CSCE 633 - Machine Learning HW1

Robert Quan
Department of Computer Science, Texas A&M University, College Station, Texas 77843-4242, USA
(Dated: September 28, 2017)

I. ABSTRACT

In this homework, I derive some mathematical relationships for our linear regression weights and Taylor expand the cost function for our Gradient Descent. I use two different algorithms to analyze a forest fire dataset. those algorithms are KNN and Linear Regression.

QUESTION I
1-dimensional linear regression:
Starting with our equation for the Residual Sum of Squares,

$$RSS(w_0, w_1) = \sum_{n=1}^{N} (y_n - w_0 - w_1 x_n)^2 \tag{1}$$

We take the partial derivatives with respect to w_0 and w_1. We obtain the following:

$$\frac{\partial RSS}{\partial w_0} = -2 \sum_{n=1}^{N} (y_n - w_0 - w_1 x_n), \tag{2a}$$

$$\frac{\partial RSS}{\partial w_1} = -2 \sum_{n=1}^{N} (y_n - w_0 - w_1 x_n)(x_n), \tag{2b}$$

By minimizing the RSS (setting both partials equal to 0), and solving for the respective weights, we obtain (a):

$$\sum_{n=1}^{N} y_n - N{w_0}^* - \sum_{n=1}^{N} w_1 x_n = 0 \tag{3}$$

Solving for ${w_0}^*$ we have:

$$\boxed{{w_0}^* = (\frac{1}{N} \sum_{n=1}^{N} y_n) - w_1 (\frac{1}{N} \sum_{n=1}^{N} x_n)} \tag{4}$$

We know that the average of any set of measurements is just the sum of each component divided by the number of components. $\bar{x} = \frac{1}{N} \sum_{n=1}^{N} x_n$ We can then rewrite the minimized ${w_0}^*$ as (b):

$$\boxed{{w_0}^* = \bar{y} - w_1 \bar{x}} \tag{5}$$

For the ${w_1}^*$ minimization, we use the previous result to replace the ${w_0}^*$ variable in our equation

$$\sum_{n=1}^{N} y_n x_n - \sum_{n=1}^{N} {w_0}^* x_n - \sum_{n=1}^{N} {w_1}^* {x_n}^2 = 0, \tag{6a}$$

$$\sum_{n=1}^{N} y_n x_n - \sum_{n=1}^{N} ((\frac{1}{N} \sum_{n=1}^{N} y_n) - \tag{6b}$$

$$w_1 (\frac{1}{N} \sum_{n=1}^{N} x_n)) x_n - \sum_{n=1}^{N} {w_1}^* {x_n}^2 = 0, \tag{6c}$$

$$\sum_{n=1}^{N} y_n x_n - N((\frac{1}{N} \sum_{n=1}^{N} y_n)(\frac{1}{N} \sum_{n=1}^{N} x_n) - \tag{6d}$$

$$w_1 (\frac{1}{N} \sum_{n=1}^{N} x_n)) x_n - \sum_{n=1}^{N} {w_1}^* {x_n}^2 = 0, \tag{6e}$$

$$\sum_{n=1}^{N} y_n x_n - N(\frac{1}{N} \sum_{n=1}^{N} y_n)(\frac{1}{N} \sum_{n=1}^{N} x_n) = \tag{6f}$$

$$w_1 (N(\frac{1}{N} \sum_{n=1}^{N} x_n) x_n - \sum_{n=1}^{N} {x_n}^2), \tag{6g}$$

$$\tag{6h}$$

Now we can solve for ${w_1}^*$ and obtain (a):

$$\boxed{{w_1}^* = \frac{\sum_{n=1}^{N} y_n x_n - N(\frac{1}{N} \sum_{n=1}^{N} y_n)(\frac{1}{N} \sum_{n=1}^{N} x_n)}{N(\frac{1}{N} \sum_{n=1}^{N} x_n) x_n + \sum_{n=1}^{N} {x_n}^2}} \tag{7}$$

we are able to rewrite this equation to a more compact form by using the average notation we used for w_0 (b).

