# Machine Learning – HW1

Li Hen Chen

*Department of Computer Science & Engineering, Texas A&M University*

## Question 1

### Ans #a & #b

In order to minimize the RSS, we must compute the partial derivatives of the RSS with respect to $\omega_0 \; and \; \omega_1$ Starting with the following equation.

$$RSS(w_0, w_1) = \sum_{n=1}^{N} (y_n - \omega_0 \; - \omega_1 x_n \;)^2$$

Then take partial derivatives with $\omega_0 \; and \; \omega_1$ and we can obtain the following.

$$\frac{\theta RSS}{\theta \omega_0} = -2 \sum_{n=1}^{N} (y_n - \omega_0 \; - \omega_1 x_n \;)$$

$$\frac{\theta RSS}{\theta \omega_1} = -2 \sum_{n=1}^{N} (y_n - \omega_0 \; - \omega_1 x_n \;)(x_n)$$

We can set both partials equal to 0 (Minimizing RSS), and solve for the respective weights, we obtain

$$\sum_{n=1}^{N} y_n - N\omega_0^* - \sum_{n=1}^{N} \omega_1 x_n = 0$$

Solving for $\omega_0^*$ we could have:

$$\omega_0^* = \frac{1}{N} \sum_{n=1}^{N} y_n - \omega_1 \frac{1}{N} \sum_{n=1}^{N} x_n$$

We know that the average of any set of measurements is just the sum of each component divided by the number of components. So we can then rewrite the minimized $\omega_0^*$ as

$$\omega_0^* = \overline{y} - \omega_1 \overline{x}$$

Similarity, we can do the same for $\omega_1$:

$$\sum_{n=1}^{N} x_n y_n - \sum_{n=1}^{N} \omega_0^* x_n - \sum_{n=1}^{N} \omega_1^* x_n^2 = 0$$

Now we can solve it for $\omega_1^*$ and obtain:

$$\omega_1^* = \frac{\sum_{n=1}^{N} x_n y_n - N(\frac{1}{N}\sum_{n=1}^{N} y_n)(\frac{1}{N}\sum_{n=1}^{N} x_n)}{\sum_{n=1}^{N} x_n^2 - N(\frac{1}{N}\sum_{n=1}^{N} x_n)x_n}$$

we are able to rewrite this equation to a more compact form by using the average notation we used for $\omega_0$.

$$\omega_1^* = \frac{\sum_{n=1}^{N} x_n y_n - N(\overline{x})(\overline{y}))}{\sum_{n=1}^{N} x_n^2 - N(\overline{x})^2}$$

Through a few tricks of adding and subtracting certain terms. Finally, we can obtain

$$\omega_1^* = \frac{\sum_{n=1}^{N} (x_n - \overline{x})(y_n - \overline{y}))}{\sum_{n=1}^{N} (x_n - \overline{x})^2}$$

### Ans #c

Both $\overline{x} \; and \; \overline{y}$ represent the sample means for both populations. Additionally, $\sum_{n=1}^{N} (x_n - \overline{x})^2$ represent the total sample variation in sample x. The same can be said for y. The Numerator of $\omega_1^*$ is the Co-variance between x and y values. The denominator is the variance of x.

## Question 2

### Ans#a

Cause w(k) is the value of w at the k$^{th}$ iteration of gradient descent. To show the second order Taylor expansion of the target function J(w) around w(k) just to let

$$x_0 = w(k)$$

$$x = w$$

$$f(x_0) = J\big(w(k)\big)$$

$$\nabla^2 J(w) = H$$

Thus according to Taylor expansion we can obtain the following equation.

$$J(w) \approx J\big(w(k)\big) + \big(\nabla J|_{w=w(k)}\big)^T \cdot \big(w - w(k)\big)$$

$$+\frac{1}{2}\big(w - w(k)\big)^T \cdot \mathrm{H}_J|_{w=w(k)} \cdot (w - w(k))$$

**Ans #b**

$$w(k+1) = w(k) - \alpha(k) \cdot \nabla J|_{w-w(k)}$$

As a result, we can substitute it into Taylor expansion and I will have the following.

