CSCE 633 Homework #1

Jacob Fenger

September 27, 2017

Problem #1

a)

To minimize the RSS, we must compute the partial derivatives of the RSS function with respect to w_0 and w_1 respectively. This results in the following two equations:

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

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

We can then solve the first equation for w_0. This is easily done by factoring in the -1 and then splitting the summation up.

$$\sum_{n=1}^{N} y_n = w_1 \sum_{n=1}^{N} x_n + N w_0$$

$$w_0 = \frac{1}{N} \sum_{n=1}^{N} y_n - w_1 \frac{1}{N} \sum_{n=1}^{N} x_n$$

Similarily, we can do the same for w_1:

$$\sum_{n=1}^{N} x_n y_n = w_1 \sum_{n=1}^{N} x_n^2 + w_0 \sum_{n=1}^{N} x_n$$

We then solve for w_1 by plugging in what we found for w_0.

$$w_1 = \frac{\sum_{n=1}^{N} x_n y_n - N(\frac{1}{N}\sum_{n=1}^{N} x_n)(\frac{1}{N}\sum_{n=1}^{N} y_n)}{\sum_{n=1}^{N} x_n^2 - N(\frac{1}{N}\sum_{n=1}^{N} x_n)^2}$$

b)

We can use our previous solution for w_0 to replace all the $\sum_{n=1}^{N} x_n$ with $\bar{x}$ and similarily for y_0.

$$w_0 = \bar{y} - w_1\bar{x}$$

Solving for w_1 requires a trick or two in order to match the expression defined in the homework. We first can do all the replacements like we did for w_0:

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

Through a few tricks of adding and subtracting certain terms that is a bit tedious, we should end up with:

$$w_1 = \frac{\sum_{n=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{n=1}^{N}(x_i - \bar{x})^2}$$

c)

Both $\bar{x}$ and $\bar{y}$ represent the sample means for both populations. Additionally, $\sum_{n=1}^{N}(x_n - \bar{x})^2$ represent the total sample variation in sample x. The same can be said for y.

Problem #2

a)

Let ∇J be the gradient vector and H_J the Hessian Matrix of J evaluated at $w(k)$.

We can write the target function using the Taylor series expansion around $w(k)$ as the following:

$$J(w) \approx J(w(k)) + (\nabla J|_{w=w(k)})^T \cdot (w - w(k)) + \frac{1}{2}(w - w(k))^T \cdot H_J|_{w=w(k)} \cdot (w - w(k))$$

b)

To solve this problem, we simply need to plug in $w(k) - \alpha(k) \cdot \nabla J|_{w=w(k)}$ for w in the equation from a).

$$J(w(k+1)) \approx J(w(k)) + (\nabla J|_{w=w(k)})^T \cdot (-\alpha(k) \cdot \nabla J|_{w=w(k)}) + \frac{1}{2}(-\alpha(k) \cdot J|_{w=w(k)})^T \cdot H_J|_{w=w(k)} \cdot (-\alpha(k) \cdot J|_{w=w(k)})$$

$$J(w(k+1)) \approx J(w(k)) - \alpha(k) \cdot \parallel \nabla J|_{w=w(k)} \parallel_2^2 + \frac{1}{2}\alpha(k)^2 \cdot (\nabla J|_{w=w(k)})^T H_J|_{w=w(k)} (\nabla J|_{w=w(k)})$$

c)

For this problem, we can set $J(w) = 0$ and take the partial derivative with respect to $\alpha(k)$:

$$\frac{\partial J}{\partial \alpha(k)} = - \parallel \nabla J|_{w=w(k)} \parallel_2^2 + \nabla J|_{w=w(k)})^T H_J|_{w=w(k)\nabla J|_{w=w(k)}} \alpha(k)$$

Then solve for $\alpha(k)$:

$$\alpha(k) = \frac{\parallel \nabla J|_{w=w(k)} \parallel_2^2}{(\nabla J|_{w=w(k)})^T H_J|_{w=w(k)} (\nabla J|_{w=w(k)})}$$

d)

Given the above expression, we must compute the gradient each iteration. Additionally we have to compute the second derivatives as well given we utilize the Hessian matrix. Overall, I would argue that the runtime $\Theta(n^2)$ for computational cost.

Problem #3

a)

Below, you will find figures with explanations of what the graph consists of and what it may be able to tell us.

