REPORT

problem #1 Voted perceptron

1. This is a given pseudo code from HW3.pdf. I have basically used this algorithm for voted perceptron.

```
Initiate k=1, c_1 = 0, w_1 = 0, t = 0;
while t <= T do
    for each training example (x_i, t_i) do
        if t_i (w_k x_i) <= 0 then
            w_k+1 = w_k + t_i x_i;
            c_k+1 = 1;
            k = k + 1
        else
            c_k += 1;
        end
    end
    t = t + 1;
end
```

```
# voted perceptron
while t < T:
  # for each training example
  for i in range(len(train_five)):
    inner = np.dot(w[k], train_five[i])
    pred = np.sign(label_five[i] * inner)
    # misclassification
    if pred <= 0:
      w.append(w[k] + np.dot(y[i], train_five[i]))
      c.append(1)
      k += 1
    else:
      c[k] += 1
  t += 1
```

- I used the given pseudo code from the HW3.pdf file and built the code above.

T = number of epoch(# of time you want to train), t = currently epoch
K = position of weight c = weight of weight(voted) w = classification vector
Label_five = just label {-1, 1}, train_five = +- training data

- First, I have changed the label 0 to -1 to train.
- And when pred <= 0, which means its misclassification(when the sign doesn't match) , then new weight and vote and then go to the next weight. When pred > 0, which means it is classified, then go to the next vote.

2. This equation is also given as an equation from the HW3.pdf file. I used this to predict.

$$\hat{y} = sign(\sum_{k=1}^{K} c_k sign(w_k x))$$

```
prediction = np.zeros(len(test_data_file))
for j in range(len(test_data_file)):
  for K in range(k):
    inner = np.sign(w[K] * test_data_file[j])
    pred = np.sign(np.sum(c[K] * inner))
  prediction[j] = pred

# change the all -1 back to 0
for count in range(prediction.shape[0]):
  if prediction[count] == -1:
    prediction[count] = 0
```

Basically, I checked the sign and store in the inner. And then I stored the prediction of the sign. np.sign only returns -1 if negative, and return 1 if positive. Thus, at the end, there is {-1, 1} in my prediction. After that, I changed back all -1 to 0 because it was given in {0, 1}

Use the last 10% of the training data as your test data. Compare Voted Perceptron on several fractions of your remaining training data.

```
train_test_split = int(0.9 * len(x))
x_train, y_label = x[:train_test_split], y[:train_test_split]
test, test_label = x[train_test_split:], y[train_test_split:]
```

Use the last 10 percent of the training data as my test data 1%,2%,5%,10%,20% and 100% of the first 90% training data to train and compare the performance of Vote Perceptron on the test data.

1% of the training, 99 % remaining, 78% accuracy

```
compare y and pred
 [ True  True False  True  True  True  True False  True  True False  True
 False  True  True  True  True False  True  True  True  True  True  True
  True  True False  True  True  True  True  True  True  True  True  True
  True False  True  True False  True False  True  True  True False False
  True  True]
[[39 11]
 [ 0  0]]
accuracy:  0.78
```

2% of the training, 98% remaining, 80% accuracy

```
compare y and pred
 [ True  True False  True  True  True  True False  True  True False  True
 False  True  True  True  True False  True  True  True  True  True  True
  True  True False  True  True  True  True  True  True  True  True  True
  True False  True  True False  True False  True  True  True False  True
  True  True]
[[39 10]
 [ 0  1]]
accuracy:  0.8
```

5% of the training, 95% remaining, 88% accuracy

```
compare y and pred
 [ True  True  True  True  True  True  True False  True  True  True  True
 False  True  True  True  True False  True  True  True  True  True  True
  True  True False  True  True  True  True  True  True  True  True  True
  True False  True  True  True False  True  True  True  True  True  True
  True  True]
[[38  5]
 [ 1  6]]
accuracy:  0.88
```

