CS542 HW2

CS 542: Machine Learning **Spring 2019**

Problem Set 2

Student: Haoying Zhou *Due: Mar 5, 2019*

1. **(60 points) Written Problems**

(a) (15 points) Bishop 3.3
(b) (15 points) Bishop 3.11
(c) (15 points) Bishop 3.14
(d) (15 points) Bishop 3.21

(a)

3.3 (⋆) Consider a data set in which each data point t_n is associated with a weighting factor $r_n > 0$, so that the sum-of-squares error function becomes

$$E_D(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} r_n \left\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\right\}^2. \tag{3.104}$$

Find an expression for the solution $\mathbf{w}^\star$ that minimizes this error function. Give two alternative interpretations of the weighted sum-of-squares error function in terms of (i) data dependent noise variance and (ii) replicated data points.

Solution:
According to the description of the problem, it can obtain that:

$$E_D(w) = \frac{1}{2}\sum_{n=1}^{N} r_n\{t_n - w^T\phi(x_n)\}^2$$

Then, it can calculate the derivative or gradient of $E_D(w)$ respect to w:

$$\nabla E_D(w) = \sum_{n=1}^{N} r_n\{t_n - w^T\phi(x_n)\}\phi(x_n)^T$$

Set the derivative to zero, then the equation becomes:

$$0 = \sum_{n=1}^{N} r_n\{t_n - w^T\phi(x_n)\}\phi(x_n)^T = \sum_{n=1}^{N} r_n t_n\phi(x_n)^T - w^T(\sum_{n=1}^{N} r_n\phi(x_n)\phi(x_n)^T)$$

To make the above equation look similar to the results of Gaussian function, just let $\Phi(x_n) = \sqrt{r_n}\phi(x_n)$ and $T_n = \sqrt{r_n}t_n$, hence the equation becomes:

$$0 = \sum_{n=1}^{N} T_n \Phi(x_n)^T - w^T(\sum_{n=1}^{N} \Phi(x_n)\Phi(x_n)^T)$$

This is a Gaussian function's form, according to the formula, it can solve that:

$$w = (\Phi^T\Phi)^{-1}\Phi^T T$$

Where $T = \begin{bmatrix} \sqrt{r_1}t_1 \\ \vdots \\ \sqrt{r_n}t_n \end{bmatrix}$ and Φ is a N × M matrix with element $\Phi(\mathrm{i},\mathrm{j}) = \sqrt{r_i}\phi_j(x_i)$

(b)

3.11 (⋆⋆) We have seen that, as the size of a data set increases, the uncertainty associated with the posterior distribution over model parameters decreases. Make use of the matrix identity (Appendix C)

$$\left(\mathbf{M} + \mathbf{v}\mathbf{v}^{\mathrm{T}}\right)^{-1} = \mathbf{M}^{-1} - \frac{\left(\mathbf{M}^{-1}\mathbf{v}\right)\left(\mathbf{v}^{\mathrm{T}}\mathbf{M}^{-1}\right)}{1 + \mathbf{v}^{\mathrm{T}}\mathbf{M}^{-1}\mathbf{v}} \tag{3.110}$$

to show that the uncertainty $\sigma_N^2(\mathbf{x})$ associated with the linear regression function given by (3.59) satisfies

$$\sigma_{N+1}^2(\mathbf{x}) \leqslant \sigma_N^2(\mathbf{x}). \tag{3.111}$$

Solution: Formula (3.59) is given as:

$$\sigma_N^2(\mathrm{x}) = \frac{1}{\beta} + \phi(x)^T S_N \phi(x)$$

To accomplish the whole problem, we need to derive the formula of S_{N+1}:
Since the prior is:

$$\mathrm{p}(\mathrm{w}) = \mathcal{N}(m_N, S_N)$$

Then use it for sample (X_{N+1}, t_{N+1}), the likelihood can be calculated as:

$$p(t_{N+1}|x_{N+1}, w) = \mathcal{N}(t_{N+1}|\mathrm{y}(x_{N+1}, w), \beta^{-1})$$

And the posterior equals to the production of likelihood function and the prior, up to a constant, then mainly focus on the exponential term:

$$exponential\ term = (w - m_N)^T S_N^{-1}(w - m_N) + \beta[t_{N+1} - w^T\phi(x_{N+1})]^2$$

