Abstract

The main task for this project is to create a machine-learning algorithm to classify two given datasets hepatitis and bankruptcy. This project uses logistic regression to perform a two-class classification, which calculates the weights of each feature to classify the output. For hepatitis, the program has to figure out if a person is a patient or a survivor. For bankruptcy, the program has to classify each data into two labels.

When we work on this project, we implement gradient_descent to calculate weights and cross-validation to choose the best weights. We found that different learning rates would influence the accuracy rate of classification, also the decrease rate of learning rate influences the speed of machine learning. A small decrease rate of the learning rate would lead the training process to run a very long time. The implementation of k-fold cross validation helps to choose better weights for classification. By further experiments, we found that by using a deducting

Introduction

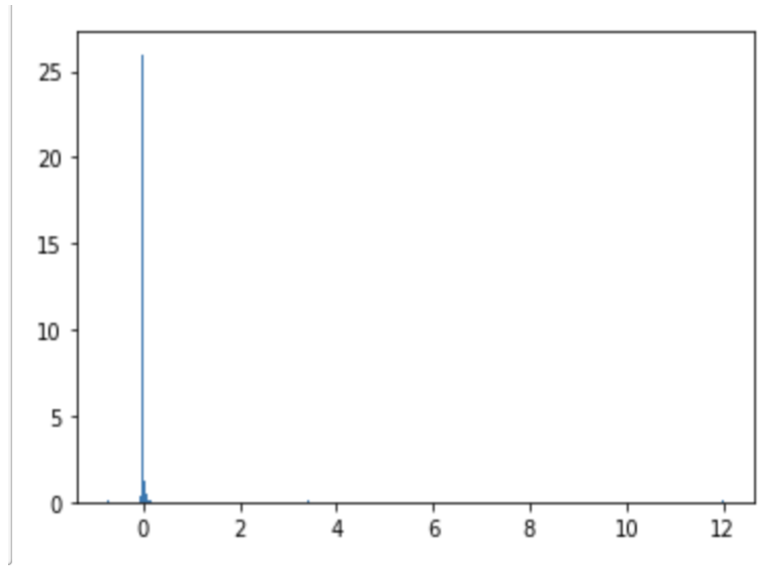
There are two datasets for our program to do the classification. Both of them need to be classified into two labels by using normalization. The main idea is to randomly initialize the weight of each feature and pass through the gradient descent to achieve the best weight. The way to support us in changing the weight is by using different learning rates. For example, we tried different initial learning rates (from 0 to 1) for calculation of the weights, it would get different values of weights. The learning rate cannot be too big or too small. Another important finding is that we need to make sure all the data are independently and identically. For example, the first half of the data of bankruptcy dataset are all labeled in 0 and the rest are all labeled in 1. This is not identically distributed, it would cause mistakes in the k-fold cross-validation, because all the folds are all labeled in 0 or all labeled in 1, so the weight would be calculated in an extreme side.

Datasets

Dataset hepatitis has 142 samples, and each of them has 13 binary features, and 6 numerical features.

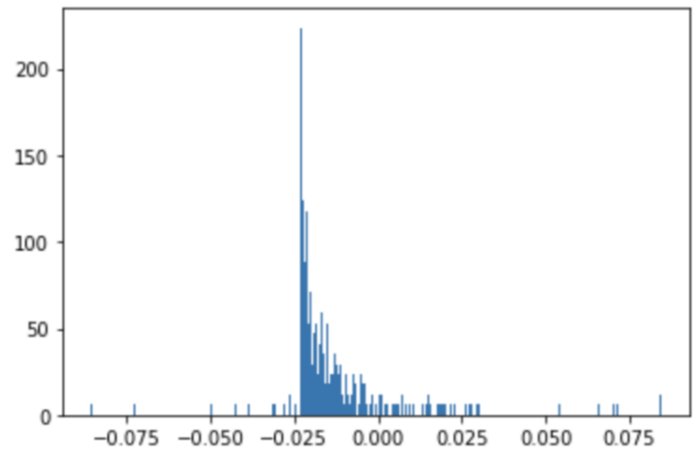
Dataset bankruptcy has 453 samples and each of them has 64 numerical features.

