1) Optimization: Prove whether or not the following fxns are convex, concave, strictly convex or strictly concave.

i) $f(x) = e^x - 1$, $x \in \mathbb{R}$. Strictly convex.
$f'(x) = e^x$. This is monotonically increasing on $\mathbb{R}$
($e^y > e^x$ iff $y > x$), so $f(x)$ is strictly convex.

ii) $f(x_1, x_2) = x_1 x_2 \quad x_1, x_2 \in \mathbb{R}^2_{++}$.

$\mathbb{R}^2_{++}$ is a convex set, so we may find
$f((1-\lambda)x + \lambda x')$ and $(1-\lambda)f(x) + \lambda f(x')$, where $x = (x_1, x_2)$, $x' = (x_1', x_2')$.

$f((1-\lambda)x_1 + \lambda x_1', (1-\lambda)x_2 + \lambda x_2') =$
$((1-\lambda)x_1 + \lambda x_1')((1-\lambda)x_2 + \lambda x_2') =$
$(1-\lambda)^2 x_1 x_2 + \lambda^2 x_1' x_2' + \lambda(1-\lambda)(x_1 x_2' + x_1' x_2)$.

similarly, $(1-\lambda)f(x) + \lambda f(x') = (1-\lambda)x_1 x_2 + \lambda x_1' x_2'$.
$f((1-\lambda)x + \lambda x') - ((1-\lambda)f(x) + \lambda f(x')) =$
$((1-\lambda)^2 - (1-\lambda))x_1 x_2 + (\lambda^2 - \lambda)x_1' x_2' + \lambda(1-\lambda)(x_1 x_2' + x_1' x_2)$
$= -\lambda(1-\lambda)(x_1 x_2 + x_1' x_2') + \lambda(1-\lambda)(x_1 x_2' + x_1' x_2)$
$= \lambda(1-\lambda)(x_1 - x_1')(x_2' - x_2)$

We note that whether or not this is ≥ 0 or ≤ 0 depends on x_1 and x_2, i.e., we cannot conclude that

$(1-\lambda)f(x) + \lambda f(x') \geq f((1-\lambda)x + \lambda x')$
or $(1-\lambda)f(x) + \lambda f(x') \leq f((1-\lambda)x + \lambda x')$.

Thus, f is neither convex nor concave.
This makes sense, as f is a hyperbolic paraboloid.

iii) $f(x_1, x_2) = x_1^{\alpha} x_2^{1-\alpha}$ for $\alpha \in [0,1]$ $x_1, x_2 \in \mathbb{R}^{++}$

We calculate the Hessian.

$$\frac{\partial f}{\partial x_1} = \alpha x_1^{\alpha-1} x_2^{(1-\alpha)} \qquad \frac{\partial f}{\partial x_2} = (1-\alpha) x_1^{\alpha} x_2^{-\alpha}$$

$$\frac{\partial^2 f}{\partial x_2 \partial x_1} = \frac{\partial^2 f}{\partial x_1 \partial x_2} = \alpha(1-\alpha) x_1^{\alpha-1} x_2^{-\alpha}, \quad \frac{\partial^2 f}{\partial x_1^2} = \alpha(\alpha-1) x_1^{\alpha-2} x_2^{(1-\alpha)}$$

$$\frac{\partial^2 f}{\partial x_2^2} = -\alpha(1-\alpha) x_1^{\alpha} x_2^{-\alpha-1}.$$

so

$$\nabla^2 f = \alpha(\alpha-1) \begin{pmatrix} x_1^{\alpha-2} x_2^{1-\alpha} & -x_1^{\alpha-1} x_2^{-\alpha} \\ -x_1^{\alpha-1} x_2^{-\alpha} & x_1^{\alpha} x_2^{-\alpha-1} \end{pmatrix}$$

$$= \alpha(1-\alpha) x_1^{\alpha-2} x_2^{-\alpha-1} \begin{pmatrix} x_2^2 & -x_1 x_2 \\ -x_1 x_2 & x_1^2 \end{pmatrix}$$

So Eigenvalues of the matrix $M = \begin{pmatrix} x_2^2 & -x_1 x_2 \\ -x_1 x_2 & x_1^2 \end{pmatrix}$ satisfy

Let $u = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \in \mathbb{R}^{++}$. Then $u^T M u =$

$$(u_1\ u_2) \begin{pmatrix} x_2^2 & -x_1 x_2 \\ -x_1 x_2 & x_1^2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = (u_1\ u_2) \begin{pmatrix} x_2^2 u_1 - x_1 x_2 u_2 \\ -x_1 x_2 u_1 + x_1^2 u_2 \end{pmatrix}$$

$$= x_2^2 u_1^2 - x_1 x_2 u_1 u_2 - x_1 x_2 u_1 u_2 + x_1^2 u_2^2$$

$$= (x_2 u_1 - x_1 u_2)^2$$ which is clearly ≥ 0 $\forall\ u \in \mathbb{R}_{++}$.