Then after observation, it can get that:

$$\mathrm{p}(\mathrm{w}) = \mathcal{N}(m_{N+1}, S_{N+1})$$

Thus, it can find that:

$$S_{N+1}^{-1} = S_N^{-1} + \beta\phi(x_{N+1})\phi(x_{N+1})^T$$

According to above equation it can get that:

$$S_{N+1} = [S_N^{-1} + \beta\phi(x_{N+1})\phi(x_{N+1})^T]^{-1}$$

And using the matrix identity which is shown in the problem, it can derive that:

$$S_{N+1} = \left[S_N^{-1} + \sqrt{\beta}\phi(x_{N+1})\sqrt{\beta}\phi(x_{N+1})^T\right]^{-1}$$

$$= S_N - \frac{S_N\sqrt{\beta}\phi(x_{N+1})\sqrt{\beta}\phi(x_{N+1})^T S_N}{1+\sqrt{\beta}\phi(x_{N+1})^T S_N\sqrt{\beta}\phi(x_{N+1})} = S_N - \frac{\beta S_N\phi(x_{N+1})\phi(x_{N+1})^T S_N}{1+\beta\phi(x_{N+1})^T S_N\phi(x_{N+1})}$$

And according to formula (3.59), it can obtain that:

$$\sigma_N^2(x) - \sigma_{N+1}^2(x) = \phi(x)^T(S_N - S_{N+1})\phi(x)$$

$$= \phi(x)^T\frac{\beta S_N\phi(x_{N+1})\phi(x_{N+1})^T S_N}{1+\beta\phi(x_{N+1})^T S_N\phi(x_{N+1})}\phi(x) = \frac{[\phi(x_{N+1})^T S_N\phi(x_{N+1})]^2}{\frac{1}{\beta}+\phi(x_{N+1})^T S_N\phi(x_{N+1})}$$

Since S_N is positive definite and $\sigma_N^2(x_{N+1})$ is non-negative, thus $\frac{[\phi(x_{N+1})^T S_N\phi(x_{N+1})]^2}{\frac{1}{\beta}+\phi(x_{N+1})^T S_N\phi(x_{N+1})}$ must be non-negative , which means $\sigma_N^2(x) \geq \sigma_{N+1}^2(x)$

(c)

3.14 ($\star\star$) In this exercise, we explore in more detail the properties of the equivalent kernel defined by (3.62), where $\mathbf{S}_N$ is defined by (3.54). Suppose that the basis functions $\phi_j(\mathbf{x})$ are linearly independent and that the number N of data points is greater than the number M of basis functions. Furthermore, let one of the basis functions be constant, say $\phi_0(\mathbf{x}) = 1$. By taking suitable linear combinations of these basis functions, we can construct a new basis set $\psi_j(\mathbf{x})$ spanning the same space but that are orthonormal, so that

$$\sum_{n=1}^{N}\psi_j(\mathbf{x}_n)\psi_k(\mathbf{x}_n) = I_{jk} \tag{3.115}$$

where I_{jk} is defined to be 1 if $j = k$ and 0 otherwise, and we take $\psi_0(\mathbf{x}) = 1$. Show that for $\alpha = 0$, the equivalent kernel can be written as $k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\psi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\psi}(\mathbf{x}')$ where $\boldsymbol{\psi} = (\psi_1, \ldots, \psi_M)^{\mathrm{T}}$. Use this result to show that the kernel satisfies the summation constraint

$$\sum_{n=1}^{N}k(\mathbf{x}, \mathbf{x}_n) = 1. \tag{3.116}$$

Proof: Since new basis set is orthonormal, then it can obtain that:

$$\psi^T\psi = I_n$$

Where $\psi = \begin{bmatrix}\psi_1 \\ \vdots \\ \psi_M\end{bmatrix}$

According to the formula $S_N^{-1} = \alpha I + \beta\Phi^T\Phi$, and $\alpha = 0$. Then it can derive that: $\beta S_N = 1$

Thus, it can obtain that:

$$k(x,x') = \beta\psi(x)^T S_N \psi(x') = \psi(x)^T \psi(x')$$

(d)

3.21 (⋆⋆) An alternative way to derive the result (3.92) for the optimal value of α in the evidence framework is to make use of the identity

$$\frac{d}{d\alpha}\ln|\mathbf{A}| = \mathrm{Tr}\left(\mathbf{A}^{-1}\frac{d}{d\alpha}\mathbf{A}\right). \tag{3.117}$$

Prove this identity by considering the eigenvalue expansion of a real, symmetric matrix $\mathbf{A}$, and making use of the standard results for the determinant and trace of $\mathbf{A}$ expressed in terms of its eigenvalues (Appendix C). Then make use of (3.117) to derive (3.92) starting from (3.86).

