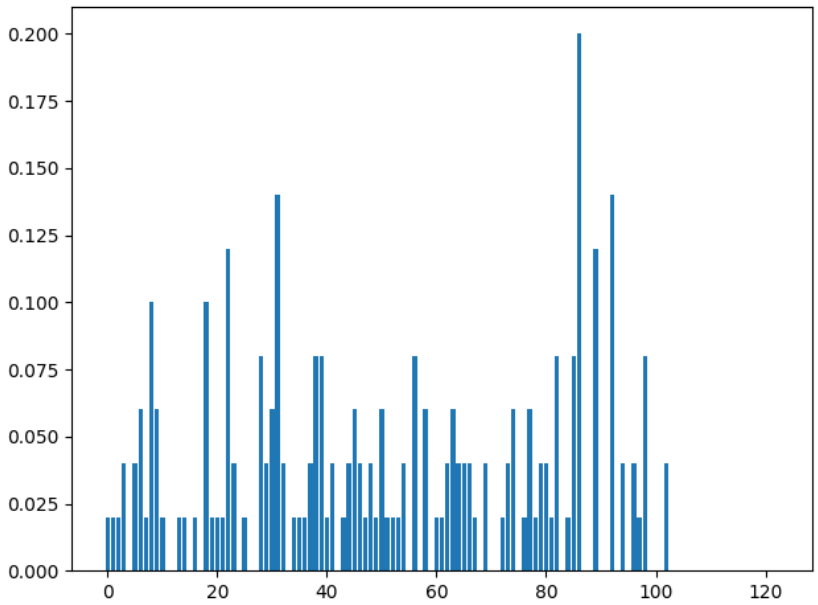
Alex Weeden

Ajw0057

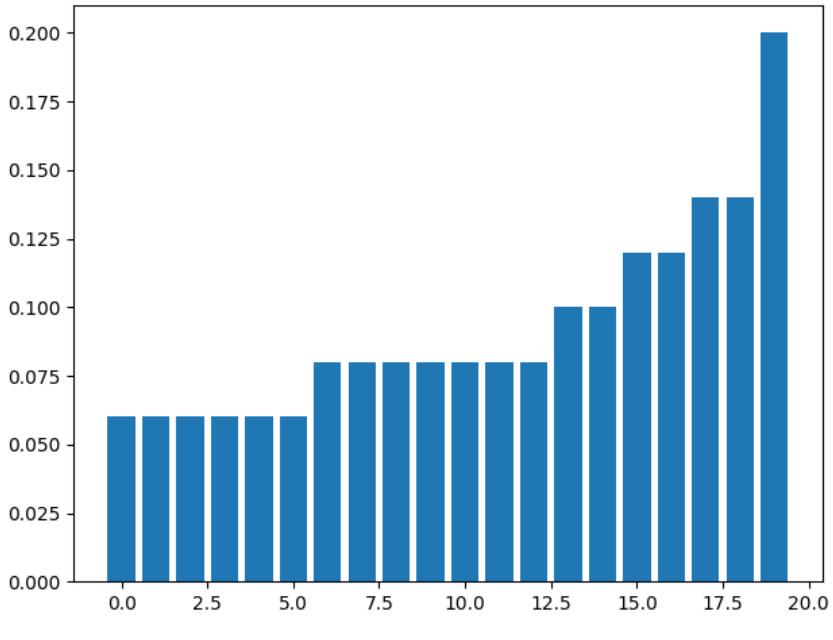
COMP 5630

Mini-Project 1

1.) K-NN
    a. K-Nearest Neighbors remains relatively similar between binary and multi-class classification tasks. The approach is simple: Take the Euclidean distance between a test point and every training point, and assign votes for a certain classification based on which k-number of training points are closest to the test point. An odd k value is important here to avoid ties. For the a4a dataset, the set is trimmed to a manageable level and has a smaller percentage devoted to the training pool than the algorithm for the iris dataset. We can record a misclassification error by measuring how many errors the algorithm makes against the test data. For the a4a dataset, this algorithm achieves an accuracy rate of 80.8%, with a misclassification rate of about 19%. The algorithm performs much better on the multi-class iris data set, with an accuracy rate of 86.7%, misclassifying only 13.3% of test points.
2.) Perceptron
    a. As recommended in the specification for homework 03, I utilized the algorithm designed there to implement perceptron, and expand it to a one-versus-all multi-class classification. The solution approach involves computing weights and biases from training data features to fit a hyperplane that allows near-instant predictions by gauging what side of the hyperplane the test data point lies on. The misclassification error when performing binary classification on the a4a dataset reached 22%, with an accuracy of 78%, which could be improved by tweaking the training/test data split, scalars, or iterations. The algorithm performs similarly on the multi-class iris data, with a misclassification error of 26%.
    b. Finding the most important 20 features is simple since we must already compute weights for each feature while training the model. Taking the absolute value of the weights (first plot), and then sorting and taking the top 20 features (second plot) allows us to see how drastically our top 20 features affect classification.



plot 1



plot 2

KNN Binary A4A data

```
from sklearn.datasets import load_svmlight_file
