University of Toronto Mississauga
Department of Mathematical and Computational Sciences
CSC 411 - Machine Learning and Data Mining, Fall 2018

Assignment 2

Due date: Monday November 12, 11:59pm.
No late assignments will be accepted.

As in all work in this course, 20% of your grade is for quality of presentation, including the use of good English, properly commented and easy-to-understand programs, and clear proofs. In general, short, simple answers are worth more than long, complicated ones. Unless stated otherwise, all answers should be justified. The TA has a limited amount of time to devote to each assignment, so what you hand in should be legible (either typed or *neatly* hand-written), well-organized and easy to evaluate. (An employer would demand no less.) All computer problems are to be done in Python with the NumPy, SciPy and scikit-learn libraries.

Hand in five files: The source code of all your programs (functions and script) in a single Python file, a pdf file of figures generated by the progams, a pdf file of all printed output, a pdf file of answers to all the non-programming questions (scanned hand-writing is fine, but *not* photographs), and a scanned, signed copy of the cover sheet at the end of the assignment. Be sure to indicate clearly which question(s) each program and piece of output refers to. All the Python code (functions and script) for a given question should appear in one location in your source file, along with a comment giving the question number. All material in all files should appear in order; *i.e.*, material for Question 1 before Question 2 before Question 3, etc. It should be easy for the TA to identify the material for each question. In particular, all figures should be titled, and all printed output should be identified with the Question number. The five files should be submitted electronically as described on the course web page. In addition, if we run your source file, it should not produce any errors, it should produce all the output that you hand in (figures and print outs), and it should be clear which question each piece of output refers to. *Output that is not labled with the Question number will not be graded. Programs that do not produce output will not be graded.*

Style: Use the solutions to Assignment 1 as a guide/model for how to present your solutions to Assignment 2.

I don't know policy: If you do not know the answer to a question (or part), and you write "I don't know", you will receive 20% of the marks of that question (or part). If you just leave a question blank with no such statement, you get 0 marks for that question.

No more questions will be added.

Tips on Scientific Programming in Python

If you haven't already done so, please read the NumPy tutorial on the course web page. Be sure to read about indexing and slicing Numpy arrays. First, indexing begins at 0, not 1. Thus, if `A` is a matrix, then `A[7,0]` is the element in row 7 and column 0. Likewise, `A[0,4]` is the element in row 0 and column 4. Slicing allows large segments of an array to be referenced. For example, `A[:,5]` returns column 5 of matrix `A`, and `A[7,[3,6,8]]` returns elements 3, 6 and 8 of row 7. Similarly, if `v` is a vector, then the statement `A[6,:]=v` copies `v` into row 6 of matrix `A`. Note that if `A` and `B` are two-dimensional Numpy arrays, then `A*B` performs *element-wise* multiplication, *not* matrix multiplication. To perform matrix multiplication, you should use `numpy.matmul(A,B)`. Whenever possible, *do not use loops*, which are very slow in Python. In particular, avoid iterating over the elements of a large vector or matrix. Instead, use numpy's vector and matrix operations, which are much faster and can be executed in parallel. For example, if `A` is a matrix and `v` is a column vector, the `A+v` will add `v` to every column of `A`. Likewise for rows and row vectors. Also, the functions `sum` and `mean` in `numpy` are useful for summing or averaging over all or part of an array. Many NumPy functions that are defined for single numbers can be passed lists, vectors and matrices instead. For example, $f([x_1, x_2, ..., x_n])$ returns the list $[f(x_1), f(x_2), ..., f(x_n)]$. The same is true for many user-defined functions. The term `numpy.inf` represents infinity. It results from dividing by 0 in numpy. It can also result from overflow (*i.e.*, from numbers that are too large to represent in the computer, like 10^{1000}). The term `numpy.nan` stands for "not a number", and it results from doing 0/0, inf/inf or inf-inf in numpy. For generating and labelling plots, the following SciPy functions in `matplotlib.pyplot` are needed: `plot`, `xlabel`, `ylabel`, `title`, `suptitle` and `figure`. You can use Google to conveniently look up SciPy functions. e.g., you can google "numpy matmul" and "pyplot suptitle".

1. (10 points) *Data Generation.*

   In this question you will generate and plot 2-dimensional data for a binary classification problem. We will call the two classes Class 0 and Class 1 (for which the target values are $t = 0$ and $t = 1$, respectively).

