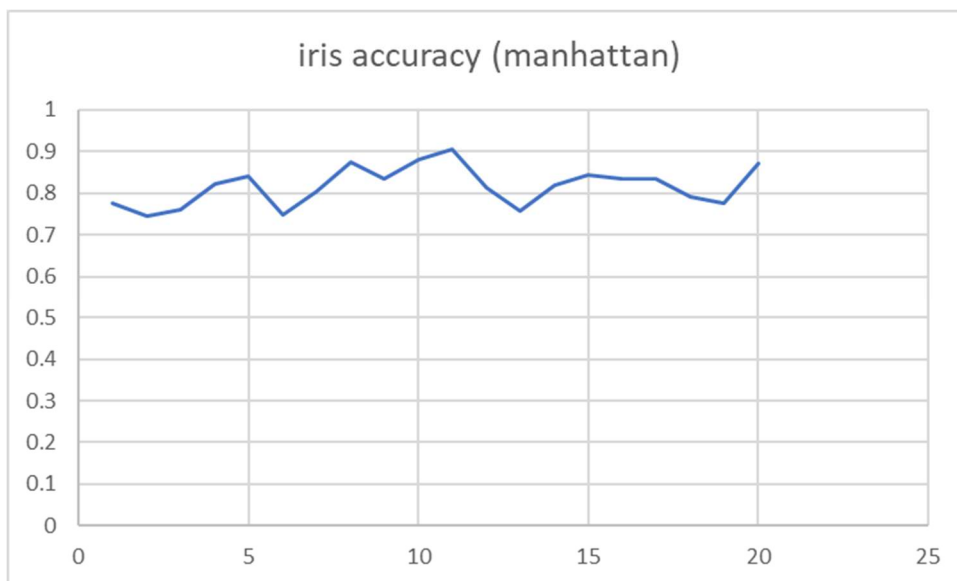
Below you can see graphs that measure the accuracy of my KNN algorithm over the values of k from (0-20). The accuracy number is calculated by running the algorithm 5 times with the same value of k, and taking the average of those accuracy numbers.

KNN accuracy using Euclidean distance on the iris dataset:

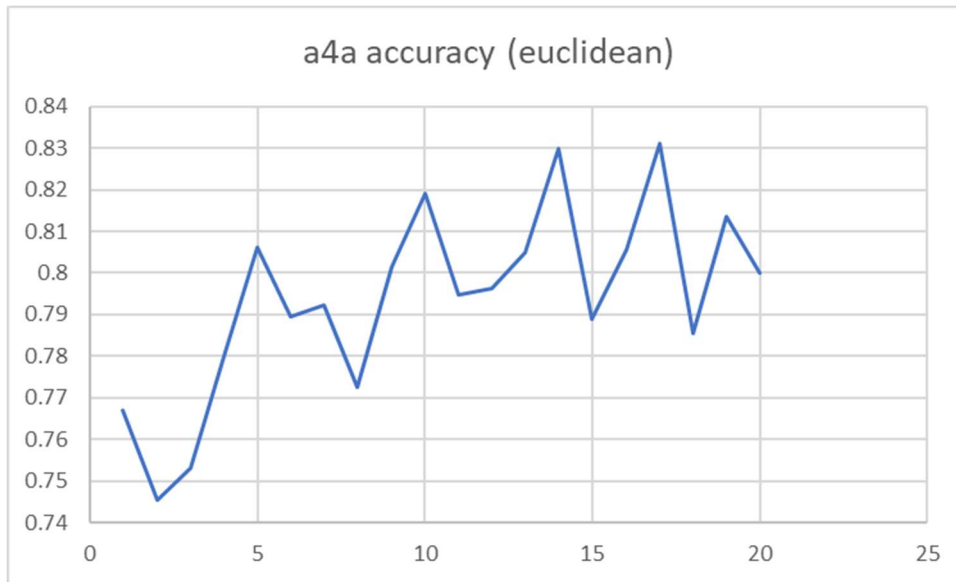


K= 7 resulted in the best average accuracy, which was roughly equal to 98.8%

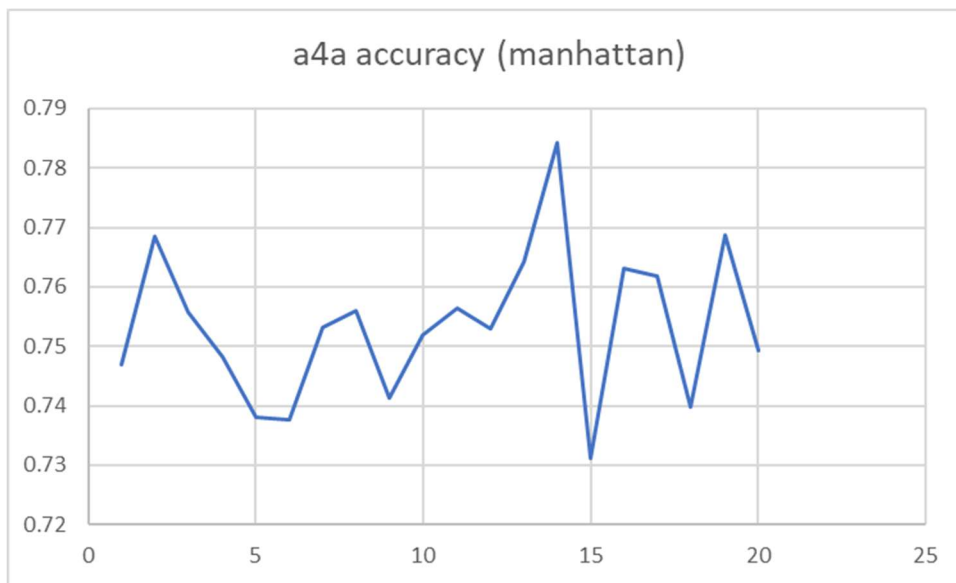
KNN accuracy using Euclidean distance on the iris dataset:



K= 11 resulted in the best average accuracy, which was roughly equal to 90.4%

KNN accuracy using Euclidean distance on a subsample of the a4a dataset:

K= 17 resulted in the best average accuracy, which was roughly equal to 83.1%

KNN accuracy using Manhattan distance on a subsample of the a4a dataset:

K= 14 resulted in the best average accuracy, which was roughly equal to 78.4%

Takeaways from analyzing the data

As I expected before running tests. Using Euclidean distance turns out to be quite a bit more accurate than using Manhattan distance as our distance metric. This makes sense since Euclidean distance is the “true” distance from one point to another.

Perceptron

For my perceptron, I went a similar route as my KNN algorithm. I coded the algorithm and then I created a separate file called "perceptron_tests". In that file, I formatted the data and ran my algorithm on the data.

Perceptron.py code:

```

1  import numpy as np
2
3
4  class Perceptron():
5
6      def __init__(self, alpha=0.2, n_iter=100):
7          self.alpha = alpha
8          self.n_iter = n_iter
9
10         # Store training samples
11         # X = training samples, y = training labels
12         def fit(self, X, y):
13             self.w = np.zeros(X.shape[1])
14             self.b = 0
15             self.errors_ = []
16             for _ in range(self.n_iter):
17                 errors = 0
18                 for xi, yi in zip(X, y):
19                     update = self.alpha * (yi - self.predict(xi))
20                     self.w += update * xi
21                     self.b += update
22                     errors += int(update != 0.0)
23
24                 if errors == 0:
25                     break
26                 self.errors_.append(errors)
27             return self
28
29         # predict functions:
30         def predict(self, x):
31             return np.where(np.dot(x, self.w) + self.b <= 0.0, -1, 1)
32
33         def predict_iris(self, x):
34             result = np.dot(x, self.w) + self.b
35             return np.where(result <= 0.0, -1, result)