import numpy as np
from collections import Counter

def kNN(trainX, trainY, test, k=5):
    distances = []
    for i in range(trainX.shape[0]):
        trainFeature = []
        for j in range(123):
            trainFeature.append(trainX[i,j])
```

```
        euclid = np.linalg.norm(np.array(trainFeature)-np.array(test))
        distances.append([euclid, trainY[i]])
    votes = [i[1] for i in sorted(distances)[:k]]
    voteResult = Counter(votes).most_common(1)[0][0]
    return voteResult

xData, yData = load_svmlight_file('a4a.t')
xData = xData[:200]
yData = yData[:200]

testSize = 0.3
xTrain = xData[:-int(testSize*xData.shape[0])]
yTrain = yData[:-int(testSize*xData.shape[0])]
xTest = xData[-int(testSize*xData.shape[0]):]
yTest = yData[-int(testSize*xData.shape[0]):]

correct = 0
total = 0

for i in range(xTest.shape[0]):
    featureList = []
    for j in range(123):
        featureList.append(xTest[i,j])
    vote = kNN(xTrain, yTrain, featureList)
    if vote == yTest[i]:
        correct +=1
    total +=1

acc = correct/total
print(acc)
```

KNN Multi Iris data

```
from sklearn.datasets import load_svmlight_file
import numpy as np
from collections import Counter

def kNN(trainX, trainY, test, k=3):
    distances = []
    for i in range(trainX.shape[0]):
        trainFeature = []
        for j in range(4):
            trainFeature.append(trainX[i,j])
        euclid = np.linalg.norm(np.array(trainFeature)-np.array(test))
        distances.append([euclid, trainY[i]])
    votes = [i[1] for i in sorted(distances)[:k]]
    voteResult = Counter(votes).most_common(1)[0][0]
    return voteResult

xData, yData = load_svmlight_file('iris.scale')

testSize = 0.2
xTrain = xData[:-int(testSize*xData.shape[0])]
yTrain = yData[:-int(testSize*xData.shape[0])]
```

```
xTest = xData[-int(testSize*xData.shape[0]):]
yTest = yData[-int(testSize*xData.shape[0]):]

correct = 0
total = 0
for i in range(xTest.shape[0]):
    featureList = []
    for j in range(4):
        featureList.append(xTest[i,j])
    vote = kNN(xTrain, yTrain, featureList)
    if vote == yTest[i]:
        correct +=1
    total +=1

acc = correct/total
print(acc)
```