   (a) Write a Python function `genData(mu0,mu1,Sigma0,Sigma1,N)` that generates two clusters of data, one for each class. Each cluster consists of `N` data points. The cluster for class 0 is centred at `mu0` and has covariance matrix `Sigma0`. The cluster for class 1 is centred at `mu1` and has covariance matrix `Sigma1`. Note that `mu0` and `mu1` and all the data points are 2-dimensional vectors. `Sigma0` and `Sigma1` are 2×2 symmetric matrices that describe the shape of the clusters: the diagonal entries specify the variance of a cluster along each of the two dimensions, and the off-diagonal entries describe how correlated the two dimensions are.

   The function should return two arrays, `X` and `t`, representing data points and target values, respectively. `X` is a $2N \times 2$ dimensional array in which each row is a data point. `t` is a $2N$-dimensional vector of 0s and 1s. Specifically, `t[i]` is 0 if `X[i]` belongs to class 0, and 1 if it belongs to class 1. The data for the two classes should be distributed randomly in the arrays. In particular, the data for class 0 should not all be in the first half of the arrays, with the data for class 1 in the second half.

   We will model each cluster as a multivariate normal distribution. Recall that the probability density of such a distribution is given by

$$P(x) = \frac{\exp\left[-(x-\mu)^T \Sigma^{-1} (x-\mu)/2\right]}{\sqrt{(2\pi)^k \det \Sigma}}$$

   where μ is the mean (cluster centre), Σ is the covariance matrix, and k is the dimensionaliy of the data (2 in our case). To generate data for a cluster, use the function `multivariate_normal` in `numpy.random`. Use the function `shuffle` in `sklearn.utils` to distribute the data randomly in the arrays.

   (b) Use your function from part (a) to generate two clusters with 10,000 points each with `mu0` $= (0, -1)$, `mu1` $= (-1, 1)$ and

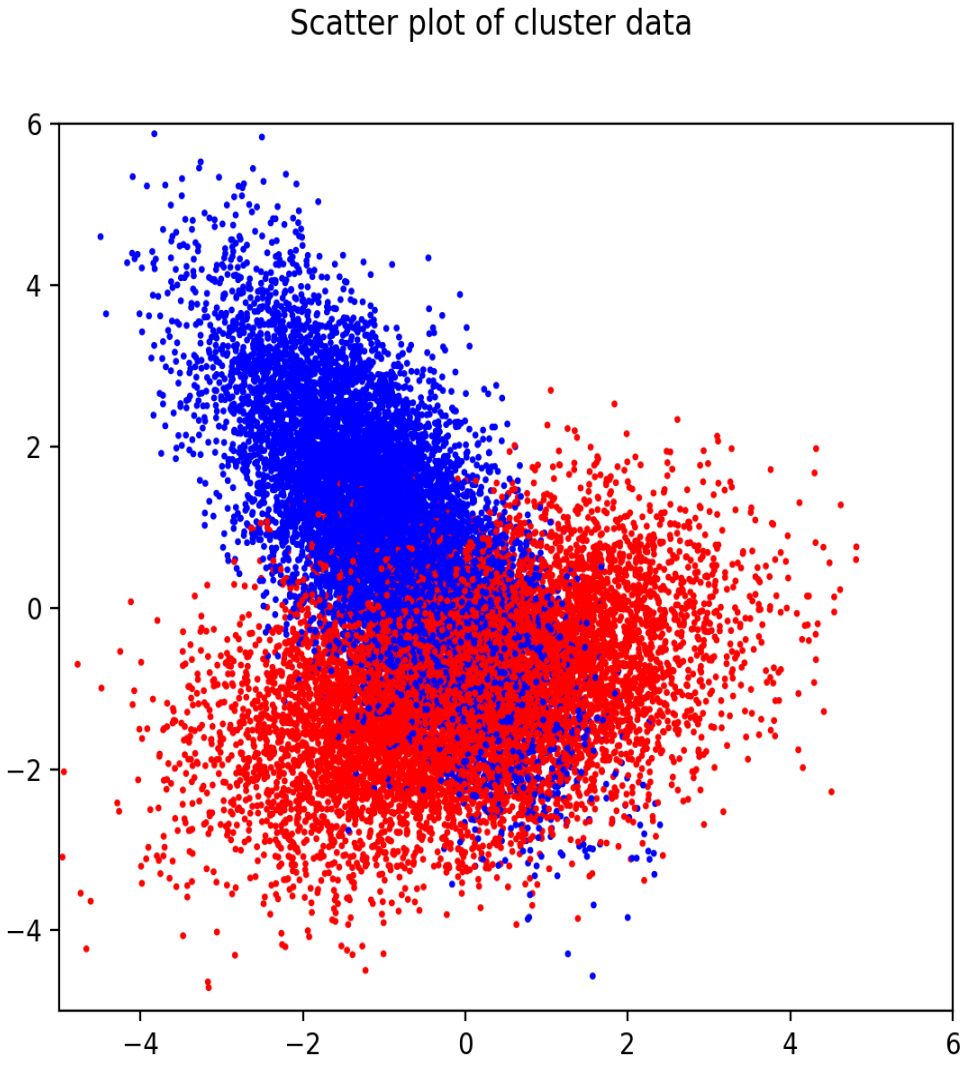
$$\texttt{Sigma0} = \begin{pmatrix} 2.0 & 0.5 \\ 0.5 & 1.0 \end{pmatrix} \qquad \texttt{Sigma1} = \begin{pmatrix} 1.0 & -1.0 \\ -1.0 & 2.0 \end{pmatrix}$$