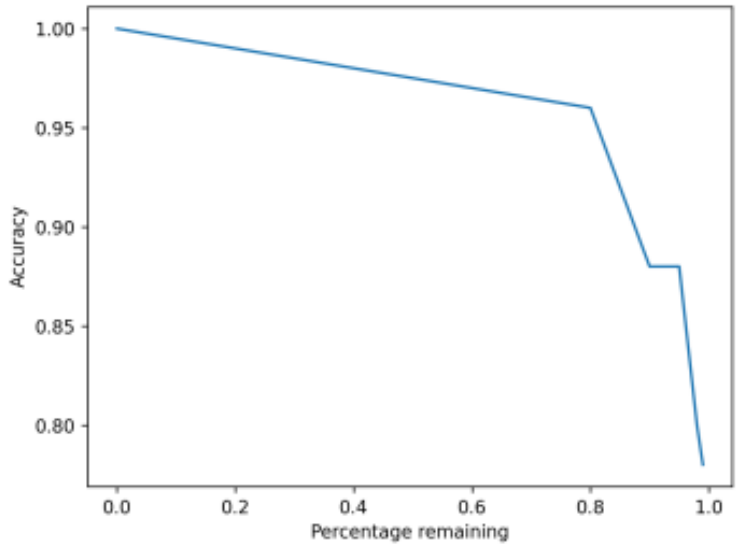
10% of the training, 90% remaining, 88% accuracy

```
compare y and pred
 [ True  True  True  True  True False  True  True False  True  True  True
  True False  True  True False  True  True False  True  True  True  True
  True  True  True  True False  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True]
[[33  0]
 [ 6 11]]
accuracy:  0.88
```

20% of the training, 80% remaining, 96% accuracy

```
compare y and pred
 [ True  True  True  True  True  True  True False  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True False  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True]
[[38  1]
 [ 1 10]]
accuracy:  0.96
```

Plot the accuracy as a function of the size of the fraction you picked (x-axis should be “percent of the remaining training data” and y-axis should be “accuracy”).



REPORT

problem #2 KNN

The KNN is a supervised machine learning algorithm. The algorithm can be used to solve both classification and regression problems. The number of nearest neighbors to a new unknown variable that has to be predicted or classified is denoted by the symbol **K**(for this problem, 1, 2,3, 4,5,10, 20, 30, 40) In this problem, I first used **euclidean_distance** to find the distance between train and test. And I sorted the distance in order to find the label. And I used these labels to find the **majority of neighbors** to predict later. As you can see in the example, Prediction is what I predicted by using KNN and I used my prediction to compare with the truth(y_label) since I set my test equal to Xtrain.csv.

```
# Calculate the Euclidean distance between two vectors
def euclidean_distance(x1, x2):
  euclidean_distance = np.sum(pow((x1 - x2), 2))
  return np.sqrt(euclidean_distance)
```

This is the **euclidean_distance()** function that I used to find the distance between two vectors(x1, x2).

```
# To find the k neighbor's label
def find_neighbors(x_train, y_label, test, k):

  # Predict neighbor = k-th closest neighbors
  Predict_Neighbors = np.zeros(k)

  # temp = store the distance btw x_train and test[0] point
  temp = np.zeros(len(x_train))

  # find the distance and use index to find the k-th closest neighbors
  for i in range(len(x_train)):
    distance = euclidean_distance(x_train[i], test)
    temp[i] = distance
    index = temp.argsort()

  # find the k-th closest label
  for j in range(k):
    Predict_Neighbors[j] = y_label[index[j]]

  return Predict_Neighbors
```

This is the **find_neighbors()** function that I used to find the K-th nearest neighbors label. **Predict_neighbors** = k number of kth closest labels(neighbors), I first used t**he euclidean_distance()** function between training data and test to find the distance for all. And store the distance data into temp. And sort it in order by using **temp.argsort()** so that I can find the k-th closest distance's index from the point. And I stored the label in the Predict_Neighbor and finally returned the k-th nearest neighbors.

```
# set k-th closest
k = 3

# array of neighbors labels(rough before applying majority rule)
neighbors = []
for i in range(len(test)):
  # to get each row line
  test_value = test[i, :]
  labels = find_neighbors(x_train, y_label, test_value, k)
  neighbors.append(labels)
```

k = 3, basically set by the user that I am going to find the 3 labels that I closest to the test point. I used **test[i, :]** because I want to use each row of the test_value(test point) to find the neighbors. And I **append** it as in neighbors.

```
prediction = np.zeros(len(test))
temp = np.zeros(0)

for i in range(len(test)):
  ''' np.unique from numpy.unique
  return_counts: If return_counts is True, also return the number of times each unique item appears in ar.
  values: the sorted unique elements of an array
  counts: the number of times each unique item appears in ar
  '''
  values, count = np.unique(neighbors[i], return_counts = True)
  temp = values[np.argmax(count)]
  prediction[i] = temp
```