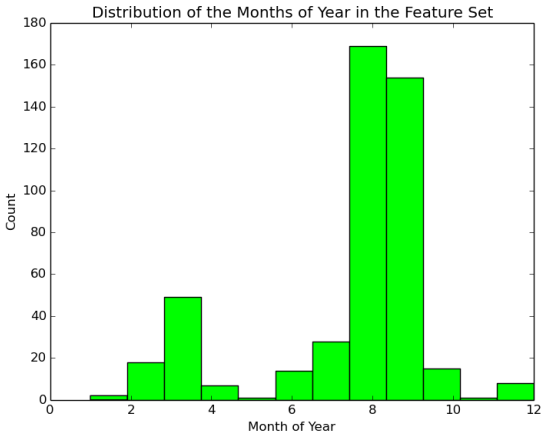

Figure 1: Looking at the distributions of the months, a categorial variable.

Categorical variables for this data set include: X, Y, month, and day. Continuous features include: FFMC, DMC, DC, ISI, temp, RH, wind, rain, and area.

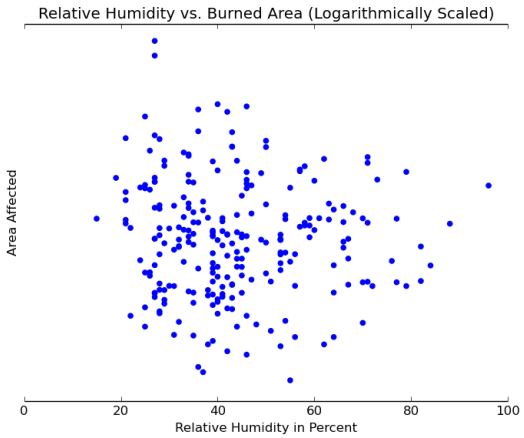

Figure 2: A comparison of humidity in percent and the log of burned area.

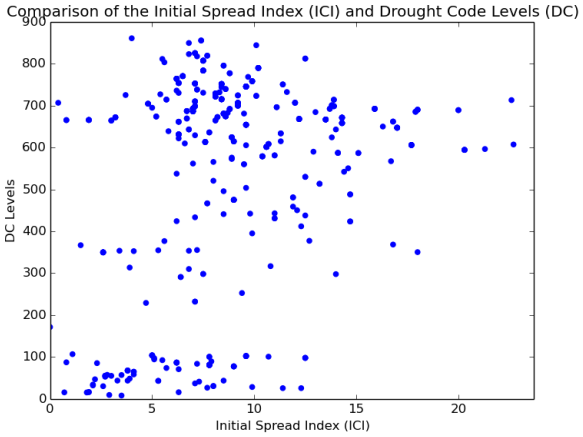

Figure 3: Comparison of the ICI and droughtcode. Not much of a correlation is visible.

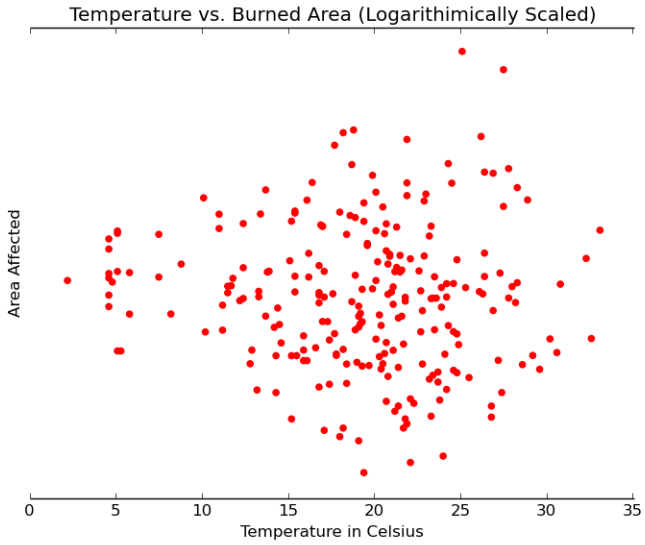

Figure 4: A comparison of temperature and the corresponding burned area (Log scaled)

b)

See code for dichotomization.

b.i)

See code.

b.ii)

The best K values found using cross validation were: 3, 5, and 7. These had the lowest errors. I utilized 10-fold cross validation to compute these values.