   You will have to encode these argument values as Numpy arrays.

   (c) Display the data from part (b) as a scatter plot, using red dots for points in cluster 0, and blue dots for points in cluster 1. Use the function `scatter` in `numpy.pyplot`. Specify a relatively small dot size by using the named argument `s=2`. Use the functions `xlim` and `ylim` to extend the x and y axes from -5 to 6. Title the plot, "Question 1(c): sample cluster data (10,000 points per cluster)".

   If you have done everything correctly, the scatter plot should look something like Figure 1, which shows two heavily overlapping clusters. In particular, the

Figure 1:



red cluster should not be displayed on top of the blue cluster (or vice versa). Instead, the red and blue dots should be intermingled just below the center of the figure, which is a result of distributing the data points randomly in the arrays. To increase the detail in the plot, put the figure into full-screen mode before saving it. Hand in the plot.

2. (61 points total) *Binary Logistic Regression.*

   In this question you will use logistic regression to generate a classifier for cluster data. You will also generate a precision-recall curve for the classifier. Use the Python class `LogisticRegression` in `sklearn.linear_model` to do the logistic regression. This class generates a Python object, much as the function `Ridge` did in Question 5 of Assignment 1. The class comes with a number of attributes and methods that you will

find useful for answering the questions below. Do not use any other functions from `sklearn`.

(a) (0 points) Use `genData` to generate training data consisting of two clusters with 1000 points each. Use the same cluster centers and covariance matrices as in Question 1(b).

(b) (5 points) Carry out logistic regression on the data in part (a). Print out the values of the bias term, w_0, the weight vector, w, and the mean accuracy of the classifier on the training data. (Accuracy is the number of correct predictions.)

(c) (5 points) Generate a scatter plot of the taining data as in Question 1(c), and draw the decision boundary of the classifier as a black line on top of the data. Title the figure, "Question 2(c): training data and decision boundary".

(d) (4 points) Recall that the standard decision boundary tends to make the number of false positives equal to the number of false negatives. However, these two kinds of error may have different costs, and we may want to shift the decision boundary to account for this. That is, instead of defining the decision boundary by $w^T x + w_0 = 0$, we may want to define it by $w^T x + w_0 = t$ for some threshold, t.

Generate a scatter plot of the training data, and plot seven different decision boundaries on top of it, for $t = 3, 2, 1, 0, -1, -2, -3$. Plot the decision boundary as a blue line when t is positive, as a red line when t is negative, and as a black line when t is 0. Title the figure, "Question 2(d): decision boundaries for seven thresholds".

(e) (5 points) Which of the seven values of t in part (d) gives the greatest number of false positives (*i.e.*, false blue predictions)? Provide a geometric interpretation of your answer.

(f) (5 points) For $t = 1$, what is the probability of a point on the decision boundary being in class 1 (*i.e.*, blue). This is a paper-and-pencil problem.

(g) (0 points) Use `genData` to generate test data consisting of two clusters with 10,000 points each. Use the same cluster centers and covariance matrices as for the training data.

(h) (17 points total) Use the test data to compute and print out the following values for $t = 1$ (1 point each):

- The number of predicted positives (*i.e.*, points predicted to be in class 1)
- The number of predicted negatives (*i.e.*, points predicted to be in class 0)
- The number of true positives (*i.e.*, predictions for class 1 that are correct).
- The number of false postives (*i.e.*, predictions for class 1 that are incorrect)
- The number of true negatives (*i.e.*, predictions for class 0 that are correct)
- The number of false negatives (*i.e.*, predictions for class 0 that are incorrect).
- The precision.
- The recall.

The number of predicted positives should be less than the number of predicted negatives. The number of true positives should be much greater than the number of false positives. *Provide a geometric interpretaation of both these points*, generating an appropriate figure to simplify your answer (10 points). Title the figure, "Question 2(h): explanatory figure".