Thus, M is positive semidefinite, meaning $\nabla^2 f$ is negative semidefinite, so f is concave.

b) The cost fxn for Linear Regression is given by $J(\theta) = \frac{1}{2} \|A^T\theta - y\|^2$ where A is the design matrix and y is the target vector.

i) Derive the gradient $\nabla_\theta J(\theta)$ and give the expression of gradient descent.

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{\ell} \left(\sum_{j=1}^{k} A^T_{ij} \theta_j - y_i \right)^2$$

So $$\frac{\partial J}{\partial \theta_m} = \frac{1}{2} \sum_{i=1}^{\ell} 2 A^T_{im} \left(\sum_{j=1}^{k} A^T_{ij} \theta_j - y_i \right)$$

$$= \sum_{i=1}^{k} A^T_{im} (A^T\theta - y)_i = \sum_{i=1}^{k} A_{mi} (A^T\theta - y)_i$$

So $$\nabla J(\theta) = A(A^T\theta - y).$$

Then the update rule for gradient descent is

$$\theta^{(t+1)} = \theta^{(t)} - \alpha A(A^T\theta^{(t)} - y). \quad \square$$

ii) Find the expression for θ that minimizes J.

We see that J is at a minimum when $\nabla J = 0$, i.e., when $A(A^T\theta - y) = 0$

multiply on the left by A^{-1}: $A^T\theta - y = 0$.

So

$$A^T\theta = y, \quad \text{i.e.,} \quad \theta = (A^T)^{-1} y = (A^{-1})^T y. \quad \square$$

② Consider a challenge in which a mathematician offers an iphone if you can correctly guess which patrons exiting a store have a 'rose gold' phone. Construct or disprove an algorithm that has a better win chance than $1/2$.

No such algorithm exists.

We are given that P(last person has a rose gold phone) to be $1/2$. Clearly, we can also tell that the probability the <u>first</u> person has a rose gold phone is $1/2$.

We appeal to symmetry: suppose we stop and guess that person $i+1$ has a rose gold phone r having observed a previous sequence of phones $x_1, x_2, x_3 \ldots x_i$. Then the probability we are right is

$$P(x_1, x_2 \ldots x_i r ?? \ldots \mid x_1, x_2 \ldots x_i ???)$$

where we have used ? to denote unknown outcomes.

However, by symmetry, we see that

$$P(x_1 x_2 \ldots x_i r ?\ldots? \mid x_1 x_2 \ldots x_i ?\ldots?) =$$
$$P(x_1 x_2 \ldots x_i ? \ldots r \mid x_1 x_2 \ldots x_i ?\ldots?)$$

Though we may know how many rose-gold phones <u>remain</u> in the store, we have no way of knowing how they are ordered in the line, and all orderings are a priori equally likely.

Thus,

$$P(x_1 x_2 \ldots x_i r ?\ldots?) = \sum_{\substack{\text{all possible} \\ \text{sequences} \\ \text{of the first} \\ i \text{ phones}}} P(x_1 x_2 \ldots x_i r ?\ldots? \mid x_1 x_2 \ldots x_i ?\ldots?) P(x_1 x_2 \ldots x_i ?\ldots?)$$

$$= \sum_{\substack{\text{all possible} \\ \text{sequences of the first} \\ i \text{ phones}}} P(x_1 x_2 \ldots x_i ? \ldots ? r) P(x_1 x_2 \ldots x_i ???) = P(x_1 x_2 \ldots x_i \ldots ? ? r)$$

But this last expression is simply

the probability that we will win if we wait until the end, which we are given is $1/2$.