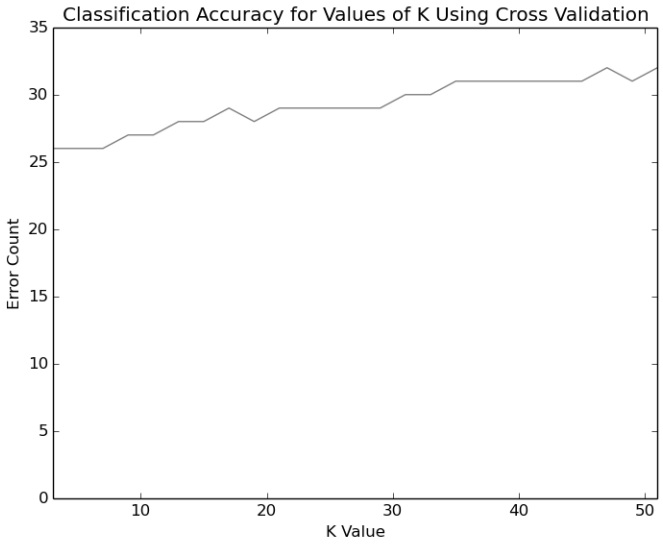

Figure 5: Number of errors corresponding to the K values when doing 10-fold cross validation.

b.iii

For the test set, I used a value of $K = 7$ since that was one of the better K values found when doing cross validation. **The accuracy ended up being:** 0.67%.

c)

c.i)

The histograms of both the burned area and the logarithm of the burned are shown below: Notice that the log(area) graph depicts a normal distribution instead of being skewed with a right tail in the first histogram.

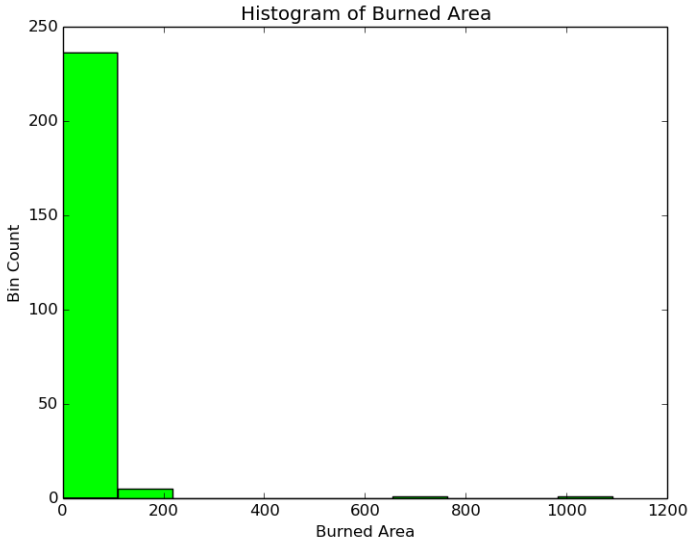

Figure 6: Histogram of the burned area.

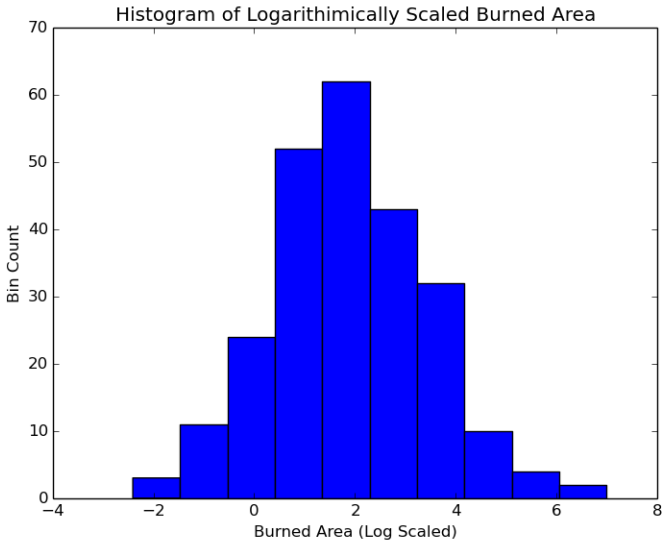

Figure 7: Histogram of the logarithm of the burned area.

c.ii)

See code.

c.iii)

The computed residual sum of squares error (RSS) was: **103808.628527**.

A correlation coefficient of **0.1652** was found. This means that there was a very weak positive correlation. In other words, I am not confident that this method of prediction is very accurate.

c.iv)

Taking the square of the input feature matrix yieled similar results with an RSS of **103023.35** and a correlation coffecient of **0.1648**.