Proof: Since A is real and symmetric and say A has eigenvalues as $\alpha + \lambda_i$
Assume A is a N × N matrix, then i = 1, ⋯ , N
Hence, it can get that:

$$|A| = \prod_{i=1}^{N}(\alpha+\lambda_i), tr(A) = \sum_{i=1}^{N}(\alpha+\lambda_i)$$

And the left side of the equation is:

$$\frac{\mathrm{d}}{\mathrm{d}\alpha}\ln|A| = \frac{\mathrm{d}}{\mathrm{d}\alpha}\ln\prod_{i=1}^{N}(\alpha+\lambda_i) = \sum_{i=1}^{N}\frac{\mathrm{d}}{\mathrm{d}\alpha}\ln(\alpha+\lambda_i) = \sum_{i=1}^{N}\frac{1}{\alpha+\lambda_i}$$

Then, to calculate the right side of the equation:

$$A^{-1}\frac{\mathrm{d}}{\mathrm{d}\alpha}A = A^{-1}I = A^{-1}$$

Since A is symmetric, thus A^{-1} has eigenvalues as $\frac{1}{\alpha+\lambda_i}$

Thus, it can obtain that:

$$\mathrm{tr}\left(A^{-1}\frac{\mathrm{d}}{\mathrm{d}\alpha}A\right) = \sum_{i=1}^{N}\frac{1}{\alpha+\lambda_i}$$

According to above results, it can derive that:

$$\frac{\mathrm{d}}{\mathrm{d}\alpha}\ln|A| = \mathrm{tr}\left(A^{-1}\frac{\mathrm{d}}{\mathrm{d}\alpha}A\right) = \sum_{i=1}^{N}\frac{1}{\alpha+\lambda_i}$$

CS542 HW2 Programming assignment report

2. **(120 points) Programming Report**
   (a) (60 points) Linear Regression
   (b) (60 points) Nearest Neighbors
      i. solution
      ii. solution
      iii. solution
      iv. solution

Files in the zip file:
Report

Five files are for problem (a):
detroit.mat : data source file
hw2_pa.m : main program for running
Normalize.m : function program for normalizing
GDMulti.m : function program for gradient descent
computeCostMulti.m : function program for calculating the cost

Two python files are for problem (b):
hw2_2_1.py : for the first problem of (b)
Be in root folder and run: *python hw2_2_1.py*
hw2_2_2.py: for the rest problems of (b)
Be in root folder and run: *python hw2_2_2.py*
All corresponding data files are in zip file as well, they are very easy to be found out

Most algorithms' meanings are presented in the comment lines, I'll just explain the main idea in this report

(a)
I. Problem

We are given data used in a study of the homicide rate (HOM) in Detroit, over the years 1961-1973. The following data were collected by J.C. Fisher, and used in his paper "Homicide in Detroit: The Role of Firearms," Criminology, vol. 14, pp. 387-400, 1976. Each row is for a year, and each column are values of a variable.

FTP	UEMP	MAN	LIC	GR	NMAN	GOV	HE	WE	HOM
260.35	11.0	455.5	178.15	215.98	538.1	133.9	2.98	117.18	8.60
269.80	7.0	480.2	156.41	180.48	547.6	137.6	3.09	134.02	8.90
272.04	5.2	506.1	198.02	209.57	562.8	143.6	3.23	141.68	8.52
272.96	4.3	535.8	222.10	231.67	591.0	150.3	3.33	147.98	8.89
272.51	3.5	576.0	301.92	297.65	626.1	164.3	3.46	159.85	13.07
261.34	3.2	601.7	391.22	367.62	659.8	179.5	3.60	157.19	14.57
268.89	4.1	577.3	665.56	616.54	686.2	187.5	3.73	155.29	21.36
295.99	3.9	596.9	1131.21	1029.75	699.6	195.4	2.91	131.75	28.03
319.87	3.6	613.5	837.60	786.23	729.9	210.3	4.25	178.74	31.49
341.43	7.1	569.3	794.90	713.77	757.8	223.8	4.47	178.30	37.39
356.59	8.4	548.8	817.74	750.43	755.3	227.7	5.04	209.54	46.26
376.69	7.7	563.4	583.17	1027.38	787.0	230.9	5.47	240.05	47.24
390.19	6.3	609.3	709.59	666.50	819.8	230.2	5.76	258.05	52.33

It turns out that three of the variables together are good predictors of the homicide rate: FTP, WE, and one more variable.

