1. (a) $f(x) = e^x - 1$, $x \in \mathbb{R}$.

$\nabla^2 f(x) = e^x > 0$.

strictly convex

(b) $\nabla^2 f(x_1, x_2) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

$x^T \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} x = 2x_2 x_1 > 0$.

since $\begin{bmatrix} -\sqrt{2} & \sqrt{2} \\ \sqrt{2} & \sqrt{2} \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -\sqrt{2} & \sqrt{2} \\ \sqrt{2} & \sqrt{2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

Not concave or convex

(c) $$\nabla^2 f(x_1, x_2) = \begin{bmatrix} a(a-1)x_1^{a-2} x_2^{1-a}, & a x_1^{a-1}(1-a) x_2^{-a} \\ (1-a) x_2^{-a} a x_1^{a-1}, & (1-a)(-a) x_2^{-a-1} x_1^{a} \end{bmatrix}$$

$$= a(a-1) x_1^{a-1} x_2^{-a} \begin{bmatrix} x_1^{-1} x_2^{1}, & -1 \\ -1, & x_1^{1} x_2^{-1} \end{bmatrix}$$

$$= a(a-1) x_1^{a-1} x_2^{-a} \begin{bmatrix} x_1^{-1} x_2^{1} \\ -1 \end{bmatrix} \begin{bmatrix} 1 & -x_1^{1} x_2^{-1} \end{bmatrix}$$

$\text{eig} \begin{bmatrix} x_1^{-1} x_2 & -1 \\ -1 & x_1 x_2^{-1} \end{bmatrix} = 0$. and it is PSD.

$f(x_1, x_2)$ is concave.

1. (b)
(I) $\nabla_\theta J(\theta) = \nabla_{(\theta)} \frac{1}{2} \|A^T\theta - y\|^2 = \nabla_{(\theta)} \frac{1}{2} (A^T\theta - y)^T (A^T\theta - y)$

$= \nabla_{(\theta)} \frac{1}{2} [\theta^T A A^T \theta - 2\theta^T A y]$

$= AA^T\theta - Ay = A(A^T\theta - y)$

$\therefore \theta^{t+1} = \theta^t - \alpha \nabla_\theta J(\theta^{(t)})$

(II) $J(\theta)$ is convex $\Rightarrow \nabla_\theta J(\theta) = 0$.

$AA^T\theta - Ay = 0.$

$\cancel{\theta^* = (A^T)^* y}$

$AA^T\theta = Ay$

$\theta^* = (AA^T)^{-1} A y = (A^T)^+ y = (A^T)^+ y$

$\theta^* = (A^T)^+ y.$

2. X_n = gold denotes nth phone is purchased by i as Rose Gold.

X_n = gray denotes nth phone - - - - Space Gray.

$$P(X_{n+1} = \text{gold} \mid X_1, X_2 \dots X_n \text{ has } m \text{ gold phone}) = \frac{50-m}{100-n}$$

$\neq$ ~~[illegible]~~

$P(X_{n+1} = \text{gold} \mid$ there are gold. observed $)$

$$= \sum_{m=0}^{m=(k \le 50)} P(X_{n+1} = \text{gold} \mid \text{observe } n, X_1 \sim X_n \text{ has } m \text{ gold phone}) * P(X_1 \sim X_n \text{ has } m \text{ gold phone})$$

$$= \frac{\sum_m (50-m)\binom{50}{m}\binom{50}{k-m}}{\binom{100}{n}(100-n)} = \frac{1}{2}$$

This means whatever the number of gold phone was purchased by the previous customer. The ~~nex~~ probability of successfuly predicting the next one will carry a gold phone out will be $\frac{1}{2}$. So there is no chance to write an algrithm out.

3.

1) The error is 0.01625
2) tokens that are most indicative of the SPAM class `['httpaddr', 'spam', 'unsubscrib', 'ebai', 'valet']`
3) For the other datasets:

Some other datasets
Data size: 50; error: 0.03875
Data size: 100; error: 0.02625
Data size: 200; error: 0.02625
Data size: 400; error: 0.01875
Data size: 800; error: 0.0175
Data size: 1400; error: 0.01625

The code I implemented is shown as below(language == Python):