(i) (5 points) Use the test data to generate a precision/recall curve for the classifier. That is, plot precision vs recall for 1000 different values of the threshold, t. You should choose the range of t values so that the curve is as long as possible. You should find that $0.5 \leq$ precision ≤ 1 and $0 \leq$ recall ≤ 1. The result should look something like Figure 2 (although the minimum precision in this curve is different). Label the axes, and title the figure, "Question 2(i): precision/recall curve".

(j) (5 points) Provide a geometric interpretation of why the minimum precision is 0.5.

(k) (5 points) Compute and print the area under the precision/recall curve. The area should be between 0.5 and 1.0. (Recall that the area under a curve (AUC) is the area between the curve and the x axis.) Do not use any operations other than multiplication and addition. (In fact, you do not need any Numpy operations at all.) You may use 1 loop.

(l) (5 points) Provide a geometric interpretation of why the area under the curve must be between 0.5 and 1.0. (You may want to include a figure in your answer. If so, title it, "Question 2(l): explanatory figure".)

3. (16 points total) *Multi-class Classification.*

In this question, you will use logistic regression and K nearest neighbors (KNN) to classify images of handwritten digits. There are ten different digits (0 to 9), so you will be using multi-class classification.

To start, download and uncompress (if necessary) the MNIST data file from the course web page. The file, called `mnist.pickle.zip`, contains training and test data. Next, start the Python interpreter and import the `pickle` module. You can then read the file `mnist.pickle` with the following command (`'rb'` opens the file for reading in binary):

```
with open('mnist.pickle','rb') as f:
    Xtrain,Ytrain,Xtest,Ytest = pickle.load(f)
```

The variables `Xtrain` and `Ytrain` contain training data, while `Xtest` and `Yest` contain test data. Use this data for training and testing in this question and in the rest of this assignment.

`Xtrain` is a Numpy array with 60,000 rows and 784 columns. Each row represents a hand-written digit. Although each digit is stored as a row vector with 784 components, it actually represents an array of pixels with 28 rows and 28 columns ($784 = 28 \times 28$). Each pixel is stored as a floating-point number, but has an integer value between 0 and 255 (i.e., the values representable in a single byte). The variable `Ytrain` is a vector of

Figure 2:

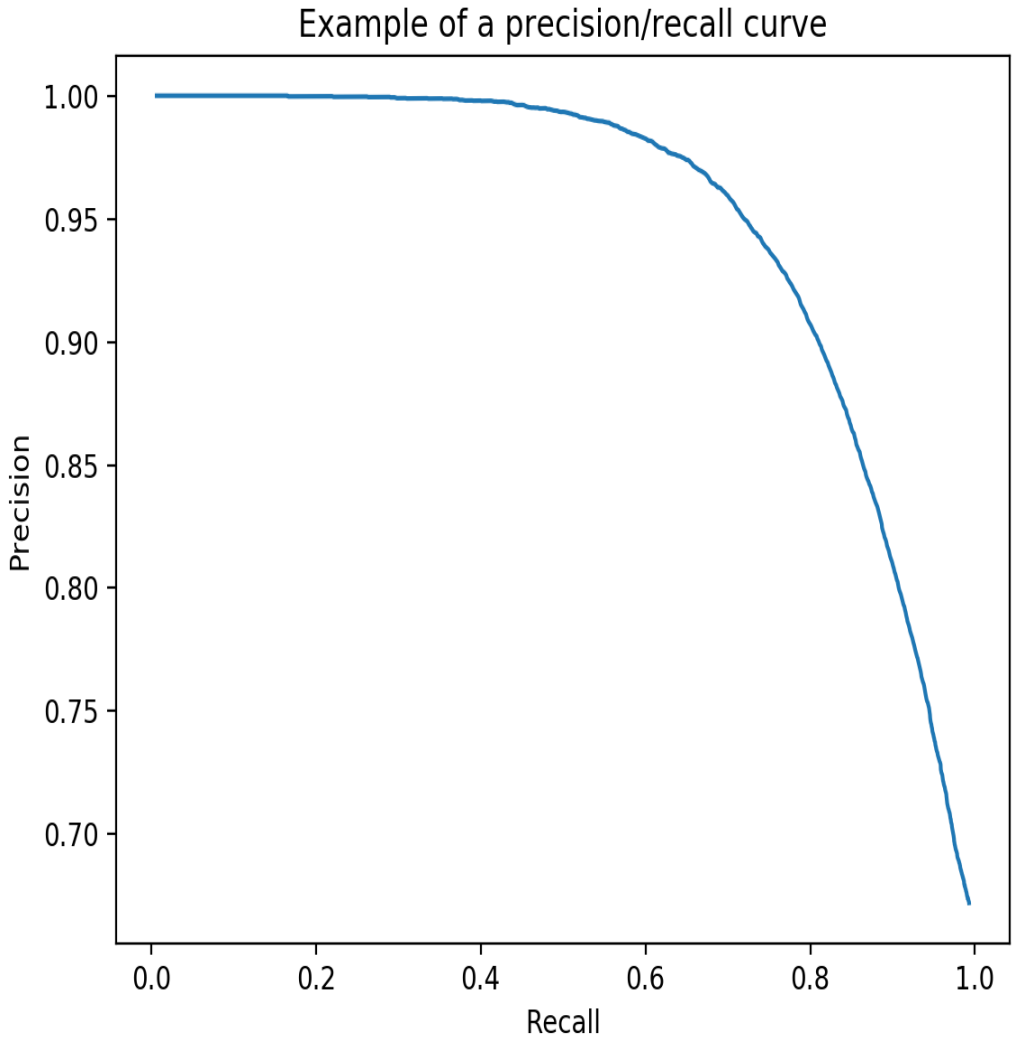
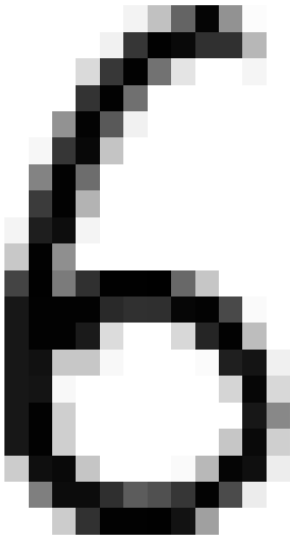