prediction = my prediction that I predict it to be, return_counts = if return_counts is true, also return the number of times each unique item appears in arr(neighbors in this case). **Values** = the sorted unique elements of an array(if [1, 2, 1, 4, 3, 1] —> then [1, 2, 3, 4]). **Count** = the number of times each unique item appears in arr(neighbors). This is from numpy.unique and I used it to count the float number in the array. And I used **np.argmax(count)** to find the index of maximum label and store in **values** that originally have a unique element of an array(neighbors). Finally, I have a maximum label, which is the **majority of labels** in the array and it is stored in **prediction**.

Use the Xtrain.csv for test_data_file.

```
Prediction = truth ____________________________________________________________
 [ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True]
Prediction ____________________________________________________________________
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4.]
Truth _________________________________________________________________________
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4.]
accuracy ______________________________________________________________________
 1.0
```

When k = 1, accuracy = 100%

```
Prediction = truth ____________________________________________________________
 [ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True False  True  True  True  True  True False  True
  True False False False  True  True  True  True  True  True  True  True
  True False  True  True]
Prediction ____________________________________________________________________
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 1. 2. 2. 2. 3. 3. 2. 3.
 3. 2. 1. 1. 3. 3. 4. 4. 4. 4. 4. 4. 4. 1. 4. 4.]
Truth _________________________________________________________________________
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4.]
accuracy ______________________________________________________________________
 0.85
```

When k = 2, accuracy = 85%

```
Prediction = truth ____________________________________________________
 [ True False  True  True  True  True  True  True  True  True  True  True
  True False  True  True  True  True  True  True  True  True  True  True
  True False False  True  True  True  True  True  True  True  True  True
  True  True  True  True]
Prediction ____________________________________________________________
 [1. 2. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 1. 2. 2. 2. 2. 2. 2. 3. 3. 3. 3.
 3. 2. 1. 3. 3. 3. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4.]
Truth _________________________________________________________________
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4.]
accuracy ______________________________________________________________
 0.9
```

When k = 3, accuracy = 90%

```
Prediction = truth ____________________________________________________
 [ True  True  True  True  True  True  True  True  True  True  True  True
  True False  True  True False  True  True  True  True  True  True  True
  True False False False  True  True  True  True  True  True  True  True
  True False  True  True]
Prediction ____________________________________________________________
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 3. 2. 2. 1. 2. 2. 2. 3. 3. 3. 3.
 3. 2. 1. 1. 3. 3. 4. 4. 4. 4. 4. 4. 4. 1. 4. 4.]
Truth _________________________________________________________________
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4.]
accuracy ______________________________________________________________
 0.85
```

When k = 4, accuracy = 85%

```
Prediction = truth ____________________________________________________
 [ True  True  True  True  True  True False False  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True False False False  True  True  True  True False  True  True  True
  True False  True  True]
Prediction ____________________________________________________________
 [1. 1. 1. 1. 1. 1. 4. 4. 1. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 3. 3. 3. 3.
 3. 1. 1. 1. 3. 3. 4. 4. 1. 4. 4. 4. 4. 1. 4. 4.]
Truth _________________________________________________________________
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4.]
accuracy ______________________________________________________________
 0.825
```

When k = 5, accuracy = 83%

```
Prediction = truth
 [ True False  True  True  True False False False  True False  True  True
  True False  True  True  True  True  True  True  True  True  True  True
  True False False  True  True  True  True  True  True  True  True  True
  True False  True  True]
Prediction
 [1. 2. 1. 1. 1. 3. 4. 4. 1. 3. 2. 2. 2. 3. 2. 2. 2. 2. 2. 2. 3. 3. 3. 3.
 3. 1. 1. 3. 3. 3. 4. 4. 4. 4. 4. 4. 4. 1. 4. 4.]
Truth
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4.]
accuracy
 0.775
```

When k = 10, accuracy = 78%

```
Prediction = truth
 [ True False  True  True  True False False False  True False  True  True
  True False  True  True  True  True  True  True  True  True  True  True
  True False  True  True  True  True  True  True False  True  True  True
  True False  True  True]
Prediction
 [1. 2. 1. 1. 1. 3. 4. 4. 1. 2. 2. 2. 2. 3. 2. 2. 2. 2. 2. 2. 3. 3. 3. 3.
 3. 2. 3. 3. 3. 3. 4. 4. 1. 4. 4. 4. 4. 1. 4. 4.]
Truth
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4.]
accuracy
 0.775
```