In this dataset, we found some of the samples that contain outlier features, for example in attribute59, most of the data are in the range of -1 to 1, howver in pyplot 1 there's two outliers which are 3.41 and 12. They influence both accuary and the cost of time.



pyplot 1

Since there are 451 data in the range of -1 to 1, it is difficult to see 3.41 and 12 on the pyplot 1.



pyplot 2

From pyplot 2, after applying our sample selection, the two outliers are cut from our database.

After apply the sample selection bankruptcy dataset reduces its 453 samples to 334 samples, however the quantity of the dataset hepatitis does not change, which means there is no distinct outliers in dataset hepatitis.

Results

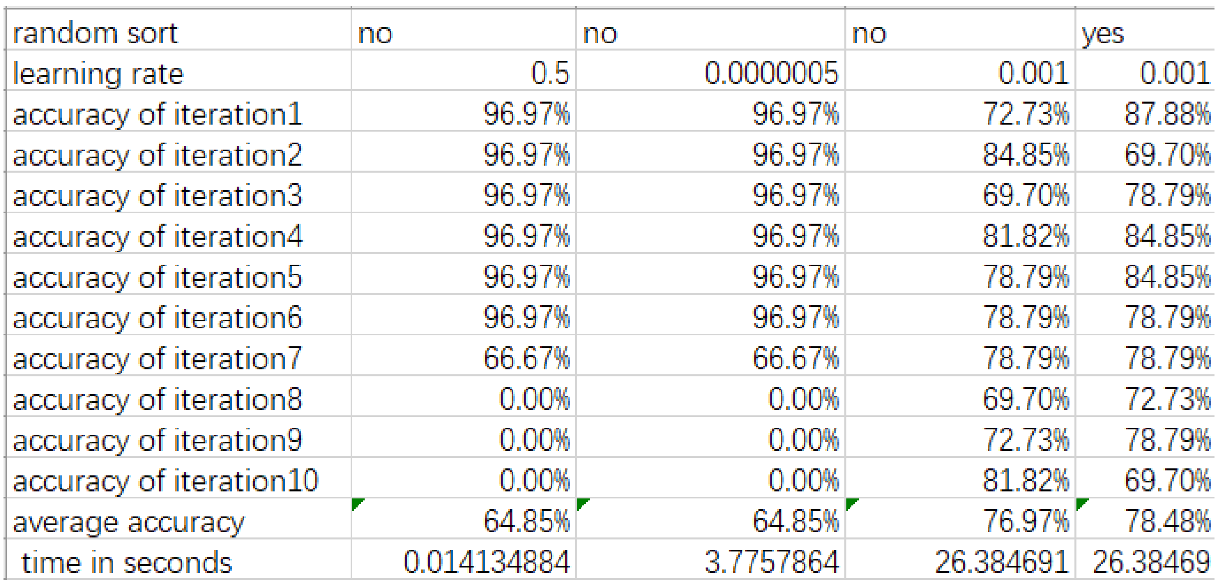
The result of bankruptcy:

We have tried several experiments with different learning rates. First we tried a very big learning rate 0.5. Here is the output which shows the accuracy of 10 fold cross validation. The accuracy is kind of low.

Then we tried a pretty small value 0.0000005.The accuracy is also not very high.

Finally we choose the value 0.001, we can see the accuracy is high and the run time is not too long.

