Homework 2

Deadline: Wednesday, Oct. 13, at 11:59pm.

Submission: You need to submit five files through MarkUs[1]:

- Your answers to Questions 1, 2, 3, and 4, as a PDF file titled `hw2_writeup.pdf`. You can produce the file however you like (e.g. LaTeX, Microsoft Word, scanner), as long as it is readable.
- Python files `run_knn.py`, `logistic.py`, and `run_logistic_regression.py` completed for Question 3.
- Python files `q4.py` completed for Question 4.

Neatness Point: One point will be given for neatness. You will receive this point as long as we don't have a hard time reading your solutions or understanding the structure of your code.

Late Submission: 10% of the marks will be deducted for each day late, up to a maximum of 3 days. After that, no submissions will be accepted.

Computing: To install Python and required libraries, see the instructions on the course web page.

Homeworks are individual work. See the Course Information handout[2] for detailed policies.

1. **[9pts] Expected Loss and Bayes Optimality** You are running an email service, and one of your key features is a spam filter. Every email is either spam or non-spam, which we represent with the target $t \in \{\mathsf{Spam}, \mathsf{NonSpam}\}$. You need to decide whether to keep it in the inbox or remove it to the spam folder. We represent this with the decision variable $y \in \{\mathsf{Keep}, \mathsf{Remove}\}$. We'd like to remove spam emails and keep non-spam ones, but the customers will be much more unhappy if we remove a non-spam email than if we keep a spam email. We can represent this with the following loss function $\mathcal{L}(y, t)$:

	NonSpam	Spam
Keep	0	1
Remove	100	0

   Your studies indicate that 10% of the emails are spam, i.e. $\Pr(t = \mathsf{Spam}) = 0.1$.

   (a) **[2pts]** Evaluate the expected loss $\mathbb{E}[\mathcal{L}(y, t)]$ for the policy that keeps every email ($y = \mathsf{Keep}$), and for the policy that removes every email ($y = \mathsf{Remove}$).

   (b) **[2pts]** Now suppose you get to observe a feature vector $\mathbf{x}$ for each email, and using your knowledge of the joint distribution $p(\mathbf{x}, t)$, you infer $p(t \mid \mathbf{x})$. What is the Bayes optimal decision rule? I.e., say how to determine the Bayes optimal decision y_* from the conditional probability $\Pr(t = \mathsf{Spam} \mid \mathbf{x})$.

[1] `https://markus.teach.cs.toronto.edu/csc311-2021-09`

[2] `http://www.cs.toronto.edu/~rgrosse/courses/csc311_f21/syllabus.pdf`

(c) **[4pts]** After some analysis, you've found two words that are indicative of an email being spam: "Viagra" and "Nigeria". You define two input features: x_1, which takes the value 1 if the email contains the word "Viagra" and 0 otherwise, and x_2, which is the same for "Nigeria". For a spam email, these features have the following joint distribution:

	$x_2 = 0$	$x_2 = 1$
$x_1 = 0$	0.4	0.3
$x_1 = 1$	0.2	0.1

For a non-spam email, these features have the following joint distribution:

	$x_2 = 0$	$x_2 = 1$
$x_1 = 0$	0.998	0.001
$x_1 = 1$	0.001	0

Determine the Bayes optimal decision rule, i.e. determine the Bayes optimal decision y_* for each value of $\mathbf{x}$. Justify your answer.

(d) **[1pts]** What is the expected loss $\mathbb{E}[\mathcal{L}(y_*, t)]$?

2. **[4pts] Feature Maps.** Suppose we have the following 1-D dataset for binary classification:

x	t
-1	1
1	0
3	1

(a) **[2pts]** Argue briefly (at most a few sentences) that this dataset is not linearly separable. (Your argument should resemble the one we used in lecture to prove XOR is not linearly separable.)

(b) **[2pts]** Now suppose we apply the feature map

$$\boldsymbol{\psi}(x) = \begin{pmatrix} \psi_1(x) \\ \psi_2(x) \end{pmatrix} = \begin{pmatrix} x \\ x^2 \end{pmatrix}.$$

Assume we have no bias term, so that the parameters are w_1 and w_2. Write down the constraint on w_1 and w_2 corresponding to each training example, and then find a pair of values (w_1, w_2) that correctly classify all the examples. Remember that there is no bias term.

3. **[15pts] kNN vs. Logistic Regression.** In this problem, you will compare the performance and characteristics of different classifiers, namely k-Nearest Neighbors and Logistic Regression. You will complete the provided code in `q3/` and experiment with the completed code. You should understand the code instead of using it as a black box.

The data you will be working with is a subset of MNIST hand-written digits, 4s and 9s, represented as 28×28 pixel arrays. We show the example digits in figure 1. There are two training sets: `mnist_train`, which contains 80 examples of each class, and `mnist_train_small`, which contains 5 examples of each class. There is also a validation set `mnist_valid` that you should use for model selection, and a test set `mnist_test` that you should use for reporting the final performance. Optionally, the code for visualizing the datasets is located at `plot_digits.py`.

Figure 1: Example digits. Top and bottom show digits of 4s and 9s, respectively.