Use methods described in Chapter 3 of the textbook to devise a mathematical formulation to determine the third variable. Implement your formulation and then conduct experiments to determine the third variable. In your report, be sure to provide the step-by-step mathematical formulation (citing Chapter 3 as needed) that corresponds to the implementation you turn in. Also give plots and a rigorous argument to justify the scheme you use and your conclusions.

Note: the file `detroit.mat` containing the data is given on the resources section of our course piazza. To load the data into matlab workspace, use `load()` command. Least-squares linear regression in Matlab can be done with the help of the backslash (\) command.

II. Theory

I will use gradient descent to get the parameters for different features. This has several reasons: firstly, if I use least-square linear regression with some zeroes in denominator, it may come across zero-division error. Secondly, if two features' difference is too large, it's too hard to accomplish

normalization and inverse-normalization when solving normal equation . Finally, gradient descent can solve for the whole set of parameters clearly after some iterations in one time.

The basic function we need to find is:

$$y = w_0 + w_1 x_1 + \cdots + w_9 x_9$$

The above equation is citing chapter 3.

To simplify the problem, assume that $x_0 = z_0 = 1$, so we write the first parameter as itself directly.

To handle the data without influence on numerical difference, the first step is to normalize the data:

$$z_i = \frac{x_i - \mu_i}{\sigma_i}, i = 1,2,\cdots,9$$

Where μ_i, σ_i are means and standard deviations of each feature.
And the above equation becomes:

$$y = \theta_0 + \theta_1 z_1 + \cdots + \theta_9 z_9 = \Theta^T Z$$

Where $\Theta = [\theta_0 \quad \cdots \quad \theta_9], Z = [z_0 \quad \cdots \quad z_9]$

Maybe $\theta s \; and \; ws$ are not identical but they have the same tendency.

To measure whether all the points are in the same lane or all the points are most close to one lane. It is necessary to define a function named as cost function:

$$J(\Theta) = \frac{1}{2m}\sum_{i=1}^{m}(\theta_i z_i)^2$$

Where m is the number of features, in the specific example, is equal to 9.
Then, we need to find Θ by iterations. The change rate is proportional to the derivative of cost function.
The gradient descent algorithm is:
Repeat

$$\{\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\Theta)\} (simultaneously \; update \; for \; every \; j = 0, \cdots, i)$$

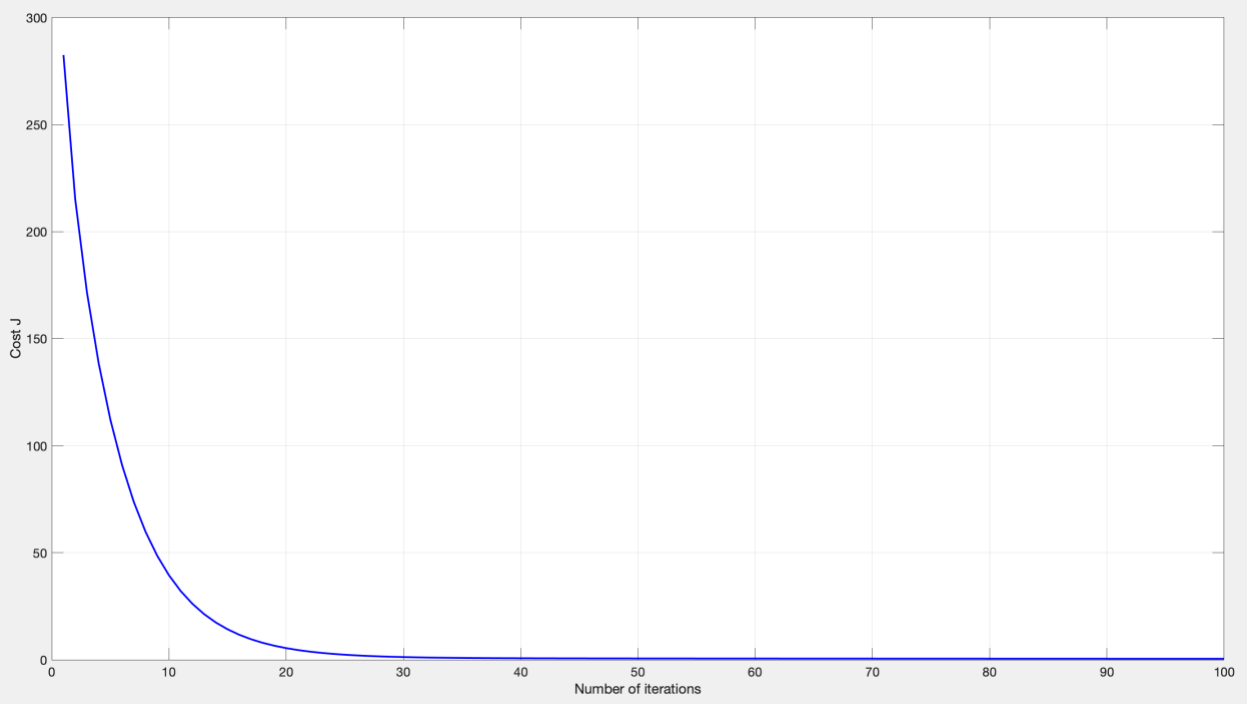
Until θ_js don't change.