Code

Main:

```
###################################
# Main file for CS 633
# Jacob Fenger
###################################

import matplotlib.pyplot as plt
import csv
import numpy as np
import inspection
import KNN

def main():
    train_ftrs, train_outcome = inspection.read_data('train.csv')
    test_ftrs, test_outcome = inspection.read_data('test.csv')

    # Dichotomize outcomes to be 0 or 1
    train_outcome = inspection.dichotomize_outcome(train_outcome)
    test_outcome = inspection.dichotomize_outcome(test_outcome)

    # Normalize features to be between 0 and 1
    train_ftrs = inspection.normalize_features(train_ftrs, 0, 1)
    test_ftrs = inspection.normalize_features(test_ftrs, 0, 1)

    # Use cross validation to find the best K, and then run that k value with
    # the test set
    #k = KNN.find_best_K(train_ftrs, train_outcome)
    v = KNN.test_classifier(train_ftrs, train_outcome, test_ftrs, test_outcome, 7)

    print "OVERALL ACCURACY FOR TEST SET: ", float(v)/len(test_outcome)
```

```
if (__name__) == '__main__':
    main()
```

Inspection of Data:

```
import csv
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing

def read_data(input_file):
    features = []
    outcome = []

    with open(input_file, 'rb') as csvfile:
        reader = csv.reader(csvfile)
        descriptions = reader.next()
        for row in reader:

            row = map(float, row)

            features.append(row[:12])
            outcome.append(row[12])

    return np.asarray(features), np.asarray(outcome)

def dichotomize_outcome(outcomes):
    for i in range(len(outcomes)):
        if outcomes[i] > 0:
            outcomes[i] = 1

    return outcomes

def normalize_features(features, min_range, max_range):

    # Used scikitlearn normalize
    min_max_scaler = preprocessing.MinMaxScaler()
    return min_max_scaler.fit_transform(features)

    # feature_min = features.min(axis=0)
    # feature_max = features.max(axis=0)
    #
    # return (features - feature_min) / (feature_max - feature_min)

# outcome corresponds to area
```

```
def explore_inputs(features, outcome):

    # We are not graphing the outcomes of 0 area affected
    outcome[outcome == 0] = np.nan

    #Scatterplot of the month of year vs. the burned area
    mnth = plt.hist(features[:,2], 12, histtype='bar', color="lime")
    plt.gcf().set_facecolor('white')
    plt.title("Distribution of the Months of Year in the Feature Set")
    plt.ylabel("Count")
    plt.xlabel("Month of Year")
    #plt.xlim([1, 13])
    #plt.ylim([0, max(outcome)+(max(outcome)/10)])

    plt.show()

    temp = plt.scatter(features[:,8], outcome, color="red")
    plt.gcf().set_facecolor('white')
    # m, b = np.polyfit(features[:, 8], outcome, 1)
    # plt.plot(outcome, m*outcome + b, '-')
    plt.title("Temperature vs. Burned Area (Logarithimically Scaled)")
    plt.ylabel("Area Affected")
    plt.xlabel("Temperature in Celsius")
    plt.ylim([0, 1800])
    plt.xlim([0, max(features[:,8])+2])
    plt.yscale('log')
    plt.show()

    temp = plt.scatter(features[:,9], outcome, color='blue')
    plt.gcf().set_facecolor('white')
    # m, b = np.polyfit(features[:, 9], outcome, 1)
    # plt.plot(outcome, m*outcome + b, '-')
    plt.title("Relative Humidity vs. Burned Area (Logarithmically Scaled)")
    plt.ylabel("Area Affected")
    plt.xlabel("Relative Humidity in Percent")
    plt.xlim([0, max(features[:,9])])
    plt.ylim([0, 1700])
    plt.yscale('log')
    plt.show()

    plt.scatter(features[:,7], features[:,6], color='blue')
    plt.gcf().set_facecolor('white')
    plt.title("Comparison of the Initial Spread Index (ICI) and Drought Code Levels (DC)
    plt.ylabel('DC Levels')
    plt.xlabel('Initial Spread Index (ICI)')
```

```
    plt.ylim([0, 900])
    plt.xlim([0, max(features[:,7]) + 1])
    plt.show()

def main():
    train_ftrs, train_outcome = read_data('train.csv')
    #explore_inputs(train_ftrs, train_outcome)


if (__name__) == '__main__':
    main()
```