$${w_1}^* = \frac{\sum_{n=1}^{N} y_n x_n - N\bar{x}\bar{y}}{\sum_{n=1}^{N} {x_n}^2 - N\bar{x}^2} \tag{8a}$$

$${w_1}^* = \frac{\sum_{n=1}^{N} y_n x_n - N\bar{x}\bar{y} - N\bar{x}\bar{y} + N\bar{x}\bar{y}}{\sum_{n=1}^{N} {x_n}^2 + N\bar{x}^2 - N\bar{x}^2 + N\bar{x}^2} \tag{8b}$$

$${w_1}^* = \frac{\sum y_n x_n - \sum x_n \bar{y} - \bar{x} \sum y_n + \sum \bar{x}\bar{y}}{\sum_{n=1}^{N} {x_n}^2 - \sum x_n \bar{x} - \sum x_n \bar{x} + \sum \bar{x}^2} \tag{8c}$$

$$\tag{8d}$$

At 8c, we see that this can be reversed FOILd into their respective equations:

$$\boxed{{w_1}^* = \frac{\sum_{n=1}^{N} (x_n - \bar{x})(y_n - \bar{y})}{\sum_{n=1}^{N} (x_n - \bar{x})^2}} \tag{9}$$

We note that the the $\bar{x}$ and $\bar{y}$ are the sample means of our measurements. The Numerator in the $w_1{}^*$ is the Co-variance that is not normalized between the x and y values. The denominator is the variance of x. Var(x). This equation is almost similar to the correlation between the set x and y but it is missing the variance of y in the denominator.

QUESTION II

To write the target function J(**w**) as a Taylor expansion, we simply note the following: Let

$$\boldsymbol{x_0} = \boldsymbol{w(k)} \tag{10a}$$
$$\boldsymbol{x} = \boldsymbol{w} \tag{10b}$$
$$f(\boldsymbol{x_0}) = J(\boldsymbol{w}(k)) \tag{10c}$$
$$\nabla^2 J(\boldsymbol{w}) = \boldsymbol{H} \tag{10d}$$

Now we can expand our minimized target function around **w**(k) to obtain (a):

$$\boxed{J(\boldsymbol{w}) \approx J(\boldsymbol{w}(k)) + (\nabla J|_{w=w(k)})^T \cdot (\boldsymbol{w} - \boldsymbol{w}(k))} \tag{11a}$$
$$\boxed{+\frac{1}{2}((\boldsymbol{w} - \boldsymbol{w}(k))^T \cdot \boldsymbol{H}_{J|_{w-w(k)}} \cdot (\boldsymbol{w} - \boldsymbol{w}(k))} \tag{11b}$$

For part (b), we can use the equation:

$$\boldsymbol{w}(k+1) = \boldsymbol{w} - \alpha(k) \cdot \nabla J|_{w=w(k)} \tag{12a}$$
$$\tag{12b}$$

By substituting this equation in the above Taylor expansion, we see that the w(k) variables will cancel out and we are left with:

$$J(\boldsymbol{w}(k+1)) \approx J(\boldsymbol{w}(k)) - \tag{13a}$$
$$(\nabla J|_{w=w(k)})^T \cdot (\boldsymbol{w}(k) - \alpha(k) \cdot \nabla J|_{w=w(k)} - \boldsymbol{w}(k)) \tag{13b}$$
$$+\frac{1}{2}((\boldsymbol{w}(k) - \alpha(k) \cdot \nabla J|_{w=w(k)} - \boldsymbol{w}(k))^T \cdot \tag{13c}$$
$$\boldsymbol{H}_{J|_{w-w(k)}} \cdot (\boldsymbol{w}(k) - \alpha(k) \cdot \nabla J|_{w=w(k)} - \boldsymbol{w}(k))) \tag{13d}$$

This reduces down to (b):

$$\boxed{J(\boldsymbol{w}(k+1)) \approx J(\boldsymbol{w}(k)) - ||\nabla J|_{w=w(k)})||_2^2 \cdot \alpha(k)} \tag{14a}$$
$$\boxed{+\frac{1}{2}(\nabla J|_{w=w(k)})^T \cdot \boldsymbol{H}_{J|_{w-w(k)}} \cdot (\nabla J|_{w=w(k)}) \cdot \alpha^2(k)} \tag{14b}$$

