HW1
Spencer Carlson

Q1 - $\lambda = \frac{1}{N}\sum_{i}^{N} x_i$

Q2 - $\lambda = \frac{N+\alpha-1}{\Sigma_i x_i+\beta}$

Q3 -

Q4 - This could make distances farther between a node having a private work class and never worked. The max distance between any class would be one in the binary version but would have a max distance of two as an ordinal value.

Q5 - Only about 24.5% of entries make above $50k a year. I think it would be a poor model. You could get about 70% accuracy if you guessed that it was under 50k every time. In my implementation each point has 85 dimensions but thinking logically it's 12 dimensions

Q6 - Since the L2 norm sums the square of elements in an array and then takes the square root. We can subtract x from z to form d and then take the L2 norm of d to find the euclidian distance. These are equivalent mathematically but significantly faster utilizing the efficiency of the numpy operations

```
equal to using linalg.norm but much slower
def distance(x, xi):
    d = 0.00000000000
    temp = xi[1:-1]
    tempx = x[1:]
    c = tempx - temp
    numpy.power(c, 2)
    index = 0
    for i in numpy.nditer(c):
        d += i
    return numpy.sqrt(d)
```

Q7 -

```
#data loading funciton - loads csv data into matrix
def load_arr(file):
    data = numpy.genfromtxt(file, dtype = float, delimiter = ',',
skip_header=1)
```

```
    return data

#distance function - calculates euclidean distance between two 1D data
point arrays
def distance(x, xi):
    temp = xi[1:-1]
    if flag == 1:
        tempx = x[1:-1]
    else:
        tempx = x[1:]
    return numpy.linalg.norm(temp - tempx)

#classifier function - takes 1D array test and training data matrix
#returns class of test case based on given data
def classify(test, train):
    avg = 0.00000000000
    dist = list()
    #test against every case
    for case in train:
        dist.append(distance(test, case))
    dex = numpy.argsort(dist)
    #select k nearest neighbors
    global k
    if flag == 1 and k == 8000:
        k = 6000
    for i in range(0,k):
        avg += train[dex[i], -1]
    avg /= k
    ident = round(avg)
    return ident
```

Q8 -

```
#main k-fold function - takes training data and returns highest performing
k value
def kmain(train):
    print("Doing 4-fold cross validation")
    global flag
    flag = 1
    #test case loop
```

```
    output = open("kcross.txt", "w")
    perf = list()
    #loop over possible k values
    for hype in hp:
        global k
        k = hype
        output.write("K = ")
        output.write(str(k))
        output.write("\n")
        print("k = ", k)
        vmean = 0.0000000000
        var = list()
        #loop over subset valdation blocks
        for i in range(0,4):
            print("subset ", (i+1))
            output.write("Subset #")
            output.write(str(i+1))
            nacc = 0.00
            start = i * 2000
            end = (i+1) * 2000
            cases = train[start:end]
            if i == 0:
                tcases = train[end:]
            elif i == 3:
                tcases = train[:start]
            else:
                t1 = train[:start]
                t2 = train[end:]
                tcases = numpy.vstack((t1,t2))
            res = list()
            #test validation data against the rest of data and store
result
            for test in cases:
                res.append(classify(test, tcases))
            index = 0
            #calculate validation performance
            for r in res:
                if cases[index, -1] == r:
                    nacc += 1
            acc = nacc/2000.0
```

```
            vmean += acc
            var.append(acc)
            output.write(" Validation Accuracy: ")
            output.write(str(acc))
            vd = list()
            #test validation data against all of the data and store result
            for test in cases:
                vd.append(classify(test, train))
            nacc = 0.00
            #calculate training performance
            index = 0
            for r in vd:
                if cases[index, -1] == r:
                    nacc += 1
            acc = nacc/2000.0
            output.write(" Training Accuracy: ")
            output.write(str(acc))
            output.write("\n")
        #calculate varience and mean
        vmean /= 4
        v = numpy.var(var)
        output.write("Mean = ")
        output.write(str(vmean))
        output.write("  Varience")
        output.write(str(var))
        output.write("\n")
        perf.append(vmean)
        print("mean", vmean)
        print("var", var)
    #determine highest performing k
    output.close()
    dex = numpy.argsort(perf)
    return hp[dex[-1]]
```