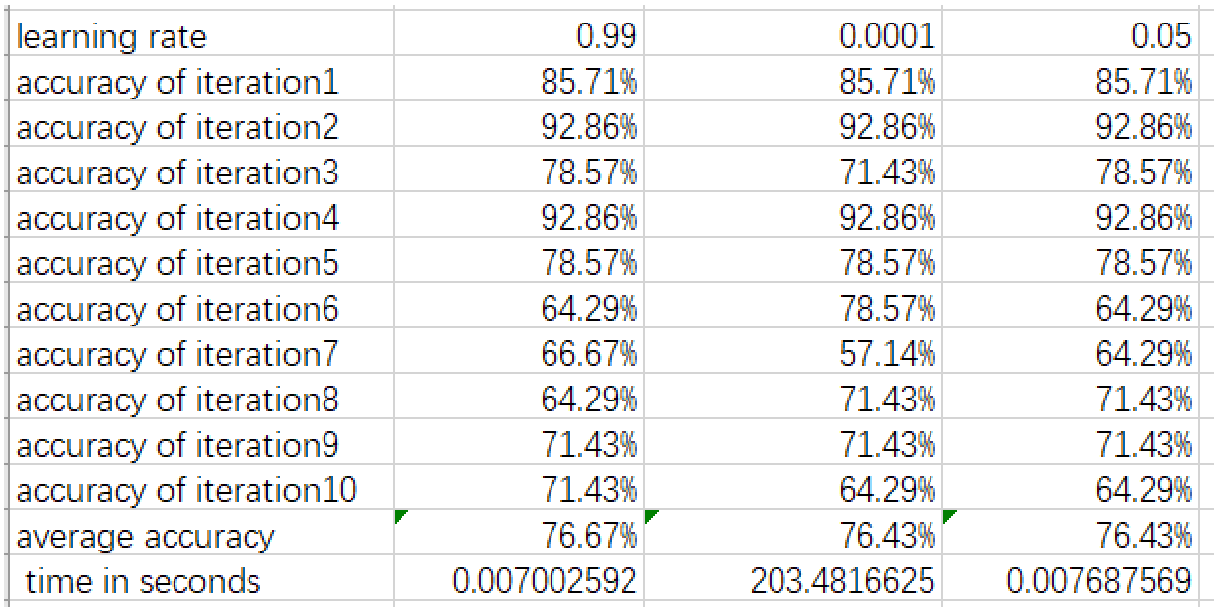
For this data set, we also tried sort those data in random order, which makes the calculation works better.



random sort	no	no	no	yes
learning rate	0.5	0.0000005	0.001	0.001
accuracy of iteration1	96.97%	96.97%	72.73%	87.88%
accuracy of iteration2	96.97%	96.97%	84.85%	69.70%
accuracy of iteration3	96.97%	96.97%	69.70%	78.79%
accuracy of iteration4	96.97%	96.97%	81.82%	84.85%
accuracy of iteration5	96.97%	96.97%	78.79%	84.85%
accuracy of iteration6	96.97%	96.97%	78.79%	78.79%
accuracy of iteration7	66.67%	66.67%	78.79%	78.79%
accuracy of iteration8	0.00%	0.00%	69.70%	72.73%
accuracy of iteration9	0.00%	0.00%	72.73%	78.79%
accuracy of iteration10	0.00%	0.00%	81.82%	69.70%
average accuracy	64.85%	64.85%	76.97%	78.48%
time in seconds	0.014134884	3.7757864	26.384691	26.38469

The result of hepatitis:

We also tried different learning rates. First we tried 0.99. Then we tried 0.00000001, the run time is very long. Finally we chose 0.05. We can see that the learning rate does not influence the accuracy too much.



learning rate	0.99	0.0001	0.05
accuracy of iteration1	85.71%	85.71%	85.71%
accuracy of iteration2	92.86%	92.86%	92.86%
accuracy of iteration3	78.57%	71.43%	78.57%
accuracy of iteration4	92.86%	92.86%	92.86%
accuracy of iteration5	78.57%	78.57%	78.57%
accuracy of iteration6	64.29%	78.57%	64.29%
accuracy of iteration7	66.67%	57.14%	64.29%
accuracy of iteration8	64.29%	71.43%	71.43%
accuracy of iteration9	71.43%	71.43%	71.43%
accuracy of iteration10	71.43%	64.29%	64.29%
average accuracy	76.67%	76.43%	76.43%
time in seconds	0.007002592	203.4816625	0.007687569

Discussion and conclusion

After doing this project, there are several things we learned. The first thing we learned is the exact sequence of training a machine learning program. The sequence is that we have the model and data, we calculate an approximate weight. Then separate the data into k folds to implement cross-validation. Each time we implement the gradient descent to calculate a better weight. Next, calculate each fold's accuracy. Finally, compare each fold''s accuracy to choose the best one.