To minimize the above equation we simply take the derivate with respect to α(k) and solve for α(k).

$$\frac{\partial J(\boldsymbol{w}(k+1))}{\partial \alpha(k)} = 0 - ||\nabla J|_{w=w(k)})||_2^2 + \tag{15a}$$
$$(\nabla J|_{w=w(k)})^T \cdot \boldsymbol{H}_{J|_{w-w(k)}} \cdot (\nabla J|_{w=w(k)}) \cdot \alpha(k) = 0 \tag{15b}$$

Now solving for α(k) we obtain the solution for part (c):

$$\boxed{\alpha(k) = \frac{||\nabla J|_{w=w(k)})||_2^2}{(\nabla J|_{w=w(k)})^T \cdot \boldsymbol{H}_{J|_{w-w(k)}} \cdot (\nabla J|_{w=w(k)})}} \tag{16a}$$

This expression gives us a closed-form solution of the step size at iteration k that minimizes the target function at the next iteration.

Since we are multiplying matrices, the time complexity is:

$$\boxed{O(n^3)} \tag{17}$$

QUESTION III part 1

Predicting Forest Fires
KNN

a) We begin by inspecting the input features and finding some interesting correlations.

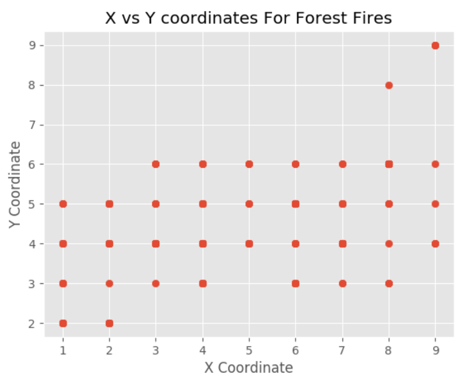

FIG. 1: Simple X vs Y scatter Plot

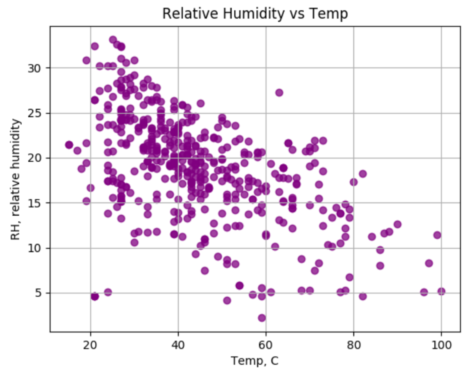

FIG. 2: RH vs. Temperature Scatter

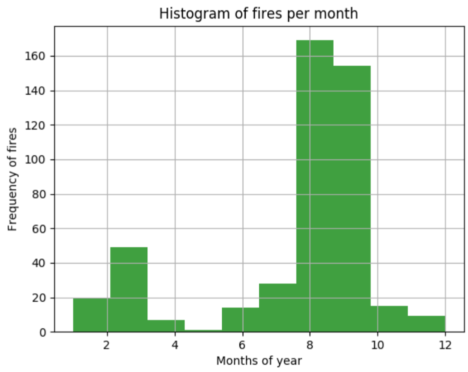

FIG. 3: Historical fires per month

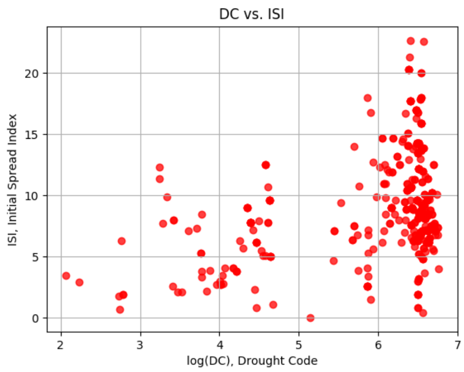

FIG. 4: DC vs ISI

Its interesting to note the downward correlation between the Relative Humidity and Temperature in FIG 2.

