Homework 2

Deadline: Monday, October 17, 2022, 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.
- Completed Python files `run_knn.py`, `logistic.py`, and `run_logistic_regression.py` for Question 3.
- Completed Python files `q4.py` 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 $\mathcal{J}(y, t)$:

	NonSpam	Spam
Keep	0	1
Remove	500	0

   Your studies indicate that 20% of the emails are spam, i.e. $\Pr(t = \mathsf{Spam}) = 0.2$.

   (a) **[2pts]** Evaluate the expected loss $\mathbb{E}[\mathcal{J}(y, t)]$ 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 $p(t \mid \mathbf{x})$. Determine how you will make Bayes optimal decision $y_* \in \{\mathsf{Keep}, \mathsf{Remove}\}$ given the conditional probability $\Pr(t = \mathsf{Spam} \mid \mathbf{x})$.

[1] https://markus.teach.cs.toronto.edu/2022-09

[2] http://www.cs.toronto.edu/~rahulgk/courses/csc311_f22/index.html

(c) **[4pts]** After some analysis, you found two words that are indicative of an email being spam: "Viagra" and "Nigeria". You define two input features:
$x_1 = 1$ if the email contains the word "Viagra" and $x_1 = 0$ otherwise, and
$x_2 = 1$ if the email contains the word "Nigeria" and $x_2 = 0$ otherwise.

For a spam email, the two features have the following joint distribution:

	$x_2 = 0$	$x_2 = 1$
$x_1 = 0$	0.45	0.25
$x_1 = 1$	0.18	0.12

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

	$x_2 = 0$	$x_2 = 1$
$x_1 = 0$	0.996	0.002
$x_1 = 1$	0.002	0

Determine how you will make Bayes optimal decision $y_* \in \{\mathsf{Keep}, \mathsf{Remove}\}$ given the values of the two features x_1 and x_2. Justify your answer.

(d) **[1pts]** What is the expected loss $\mathbb{E}[\mathcal{J}(y_*, t)]$ for the Bayes optimal decision rule from part (c)?

2. **[3pts] Feature Maps.**

Suppose we have the following 2-D data-set for binary classification:

x_1	x_2	y
-2	-1	1
1	2	0
2	3	1

(a) **[3pts]** Explain why this data-set is NOT linearly separable in a few sentences. Hint: Your explanation should mention convexity.

(b) **[0pts]** Suppose you are interested in studying if the above data-set can be separable by a quadratic functions $y = w_1 x_1 + w_2 x_2^2$. Write all the constraints that w_1, w_2 have to satisfy. You do not need to solve for w_1, w_2 using the constraints.

Solution (provided to help prep for future tests). The constraint on w_1 and w_2 are as follows:

$$\begin{aligned} -2w_1 + w_2 &\geq 0 \\ w_1 + 4w_2 &< 0 \\ 2w_1 + 9w_2 &\geq 0 \end{aligned}$$

3. **[16pts] kNN vs. Logistic Regression.** In this problem, you will compare the performance and characteristics of two classifiers: k-Nearest Neighbors and Logistic Regression. You will complete the provided code in `q2/` and experiment with the completed code. You should understand the code instead of using it as a black box.

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

   (a) **[3pts]** Implement the 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 accuracy on the validation set as a function of k. The classification accuracy is the number of correctly predicted data points divided by the total number of data points. Include the plot in the write-up.

   (b) **[2pts]** Choose a value of k (strictly positive integer) and justify your choice. Let's denote the chosen value by k^*. Report the validation and test accuracies of KNN for k^*. Also, report the validation and test accuracies of KNN for $k^* + 2$ and $k^* - 2$ (report the latter if your value of k^* is greater than two).
   How do the test and validation accuracies change as we change the value of k?
   In general, you shouldn't peek at the test set multiple times, but we do this for this question as an illustrative exercise.

   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:

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

   where N is the total number of data. Note that the cost function is the total loss instead of the average loss.

   (a) **[4pts]** Implement functions `logistic_predict`, `evaluate`, and `logistic` in the file `logistic.py`.

   (b) **[5pts]** Complete the missing parts in the function `run_logistic_regression`. 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 hyper-parameters for the learning rate, the number of iterations, and the initial values of the weights. If you have a smaller learning rate, your model will take longer to converge.

   If you get `NaN/Inf` errors, you may try to reduce your learning rate or initialize with smaller weights. Report the best hyper-parameter settings you found and the final cross entropy and classification accuracy on the training, validation, and test sets. You should only compute the test error once you have selected your best hyper-parameter 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]** Let the training set be denoted by $\{(\mathbf{x}^{(1)}, y^{(1)}), .., (\mathbf{x}^{(N)}, y^{(N)})\}$. Consider positive weights $a^{(1)}, ..., a^{(N)}$. We will define the *weighted* least squares problem as follows.

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

   Show that the closed-form solution to the weighted least squares problem is given by the formula below.

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

   where $\mathbf{X}$ is the N by D design matrix and $\mathbf{A}$ is a diagonal matrix where $\mathbf{A}_{ii} = a^{(i)}$.
   Note that the loss function is defined to be the total loss (not the average loss) over all the training examples.

   (b) **[2pts]** Locally reweighted least squares combines ideas from k-NN and linear regression. For each new test example $\mathbf{x}$, we
   (1) compute distance-based weights for each training example

$$a^{(i)} = \frac{\exp\left(-\frac{||\mathbf{x} - \mathbf{x}^{(i)}||^2}{2\tau^2}\right)}{\sum_j \exp\left(-\frac{||\mathbf{x} - \mathbf{x}^{(j)}||^2}{2\tau^2}\right)},$$

   (2) computes the optimal weights

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

   (3) and predicts

$$\hat{y} = \mathbf{x}^T \mathbf{w}^*.$$

   Complete the implementation of locally reweighted least squares by filling in the missing parts in `q4.py`.

   (c) **[2pts]** 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) **[1pt]** How would this algorithm behave as $\tau \to \infty$? How would the algorithm behave when $\tau \to 0$?

   (e) **[1pt]** Compared to the traditional linear regression, what is one advantage and one disadvantage of locally weighted regression?