Example of a precision/recall curve
Precision
Recall
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.0
0.2
0.4
0.6
0.8
1.0

Figure 3:
An MNIST image



60,000 image labels, where a label is an integer betwen 0 and 9. For example, if row `n` of `Xtrain` is an image of the digit 7, then `Ytrain[n]` = 7. Likewise for `Xtest` and `Ytest`, which represent 10,000 test images.

To view a digit, you must first convert it to a 28×28 array using the function `numpy.reshape`. To display a 2-dimensional array as an image, you can use the function `imshow` in `matplotlib.pyplot`. To see an image in black-and-white, add the keyword argument `cmap='Greys'` to `imshow`. To remove the smoothing and see the 784 pixels clearly, add the keyword argument `interpolation='nearest'`. Try displaying a few digits as images. (Figure 3 shows an example.) For comparison, try printing them as vectors. (Do not hand this in.)

What to do: Write Python programs to carry out the following tasks:

(a) (5 points) Choose 36 MNIST images at random (without replacement) and display them in a single figure, arranged in a 6×6 grid. Turn off the axes in each image using the function `matplotlib.pyplot.axis`. The images should be in black-

and-white and should not use any smoothing. Title the figure, "Question 3(a): 36 random MNIST images."

(b) (4 points) Train a logistic-regression classifier on the MNIST training data and print out the training accuracy and the test accuracy. The following code defines the correct kind of multi-class classifier:

```
import sklearn.linear_model as lin
clf = lin.LogisticRegression(multi_class='multinomial',solver='lbfgs')
```

In particiular, the resulting classifier, `clf`, will *not* use a one-vs-one or one-vs-the-rest scheme. Instead, it will use a separate, linear discriminant function for each class, a softmax function to convert the discriminant values to probabilities, and a cross-entropy loss function (as described in class). You should fit the classifier to the training data, and then compute and print its mean accuracy (score) on the training data and on the test data. Print the mean accuracy as a percent. If you have done everything correctly, the test accuracy should be a little less than the training accuracy, and both should be a little bit bigger than 90%. Training the classifier may take a little time (about 15 seconds on my laptop), but computing the accuracy should be quite fast (less than a second).

(c) (7 points) Train and test a KNN classifier on the MNIST data using `KNeighborsClassifier` in `sklearn.neighbors`. Specify the 'brute' algorithm, which does not build any fancy data structures for storing the training data, but does a simple brute-force search through the data to find the nearest neighbours. Do this for K = 1,2,...,20. For each value of K, compute the mean test accuracy. Plot the mean test accuracies (as percentages), with K on the horizontal axis. The plot should be quite jagged, especially for small values of K. Label the axes, and title the plot, 'Figure 3(c): KNN test accuracy'. Print out the best value of K. You should find that fitting a KNN classifier to the training data is very fast, but computing the test accuracy is quite slow (about 12 seconds on my laptop for each value of K). You should also find that KNN has better test accuracy than logistic regression for all 20 values of K.