```

Perceptron_tests.py code:

```
1  import numpy as np
2  import matplotlib
3  import matplotlib.pyplot as plt
4  from libsvm.svmutil import svm_read_problem
5  from Perceptron import Perceptron
6
7
8  # -----
9  # function to calculate accuracy of knn
10 # -----
11 def accuracy(y_true, y_pred):
12     accuracy = np.round(y_true * 100 / len(y_pred))
13     return accuracy
14
15 # -----
16 # Function To Shuffle and Split the Data
17 # -----
18 def shuffle_split_data(X, y, ratio):
19     split = np.random.rand(X.shape[0]) < ratio
20
21     X_Train = X[split]
22     y_Train = y[split]
23     X_Test = X[~split]
24     y_Test = y[~split]
25
26     return X_Train, y_Train, X_Test, y_Test
```

```

28 # -----
29 # Reading in data
30 # -----
31 y_raw, x_raw = svm_read_problem('a4a')
32
33 dataset = np.zeros((len(y_raw), 123))
34
35 for i in range(len(y_raw)):
36     line = x_raw[i]
37     for k, v in line.items():
38         dataset[i][k - 1] = v
39
40 y = np.array(y_raw)
41
42 perceptron = Perceptron(0.2, 300)
43 perceptron.fit(dataset, y)
44
45
46 y_test, x_test = svm_read_problem('a4a.t')
47 dataset_test = np.zeros((len(y_test), 123))
48
49 for i in range(len(y_test)):
50     line = x_test[i]
51     for k, v in line.items():
52         dataset_test[i][k - 1] = v
53
54 y_test = np.array(y_test)
55
56
57 predicted = np.zeros(len(y_test))
58 for i in range(len(y_test)):
59     predicted[i] = perceptron.predict(dataset_test[i])
60

```

```

64 # -----
65 # Calculating and printing out results
66 # -----
67 print('\n')
68 print('===== a4a =====')
69 print('Total predicted: ', len(predicted), sep='')
70 correct_num = np.sum(predicted == y_test)
71 print('Total correct: ', correct_num, sep='')
72 print('Accuracy = ', accuracy(correct_num, predicted), '%', sep='')
73
74 y_raw, x_raw = svm_read_problem('iris.scale')
75
76 y = np.array(y_raw)
77 x = np.zeros((len(y_raw), 4))
78 for i in range(len(y_raw)):
79     line = x_raw[i]
80     for k, v in line.items():
81         x[i][k - 1] = v
82
83 x_train, y_train, x_test, y_test = shuffle_split_data(x, y, 0.3)
84
85 y_train1 = np.copy(y_train)
86 y_train1[y_train1 != 1] = -1
87 y_train1[y_train1 != -1] = 1
88
89 y_train2 = np.copy(y_train)
90 y_train2[y_train2 != 2] = -1
91 y_train2[y_train2 != -1] = 1
92
93 y_train3 = np.copy(y_train)
94 y_train3[y_train3 != 3] = -1
95 y_train3[y_train3 != -1] = 1
96
97 perceptron_iris1 = Perceptron(0.2, 1000)
98 perceptron_iris1.fit(x_train, y_train1)
99 perceptron_iris2 = Perceptron(0.2, 1000)
100 perceptron_iris2.fit(x_train, y_train2)
101 perceptron_iris3 = Perceptron(0.2, 1000)
102 perceptron_iris3.fit(x_train, y_train3)