$$J\big(w(k+1)\big) \approx$$

$$J\big(w(k)\big) + \big(\nabla J|_{w=w(k)}\big)^T$$

$$\cdot \Big(w(k) - \alpha(k) \cdot \nabla J|_{w-w(k)} - w(k)\Big)$$

$$+\frac{1}{2}\big(w - w(k) - \alpha(k) \cdot \nabla J|_{w-w(k)}\big)^T \cdot \mathrm{H}_J|_{w=w(k)}$$

$$\cdot (w(k) - \alpha(k) \cdot \nabla J|_{w-w(k)} - w(k))$$

We can see $w(k)$ can be subtracted as the following result.

$$J\big(w(k+1)\big) \approx J\big(w(k)\big) - \big||\nabla \mathrm{J}|_{w=w(k)}\big|_2^2 \cdot \alpha(k) + \frac{1}{2}\big(\nabla J|_{w=w(k)}\big)^T \cdot \mathrm{H}_{J|w=w(k)} \cdot \big(\nabla \mathrm{J}|_{w=w(k)}\big) \cdot \alpha^2(k)$$

**Ans #c**

Minimizing the above expression with respect to the step size α(k) means we derivate the above equation with respect to $\alpha$.

$$\frac{\partial J\big(w(k+1)\big)}{\partial \alpha(k)} =$$

$$-\big||\nabla J|_{w=w(k)}\big|_2^2 + \big(\nabla J|_{w=w(k)}\big)^T \cdot H_{J|_{w=w(k)}} \cdot \big(\nabla J_{w=w(k)}\big) \cdot \alpha(k) = 0$$

Then solve the above equation we can have

$$\alpha(k) = \frac{\big||\nabla J|_{w=w(k)}\big|_2^2}{\big(\nabla J|_{w=w(k)}\big)^T \cdot H_{J|_{w-w(k)}} \cdot \big(\nabla J_{w=w(k)}\big)}$$

**Ans #d**

According to the above equation, cause $H_{J|_{w=w(k)}}$ is second derivation, so the time complexity is $O(n^3)$.

**Question 3**

**Ans #a**

We start to inspect the data and we find out **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.** As a result, there are 13 features. First we can preprocess the data like using the statistical data to show the whole distribution. Also

by creating “Class” feature (Class 0: Forests not affected by the fire, i.e. area = 0 , Class 1: Forests affected by the fire, i.e. area > 0) and “Log-Area” feature, we can better interpret the data. Finally, We visualize them to find if there are some correlations between them.

Form 1. All Data Description

description

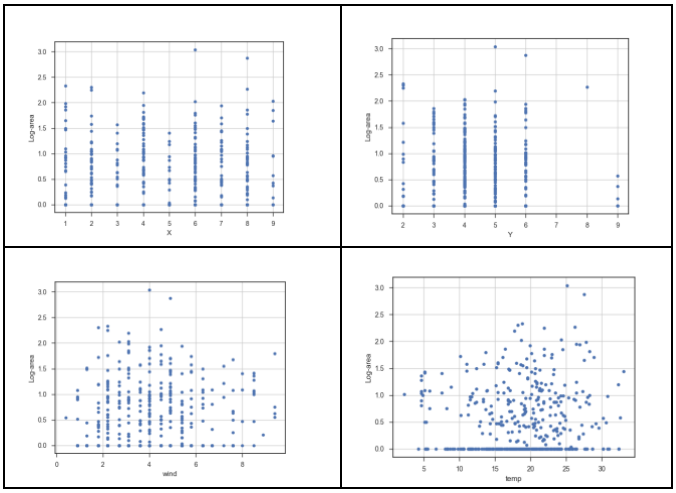
| | X | Y | month | day | FFMC | DMC | DC | ISI | temp | RH | wind | rain | area | Log-area | category |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 466 | 466 | 466 | 466 | 466 | 466 | 466 | 466 | 466 | 466 | 466 | 466 | 466 | 466 | 466 |
| mean | 4.67 | 4.29 | 7.5 | 4.27 | 90.68 | 111.26 | 550.37 | 8.96 | 18.92 | 43.98 | 4.04 | 0.02 | 12.87 | 0.49 | 0.52 |
| std | 2.34 | 1.25 | 2.28 | 2.07 | 5.48 | 63.19 | 245.36 | 4.04 | 5.79 | 16.29 | 1.81 | 0.31 | 65.24 | 0.61 | 0.5 |
| min | 1 | 2 | 1 | 1 | 18.7 | 1.1 | 7.9 | 0 | 2.2 | 15 | 0.4 | 0 | 0 | 0 | 0 |
| 25% | 3 | 4 | 7 | 2 | 90.3 | 69.15 | 444.72 | 6.7 | 15.6 | 32 | 2.7 | 0 | 0 | 0 | 0 |
| 50% | 4 | 4 | 8 | 5 | 91.6 | 108.4 | 661.8 | 8.4 | 19.3 | 41 | 4 | 0 | 0.49 | 0.17 | 1 |
| 75% | 7 | 5 | 9 | 6 | 92.9 | 142.12 | 712.23 | 10.78 | 22.8 | 53 | 5.4 | 0 | 7 | 0.9 | 1 |
| max | 9 | 9 | 12 | 7 | 96.2 | 291.3 | 860.6 | 22.7 | 33.1 | 100 | 9.4 | 6.4 | 1090.84 | 3.04 | 1 |