α is named as learning rate. Since it is a not a large data set, we could choose learning rate as 0.1 to avoid local maximum. And iteration number

is 100. And we can choose the three largest parameters then we could find the corresponding three features.

III. Results

The code is in MATLAB, I write some basic comments in it.
This is the plot of cost function:



It can see that after 100 times iterations the cost is almost zero.

The main programming file is hw2_pa.m

And Θ is:
Theta computed from gradient descent:
25.126256
3.624453
2.201921
0.094063
3.239165
1.322575
2.308523
2.124974
2.390091
2.745974

IV. Conclusion

Since the first term is constant term, we'll skip the first term. Then the second, last and fifth is the three largest term is corresponded to the first, fourth and last features. The features' names are: *FTP, WE, LIC*

(b)

Problem:

i. Write down (on the written part of your solution) exactly how you imputed missing values for each feature (i.e., replaced all missing values of feature 1 with b) for the CA dataset. Note that you are free to impute the values using statistics over the entire dataset (training and testing combined) or just training, but please state your method. Create a program that can be run on the BU computing environment with the command

ii. Write a k-NN algorithm with L2 distance, $\mathcal{D}_{L2}(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_i (a_i - b_i)^2}$, program that can be run with the commands

respectively where k is an integer indicating the k in your k-NN algorithm. The output of your program should be the testing file with an additional comma-separated field indicating the prediction of your classifier.

iii. Generate a table that reports the accuracy on both data sets for at least two different values of k in each case. I have included a small script that you can use to calculate the accuracy and check if your code works (the numbers are made up just for instructional purposes).

i. For all string-typed missing elements, take the median of the feature's dataset as backup because if I have to choose from median and mean, the mean will never make sense. For all int-typed or float -typed elements, take the mean of the feature's dataset as backup because it won't change the mean of the dataset. For all missing elements, the positive and negative are recovered separately: positive to positive, negative to negative.

ii. The basic algorithm of k-NN is that find k nearest neighbor for a specific data point then decide its label which is + or - . For my voting strategy, take the label which has larger amount; If there is a tie, then return the label of the nearest one. And the distance of string-typed feature is that if they

are the same the value is 0, otherwise it's 1. Normalize the data at the very beginning.

iii.

The value of k	Accuracy in %
1	82.6087
2	82.6087
3	85.5072
4	86.9565
5	84.0580
6	86.2319
7	84.7826
8	85.5072
9	85.5072
10	87.6812