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Oct 12 13:56:53 2017

@author: liuchangbai
"""

import csv
import os
import numpy as np
import pandas as pd
import nb


os.chdir("/Users/liuchangbai/Desktop/courses/Machine-Learning/Homework/HW2")
token_list = []
word_list = []
token_path = os.path.expanduser('./spam_classification/TOKENS_LIST')

with open(token_path,newline='') as token:
  reader = csv.reader(token, delimiter=' ')
  for row in reader:
    token_list.append(row)

for i in token_list:
```

```
    word_list.append(i[1])


train_path = os.path.expanduser('./spam_classification/SPARSE.TRAIN')
with open(train_path, newline='') as train:
    reader = csv.reader(train, delimiter=' ')
    for row in reader:
        label.append(int(row[0]))
label = np.asarray(label,dtype=int)

count_d_w = np.zeros([nd,nw],dtype=int)
with open(train_path, newline='') as train:
    reader = csv.reader(train, delimiter=' ')
    for d_id, row in enumerate(reader):
        current_email = csv.reader(row[2:-1],delimiter=':')
        for rows in current_email:
            w_id = int(rows[0])
            count = int(rows[1])
            count_d_w[d_id][w_id-1] = count

df_train = pd.DataFrame(count_d_w, columns = [word_list])
df_train["label"] = pd.Series(label)



label_test_buf = list()
test_path = os.path.expanduser('./spam_classification/SPARSE.TEST')
with open(test_path, newline='') as test:
    reader = csv.reader(test, delimiter=' ')
    for row in reader:
        label_test_buf.append(int(row[0]))
label_test = np.asarray(label_test_buf,dtype=int)

nd_test = len(label_test)
count_d_w_test = np.zeros([nd_test,nw],dtype=int)
with open(test_path, newline='') as test:
    reader = csv.reader(test, delimiter=' ')
    for d_id, row in enumerate(reader):
        current_email = csv.reader(row[2:-1],delimiter=':')
        for rows in current_email:
            w_id = int(rows[0])
            count = int(rows[1])
            count_d_w_test[d_id][w_id-1] = count




df_test = pd.DataFrame(count_d_w_test)
nb_model = nb.train(df_train)
```

```
nb_predictions = nb.test(nb_model, df_test)
y = pd.Series(label_test)
nb_error = nb.compute_error(y, nb_predictions)
print('NB Test error: {}'.format(nb_error))

words = nb.k_most_indicative_words(5, nb_model.to_dataframe().iloc[:,:-1])
print('The {} most spam-worthy words are: {}'.format(len(words), words))
```

4. (1)
L2-norm:
k = 1: accuracy = 94%
k = 5: accuracy = 98%
k = 9: accuracy = 96%
k = 13: accuracy = 96%
(2)
L1-norm:
k = 1: accuracy = 92%
k = 5: accuracy = 96%
k = 9: accuracy = 95%
k = 13: accuracy = 94%

It seems L2 has better accuracy than L1-norm. And when k = 5 has the best accuracy.
I think it is because this matrix is a sparse matrix and L2-norm magnifies big entries, so that it could be better than L1-norm.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Oct 12 19:52:44 2017

@author: liuchangbai
"""

import os
import numpy as np
import scipy.io as sio
from sklearn import cross_validation, neighbors
import operator

os.chdir("/Users/liuchangbai/Desktop/courses/Machine-Learning/Homework/HW2")

mnist_data = sio.loadmat('mnist_data.mat')
```

```
train_array = np.array(mnist_data['train'])
test_array = np.array(mnist_data['test'])

random_numbers = np.random.choice(10000,50)
test_data = test_array[random_numbers]



def KNN(feature, predict, k):
    X = feature
    y = predict
    X_train, X_test, y_train, y_test = cross_validation.train_test_split(X,y,test_size = 0.2)

    classifier = neighbors.KNeighborsClassifier()
    classifier.fit(X_train,y_train)

    accuracy = classifier.score(X_test, y_test)
    return accuracy


def l1distance(instance1, instance2, length):
    distance = 0
    for x in range(1,length+1):
        distance += abs(instance1[x]-instance2[x])
    return distance

 def l2distance(instance1, instance2, length):
    distance = 0
    for x in range(1,length+1):
        distance+=pow((instance1[x]-instance2[x]),2)
    return distance

def Neighbors(trainingSet, testInstance, k):
    distance=[]
    length = len(testInstance)-1
    for x in range(len(trainingSet)):
        dist = euclideanDistance(testInstance, trainingSet[x], length)
        distance.append((trainingSet[x],dist))
    distance.sort(key=operator.itemgetter(1))
    neighbors=[]
    for x in range(k):
        neighbors.append(distance[x][0])
    classVotes={}
```

```
    for x in range(len(neighbors)):
        response = neighbors[x][0]
        if response in classVotes:
            classVotes[response] += 1
        else:
            classVotes[response] = 1
    sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1),reverse=True)
    return sortedVotes[0][0]

def getAccuracy(testSet, predictions):
    correct=0
    for x in range(len(testSet)):
        if testSet[x][0]==predictions[x]:
            correct+=1
    return (correct/float(len(testSet))) *100


K = 5 #{1,5,9,13}

for x in range(len(test_data)):
    result = Neighbors(train_array, test_data[x], K)
    predictions.append(result)

accuracy = getAccuracy(test_data, predictions)
```