Form 2. Burned Area Description

description_Nozeros

| | X | Y | month | day | FFMC | DMC | DC | ISI | temp | RH | wind | rain | area | Log-area | category |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 243 | 243 | 243 | 243 | 243 | 243 | 243 | 243 | 243 | 243 | 243 | 243 | 243 | 243 | 243 |
| mean | 4.86 | 4.37 | 7.74 | 4.17 | 90.99 | 113.83 | 567.4 | 9.13 | 19.23 | 43.6 | 4.16 | 0.03 | 24.68 | 0.93 | 1 |
| std | 2.39 | 1.18 | 2.15 | 2.06 | 3.81 | 61.39 | 232.38 | 4.13 | 6.13 | 15.15 | 1.92 | 0.42 | 88.81 | 0.54 | 0 |
| min | 1 | 2 | 2 | 1 | 63.5 | 3.2 | 15.3 | 0.8 | 2.2 | 15 | 0.4 | 0 | 0.09 | 0.04 | 1 |
| 25% | 3 | 4 | 8 | 2 | 90.5 | 81.35 | 480.8 | 7 | 16 | 32.5 | 2.7 | 0 | 2.14 | 0.5 | 1 |
| 50% | 5 | 4 | 8 | 4 | 91.7 | 112.4 | 665.3 | 8.4 | 20.1 | 41 | 4 | 0 | 6.58 | 0.88 | 1 |
| 75% | 7 | 5 | 9 | 6 | 93.1 | 141.15 | 721.4 | 11.2 | 23.3 | 53 | 5.4 | 0 | 15.82 | 1.23 | 1 |
| max | 9 | 9 | 12 | 7 | 96.2 | 291.3 | 860.6 | 22.7 | 33.1 | 96 | 9.4 | 6.4 | 1090.84 | 3.04 | 1 |

We can tell from the above two forms that the burned area and non-burned area is nearly 1:1. The burned area’s standard deviation is relatively large.