Actually I run k from 1 to 30, the highest one is k = 28/30, accuracy = 89.1304%

CS542 HW2 Source code

hw2_2_1.py:

```
import numpy as np

''' read the files'''
def getlist(filename):
	read_list = []
	with open(filename) as f:
		for line in f:
			line_data = line.strip()
			line_data = line.split(',')
			read_list.append(line_data)
	return read_list

'''median letter of letters'''
def midletter(lst):
	sortedlst = sorted(lst)
	index = (len(lst)-1)//2
	return sortedlst[index]

'''get rid of '?''''
def dequestion(lst):
	outlst = [[] for _ in range(len(lst))]
	for i in range(len(lst)):
		for j in range(len(lst[i])):
			if lst[i][j] == '?':
				continue
			else:
				outlst[i].append(lst[i][j])
	return outlst

'''change str into int or float if possible'''
def mknum(lst):
	outlst = [[] for _ in range(len(lst))]
	for i in range(len(lst)):
```

```
            for j in range(len(lst[i])):
                try:
                    outlst[i].append(int(lst[i][j]))
                except:
                    try:
                        outlst[i].append(float(lst[i][j]))
                    except:
                        outlst[i].append(lst[i][j])
        return outlst

'''backup digit  eg.  a = '1', s = a.zfill(5)'''

'''find the replace element'''
def findr(lst):
        outlst = []
        for i in range(len(lst)):
            if (type(lst[i][1]) == int) or (type(lst[i][1]) == float) :
                outlst.append(np.mean(lst[i]))
            else:
                outlst.append(midletter(lst[i]))
        return outlst

'''modify the results of replacement element'''
def modfr(lst):
        outlst = []
        outlst.append(lst[0])
        outlst.append(str(round(lst[1],2)))
        outlst.append(str(round(lst[2],2)))
        outlst.append(lst[3])
        outlst.append(lst[4])
        outlst.append(lst[5])
        outlst.append(lst[6])
        outlst.append(str(round(lst[7],3)))
        outlst.append(lst[8])
        outlst.append(lst[9])
```

```
        a = str(int(lst[10]))
        outlst.append(a.zfill(2))
        outlst.append(lst[11])
        outlst.append(lst[12])
        b = str(int(lst[13]))
        outlst.append(b.zfill(5))
        outlst.append(str(int(lst[14])))
        return outlst


'''main function'''
def process(filename):
        #use entire dataset as source
        sourcefl = 'crx.data'
        source_list = getlist(sourcefl)
        # read the data first
        read_list = getlist(filename)
        # generate output list
        out_list = [[] for _ in range(len(read_list))]
        # initialize lists
        # source lists
        n = len(source_list[15])
        sflsts = [[] for _ in range(n)]
        sflsts_p = [[] for _ in range(n-1)]
        sflsts_n = [[] for _ in range(n-1)]
        resflsts_p = []
        resflsts_n = []
        results_p = []
        results_n = []
        # loaded lists
        rflsts = [[] for _ in range(n)]
        # input loaded data into empty lists
        for i in range(len(read_list)):
                for j in range(n):
                        rflsts[j].append(read_list[i][j])
```

```
# input source data into empty lists
for i in range(len(source_list)):
	for j in range(n):
		sflsts[j].append(source_list[i][j])
# get data for positive and negative sign
for i in range(len(source_list)):
	for j in range(n-1):
		if source_list[i][15] == '+\n':
			sflsts_p[j].append(source_list[i][j])
		else:
			sflsts_n[j].append(source_list[i][j])

# get rid of '?'
sflsts_p = dequestion(sflsts_p)
sflsts_n = dequestion(sflsts_n)
# change into int or float if possible
sflsts_p = mknum(sflsts_p)
sflsts_n = mknum(sflsts_n)
# find replace elements
resflsts_p = findr(sflsts_p)
resflsts_n = findr(sflsts_n)
# modify the form
results_p = modfr(resflsts_p)
results_n = modfr(resflsts_n)


'''main part'''
for i in range(len(rflsts)):
	for j in range(len(rflsts[i])):
		if rflsts[i][15] == '+\n':
			if rflsts[i][j] == '?':
				out_list[j].append(results_p[i])
			else:
				out_list[j].append(rflsts[i][j])
		else:
```

```
                        if rflsts[i][j] == '?':
                                out_list[j].append(results_n[i])
                        else:
                                out_list[j].append(rflsts[i][j])

        # ready for writing
        outlst = []
        for i in range(len(out_list)):
                for j in range(len(out_list[i])):
                        if j == 15:
                                outlst.append(out_list[i][j])
                        else:
                                outlst.append(out_list[i][j] + ',')

        # write into file
        newfile = open(filename[0:3] + filename[8:] + '.processed','w')
        for k in range(len(outlst)):
                newfile.write(outlst[k])
        newfile.close()

# actions when running the python file
if __name__ == '__main__':
        process('crx.data.training')
        process('crx.data.testing')
```