The second thing we learned is about choosing the learning rate. This is a very important parameter for calculating weights. If it is too big, the program cannot find the correct weight, if it is too small, the calculation of weight may end up with the local minimum or run a long time. In the feature, if we deal with this kind of experiment, we will consider calculating the accuracy of the weights by using different learning rates to see the performance. The learning rate is different depending on the data and model that we are using.

The last thing we learned is about choosing data. In this project, we provided two datasets, but we still need to do some operation on the given data. We need to choose data for each fold, then choose the validation set and training set. We need to make sure the data is independently and identically. Otherwise, we need to make the data randomly so that it is close to the natural distribution. For example, in the distribution of bankruptcy, this dataset has all the data with the same output together. If we don't make the data sort in random order first, the weight calculated by each fold is only suitable for its own fold.

We also used IQR to reduce the quantity of the sample. The interquartile range (IQR) is a measure of statistical dispersion and is calculated as the difference between the 75th and 25th percentiles. In our implementation we limite the difference between 95th and 5th percentiles which fits the datasets the best. It is represented by the formula IQR = Q3 − Q1.

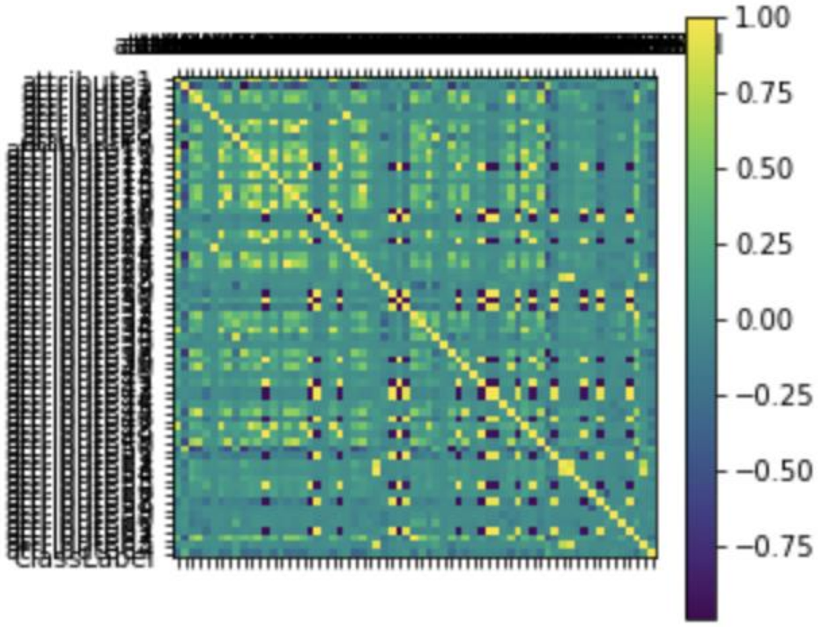
After the implementation of the code. We have another inverstgation. e compare the correlation between each features. For exmaple, there are n features in a data set, we compare feature 1 vs. feature 1, feature 1 vs. 2 ... feature 1 vs. feature n, feature 2 vs. feature 1, feature 2 vs. feature 3 ... feature2 vs feature n .

Correlation values range between -1 and 1.

There are two key components of a correlation value:

- magnitude – The larger the magnitude (closer to 1 or -1), the stronger the correlation
- sign – If negative, there is an inverse correlation. If positive, there is a regular correlation.