Perceptron A4A Binary

```
from sklearn.datasets import load_svmlight_file
import numpy as np
import matplotlib.pyplot as plt
class Perceptron:

    # Initialize perceptron
    # Learning rate scalar, iteration count, and weights/bias
    def __init__(self, scalar=0.01, iter=1000):
        self.scalar = scalar
        self.iter = iter
        #self.stepFunc = self.unitStepFunc()
        self.weight = None
        self.bias = None

    # Fits hyperplane to provided data to test against y, a vector of 1's and
-1's
    def fit(self, X, y):
        samples, features = X.shape

        self.weight = np.zeros(features)
        self.bias = 0

        y_ = np.array([1 if i > 0 else 0 for i in y])

        for _ in range(self.iter):
            for idx, x in enumerate(X):
                output = np.dot(x, self.weight) + self.bias
                predictedY = self.unitStepFunc(output)
                update = self.scalar * (y_[idx] - predictedY)
                self.weight += update * x
                self.bias += update
        return self.weight

    # Predicts class of y based on current weights and biases
    def predict(self,X):
```

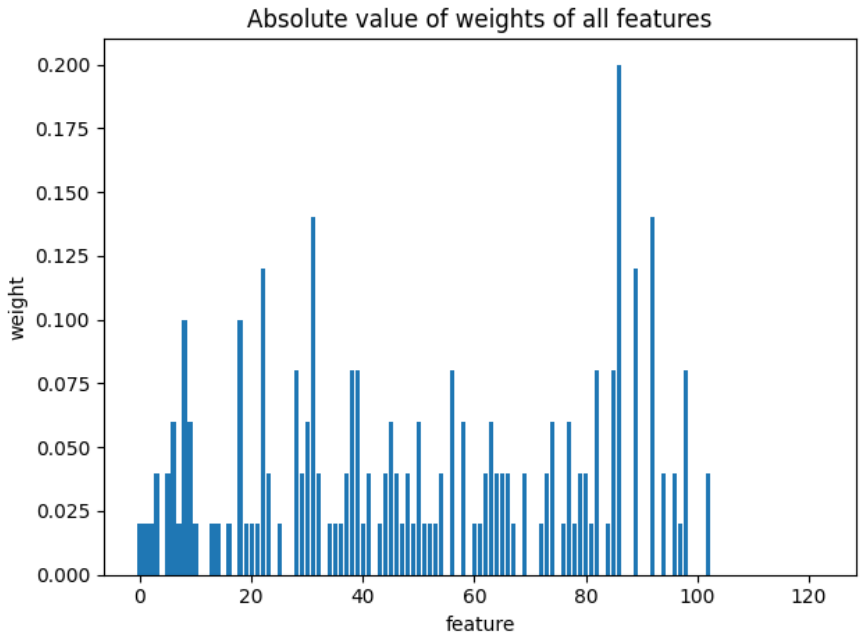
```
        linearOutput = np.dot(X, self.weight) + self.bias
        predictedY = self.unitStepFunc(linearOutput)
        return predictedY

    def unitStepFunc(self,x):
        return np.where(x<0,-1,1)

xData, yData = load_svmlight_file('a4a.t')
xData = xData[:200]
yData = yData[:200]

xdf = []
for i in range(xData.shape[0]):
    tempList = []
    for j in range(123):
        tempList.append(xData[i,j])
    xdf.append(tempList)
ydf = []
for i in range(yData.shape[0]):
    ydf.append(yData[i])

testSize = 0.15
xTrain = xdf[:-int(testSize*len(xdf))]
yTrain = ydf[:-int(testSize*len(ydf))]
xTest = xdf[-int(testSize*len(xdf)):]
yTest = ydf[-int(testSize*len(ydf)):]

perceptron = Perceptron()
weights = np.array(perceptron.fit(np.array(xTrain), yTrain))
weights = np.absolute(weights)









prediction = np.array(perceptron.predict(np.array(xTest)))
actual = np.array(yTest)

total = 0
correct = 0
for i in range(len(prediction)):
    total +=1
    if int(prediction[i]) == int(actual[i]):
        correct +=1

print(correct/total)

plt.bar(np.arange(len(weights)),weights)
plt.show()
```