Thus, if we decide to stop at <u>any</u> point in the line, the probability we will win the challenge is the same: $1/2$. Thus, no viable algorithm exists. $\square$

3a) I construct a naive Bayes classifier to filter spam emails and report the error for a number of training sets. My code is below:

```
import numpy as np
import heapq

probwordgivspam, probwordgivgood, spamsum, goodsum, logprobs = [0]*1448, [0]*1448, [0]*1448,
[0]*1448, [0]*1448
numspam = 0
numgood = 0

with open('SPARSE.TRAIN.1400') as f:
#get labels, calculate frequency sums
 for i, line in enumerate(f):
  label = line.split('  ')[0]
  text = line.split('  ')[1]
  for cell in text.strip().split(' '):
   word, freq = cell.split(':')
   if int(label) == 1:
    spamsum[int(word)-1] += int(freq)
    numspam += 1
   if int(label) == -1:
    goodsum[int(word)-1] += int(freq)
    numgood += 1

 #get probability of good & spam
probspam = numspam / (numspam + numgood)
probgood = numgood / (numspam + numgood)

 #use frequency sums to get conditionals
for i in range(0, 1448):
 probwordgivspam[i] = (1 + spamsum[i])/(len(spamsum)+sum(spamsum))
 probwordgivgood[i] = (1 + goodsum[i])/(len(goodsum)+sum(goodsum))
 logprobs[i] = np.log(probwordgivspam[i]/probwordgivgood[i])

with open('SPARSE.TEST') as g:
 numdocs = 0
 misclass = 0
#get labels
 for i, line in enumerate(g):
  numdocs += 1
  guesslabel = 0
  logpost = np.log(probspam/probgood)
  testwords = []
  testlabel = line.split('  ')[0]
  testtext = line.split('  ')[1]
  for cell in testtext.strip().split(' '):
   testword, testfreq = cell.split(':')
   testwords.append(int(testword)-1)
```

```
  for test in testwords:
   logpost += np.log(probwordgivspam[test]/probwordgivgood[test])
  if (logpost > 0):
   guesslabel = 1
  elif (logpost < 0):
   guesslabel = -1
  if (int(testlabel) != guesslabel):
   misclass += 1

error = (misclass/numdocs)*100
print("error is")
print(error)
print("spammiest words are at")

spammiest = heapq.nlargest(5, range(len(logprobs)), logprobs.__getitem__)
print(spammiest + np.ones(len(spammiest)))
```

Training on the SPARSE.TRAIN.50 sample gives an error of 1.875%.
Training on the SPARSE.TRAIN.100 sample gives an error of 1.375%.
Training on the SPARSE.TRAIN.200 sample gives an error of 1.125%.
Training on the SPARSE.TRAIN.400 sample gives an error of 1.25%.
Training on the SPARSE.TRAIN.800 sample gives an error of 1.375%.
Training on the SPARSE.TRAIN.1400 sample gives an error of 1.25%.

b) Using the probability measures calculated, we find the words most indicative of spam. Training on the SPARSE.TRAIN.1400 dataset, we find that the ‘spammiest’ words are ‘httpaddr’, ‘spam’, ‘unsubscrib’, ‘ebai’, and ‘diploma’.

4. Construct a K-Nearest Neighbors classifier for the MNIST dataset.

My code is as follows:

```
“import numpy as np
import scipy.io as sio
from collections import Counter

mnist_data = sio.loadmat('mnist_data.mat')
train_data = mnist_data['train']
test_data = mnist_data['test']

train_images = np.asarray(train_data[:,1:785])
train_labels = np.asarray(train_data[:,0])
test_images = np.asarray(test_data[:,1:785])
test_labels = np.asarray(test_data[:,0])

errorcount = 0
klist = [1, 5, 9, 13]
for k in klist:
 for i in range(100):
  randnum = np.random.choice(test_labels.size)
  test_image = test_images[randnum]
  test_label = test_labels[randnum]
  distances = [(np.linalg.norm(test_image - image), label) for (image, label) in zip(train_images,
train_labels)]
  distsort = sorted(distances, key = lambda tup: tup[0])
  k_labels = [label for (_, label) in distsort[0:k]]
  win_label, freq = Counter(k_labels).most_common()[0]
  if (int(win_label) != int(test_label)):
   errorcount += 1
 print('for k = ', end =' ')
 print(k, end =' ')
 print(', the error is ', end =' '))
 print(errorcount/100)”
```

Sample output is:

for k = 1 , the error is 0.02
for k = 5 , the error is 0.03
for k = 9 , the error is 0.07
for k = 13 , the error is 0.11

We thus conclude that k=9 gives the best classification accuracy.

b) Consider the L1-Norm, and choose between L1- and L2-normed classifiers.

I change the distance metric from L2-Norm to L1-Norm. This is simply done by changing the “np.linalg.norm(test_image – image)” to “np.linalg.norm((test_image – image), 1). Doing this gives the following output:

for k = 1 , the error is 0.03
for k = 5 , the error is 0.08
for k = 9 , the error is 0.12
for k = 13 , the error is 0.15

We note that for all values of k in our set, the L2-Norm produces a lower error, so we select this as our distance measure.