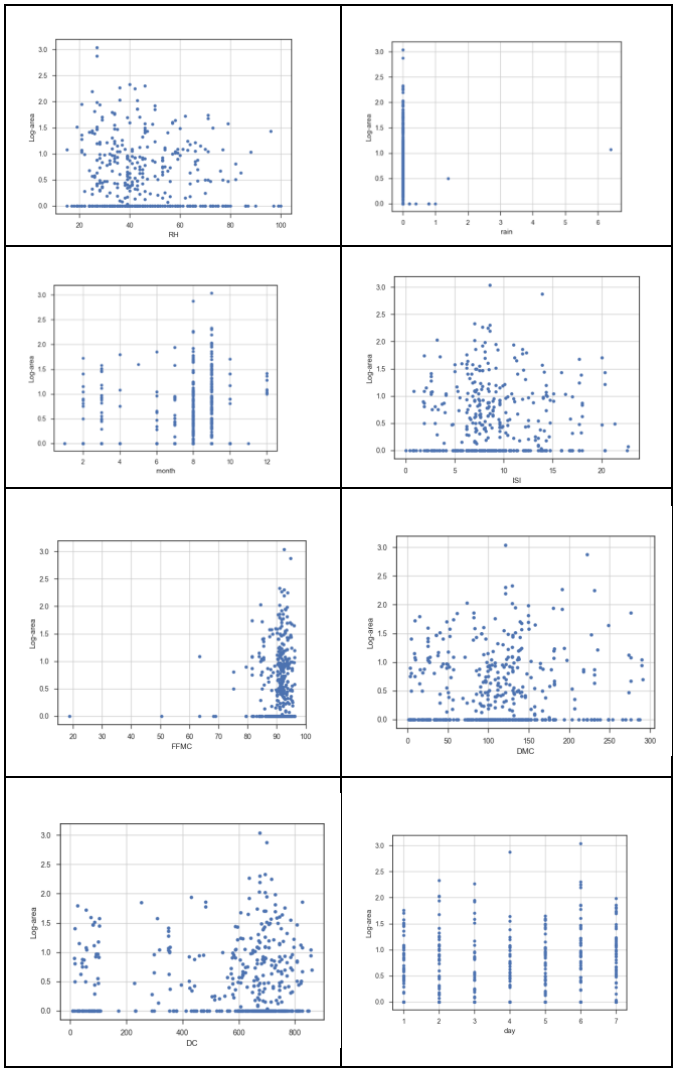




Fig 1. Log-Area & Features Correlation

As mentioned above, burned area’s standard deviation is relatively large, so I plot the above figures with the row of Log-Area and column of each features. On the above form, the burned area seems doesn’t relate to any attribute. In another word, their correlations are relatively low.

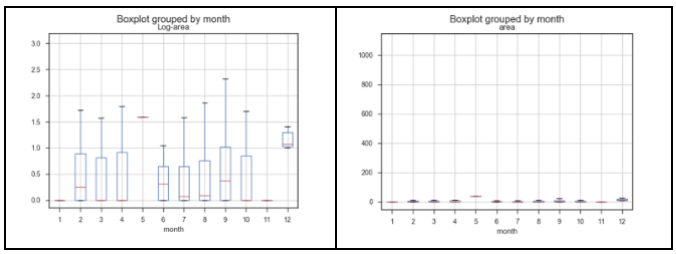



Fig 2. Box figure relating to month

I consider month may impact the burned area to certain degree. Because the std of burned area is high, it has outliers. In order to visualize well, we take the

log of burned-area. The result shows the burned area is larger during the summer.

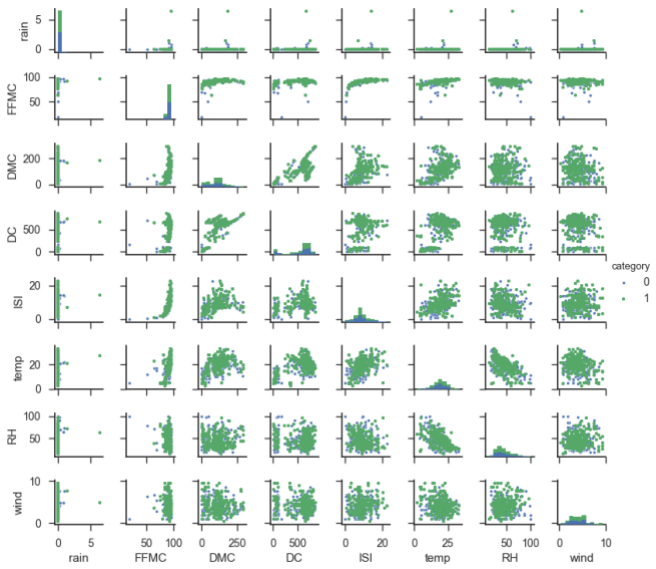



Fig 3. Correlation between Class1 & Class2 to Features

At the question (b), we need classify the data into two categories burned and unburned. I regard every two continuous features could result in some kind of correlation to the categories. However, we indeed could not see some correlation by our eyes. That's what machine learning do.