KNN Functions:

```
import math
import copy
import numpy as np
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt

# Compute difference between 2 points
def euclidean_distance(a, b):
    d = 0
    for i in range(len(a)):
        d += (a[i] - b[i]) ** 2
    return math.sqrt(d)

def find_neighbors(K, train, point):
    distances = []

    for i in range(len(train)):
        d = euclidean_distance(train[i], point)
        distances.append((d, i)) # Create a tuple of the distance and sample #

    # Only return the top K distances
    return sorted(distances)[:K]

def run_knn(K, truth, train, point):

    neighbors = find_neighbors(K, train, point)

    yes_votes = 0
    no_votes = 0
    for point in neighbors:
        if truth[point[1]] == 1:
```

```
            yes_votes += 1
        else:
            no_votes += 1

    if yes_votes > no_votes:
        return 1
    else:
        return 0

def cross_validation(feature_set, truth_set, K, num_splits):

    kf = KFold(n_splits=num_splits)
    set_accuracy = []
    total = 0
    # Split the set indices between training and testing sets
    for train_index, test_index in kf.split(feature_set):
        correct = 0

        X_train, x_test = feature_set[train_index], feature_set[test_index]
        y_train, y_test = truth_set[train_index], truth_set[test_index]

        total = len(y_test)

        for i in range(len(x_test)):
            classification = run_knn(K, y_train, X_train, x_test[i])

            if classification == y_test[i]:
                correct += 1

        set_accuracy.append(correct)


    #print("K VALUE:", K, "---- ERRORS:", sum(set_accuracy)/len(set_accuracy))
    print(float(sum(set_accuracy))/len(set_accuracy))/total
    return float((sum(set_accuracy)))/(len(set_accuracy))/total

def find_best_K(feature_set, truth_set):

    K = [ i for i in range(3, 53, 2)]
    accuracy = [] # Keep track of errors for each K value

    best_K = 1
    error_count = 10000000000
    for k in K:
```

```
        x = cross_validation(feature_set, truth_set, k, 10)

        accuracy.append(x)

        if x < error_count:
            error_count = x
            best_K = k

    plt.plot(K, accuracy, color='Grey')
    plt.xlim([0, max(K)+3])
    plt.title('Classification Accuracy for Values of K Using Cross Validation')
    plt.xlabel('K Value')
    plt.ylabel('Accuracy')
    plt.ylim([0, 1])
    plt.xlim([min(K), max(K)])
    plt.show()

    print "BEST K", best_K, " --- Errors: ", error_count
    return best_K

def test_classifier(train_features, train_truth, test_features, test_truth, K):

    correct = 0

    for p in range(len(test_features)):
        classification = run_knn(K, train_truth, train_features, test_features[p])

        if classification == test_truth[p]:
            correct += 1

    return correct
```

Linear Regression Implementation:

```
import matplotlib.pyplot as plt
from numpy.linalg import inv
import csv
import numpy as np
import inspection

def strip_useless_data(features, outcomes):

    new_features = []
    new_outcomes = []
```

```
    for i in range(len(outcomes)):
        if outcomes[i] != 0:
            new_features.append(features[i])
            new_outcomes.append(outcomes[i])
    return new_features, new_outcomes


# The log graph addresses skewed data and we can view a more normal distribution
# of the outcomes with the log graph. It decreases the variability of the data.
def plot_outcome_histogram(outcomes):
    plt.hist(outcomes, color='lime')
    plt.gcf().set_facecolor('white')
    plt.title('Histogram of Burned Area ')
    plt.ylabel('Bin Count')
    plt.xlabel('Burned Area')
    plt.show()

    plt.hist(np.log(outcomes))
    plt.gcf().set_facecolor('white')
    plt.title('Histogram of Logarithimically Scaled Burned Area')
    plt.ylabel('Bin Count')
    plt.xlabel('Burned Area (Log Scaled)')
    plt.show()

def compute_ols(X, Y):
        X_t = np.transpose(X)
        w_1 = inv(np.dot(X_t, X))
        w_2 = np.dot(X_t, Y)

        # Compute optimal weight vector
        w = np.dot(w_1, w_2)
        return w

# Computes RSS with testin data and computed weight vector
def compute_RSS(w, X, Y):
    RSS = np.dot(np.transpose((Y - np.dot(X, w))), (Y - np.dot(X, w)))
    print "RSS:", RSS

    return np.dot(X, w)

def main():
    train_ftrs, train_outcome = inspection.read_data('train.csv')
    test_ftrs, test_outcome = inspection.read_data('test.csv')

    train_ftrs, train_outcome = strip_useless_data(train_ftrs, train_outcome)
```

```
    test_ftrs, test_outcome = strip_useless_data(test_ftrs, test_outcome)

    # Append 1s to the first column for the feature data
    train_ftrs = np.insert(train_ftrs, 0, 1, axis=1)
    test_ftrs = np.insert(test_ftrs, 0, 1, axis=1)

    train_ftrs = np.square(train_ftrs)
    test_ftrs = np.square(test_ftrs)

    #plot_outcome_histogram(train_outcome)

    weight_vector = compute_ols(train_ftrs, train_outcome)
    predicted = compute_RSS(weight_vector, test_ftrs, test_outcome)

    print np.corrcoef(test_outcome, predicted)[0, 1]

if (__name__) == '__main__':
    main()
```