```



```

104 predicted = np.zeros(len(y_test))
105 for i in range(len(y_test)):
106     pred = np.zeros(3)
107     pred[0] = perceptron_iris1.predict(x_test[i])
108     pred[1] = perceptron_iris2.predict(x_test[i])
109     pred[2] = perceptron_iris3.predict(x_test[i])
110     predicted[i] = np.argmax(pred) + 1
111
112 print('\n')
113 print('==== iris =====')
114 print('Total predicted: ', len(predicted), sep='')
115 correct_num = np.sum(predicted == y_test)
116 print('Total correct: ', correct_num, sep='')
117 print('Accuracy = ', accuracy(correct_num, predicted), '%', sep='')
118
119
120 # -----
121 #   Creating Graphs
122 # -----
123 weights = perceptron.w
124 top_index = np.argsort(-np.abs(weights))[:20]
125 plt.figure()
126 plt.bar(list(map(str, top_index)), weights[top_index])
127 plt.xlabel('Feature Number')
128 plt.ylabel('Weight')
129 plt.title('Top 20 Weights in a4a Dataset')
130 plt.show(block=False)
131
132 weights = perceptron_iris1.w
133 top_index = np.argsort(-np.abs(weights))
134 plt.figure()
135 plt.bar(list(map(str, top_index)), weights[top_index])
136 plt.xlabel('Feature Number')
137 plt.ylabel('Weight')
138 plt.title('Weights in iris Dataset (Class 1)')
139 plt.show(block=False)
140
141 weights = perceptron_iris2.w
142 top_index = np.argsort(-np.abs(weights))
143 plt.figure()
144 plt.bar(list(map(str, top_index)), weights[top_index])
145 plt.xlabel('Feature Number')
146 plt.ylabel('Weight')
147 plt.title('Weights in a4a Dataset (Class 2)')
148 plt.show(block=False)
149
150 weights = perceptron_iris3.w
151 top_index = np.argsort(-np.abs(weights))
152 plt.figure()
153 plt.bar(list(map(str, top_index)), weights[top_index])
154 plt.xlabel('Feature Number')
155 plt.ylabel('Weight')
156 plt.title('Weights in a4a Dataset (Class 3)')
157 plt.show()

```

Accuracy of the Perceptron algorithm

While playing around with the data I've found that changing the learning rate is what had the biggest impact on the accuracy of my predictions. A learning rate of 0.01 worked very well on the iris dataset, this gave me an accuracy of 78%. The same goes for the a4a dataset where I had 82% accuracy when it came to my predictions. I struggled a lot with the iris dataset and I think it was because my training set was on the small side, I could not increase it by much since the entire dataset is quite small.

Highest accuracy (terminal screenshot):

```

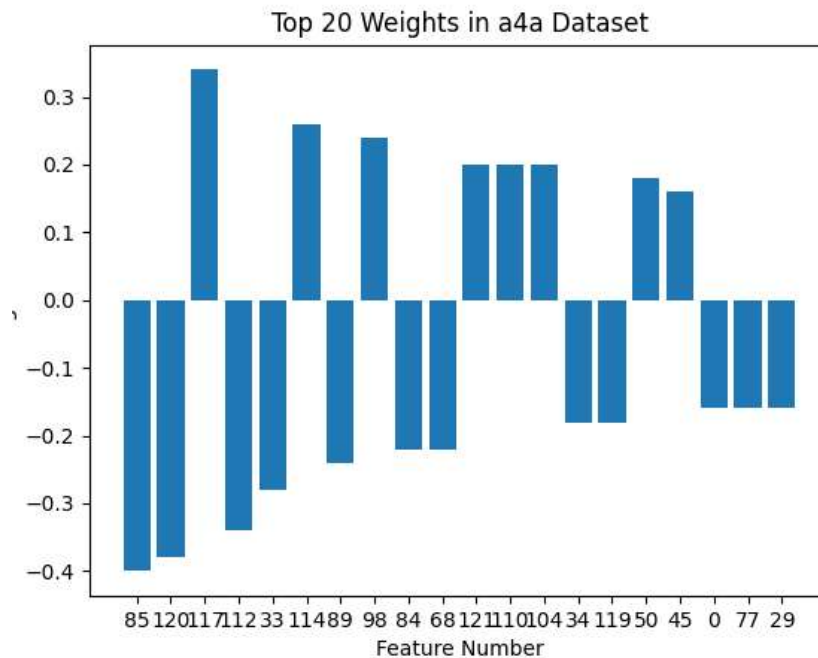
===== a4a =====
Total predicted: 27780
Total correct: 22792
Accuracy = 82.0%

===== iris =====
Total predicted: 23
Total correct: 18
Accuracy = 78.0%

```

Weights of the perceptron algorithm

Below is a picture of the top 20 attributes in the a4a dataset. Note: While about 50% of the weights are negative numbers, we look at the absolute value of the number and not the general value.



Below are some of the other graphs I've created, that are not required for our project specifications.