If the testing phase seems to take forever, then your computer may not have enough memory. In this case, you can try running your program on one of the lab machines, which have 8GB of memory each. Alternatively, you can divide the test data into ten groups of 1,000 points each. Then evaluate the test accuracy for each group separately, giving ten different accuracies. (You may use a loop for this.) Finally, take the average of the ten accuracies. This should require only one tenth of the memory.

4. (25 points total) *Multi-class Logistic Regression: Theory.*

In this question, each data point has the form $(x^{(n)}, t^{(n)})$ where $x^{(n)}$ is a real vector and $t^{(n)}$ is a binary vector, i.e., a 1-of-K encoding of the class of $x^{(n)}$. We let $t_k^{(n)}$ denote the k^{th} component of vector $t^{(n)}$. The loss function is then given by the cross entropy:

$$E = -\sum_{nk} t_k^{(n)} \log y_k^{(n)} \tag{1}$$

where the sum is over all training points, n, and all classes, k. Here, $y_k^{(n)} = \exp z_k^{(n)} / \sum_j \exp z_j^{(n)}$ where $z_j^{(n)} = w_j^T x^{(n)} + w_{j0}$. Intuitively, $y_k^{(n)}$ is the predicted probability that input $x^{(n)}$ is in class k. Let $y^{(n)}$ denote the vector whose k^{th} component is $y_k^{(n)}$. This is a vector of class predictions (probabilities) for $x^{(n)}$. By minimizing cross entropy, logistic regression tries to make these predictions as close as possible to the target vectors, $t^{(n)}$.

To make your proofs below easier to mark, use the indices m and n to range over training instances, use j and k to range over classes, and use i to range over features of an input vector, x.

(a) (13 points) Prove the second-last equation on slide 14 of Lecture 7:

$$\partial E / \partial z_k^{(n)} \;=\; y_k^{(n)} - t_k^{(n)}$$

You may use the third last-equation in your proof:

$$\partial y_j^{(n)} / \partial z_k^{(n)} \;=\; \delta_{jk} y_j^{(n)} - y_j^{(n)} y_k^{(n)} \tag{2}$$

where $\delta_{kj} = 1$ if $k = j$, and 0 otherwise.

(b) (9 points) Let W be a matrix whose k^{th} column is the vector w_k, and let X, T and Y be matrices whose n^{th} rows are the vectors $x^{(n)}$, $t^{(n)}$, and $y^{(n)}$, respectively. Prove the following equation for the gradient of the cross entropy:

$$\partial E / \partial W \;=\; X^T (Y - T)$$

Note that the gradient is a matrix, not a vector. You may use the last equation on slide 14 of Lecture 7 in your proof:

$$\partial E / \partial w_{ki} \;=\; \sum_n [y_k^{(n)} - t_k^{(n)}] x_i^{(n)} \tag{3}$$

where w_{ki} and $x_i^{(n)}$ are the i^{th} components of vectors w_k and $x^{(n)}$, respectively.

(c) (3 points) Without going through a separate proof, argue that

$$\partial E / \partial w_{k0} \;=\; = \sum_n [y_k^{(n)} - t_k^{(n)}]$$

5. (23 points total) *Softmax.*

To implement multi-class logistic regression, we must first implement the softmax function. Recall that the softmax function normalizes a vector, z, by converting it into a probability distribution. Specifically, if z is a K-dimensional vector and $y = \text{softmax}(z)$, then y is a K-dimensional vector where

$$y_k \;=\; \frac{\exp z_k}{\sum_{j=1}^K \exp z_j} \tag{4}$$

Notice that $y_k > 0$, so we should always be able to evaluate $\log(y_k)$ in Equation (1), so the cross entropy is well-defined.

Unfortunately, the obvious implementation of softmax causes numerical problems that make it unusable. This question illustrates these problems and points to a solution.

(a) (3 points) Write a function `softmax1(z)` that computes the softmax function of vector `z`. You should implement this function in the obvious way, using at most 3 lines of Python code. (This is also called the naive implementation.)

Test your function by evaluating `softmax1((0,0))`, `softmax1((1000,0))` and `log(softmax1((-1000,0)))`, and printing out the answers. If you have done everything correctly, then the first function call should should return `(0.5,0.5)`, but the second should cause warnings and return `(nan,0)`, and the third should cause a warning and return `(-inf,0)`. The last problem means that we cannot always evaluate the cross entropy in Equation (1).