We have not implement it yet, and it is the direction of our future direction.



graph 1

Statement of Contributions

Chenyi Song and Peng Li wrote one Version, Zeye Gu wrote another version. Later on, we combined two versions together, Chenyi Song and Peng Li focused on the cross validation part, Zeye Gu was doing visualization,   chosing data, and   supporting his teammate.

Appendix

Code of bankrupcy.py:

```
# -*- coding: utf-8 -*-

"""bankrupcy.ipynb


Automatically generated by Colaboratory.


Original file is located at
```

```
https://colab.research.google.com/drive/1nTMa_9OtO9dWorU7kZB5tz2Qb_u5w5b8

"""


import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import time


def read_data():

    df = pd.read_csv('bankrupcy.csv')

    Q1 = df.quantile(0.05)

    Q3 = df.quantile(0.95)

    IQR = Q3 - Q1

    dataset_out = df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]

    return dataset_out.to_numpy()

#convert to numpy make it faster


# data set to 0 - 1 impact remains the same

def normalize(data):

    minimums = np.min(data,axis=0)

    maximums =np.max(data,axis=0)

    rng= maximums-minimums

    normalize_data = 1-((maximums-data)/rng)
    return normalize_data


# def cor_selector(X, y,num_feats):
#       cor_list = []
#       feature_name = X.columns.tolist()
#       # calculate the correlation with y for each feature
#       for i in X.columns.tolist():
#             cor = np.corrcoef(X[i], y)[0, 1]
#             cor_list.append(cor)
#       # replace NaN with 0
#       cor_list = [0 if np.isnan(i) else i for i in cor_list]
#       # feature name
#       cor_feature = X.iloc[:,np.argsort(np.abs(cor_list))[-num_feats:]].columns.tolist()
#       # feature selection? 0 for not select, 1 for select
#       cor_support = [True if i in cor_feature else False for i in feature_name]
#       return cor_support, cor_feature
#       cor_support, cor_feature = cor_selector(X, y,num_feats)
#       print(str(len(cor_feature)), 'selected features')


#   1/(1+e^-a)
def logistc_function(thetas,X):
    #sygmoid
    return 1.0/(1+np.exp(-np.dot(X,thetas.T)))
```

```
def cost_func(thetas, X, y):
    log_func_value = logistc_function(thetas, X)
    step1 = y* np.log(log_func_value)
    step2 = (1 - y) * np.log(1 - log_func_value)
    return np.mean(-step1 - step2)

# return direction
def log_gradient(thetas, X, y):
    calc_1 = logistc_function(thetas, X) - y.reshape(X.shape[0],1)
    calc_f = np.dot(calc_1.T, X)
    return calc_f

#thetas is random but will go to the global minima
def gradient_descent(X, y, thetas, learning_rate, convergance_criteria):
    cost = cost_func(thetas , X, y)
    change = 1
    iter_count = 1

    while(change > convergance_criteria):
        old_cost = cost
        thetas = thetas - (learning_rate * log_gradient(thetas, X, y))
        cost = cost_func(thetas, X, y)
        change = old_cost - cost

        iter_count += 1


    return thetas , iter_count


def cross_validation(train_X,train_Y,w,learning_rate,epsilon, length):