We note that the **categorical** and the **continuous** values of our data set are:
Categorical: X, Y, month, day
Continuous: FFMC, DMC, DC, ISI, temp, RH, wind, rain, area

The KNN implementation works by comparing the number of K neighbors of our predicted set. By classifying my training set based on whether or not the forest was burned for that reading (**Class 1** for burned and **Class 0** for not burned). This was simply done by choosing whether or not the area column has a non zero value. For the testing set, I saved the actual class categories from the set and then tested the other values to predict whether or not my algorithm could show that the features would have a fire or not. By simply calculating the distance of each entries feature from the training features entries, i obtained the K nearest neighbors and then used a voting count to predict using a odd number of neighbors whether or not the testing point belonged to the majority class. This implementation showed promising results and after varying the number of neighbors and cross-validation, I found that there is an interesting trend in the accuracy correct predictions using the KNN algorithm. The graph of the accuracy per number of k nearest neighbors is below:

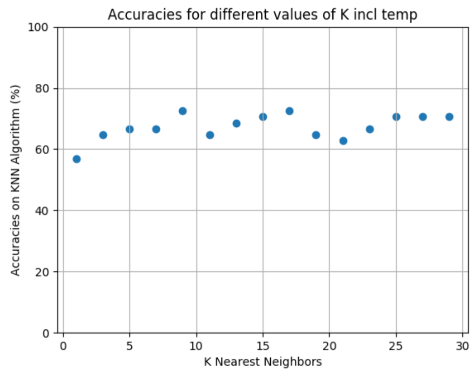

FIG. 5: K vs. KNN accuracy

The highest accuracy that my KNN implementation could reach was 76% at K=7 neighbors.

After they analysis of the accuracies per nearest neighbor, I decided to revisit the code that I had written and better the cross validation input method that I had written. After splitting my data into 3 separate testing and training arrays, I discovered the following trend in my accuracies when running the algorithm over different values of K. (Fig 6)

I have included the new code below.

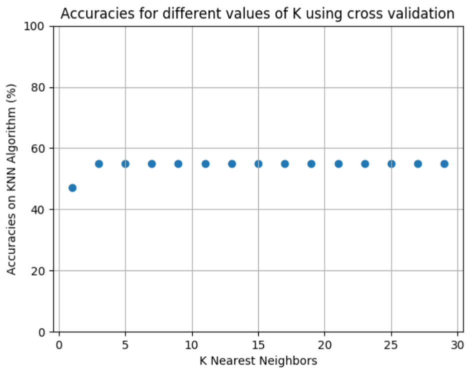

FIG. 6: K vs. KNN accuracy using different cross validation

Question III part 2

Linear Regression

The plot of the area and the log area are below:

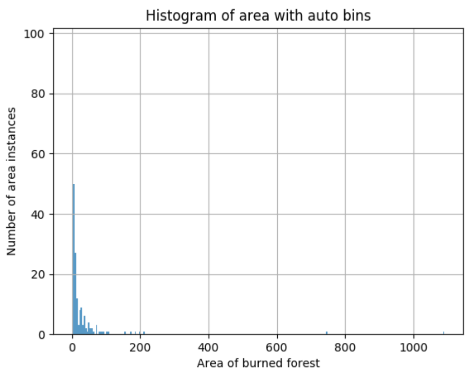

FIG. 7: Histogram of Area

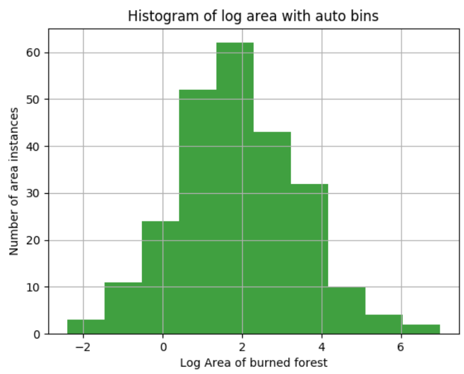

FIG. 8: Histogram of Log(Area)

When plotting in log scale, we can see that the histogram resembles a normal distribution. Fig 6 and Fig 7 respectively. How interesting.

In the Linear Regression model, we have our input vector and output vector and we use these two to fit a linear function with the use of a weight vector to store the parameters of the fit. After computing the Ordinary Least Squares matrix w^*, we use this vector to predict the new values on our testing set data. I have defined function for both operations in the python code below.

$$RSS(w) = \sum_{n=1}^{N} (y_i - w^T x_i)^2 \approx f(\mathbf{w}|x) \tag{18}$$

To evaluate how well the method work for predicting the outcomes of the testing set, we calculate the Residual Sum of Squares errors. I wrote a function below that will allow me to perform the calculation. The RSS value that I obtain using a linear model, is 103761.511 which is surprisingly big.