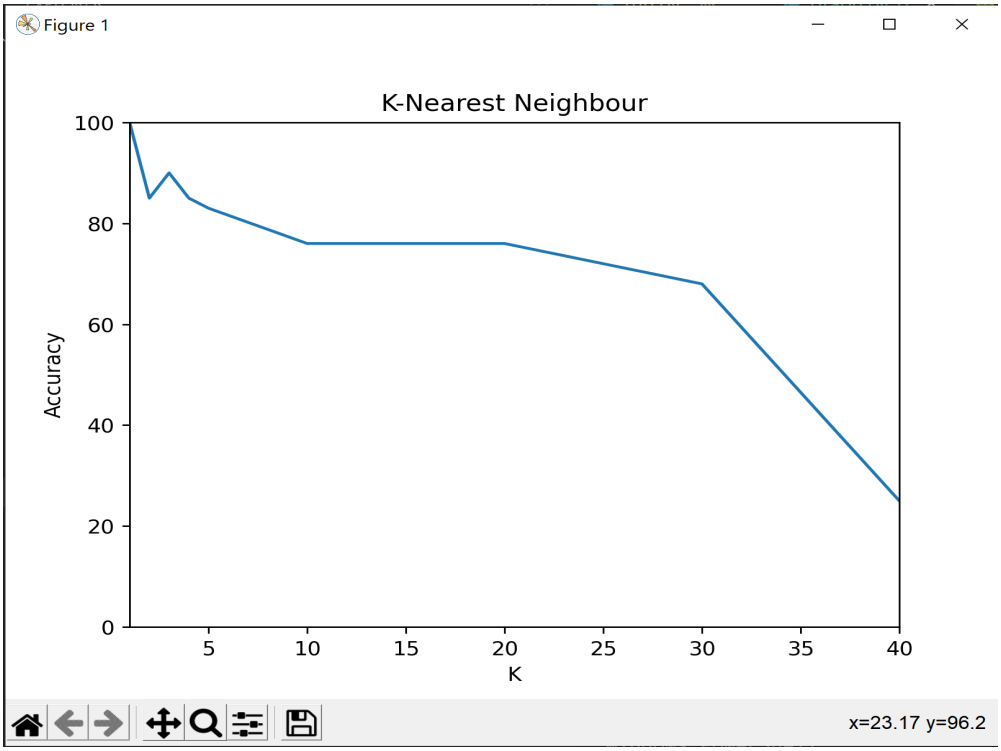
When k = 20, accuracy = 78%

```
Prediction = truth
 [ True  True  True  True  True  True  True  True  True False  True  True
  True False  True  True False  True  True  True  True False False False
  True False False  True  True  True  True  True False False  True False
 False False  True  True]
Prediction
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 3. 2. 2. 2. 3. 2. 2. 1. 2. 2. 2. 3. 2. 2. 2.
 3. 1. 1. 3. 3. 3. 4. 4. 1. 1. 4. 1. 1. 1. 4. 4.]
Truth
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4.]
accuracy
 0.675
```

When k = 30, accuracy = 68%

```
Prediction = truth ____________________________________________________________
 [ True  True  True  True  True  True  True  True  True  True False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False]
Prediction ____________________________________________________________________
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Truth _________________________________________________________________________
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 4. 4. 4. 4. 4. 4. 4. 4. 4. 4.]
accuracy ______________________________________________________________________
 0.25
```

When k = 40, accuracy = 25%



The graph,

X axis = k(whatever I set it to)

Y axis = accuracy(percentage of each k)

In conclusion, I observed that the KNN algorithm gives 100 percent accuracy when k =1. However, it is not a good prediction since it is very specific in that it only predicts based on the 1-th closest training data. And also k = 40 gives the lowest accuracy because it is too general that It gives an unreliable prediction. Therefore, it is very important to have appropriate k value to have good and reliable accuracy. That is actually why I used the k =3 in this problem and gave the highest accuracy besides k = 1. Also, I personally think that this is a very good algorithm for multiclass classification. Compared to the problem #1, it is more suitable when it needs to classify the data in more than two categories.