    Err_A=0

    lowest=1000;

    bestw=w;

    initw=w;

    for k in range(1,11):

        low= length*(k-1)

        up = length*k - 1

        w=initw

        validate_x=train_X[low:up,:]

        t_x=np.append(train_X[0:low,:],train_X[up:(length*10-1),:],axis=0)

        validate_y=train_Y[low:up,:]

        validate_y = validate_y[:,-1]

        t_y=np.append(train_Y[0:low,:],train_Y[up:(length*10-1),:],axis=0)

        t_y = t_y[:,-1]

        w ,count= gradient_descent(t_x,t_y,w,learning_rate,0.001)

        Xw = np.dot(validate_x,w.T)

        a = np.dot(validate_y.T,validate_y)
        b = 2 * np.dot(validate_y.T,Xw)

        c = np.dot(validate_x,w.T)

        d = np.dot(c.T,Xw)

        err = a - b + d

        # print("Err =" + str(err))

        y_pred = predict(w,validate_x)

        # y_pred = y_pred[:,None]

        score = Accu_eval(y_pred, validate_y, length)

        print("iteration: ",k)

        print("Accuracy: " + str(score) + "%")

        Err_A += err

        if(err<lowest):

            lowest=err

            bestw=w


    Err_A = Err_A / 10

    return (Err_A,w)


def predict(thetas, X):

    prob = logistc_function(thetas,X)

    #dicision boundary

    predicted_value = np.where(prob > 0.5, 1, 0)

    return np.squeeze(predicted_value)
```

```
def Accu_eval(y_pred, y, length):
    corrects = np.sum(y == y_pred) / length * 100
    return corrects


#main
start_time = time.time()
learning_rate =0.001
covergance_criteria = 0.001
dataset = read_data()
np.random.shuffle(dataset)


trainNum = dataset.shape[0]
featureNum = dataset.shape[1]-1


while(trainNum % 10 != 0):
    trainNum = trainNum -1


print(trainNum)
testNum = int(trainNum/10)


normalized = normalize(dataset)
print("dataset shape", normalized.shape)
```

```
# copy all the rows apart from last column

X = normalized[:trainNum,:-1]

test_X = normalized[-testNum:,:-1]

test_X = np.hstack((np.matrix(np.ones(test_X.shape[0])).T,test_X))

X = np.hstack((np.matrix(np.ones(X.shape[0])).T,X))

y = normalized[:trainNum,-1]

test_y = normalized[-testNum:,-1]




print("main: " + str(X.shape))

#init thetas just zeros

thetas = np.matrix(np.zeros(X.shape[1]))


#training the thetas

(final_thetas, iter_count) =
gradient_descent(X,y,thetas,learning_rate,covergance_criteria)

# print(final_thetas, iter_count)

y_pred = predict(final_thetas,test_X)

corrects = np.sum(test_y == y_pred)

total = test_y.__len__()

print("fit function(Gradient descent): ")
print("Accuracy without cross validation ", corrects/total * 100)

print("--- %s seconds ---" % (time.time() - start_time))

newTime = time.time()


print("10 folder cross validation:")

y=y[:,None]

err, final_thetas =
cross_validation(X,y,thetas,learning_rate,covergance_criteria,testNum)

print("--- %s seconds ---" % (time.time() - newTime))


# df = pd.read_csv('bankrupcy.csv')

# X = df.iloc[:,0:-1]

# y = df.iloc[:,-1:]

# print("sas",X.shape,y.shape)

# cor_selector(y,X,452)
```