Perceptron Iris Multi

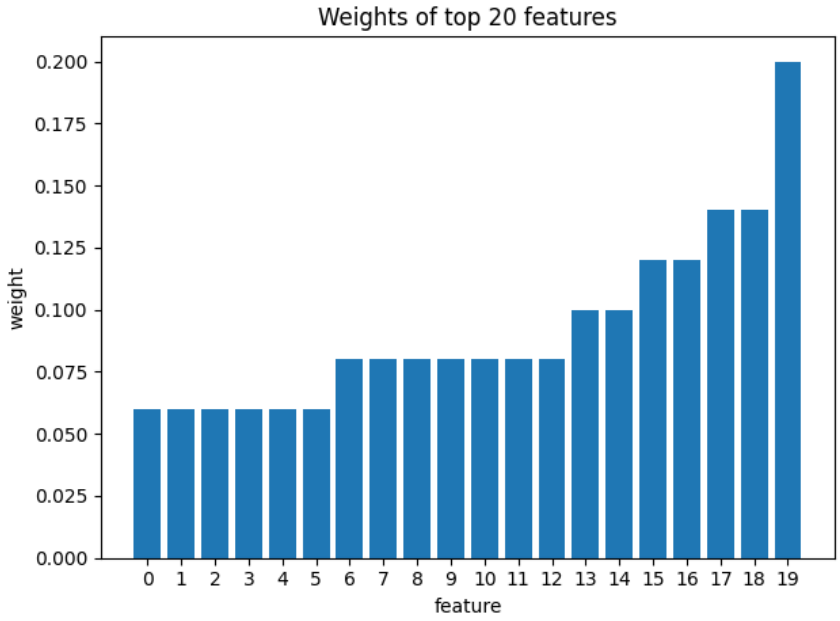
```
from sklearn.datasets import load_svmlight_file
import numpy as np
```

```
class Perceptron:

    # Initialize perceptron
    # Learning rate scalar, iteration count, and weights/bias
    def __init__(self, scalar=0.01, iter=1000):
        self.scalar = scalar
        self.iter = iter
        #self.stepFunc = self.unitStepFunc()
        self.weight = None
        self.bias = None

    # Fits hyperplane to provided data to test against y, a vector of 1's and
-1's
    def fit(self, X, y):
        samples, features = X.shape

        self.weight = np.zeros(features)
        self.bias = 0

        y_ = np.array([1 if i > 0 else 0 for i in y])

        for _ in range(self.iter):
            for idx, x in enumerate(X):
                output = np.dot(x, self.weight) + self.bias
                predictedY = self.unitStepFunc(output)
                update = self.scalar * (y_[idx] - predictedY)
                self.weight += update * x
                self.bias += update

    # Predicts class of y based on current weights and biases
    def predict(self,X):
        linearOutput = np.dot(X, self.weight) + self.bias
        predictedY = self.unitStepFunc(linearOutput)
        return predictedY

    def unitStepFunc(self,x):
        return np.where(x>1,x,2)

xData, yData = load_svmlight_file('iris.scale')
xdf = []
for i in range(xData.shape[0]):
    tempList = []
    for j in range(4):
        tempList.append(xData[i,j])
    xdf.append(tempList)
ydf = []
for i in range(yData.shape[0]):
    ydf.append(yData[i])
np.random.shuffle(xdf)
np.random.shuffle(ydf)

testSize = 0.15
xTrain = xdf[:-int(testSize*len(xdf))]
yTrain = ydf[:-int(testSize*len(ydf))]
xTest = xdf[-int(testSize*len(xdf)):]
```

```
yTest = ydf[-int(testSize*len(ydf)):]
perceptron = Perceptron()
perceptron.fit(np.array(xTrain), yTrain)
prediction = np.array(perceptron.predict(np.array(xTest)))
actual = np.array(yTest)

total = 0
correct = 0
for i in range(len(prediction)):
    total +=1
    if int(prediction[i]) == int(actual[i]):
        correct +=1

print(correct/total)
```


Absolute value of weights of all features
weight
feature
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
0
20
40
60
80
100
120

plot 1


Weights of top 20 features
weight
feature
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

plot 2