3.1. k**-Nearest Neighbors.** Use the supplied kNN implementation to predict labels for `mnist_valid`, using the training set `mnist_train`.

(a) **[2pts]** Implement a function `run_knn` in `run_knn.py` that runs kNN for different values of $k \in \{1, 3, 5, 7, 9\}$ and plots the classification rate on the validation set (number of correctly predicted cases, divided by total number of data points) as a function of k. Report the plot in the write-up.

(b) **[2pts]** Comment on the performance of the classifier and argue which value of k you would choose. What is the classification rate for k^*, your chosen value of k? Also report the classification rate for $k^* + 2$ and $k^* - 2$. How does the test performance of these values of k correspond to the validation performance[3]?

3.2. **Logistic Regression.** Read the provided code in `run_logistic_regression.py` and `logistic.py`. You need to implement the logistic regression model, where the cost is defined as:

$$\mathcal{J} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_{\text{CE}}(y^{(i)}, t^{(i)}) = \frac{1}{N} \sum_{i=1}^{N} \left(-t^{(i)} \log y^{(i)} - (1 - t^{(i)}) \log(1 - y^{(i)}) \right),$$

where N is the total number of data points.

(a) **[4pts]** Implement functions `logistic_predict`, `evaluate`, and `logistic` located at `logistic.py`.

(b) **[5pts]** Complete the missing parts in a function `run_logistic_regression` located at `run_logistic_regression.py`. You may use the implemented functions from part (a). Run the code on both `mnist_train` and `mnist_train_small`. Check whether the value returned by `run_check_grad` is small to make sure your implementation in part (a) is correct. Experiment with the hyperparameters for the learning rate, the number of iterations (if you have a smaller learning rate, your model will take longer to converge), and the way in which you initialize the weights. If you get `NaN/Inf` errors, you may try to reduce your learning rate or initialize with

[3]In general, you shouldn't peek at the test set multiple times, but we do this for this question as an illustrative exercise.

smaller weights. For each dataset, report which hyperparameter settings you found worked the best and the final cross entropy and classification error on the training, validation, and test sets. Note that you should only compute the test error once you have selected your best hyperparameter settings using the validation set.

(c) **[2pts]** Examine how the cross entropy changes as training progresses. Generate and report 2 plots, one for each of `mnist_train` and `mnist_train_small`. In each plot, you need show two curves: one for the training set and one for the validation set. Run your code several times and observe if the results change. If they do, how would you choose the best parameter settings?

4. **[8pts] Locally Weighted Regression.**

(a) **[2pts]** Given $\{(\mathbf{x}^{(1)}, y^{(1)}), .., (\mathbf{x}^{(N)}, y^{(N)})\}$ and positive weights $a^{(1)}, ..., a^{(N)}$ show that the solution to the *weighted* least squares problem

$$\mathbf{w}^* = \arg\min \frac{1}{2} \sum_{i=1}^{N} a^{(i)} (y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)})^2 + \frac{\lambda}{2} ||\mathbf{w}||^2 \tag{1}$$

is given by the formula

$$\mathbf{w}^* = \left(\mathbf{X}^T \mathbf{A} \mathbf{X} + \lambda \mathbf{I}\right)^{-1} \mathbf{X}^T \mathbf{A} \mathbf{y} \tag{2}$$

where $\mathbf{X}$ is the design matrix (defined in class) and $\mathbf{A}$ is a diagonal matrix where $\mathbf{A}_{ii} = a^{(i)}$

(b) **[2pts]** Locally reweighted least squares combines ideas from k-NN and linear regression. For each new test example $\mathbf{x}$ we compute distance-based weights for each training example $a^{(i)} = \frac{\exp(-||\mathbf{x}-\mathbf{x}^{(i)}||^2/2\tau^2)}{\sum_j \exp(-||\mathbf{x}-\mathbf{x}^{(j)}||^2/2\tau^2)}$, computes $\mathbf{w}^* = \arg\min \frac{1}{2} \sum_{i=1}^{N} a^{(i)} (y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)})^2 + \frac{\lambda}{2} ||\mathbf{w}||^2$ and predicts $\hat{y} = \mathbf{x}^T \mathbf{w}^*$. Complete the implementation of locally reweighted least squares by providing the missing parts for `q4.py`.

Important things to notice while implementing: First, do not invert any matrix, use a linear solver (numpy.linalg.solve is one example). Second, notice that $\frac{\exp(A_i)}{\sum_j \exp(A_j)} = \frac{\exp(A_i - B)}{\sum_j \exp(A_j - B)}$ but if we use $B = \max_j A_j$ it is much more numerically stable as $\frac{\exp(A_i)}{\sum_j \exp(A_j)}$ overflows/underflows easily. *This is handled automatically in the scipy package with the* `scipy.misc.logsumexp` *function*[4].

(c) **[2pt]** Randomly hold out 30% of the dataset as a validation set. Compute the average loss for different values of τ in the range [10,1000] on both the training set and the validation set. Plot the training and validation losses as a function of τ (using a log scale for τ).

(d) **[2pt]** How would you expect this algorithm to behave as $\tau \to \infty$? When $\tau \to 0$? Is this what actually happened?

[4] `https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.misc.logsumexp.html`