The first problem is due to numerical overflow. That is, `exp(1000)` is too large to represent in the computer, so it evaluates to the special symbol `inf`, representing infinity. Evaluating the first component of `softmax1((1000,0))` then results in the division `inf/inf`, which evaluates to `nan`.

The second problem is due to numerical underflow. That is, `exp(-1000)` is too small to represent in the computer, so it is truncated to 0, so the first component of `softmax1((-1000,0))` is also 0. When we try to compute the log of the softmax, the first component becomes `-inf`.

(b) (15 points total) The two problems illustrated in part (a) are common in machine learning (and in most fields that use numerical methods). The two points below outline ways for solving these problems. Your job here is to complete the solution. These are pencil-and-paper problems.

i. (8 points total) Transform vector z into z' where $z'_k = z_k - \max(z)$, where $\max(z)$ is the maximum element in z. Your job here is to prove that $\text{softmax}(z') = \text{softmax}(z)$ (3 points), and to show that computing $\text{softmax}(z')$ does not cause overflow (5 points). Hint: there are three main causes of overflow.

ii. (7 points total) The second problem (underflow) only becomes an issue when we want to compute $\log y_k$. This can be solved if we can find a way of computing $\log y_k$ without computingq $\exp z_k$ in the numerator of equation (4) (and without causing an underflow problem elsewhere). Your job here is to find a way of doing this. You should prove that your method is correct, i.e., that it computes $\log(y_k)$ (3 points), and explain why the method does not have underflow problems (4 points). Hint: there are two main causes of underflow problems.

(c) (5 points) Write a Python function `softmax2(z)` that computes the softmax function of vector `z` as well as the log of the softmax. That is, the function should return two vectors, `y` and `logy`, where `y` = softmax(`z`) and `logy` = log[softmax(`z`)]. Test your function by using it to evaluate softmax((0, 0)), softmax((1000,0)) and log[softmax((-1000,0))]. Print out the answers. If you have done everything correctly, then there should be no warnings and the answers should all be vectors of real numbers. You should be able to write `softmax2` in at most 7 lines of Python code.

6. (30 points) *Batch Gradient Descent.*

In this question, you will use the theory developed in Question 4 and the ideas developed in Question 5 to implement multi-class logistic regression using batch gradient descent. Because this is batch training, each gradient computation uses all the training data, and each step of gradient descent (gradient computations and weight updates) is called one epoch of training. You will test your programs on the MNIST data, though they should work on any data set that fits the description at the beginning of Question 4. Except where otherwise specified, use only `numpy` functions in this question, and not functions in `sklearn`. Note that you will have to convert the target values, t, from the integers used in Question 3 to the 1-of-K encoding used in Question 4.

We will use the mean cross-entropy per data point, E/N, as the loss function, where E is the cross entropy defined by the sum in Equation (1), and N is the number of data points in the sum. (We use the mean instead of the sum itself so that we can compare the test loss to the training loss, which are based on different numbers of points.) You will also be evaluating the mean accuracy of the classifier. This is the proportion of examples that the classifier correctly predicts. Recall that for an input, x, the predicted class of x is the value of k that gives the largest probability, y_k, where $y_k = \exp z_k / \sum_j \exp z_j$ and $z_k = w_k^T x + w_{k0}$. We say that $k = \text{argmax}_j(y_j)$, i.e., k is the value of j that maximizes y_j. (See the Numpy function `argmax`.)

Your program should do the following:

(a) Use the function `randn` in `numpy.random` to initialize the weights using random numbers from a Gaussian distribution with a mean of 0 and a standard deviation of 0.01. Initialize the bias terms to be all zero.

(b) Using the MNIST data, repeatedly update the weights and bias terms using 5,000 steps (epochs) of batch gradient descent. (This may take several hours, so you may want to run it over night.)

(c) Every ten epochs, compute and record the training loss, test loss, mean training accuracy and mean test accuacy. (You may want to print these out during program development, to monitor the execution of your program, but do not hand in these print outs.)

(d) After the 5,000 epochs have finished, print the learning rate, the final training loss and accuracy, and the final test loss and accuracy. Accuracies should be printed as percentages. You should find that training accuracy > test accuracy, and training loss < test loss.

(e) Plot the history of training and test accuracies in a single graph. Use blue for the test accuracy and orange for the training accuracy. Put epoch on the horizonal axis using a log scale. Label the axes appropriately, and label the figure, "Question 6(e): training and test accuracy for batch gradient descent." The accuracies should increase more-or-less smoothly from left to right.