Messing around with non-linear models

$$RSS(w) = \sum_{n=1}^{N} (y_i - w^T x_i^2)^2 \approx f(\mathbf{w}|\phi(x)) \tag{19}$$

When I train my model using the input features **X** squared (X^TX), I obtain a lower RSS value of RSS = 101493.88, which is lower than the RSS value for the linear model. When I square it again, my RSS value increases again to 103506.23.

```
#!/usr/bin/python ============Linear Regression ========== Robert Quan
import pandas as pd
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
def import_data(dir):
#Import the data set using a pandas frame with condition that area>0
      data = pd.read_csv(dir)
      data=data[data['area'] > 0]
      return data
#x is the features matrix, y is the output vector, w is weight
def OLS(x, y):
      w = np.dot(np.linalg.inv(np.dot(np.matrix.transpose(x), x)), 
      np.dot(np.matrix.transpose(x),y))
      return w
def RSS(x_test,w,y_test):
      RSS = np.dot( (y_test - np.dot(x_test, w) ), (y_test - np.dot(x_test, 
      w)))
      return RSS
def Nonlin(x):
print "You are running the Non-linear training"
      return np.square(x)
#---------------------Run main function
#import into np arrays
training = import_data('train.csv')
testing = import_data('test.csv')
#get output matrix
y_train = np.array(training['area'])
y_test = np.array(testing['area'])
#get the features matrix
del training['area'], testing['area']
norm = training.apply(lambda x: np.sqrt(x**2).sum()/x.shape[0])
training /= norm
testing /= norm
x_train = np.array(training)
x_test = np.array(testing)
# ---------------------------#To make a non-linear model
x_train = Nonlin(x_train)
#our trained weights from the training data
w = OLS(x_train, y_train)
#Lets get our RSS!
print "The value of RSS is: ", RSS(x_test, w, y_test)
```

```python
#!/usr/bin/python=========== KNN Classification ============ Robert Quan
import pandas as pd
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
import math
#----------------testing point, data locations, training data, k
def knearestneighbor(p, x, v, k):
      assert(len(v) > k)
      x = x.astype(float)
      p = np.array(p).astype(float)
      norm = np.linalg.norm(x, axis=0)
      x[:] /= norm
      p[:] /= norm
      distance = np.linalg.norm(p[:] - x[:], ord=2, axis=1)
      voters = v.iloc[np.argsort(distance)[:k]]
      return(voters['class'].mode())
def concatrain(str):
      return np.expand_dims(training[str].values, 0)
def concatest(str):
      return np.expand_dims(testing[str].values, 0)
#---------------------import data function
def import_data(dir):
      data = pd.read_csv(dir)
      data['class'] = (data['area'] > 0).astype(int)
      return data
#--------------------Main
print("Enter how many neighbors should we test with (choose an odd k value) :")
k = input()
#---------------------Import the training set
training = import_data('train.csv')
location = np.transpose(np.concatenate([concatrain('X'), 
concatrain('Y'),concatrain('month'),
concatrain('day'), 
concatrain('FFMC'),concatrain('DMC'),concatrain('DC'),concatrain('ISI'),concatr
ain('temp'),
concatrain('RH'),concatrain('wind'), concatrain('rain')]))
#--------------------Write the function for the knearestneighbor
nearest = lambda x: knearestneighbor(x, location, training, k)
```

```
#---------------------Import the testing set
testing = import_data('test.csv')
testing_set = np.transpose(np.concatenate([concatest('X'),
concatest('Y'),concatest('month'),
concatest('day'),
concatest('FFMC'),concatest('DMC'),concatest('DC'),concatest('ISI'),concatest('
temp'),
concatest('RH'),concatest('wind'), concatest('rain')]))
#--------------------Lets use the KNN on our testing set
knn_values = [np.asscalar(nearest(test)) for test in testing_set]
real_test = testing['class']
#---------------------Test accuracies
correct = (np.array(real_test) == np.array(knn_values)).astype(int)