**Ans #b**

The implementation of KNN is by comparing the number of K neighbors of our predicted set. This was 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. Caution to normalize the training features entries and y simply calculating the distance of each entries feature from the training features entries, we can obtain the K nearest neighbors and then used a voting count to predict using an odd number of neighbors whether or not the testing point belonged to the majority class.

However, due to choosing the parameter of K can be tricky. I use 10-cross validation to determine which k is the most suitable parameter.

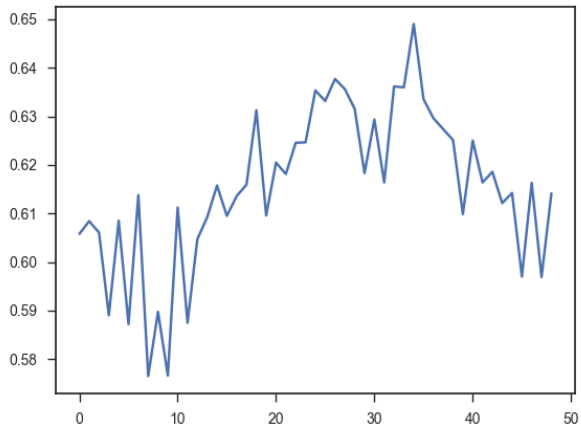



Fig 4. Cross Validation to determine K

The highest accuracy that my KNN implementation could reach accuracy of 65% at K=33 neighbors.

Finally, I adopt the K=33 to train the KNN model. When the model test with the testing set. **The accuracy is 67%.**

Besides Euclidean distance, I also try Manhattan distance. The definition of Manhattan distance $d_1$ is that between two vectors $p, q$ in an n-dimensional real vector space with fixed Cartesian coordinate system, is the sum of the lengths of the projections of the line segment between the points onto the coordinate axes(). When trying Manhattan distance I also adopt the K=33 to train the KNN model. When the model test with the testing set. The accuracy is nearly the same which is 66%.

$$d_1 = (p, q) = \left|\left|p - q\right|\right|_1 = \sum_{i=1}^{n} |p_i - q_i|$$

**Ans #C**

At the part C, we only utilize the data with burned area to predict the burned area by features.

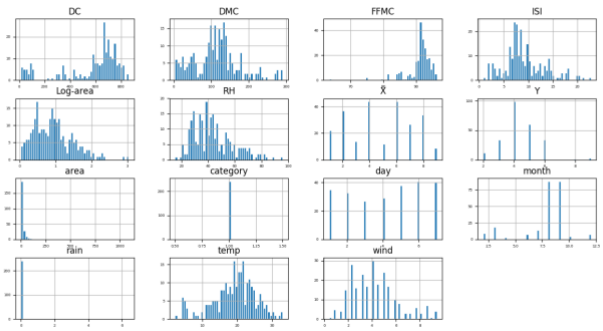



Fig 5. Histogram of each Features

As the figure above. Because there are some outliers and the Std of burned area is large so when plotting in log scale, we can see that the histogram resembles a normal distribution

In Linear Regression, we approach to model the relationship between dependent variable and one or more explanatory independent variables. At these case, the input is features except burned area, and the output is the burned area. We use weight vector to store the parameters of mode. After computing the OSL (Ordinary Least Square) matrix $\omega^*$, we use it to predict the burned area on testing data.

$$RSS(w_0, w_1) = \sum_{n=1}^{N} (y_n - \omega_0 - \omega_1 x_n)^2$$

To evaluate how well the model, we have to calculate the Residual Sum of Square errors. The RSS of this model is 103808.62 which is surprising high. We can discuss why it is so high later.

When it comes to non-linear I experimented two functions as below.

$$RSS(w_0, w_1) = \sum_{n=1}^{N} (y_n - \omega_0 - \omega_1 x_n^2)^2$$

$$RSS(w_0, w_1) = \sum_{n=1}^{N} (y_n - \omega_0 - \omega_1 x_n^3)^2$$

The RSS result is 103023.34 and 103848.25 respectively. We can tell from the number the function of $x^2$ has a better result.