(f) In a separate figure, plot the history of the training and test losses. Use blue for the test loss and orange for the training loss. Put epoch on the horizonal axis using a log scale. Label the axes appropriately, and label the figure, "Question

6(f): training and test loss for batch gradient descent." The losses should decrease smoothly from left to right.

(g) Plot the last 200 training accuracies (but not the test accuracies). Label the axes appropriately, and label the figure, "Question 6(g): training accuracy for last 2000 epochs of bgd." The accuracies should be very jagged but should tend to increase from left to right.

(h) Plot the last 200 training losses (but not the test losses). Label the axes appropriately, and label the figure, "Question 6(h): training loss for last 2000 epochs of bgd." The losses should decrease smoothly from left to right.

Experiment with the learning rate, and find a value that leads to a test accuracy that, after 5,000 epochs, is at least as good as in Question 3(b). Try learning rates that differ by an order of magnitude, e.g., 100, 10, 1, 0.1, 0.01, 0.001, etc. If the losses or accuracies tend to go up and down in waves, then the learning rate is too large. Likewise if any of them are `nan` or `inf`. However, if the learning rate is too small, then the accuracies will increase too slowly and will not be high enough enough after 5,000 epochs. You may have to run the program several times (with each complete run taking several hours) to find a good learning rate, so try to decide early in a run if the accuracies are increasing fast enough.

7. (20 points total) *Stochastic Gradient Descent.*

Modify your program in Question 6 to peform stochastic gradient descent with mini-batches. That is, instead of computing the gradient of the loss function on the entire training set at once, compute the gradient on a small, random subset of the training data (called a mini-batch), perform weight updates, and then move on to the next mini-batch, and so on. As you will see, this can lead to much faster convergence.

To produce random mini-batches of the training data, first shuffle the training data randomly, then sweep across the shuffled data from start to finish. For example, if we want mini-batches of size 100, then the first mini-batch is the first 100 points in the training set. The second mini-batch is the next 100 points. The third mini-batch is the next 100 points, etc. (If the number of training points is not a multiple of 100 then the last mini-batch in a sweep will have fewer than 100 points in it.) Each such sweep of the training data is called an epoch.

Your program should be similar to the one in Question 6, but with the following differences:

(a) Use the variable `batchSize` for the size of a mini-batch.

(b) Perform 500 epochs of training, instead of 5,000.

(c) During each epoch, perform a step of gradient descent (gradient computation and weight updates) one mini-batch at a time.

(d) Compute and record the training and test loss and the training and test accuracy after every epoch. (Use the full data sets for this, not mini-batches.)

(e) After the 500 epochs have finished, plot the history of the training and test accuracies in a single graph. Use blue for the test accuracy and orange for the training accuracy. Put epoch on the horizonal axis using a log scale. Label the axes appropriately, and label the figure, "Question 7(e): training and test accuracy for stochastic gradient descent." The training accuracy should increase from left to right. However, the testing accuracy should first increase and then level off or even decrease. The curves will be a bit jagged, especially the testing curve.

(f) In a separate figure, plot the history of the training and test losses. Use blue for the test loss and orange for the training loss. Put epoch on the horizonal axis using a log scale. Label the axes appropriately, and label the figure, "Question 7(f): training and test loss for stochastic gradient descent." The training loss should decrease from left to right. However, the testing loss should first decrease and then increase. Both curves should be smooth.

(g) Print out the number of the epoch at which the test loss is minimized. For this epoch, also print out the training loss and accuracy, and the test loss and accuracy. You should find that training accuracy $>$ test accuracy, and training loss $<$ test loss.

No other figures or print outs are required. All accuracies should be printed as percentages. Do not use any functions from `sklearn` except for randomly shuffling data.

Note that if `batchSize=60000`, then your program here should produce the same histories of loss and accuracy as your program in Question 6. You can test your program for some (but not all) errors this way. (Do not hand in any of these tests.)

Run your program on the MNIST data. Use a batch size of 1000 and the same learning rate as in Question 6. You should observe that the values of loss and accuracy printed out in part (g) are very similar to those in Question 6(d). (i.e., the first two digits of each printed value should be the same.) However you should also observe that after this point, the training and testing curves begin to diverge, with training loss continuing to decrease, while test loss increases. (We will see this again later in the course with neural nets.) *Give an explanation for this.* (3 points)

185 points total

University of Toronto Mississauga
CSC 411 - Machine Learning and Data Mining

Cover sheet for Assignment 2

Complete this page and hand it in with your assignment.

Name: _____
(Underline your last name)

Student number: _____

I declare that the solutions to Assignment 2 that I have handed in are solely my own work, and they are in accordance with the University of Toronto Code of Behavior on Academic Matters.

Signature: _____