Code of hepatitis.py:

```
# -*- coding: utf-8 -*-

"""hepatitis.ipynb


Automatically generated by Colaboratory.


Original file is located at
```

```
    https://colab.research.google.com/drive/1g7EAVRR3-o7AxyEcukY66jUuF0VutyYw
"""


import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import time


def read_data():

    df = pd.read_csv('hepatitis.csv')

    Q1 = df.quantile(0.05)

    Q3 = df.quantile(0.95)

    IQR = Q3 - Q1

    dataset_out = df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]

    return dataset_out.to_numpy()

#convert to numpy make it faster


# data set to 0 - 1 impact remains the same

def normalize(data):

    minimums = np.min(data,axis=0)

    maximums =np.max(data,axis=0)

    rng= maximums-minimums

    normalize_data = 1-((maximums-data)/rng)
    return normalize_data

# normalize_data = 1-((maximums-dataset)/rng)
# normalize_data

# def visualize(X,y,thetas):
#       x_0 = X[np.where(y==0.0)]
#       x_1 = X[np.where(y==1.0)]

#       plt.scatter([x_0[:,1]],[x_0[:,2]],c ='b',label='y=0')
#       plt.scatter([x_1[:,1]],[x_1[:,2]],c ='r',label='y=1')

#       x1 = np.arange(0, 1, 0.1)
#       x2 = -(thetas[0,0] + thetas[0,1]*x1)/thetas[0,2]
#       plt.plot(x1, x2, c = 'k', label = 'reg line')

#       plt.xlabel('x1')
#       plt.ylabel('x2')
#       plt.legend()

#       plt.show()

#   1/(1+e^-a)
```

```
def logistc_function(thetas,X):

    #sygmoid

    return 1.0/(1+np.exp(-np.dot(X,thetas.T)))


def cost_func(thetas, X, y):

    log_func_value = logistc_function(thetas, X)

    step1 = y.T* np.log(log_func_value)

    step2 = (1 - y).T * np.log(1 - log_func_value)

    return np.mean(-step1 - step2)


# return direction

def log_gradient(thetas, X, y):

    calc_1 = logistc_function(thetas, X) - y.reshape(X.shape[0],1)

    calc_f = np.dot(calc_1.T, X)

    return calc_f


#thetas is random but will go to the global minima

def fit(X, y, thetas, learning_rate, convergance_criteria):

    cost = cost_func(thetas , X, y)

    change = 1

    iter_count = 1


    while(change > convergance_criteria):
        # learning_rate *= 0.9

        old_cost = cost

        thetas = thetas - (learning_rate * log_gradient(thetas, X, y))

        cost = cost_func(thetas, X, y)

        change = old_cost - cost

        iter_count += 1


    return thetas , iter_count


def cross_validation(train_X,train_Y,w,learning_rate,epsilon,length):

    Err_A=0

    lowest=1000;

    bestw=w;

    initw=w;

    for k in range(1,11):

        low= length * (k - 1)

        up = length * k - 1

        w=initw

        validate_x=train_X[low:up,:]

        t_x=np.append(train_X[0:low,:],train_X[up:(length * 10 - 1),:],axis=0)

        validate_y=train_Y[low:up,:]

        t_y=np.append(train_Y[0:low,:],train_Y[up:(length * 10 - 1),:],axis=0)

        w ,count= fit(t_x,t_y,w,learning_rate,epsilon)
        Xw = np.dot(validate_x,w.T)

        a = np.dot(validate_y.T,validate_y)

        b = 2 * np.dot(validate_y.T,Xw)

        c = np.dot(validate_x,w.T)

        d = np.dot(c.T,Xw)

        err = a - b + d

        # print("Err =" + str(err))

        y_pred = predict(w,validate_x)

        y_pred = y_pred[:,None]

        corrects = Accu_eval(y_pred,validate_y,length)

        print("Accuracy: " + str(corrects) + "%")


        Err_A += err

        if(err<lowest):

            lowest=err

            bestw=w


    Err_A = Err_A / 10

    return (Err_A,w)


def predict(thetas, X):

    prob = logistc_function(thetas,X)

    #dicision boundary
    predicted_value = np.where(prob > 0.5, 1, 0)

    return np.squeeze(predicted_value)


def Accu_eval(y_pred,y,length):

    corrects = np.sum(y == y_pred)/length * 100

    return corrects


#main

start_time = time.time()




learning_rate =0.01

covergance_criteria = 0.000001

dataset = read_data()

trainNum = dataset.shape[0]