## Discussion

The above result in the machine learning field is not quite well. I guess because not every feature is highly related to the area like the confusion matrix and I did not preprocess the data like feature selection or feature. That's an import process in machine learning. I draw the features that have higher relation to Area (Fig. Correlation to Area). We still can not tell from the figure whether there is any relationship. Maybe other machine learning technique we learn later can distinguish. Another point is I compare the self-made KNN and linear regression to the ones in the sklearn library, but the result was slightly different and my program result is better. I speculate that sklearn's model may be implement different ways and I did not tune the parameters well.


Fig 6. Confusion Matrix

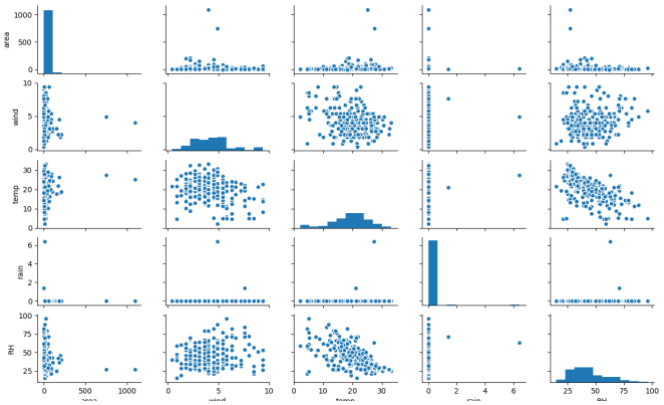



Fig 7. Correlation to Area

# Source Code

Hw 3-1

```
import numpy as np
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import os
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn import preprocessing
import seaborn as sns
from sklearn.cross_validation import cross_val_score,cross_val_predict
sns.set(style="ticks", color_codes=False)



def import_data(dir):
    #Import the data set using a pandas frame with condition that area>0
    data = pd.read_csv(dir)
    data['category'] = (data['area'] > 0).astype(int)
    return data

def add_Class(fileName, skipZeros=False):
    data = pd.read_csv(fileName)
    data['Log-area'] = np.log10(data['area']+1)
    data['category'] = (data['area'] > 0).astype(int)
    data.describe().to_csv("data_description/description.csv",
float_format="%.2f")
    data_without = data.where(data['area'] != 0)
    data_without = data_without[data_without['area'].notnull()]
    data_without.describe().to_csv(
        "data_description/description_Nozeros.csv", float_format="%.2f")
    return data_without if skipZeros else data

def plot_pair(df):
    df=df.drop(columns=['X','Y','month','day','Log-area','area'])
    #print(df)
    g = sns.pairplot(df, hue="category", size=1,
                     markers=["o", "+"], plot_kws={"s": 7}
                     ,vars=['rain',
'FFMC','DMC','DC','ISI','temp','RH','wind'])
    #g = g.map(plt.scatter, linewidths=1, edgecolor="w", s=5)
    #plt.show()

def plot_forest(dataFrame):
    data = dataFrame
    basedir = 'images/'
    #print(data_without)
    """
    fig, axes = plt.subplots(1, len(data.columns.values)-1, sharey=True)

    for i, col in enumerate(data_without.columns.values[:-2]):
        data_without.plot(x=[col], y=["Log-area"], kind="scatter",
ax=axes[i])
    """
    #plt.savefig('all.png')
    #df.boxplot(column='Log-area', by='month')
    for i in data.describe().columns[:-2]:
        data.plot.scatter(i, 'Log-area', grid=True)
        name = basedir+"col_{}.png".format(i)
        plt.savefig(name)

    data.boxplot(column='Log-area', by='month')

    plt.savefig('box_month_log.png')
    #plt.show()



#def Feature_Selection(dataFrame):



def KNN_train(X,y,parameter):
    #X_normal = preprocessing.normalize(X,norm='l1')
    #print(len(X),len(y)
    neighbor = KNeighborsClassifier(n_neighbors=parameter,
algorithm='brute', p=1)
    # minmax = preprocessing.MinMaxScaler()
    # X_normal = minmax.fit_transform(X)
    #X_normal = preprocessing.normalize(X, norm='l1')
```

```
    neighbor.fit(X,y)
    #model = feature_selection.SelectFromModel(neighbor,prefit=True)
    #x_new = model.transform(X_normal)

    return neighbor


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)


    return float((sum(set_accuracy)))/(len(set_accuracy))/total

def determinN(X,y):
    # creating odd list of K for KNN
    myList = list(range(1, 50))
    # subsetting just the odd ones
    neighbors = filter(lambda x: x % 2 != 0, myList)
    # empty list that will hold cv scores
    cv_scores = []
    xx,yy = [],[]
    # perform 10-fold cross validation
    for k in neighbors:
        knn = KNeighborsClassifier(n_neighbors=k, algorithm='brute',p=2)
        scores = model_selection.cross_validate(knn,X,y,cv=10)
        cv_scores.append(scores['test_score'].mean())

        #Y_pred = model_selection.cross_val_predict(knn, X,y, cv=10,
method='predict')
        #print(sum(1 for i, j in zip(y, Y_pred) if i != j))


    print(cv_scores)

def KNN_Scores(X,y,model):
    #minmax = preprocessing.MinMaxScaler()
    #X_normal = minmax.fit_transform(X)
    print(model.score(X,y))


def euclideanDistance(instance1, instance2, length):
    distance = 0
    for x in range(length):
        distance += pow((instance1[x] - instance2[x]), 2)
    return math.sqrt(distance)


def knn(trainingSet, testInstance, k):    #self inplenmet
    distances = []
    length = len(testInstance)-1
    for x in range(len(trainingSet)):
        dist = euclideanDistance(testInstance, trainingSet[x], length)
        distances.append((trainingSet[x], dist))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(distances[x][0])

    classVotes = {}
    for x in range(len(neighbors)):
        response = neighbors[x][-1]
        if response in classVotes:
            classVotes[response] += 1
```

```
            else:
                classVotes[response] = 1
        sortedVotes = sorted(classVotes.iteritems(),
                             key=operator.itemgetter(1), reverse=True)
        return sortedVotes[0][0]

def main():

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

HW 3-2