hw2_2_2.py:

```
import numpy as np

''' read the files'''
def getlist(filename):
	read_list = []
	with open(filename) as f:
		for line in f:
			line_data = line.strip()
			line_data = line.split(',')
			read_list.append(line_data)
	return read_list

''' modify the last term'''
def mdflt(lst):
	outlst = [[] for _ in range(len(lst))]
	for i in range(len(lst)):
		for j in range(len(lst[i])):
			if j == len(lst[i])-1:
				outlst[i].append(lst[i][-1][0])
			else:
				outlst[i].append(lst[i][j])
	return outlst

'''change str into int or float if possible'''
def mknum(lst):
	outlst = [[] for _ in range(len(lst))]
	for i in range(len(lst)):
		for j in range(len(lst[i])):
			try:
				outlst[i].append(int(lst[i][j]))
			except:
				try:
					outlst[i].append(float(lst[i][j]))
				except:
```

```
                                outlst[i].append(lst[i][j])
        return outlst

'''find the mean'''
def findm(lst):
        outlst = []
        for i in range(len(lst)):
                try:
                        outlst.append(float(np.mean(lst[i])))
                except:
                        outlst.append(0)
        return outlst

'''find the standard deviation'''
def findstd(lst):
        outlst = []
        for i in range(len(lst)):
                try:
                        outlst.append(float(np.std(lst[i])))
                except:
                        outlst.append(0)
        return outlst

'''normalize the data'''
def nmlz(means, stds, lst):
        outlst = [[] for _ in range(len(lst))]
        for i in range(len(lst)):
                for j in range(len(lst[i])):
                        if type(lst[i][j]) == int or type(lst[i][j]) == float:
                                outlst[i].append((lst[i][j]-means[j])/stds[j])
                        else:
                                outlst[i].append(lst[i][j])
        return outlst

''' add number as final list'''
```

```
def addo(lst):
	outlst = [[] for _ in range(len(lst[0]))]
	for j in range(len(lst[0])):
		for i in range(len(lst)+1):
			if i == len(lst):
				outlst[j].append(j)
			else:
				outlst[j].append(lst[i][j])
	return outlst

''' calculate the D2 distance'''
def caldis(slst,rlst):
	outlst = [[] for _ in range(len(rlst))]
	for i in range(len(rlst)):
		for j in range(len(slst)):
				if i == len(rlst)-1:
					#a = 0
					outlst[-1].append(slst[j][-1])
				elif type(rlst[i]) == str:
					if rlst[i] == slst[j][i]:
						outlst[i].append(0)
					else:
						outlst[i].append(1)
				else:
					outlst[i].append(rlst[i]-slst[j][i])
	return outlst

''' calculate sum'''
def calsum(lst):
	innerlst = [[],[]]
	outlst = []
	for i in range(len(lst[0])):
		nsum = 0
		for j in range(len(lst)-1):
			nsum += lst[j][i]*lst[j][i]
```

```
            innerlst[0].append(float(np.sqrt(nsum)))
            innerlst[1].append(lst[-1][i])
    outlst = addo(innerlst)
    return outlst

'''voting strategy'''
def votf(k, lst):
    ranklst = [[] for _ in range(len(lst[0]))]
    sortlst = []
    prelst = [[] for _ in range(len(lst))]
    outlst = []
    for i in range(len(lst[0])):
        for j in range(len(lst)):
            ranklst[i].append(lst[j][i])
    sortlst = sorted(ranklst[0])
    for n in range(len(sortlst)):
        for m in range(len(sortlst)):
            if n!=m:
                if sortlst[n] == sortlst[m]:
                    print('have same sum')
            if sortlst[n] == ranklst[0][m]:
                prelst[n].append(ranklst[0][m])
                prelst[n].append(ranklst[1][m])
                prelst[n].append(ranklst[2][m])
    if k == 1:
        outlst.append(prelst[0][1])
    elif k <= 0:
        print('error')
    else:
        posno = 0
        negno = 0
        for i in range(k):
            if prelst[i][1] == '+':
                posno += 1
            else:
```

```
                        negno += 1
            if posno > negno:
                outlst.append('+')
            elif posno < negno:
                outlst.append('-')
            else:
                outlst.append(prelst[0][1])
        outlst = str(outlst[0])
        return outlst

'''calculate the accuracy'''
def acur(rlst,slst):
        if len(rlst) != len(slst):
                print('Input error!')
        outp = 0
        totalnum = len(rlst)
        right_value = 0
        for i in range(len(rlst)):
                if rlst[i] == slst[i][-1]:
                        right_value += 1
        outp = (right_value/totalnum)*100
        return outp


'''main function: k is the k-NN's k, and sourcename as well as file
name are as their names'''
def kNNC(k,sourcename,filename):
        # get source name
        source_list = getlist(sourcename)
        # read the data first
        read_list = getlist(filename)
        # get rid of '\n'
        source_list = mdflt(source_list)
        read_list = mdflt(read_list)
```

```
# generate output list
out_list = [[],[]]
# initialize lists
# source lists
n = len(source_list[0])
# source lists
sflsts = [[] for _ in range(n)]
nsflsts = []
# other sources
rmeans = []
smeans = []
rstds = []
sstds = []
numrl = []
numsl = []
results = []
# loaded lists
rflsts = [[] for _ in range(n)]
nrflsts = []
### make the data to be its transpose
# input loaded data into empty lists
for i in range(len(read_list)):
        for j in range(n):
                rflsts[j].append(read_list[i][j])
# input source data into empty lists
for i in range(len(source_list)):
        for j in range(n):
                sflsts[j].append(source_list[i][j])
# modify lists
rflsts = mknum(rflsts)
sflsts = mknum(sflsts)
numrl = mknum(read_list)
numsl = mknum(source_list)
# calculate means and stds
rmeans = findm(rflsts)
```

```
        smeans = findm(sflsts)
        rstds = findstd(rflsts)
        sstds = findstd(sflsts)
        # normalize the data
        nrflsts = nmlz(rmeans, rstds, numrl)
        nsflsts = nmlz(smeans, sstds, numsl)
        # generate empty list for saving results
        showlst = []
        # kNN algorithm
        for i in range(len(nrflsts)):
                results = caldis(nsflsts, nrflsts[i])
                #results = modfstr(strno, results)
                results = calsum(results)
                results = votf(k, results)
                showlst.append(results)
        # calculate the accuracy
        accuracy = acur(showlst, read_list)
        # store the data into list
        out_list.append(showlst)
        out_list.append(accuracy)
        #####print out the results#####

        print('According to kNN algorithm, the labels in testing
dataset should be:')
        for i in range(len(showlst)-1):
                print(showlst[i],end = ', ')
        print(showlst[-1])
        print('The accuracy of kNN algorithm is:', end = '    ')
        print(str(round(accuracy,4))+' %')
        return out_list


# actions when running .py file
if __name__ == '__main__':
        kNNC(1,'crx.training.processed','crx.testing.processed')
```