Q9 - A K value of 9 seemed to perform the best overall. When K was 0 the training error was very near 0% but not quite. I am unsure what caused this. The training accuracy went down as k increased. This is because when k = 1 every data point has its own bubble between itself and

points of another class. The validation accuracy increased and then fell after after k =99. I did not expect the performance for this to peak for k = 99 based on my kaggle results with different k values. When k was close to 1 it was overfit and when k was 8000 it was undefit

K = 1
Subset #1 Validation Accuracy: 0.7865 Training Accuracy: 0.99
Subset #2 Validation Accuracy: 0.795 Training Accuracy: 0.9885
Subset #3 Validation Accuracy: 0.7845 Training Accuracy: 0.988
Subset #4 Validation Accuracy: 0.7815 Training Accuracy: 0.9805
Mean = 0.786875 Varience2.5171875000000268e-05
K = 3
Subset #1 Validation Accuracy: 0.8105 Training Accuracy: 0.886
Subset #2 Validation Accuracy: 0.805 Training Accuracy: 0.9025
Subset #3 Validation Accuracy: 0.7975 Training Accuracy: 0.885
Subset #4 Validation Accuracy: 0.8025 Training Accuracy: 0.8855
Mean = 0.8038749999999999 Varience2.192187500000007e-05
K = 5
Subset #1 Validation Accuracy: 0.817 Training Accuracy: 0.8735
Subset #2 Validation Accuracy: 0.8215 Training Accuracy: 0.8785
Subset #3 Validation Accuracy: 0.803 Training Accuracy: 0.863
Subset #4 Validation Accuracy: 0.8175 Training Accuracy: 0.868
Mean = 0.81475 Varience4.90624999999997e-05
K = 7
Subset #1 Validation Accuracy: 0.8195 Training Accuracy: 0.8685
Subset #2 Validation Accuracy: 0.8235 Training Accuracy: 0.8685
Subset #3 Validation Accuracy: 0.8065 Training Accuracy: 0.8485
Subset #4 Validation Accuracy: 0.811 Training Accuracy: 0.8655
Mean = 0.815125 Varience4.5171874999999965e-05
K = 9
Subset #1 Validation Accuracy: 0.826 Training Accuracy: 0.858
Subset #2 Validation Accuracy: 0.823 Training Accuracy: 0.8605
Subset #3 Validation Accuracy: 0.8095 Training Accuracy: 0.844
Subset #4 Validation Accuracy: 0.814 Training Accuracy: 0.8565
Mean = 0.818125 Varience4.4296874999999835e-05
K = 99
Subset #1 Validation Accuracy: 0.8325 Training Accuracy: 0.8375
Subset #2 Validation Accuracy: 0.8255 Training Accuracy: 0.8345
Subset #3 Validation Accuracy: 0.8175 Training Accuracy: 0.824
Subset #4 Validation Accuracy: 0.8295 Training Accuracy: 0.8335
Mean = 0.8262499999999999 Varience3.168750000000006e-05
K = 999
Subset #1 Validation Accuracy: 0.815 Training Accuracy: 0.8185
Subset #2 Validation Accuracy: 0.827 Training Accuracy: 0.83
Subset #3 Validation Accuracy: 0.81 Training Accuracy: 0.817

Subset #4 Validation Accuracy: 0.8265 Training Accuracy: 0.8265
Mean = 0.819625 Varience5.392187499999975e-05
K = 8000
Subset #1 Validation Accuracy: 0.761 Training Accuracy: 0.761
Subset #2 Validation Accuracy: 0.7555 Training Accuracy: 0.7555
Subset #3 Validation Accuracy: 0.7455 Training Accuracy: 0.7455
Subset #4 Validation Accuracy: 0.7545 Training Accuracy: 0.7545
Mean = 0.7541249999999999 Varience3.092187499999976e-05

Q10 - I used a k value of 8 and got an accuracy of 81.818%. I tried to implement a weight function but it didnt go very well and got better results without it.

Debrief -

1. About 12 hours but I spent a lot of time sitting around waiting for it to finish because the k-fold takes a long time
2. Moderately easy
3. I worked fully alone
4. 90% (10% of the math I don't understand in the first 3 questions)
5. With such large datasets some instruction on ways to efficiently compute them would be cool. Just a quick tip would be enough