```
import numpy as np
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import seaborn as sns

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

def add_Class(fileName, skipZeros=False):
    data = pd.read_csv(fileName)
    data['Log-area'] = np.log10(data['area']+1)
    data['category'] = (data['area'] > 0).astype(int)
    #data.describe().to_csv("data_description/description.csv",
float_format="%.2f")
    data_without = data.where(data['area'] != 0)
    data_without = data_without[data_without['area'].notnull()]
    #data_without.describe().to_csv(
    #       "data_description/description_Nozeros.csv", float_format="%.2f")
    return data_without if skipZeros else data

def train_regression(X,y):
    reg = LinearRegression(fit_intercept=True,normalize=False).fit(X, y)
    #model = sm.OLS(y, X)
    #result = model.fit()
    #print(reg.score(X,y))

    return reg

def remove_outlier(forest_fire):
    forest_fire['area_cat'] = pd.cut(forest_fire['area'], bins=[0, 5, 10, 50, 100,
1100], include_lowest=True,
    labels=['0-5', '5-10', '10-50', '50-100', '>100'])
    forest_fire.area_cat.value_counts()
    forest_fire.drop(forest_fire[forest_fire.area >100].index, axis=0,
inplace=True)
    return forest_fire

def draw_plot(forest_fire):
    forest_fire.hist(bins=50, figsize=(30, 20), ec='w',
                     xlabelsize=5, ylabelsize=5)
    corr_matrix = forest_fire.corr(method='spearman')
    ax = plt.figure(figsize=(12, 8))
    ax = sns.heatmap(corr_matrix, cmap='PiYG')
    plt.show()
    print(corr_matrix.area.sort_values(ascending=False))

    #visualizing relations of most related attributes
    attributes = ['area', 'wind', 'temp', 'rain', 'RH']
    sns.pairplot(forest_fire[attributes])
    plt.show()

def OLS(x, y): #self OLS
    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):    #self RSS
    RSS = np.dot((y_test - np.dot(x_test, w)), (y_test - np.dot(x_test, w)))

if __name__ == "__main__":
```