MATLAB files

```
%% Initialization
clear all;
close all;
clc;
fprintf('Loading data ...\n');

%% Load data
input = load('detroit.mat');
X = input.data(:,1:9);
y = input.data(:,10);
m = length(y);

fprintf('Loaded, program paused. Press enter to continue.\n');
pause;

%% Normalize data

fprintf('Normalizing Features ...\n');

[X mu sigma] = Normalize(X);

% Add intercept term to X
X = [ones(m,1) X];

%% Gradient Descent

fprintf('Running gradient descent...\n');

% Choose some alpha value
alpha = 0.1;
num_iters = 100;

% Initialize theta and run gradient descent
theta = zeros(size(X,2),1);
[theta, J_old] = GDMulti(X, y, theta, alpha, num_iters);

% Plot the convergence graph
figure
plot(1:numel(J_old), J_old, '-b', 'LineWidth', 2);
xlabel('Number of iterations');
ylabel('Cost J');
grid on

% Display gradient descent's result
fprintf('Theta computed from gradient descent:\n');
fprintf(' %f \n', theta);
fprintf('\n');



function [X_norm, mu, sigma] = Normalize(X)
% Normalizes the features in X
```

```
%   Normalize(X) returns a normalized version of X where
%   the mean value of each feature is 0 and the standard deviation
%   is 1. This is often a good preprocessing step to do when
%   working with learning algorithms.


mu = zeros(1, size(X, 2));
sigma = zeros(1, size(X, 2));

mu = mean(X);
sigma=std(X);
X_norm = (X-mu)./sigma;


end



function [theta, J_old] = GDMulti(X, y, theta, alpha, num_iters)
%Performs gradient descent to learn theta
%   theta = GDMulti(x, y, theta, alpha, num_iters) updates theta by
%   taking num_iters gradient steps with learning rate alpha

% Initialize some useful values
m = length(y); % number of training examples
J_old = zeros(num_iters, 1);
temp = zeros(length(theta),1);

for iter = 1:num_iters



    Jac = zeros(length(theta),1);
    for j = 1:m
        Jac = Jac + (1/m)*(X(j,:)*theta-y(j))*X(j,:)';
    end
    temp = theta;
    theta = theta - alpha*Jac;
    if theta == temp
        break
    end


    % Save the cost J in every iteration
    J_old(iter) = computeCostMulti(X, y, theta);

end

end

function J = computeCostMulti(X, y, theta)
%Compute cost for linear regression with multiple variables
%   J = COMPUTECOSTMULTI(X, y, theta) computes the cost of using theta as the
%   parameter for linear regression to fit the data points in X and y
```

```
% Initialize some useful values
m = length(y); % number of training examples

% You need to return the following variables correctly
J = 0;


for i=1:m
    J = J +(1/(2*m))*(X(i,:)*theta-y(i))'*(X(i,:)*theta-y(i));
end



end
```