print "Correct predictions", np.sum(correct)
print "Accuracy ", np.mean(correct)*100
```

```
#!/usr/bin/python=================KNN with Cross Validation===== Robert Quan
import pandas as pd
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
import math
#----------------testing point, data locations, training data, k
def knearestneighbor(p, x, v, k):
      assert(len(v) > k)
      x = x.astype(float)
      p = np.array(p).astype(float)
      norm = np.linalg.norm(x, axis=0)
      x[:] /= norm
      p[:] /= norm
      distance = np.linalg.norm(p[:] - x[:], ord=2, axis=1)
      voters = v.iloc[np.argsort(distance)[:k]]
      return(voters['class'].mode())
def concatrain(str):
      return np.expand_dims(training[str].values, 0)
def concatest(str):
      return np.expand_dims(testing[str].values, 0)
#---------------------import data function
def import_data(dir):
      data = pd.read_csv(dir)
      data['class'] = (data['area'] > 0).astype(int)
      return data
#--------------------Main
accuracies = []
kvals = []
best = []
for k in range(0,15):
      #---------------------Import the training set
      print "Starting with K =", 2*k +1
      training = import_data('train.csv')
      location = np.transpose(np.concatenate([concatrain('X'),
      concatrain('Y'),concatrain('month'),
      concatrain('day'),
      concatrain('FFMC'),concatrain('DMC'),concatrain('DC'),concatrain('ISI'),
      concatrain('temp'),
      concatrain('RH'),concatrain('wind'), concatrain('rain')]))
      t1,t2,t3,t4 = location[:150], location[150:250], location[250:350],
      location[350:]
      y1,y2,y3,y4 =
      training['class'],training['class'],training['class'],training['class']
      #-------------------Write the function for the knearestneighbor
      nearest1 = lambda x: knearestneighbor(x, t1, training, 2*k+1)
      nearest2 = lambda x: knearestneighbor(x, t2, training, 2*k+1)
      nearest3 = lambda x: knearestneighbor(x, t3, training, 2*k+1)
      nearest4 = lambda x: knearestneighbor(x, t4, training, 2*k+1)
      #---------------------Import the testing set
      testing = import_data('test.csv')
      testing_set = np.transpose(np.concatenate([concatest('X'),
      concatest('Y'),concatest('month'),
      concatest('day'),
      concatest('FFMC'),concatest('DMC'),concatest('DC'),concatest('ISI'),
```

```
      concatest('temp'), concatest('RH'),concatest('wind'),
      concatest('rain')]))
      #---------------------Lets use the KNN on our testing set
      knn_values1 = [np.asscalar(nearest1(test)) for test in testing_set]
      knn_values2 = [np.asscalar(nearest2(test)) for test in testing_set]
      knn_values3 = [np.asscalar(nearest3(test)) for test in testing_set]
      knn_values4 = [np.asscalar(nearest4(test)) for test in testing_set]
      real_test = testing['class']
      #----------------------Cross validation
      correct = (np.array(real_test) == np.array(knn_values1)).astype(int)
      print "Accuracy for set 1", np.mean(correct)*100
      accuracies.append(np.mean(correct)*100)

      correct = (np.array(real_test) == np.array(knn_values2)).astype(int)
      print "Accuracy for set 2", np.mean(correct)*100
      accuracies.append(np.mean(correct)*100)

      correct = (np.array(real_test) == np.array(knn_values3)).astype(int)
      print "Accuracy for set 3", np.mean(correct)*100
      accuracies.append(np.mean(correct)*100)

      correct = (np.array(real_test) == np.array(knn_values4)).astype(int)
      print "Accuracy for set 4", np.mean(correct)*100
      accuracies.append(np.mean(correct)*100)
      kvals.append(2*k+1)
      best.append(max(accuracies))
      print
plt.scatter(kvals,best,s=None)
plt.title("Accuracies for different values of K incl temp")
plt.ylim(0,100)
plt.xlabel("K Nearest Neighbors")
plt.ylabel("Accuracies on KNN Algorithm (%)")
plt.grid(True)
plt.show()
```