while(trainNum % 10 != 0):

    trainNum = trainNum -1

print(trainNum)

testNum = int(trainNum/10)

normalized = normalize(dataset)


# copy all the rows apart from last column

X = normalized[:trainNum,:-1]
```

```
test_X = normalized[-testNum:,:-1]


#add ones as first column

# print(X.shape)

# np.hstack add 1s on first column

test_X = np.hstack((np.matrix(np.ones(test_X.shape[0])).T,test_X))

X = np.hstack((np.matrix(np.ones(X.shape[0])).T,X))

# copy last column

# y = normalized[:127,-1]

y = normalized[:trainNum,-1]

test_y = normalized[-testNum:,-1]

print(test_y)


#init thetas just zeros

thetas = np.matrix(np.zeros(X.shape[1]))

#training the thetas

(final_thetas, iter_count) = fit(X,y,thetas,learning_rate,covergance_criteria)

y_pred = predict(final_thetas,test_X)

corrects = np.sum(test_y == y_pred)

total = test_y.__len__()

print("fit function(Gradient descent): ")


print("Accuracy without cross validation ", corrects/total * 100)
print("--- %s seconds ---" % (time.time() - start_time))

newTime = time.time()


print("\n 10 folder corss validation:")

y=y[:,None]


err, final_thetas =
cross_validation(X,y,thetas,learning_rate,covergance_criteria,testNum)

print("--- %s seconds ---" % (time.time() - newTime))


# dataset = pd.read_csv('hepatitis.csv')

dataset = pd.read_csv('bankrupcy.csv')


# plt.hist(dataset["attribute59"], density=True, bins=453)

dataset.describe()

Q1 = dataset.quantile(0.05)

Q3 = dataset.quantile(0.95)

IQR = Q3 - Q1

dataset_out = dataset[~((dataset < (Q1 - 1.5 * IQR)) |(dataset > (Q3 + 1.5 *
IQR))).any(axis=1)]

plt.hist(dataset_out["attribute59"], density=True, bins=453)

print(dataset < (Q1 - 1.5 * IQR))

print(dataset.shape)
print(dataset_out.shape)


print(dataset["sgot"])

plt.hist(dataset["sgot"], density=True, bins=141)


dataset[ (dataset['ClassLabel']==1)].__len__()


dataset[ (dataset['ClassLabel']==0)].__len__()


dataset[ (dataset['ClassLabel']==0) & (dataset['sex'] == 1)].__len__()


dataset[ (dataset['ClassLabel']==0)].__len__()


dataset[ (dataset['ClassLabel']==0) & (dataset['sex'] == 2)].__len__()


y = dataset[dataset['ClassLabel']==1].groupby('sex')['protime'].count()

print(y)

x=['Females','Males']

plt.bar(x,y)

plt.title('Class1 wrt Gender')


y = dataset[dataset['ClassLabel']==0].groupby('age')['protime'].count()
```

```
print(y)

x=dataset[ (dataset['ClassLabel']==0)]['age'].unique()

print(x)

plt.bar(x,y)

plt.title('Class0 wrt Age')

plt.ylim([0,10])

plt.xlim([20,80])


datasetNoFirstColumn = read_data();

print(datasetNoFirstColumn)

print(datasetNoFirstColumn.shape)


datasetNoFirstColumn.describe()




df = pd.read_csv('bankrupcy.csv')

df.corr()


plt.matshow(df.corr())

plt.xticks(range(len(df.columns)), df.columns)

plt.yticks(range(len(df.columns)), df.columns)

plt.colorbar()
plt.show()


import seaborn as sns

# creating set to hold the correlated features

df = pd.read_csv('bankrupcy.csv')

# df = df.to_numpy()

x_train= df.iloc[:,0:-1]

# normalized = normalize(df)

# x_train= normalized[:trainNum,:-1]


corr_features = set()


# create the correlation matrix (default to pearson)

corr_matrix = x_train.corr()


# optional: display a heatmap of the correlation matrix

plt.figure(figsize=(64,64))

sns.heatmap(corr_matrix)


for i in range(len(corr_matrix .columns)):

    for j in range(i):

        if abs(corr_matrix.iloc[i, j]) > 0.8:

            colname = corr_matrix.columns[i]
            corr_features.add(colname)


print(x_train.shape)

x_train.drop(labels=corr_features, axis=1, inplace=True)

print(x_train.shape)
```

Reference

Kendall’s rank correlation coefficient

https://heartbeat.fritz.ai/hands-on-with-feature-selection-techniques-filter-methods-f248e0436ce5

IQR Score

https://www.pluralsight.com/guides/cleaning-up-data-from-outliers

Correlation in python

https://benalexkeen.com/correlation-in-python/