Homework 2

Deadline: February 17, 2023, 5PM ET.

Submission: You need to submit four files:

- Your answers to Questions 1, 2, 3 as a PDF file titled `hw2_writeup.pdf`. You can produce the file however you like (e.g. LaTeX, Microsoft Word, handwritten and scanned), as long as it is readable. Submit this file through Markus.
- Completed Python files `run_knn.py`, `logistic.py`, and `run_logistic_regression.py`. Submit these files through MarkUs.

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: Everyone will receive 3 grace days for the course, which can be used at any point during the semester on the three assignments. No credit will be given for assignments submitted after 3 days.

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

Homeworks are individual work. See the Course Information handout[1] 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	100	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://www.cs.toronto.edu/ michael/teaching/csc311_w23/index.html

(c) **[4pts]** After some analysis, you found two words that are indicative of an email being spam: "Sale" and "Prince". You define two input features:
$x_1 = 1$ if the email contains the word "Sale" and $x_1 = 0$ otherwise, and
$x_2 = 1$ if the email contains the word "Prince" 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, which should address all four possible combinations of the features.

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

2. **[5pts] Feature Maps.**

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

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

   (a) **[2pts]** Explain why this data-set is NOT linearly separable in a few sentences.

   (b) **[3pts]** Suppose you are interested in studying if the above data-set can be separable by a quadratic functions $y = w_1x_1 + w_2x_2 + w_3x_2^2$. Write all the constraints that w_1, w_2, w_3 have to satisfy. You do not need to solve for the w using the constraints.

3. **[17pts] 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 `q3/` and run experiments with the completed code. You should understand the code instead of using it as a black box.

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

      i. **[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.

      ii. **[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.

   (b) **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:

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

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

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

      ii. **[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 hyperparameters for the learning rate, the number of iterations. You may optionally also want to consider setting weight regularization or changing the initialization. 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 hyperparameter 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. You should aim for a validation accuracy of at least 87%.

iii. **[2pts]** Make a plot for how the cross entropy changes as training progresses for the hyperparameters you found which achieve the highest validation accuracy. 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.

iv. **[1pt]** Run your code three times and report if the results change. Suppose there is some variation in each run–how would you choose the best hyperparameter settings?