Homework 3

Deadline: Monday, Nov. 15, at 11:59pm.

Submission: You need to submit a **single** PDF file titled `hw3_writeup.pdf` through Quercus with your answers to Questions 1, 2, and 3, and outputs requested for Question 1, 2 and 3 (e.g. screenshots of your code). You can produce the file however you like (e.g. LaTeX, Microsoft Word, scanner), as long as it is readable.

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 total possible 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 to be done alone or in pairs. See the Course Information handout[1] for detailed policies.

1. **[15 pts] Logistic Regression.** In this problem, you will implement logistic regression by completing the provided code in `logistic_regression.py` and experiment with the completed code.

   Throughout this homework, you will be working with a subset of hand-written digits, 2's and 3's, represented as 16×16 pixel arrays. We show the example digits in Figure 1. The pixel intensities are between 0 and 1, and were read into the vectors in a raster-scan manner. You are given two training sets: `digits_train` which contains 300 examples of each class and `digits_train_small` which contains 2 examples of each class. You can access these training sets by using functions `load_train` and `load_train_small` in `utils.py`. You are also given a validation set that you should use for model selection and a test set that you should use for reporting the final performance. Optionally, the code for visualizing the dataset is located at `utils.py`.

   Carefully read the provided code in `logistic_regression.py`. You should understand the code instead of using it as a black box. You need to implement the penalized logistic regression model, where the cost is defined as:

$$\begin{aligned}\mathcal{J} &= \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_{\mathrm{CE}}(y^{(i)}, t^{(i)}) + \frac{\lambda}{2}\|\mathbf{w}\|^2 \\ &= \frac{1}{N}\sum_{i=1}^{N}\Big(-t^{(i)}\log y^{(i)} - (1-t^{(i)})\log(1-y^{(i)})\Big) + \frac{\lambda}{2}\|\mathbf{w}\|^2,\end{aligned}$$

   where N is the total number of data points and $\mathbf{w}$ is the weight of logistic regression model. Note that you should only penalize the weights and not the bias term.

   (a) **[4 pts]** Implement the functions `logistic_predict`, `evaluate`, and `logistic` located at `logistic_regression.py`. While implementing the functions, remember to vectorize

[1] https://www.cs.toronto.edu/~cmaddis/courses/sta314_f21/sta314_f21_syllabus.pdf

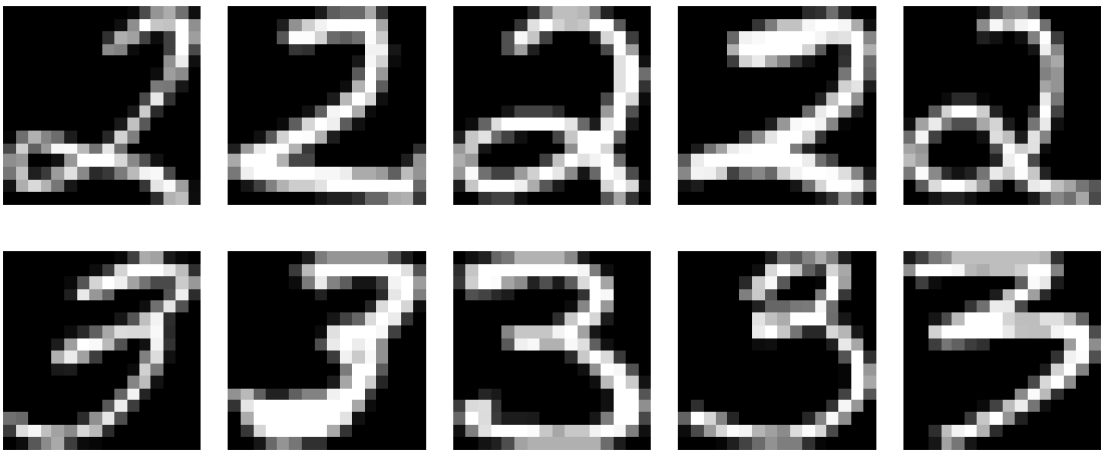

Figure 1: Example digits. Top and bottom show digits of 2s and 3s, respectively.

the operations; you should not have have any `for-loops` in these functions. Include your code in the report.

(b) **[5 pts]** Complete the missing parts in a function `run_logistic_regression`. The function should train the logistic regression model using gradient descent on `digits_train` training set. You may use the implemented functions from part (a). Experiment with the hyperparameters for the learning rate and the number of iterations (if you have a smaller learning rate, your model will take longer to converge). You will fix your ℓ_2 weight regularization to 0 for this part. If you get `NaN/Inf` errors, you may need to reduce your learning rate. Include your code in the report.

Moreover, in the write-up, report which hyperparameter settings you found worked the best and the final cross-entropy and classification accuracy 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) **[2 pts]** Examine how the cross-entropy changes as the training progresses. Generate and report a plot that shows the training curve (iteration counter on x-axis and cross-entropy on y-axis). The plot should have 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 hyperparameter settings?

(d) **[2 pts]** Using the same hyperparameter settings (for learning rate and number of iteration) that worked well in part (b), train the logistic regression model with different values of weight regularization $\lambda \in \{0., 0.001, 0.01, 0.1, 1.0\}$. Generate and report a plot that shows how the validation cross-entropy changes as you train with different weight regularization λ.

Repeat the same experiment and report an additional plot with `digits_train_small` training set. You may need to additionally tune the hyperparameter settings (e.g. learning rate). You should have two plots in total that show the relationship between weight regularization λ and validation cross-entropy: one for each training set `digits_train` and `digits_train_small`.

(e) **[2 pts]** For each dataset (`digits_train` and `digits_train_small`), how does the train and validation cross-entropy change when you increase λ? Do they go up, down first up and down, or down and then up? Explain why you think they behave this way. Which

is the best value of λ based on your experiment? Report the test cross-entropy and classification accuracy for the best value of λ.

2. **[10 pts] K-Means Clustering.** In this question you will be reasoning about the set of optima of the K-Means algorithm. An optimal configuration is a specific configuration of cluster centers $\{\mathbf{m}_k\}_{k=1}^K \subseteq \mathbb{R}^D$ and one-hot assignment vectors $\{\mathbf{r}^{(i)}\}_{i=1}^N \subseteq \{0,1\}^K$ for which the K-Means objective with metric d

$$\min_{\mathbf{m}_k, \mathbf{r}^{(i)}} \sum_{i=1}^N \sum_{k=1}^K r_k^{(i)} d(\mathbf{m}_k, \mathbf{x}^{(i)})^2$$

does not change during the refitting $\mathbf{m}_k$ and reassignment $\mathbf{r}^{(i)}$ steps of the K-means algorithm. An *optimal clustering* is a partition of the training set $\{\mathbf{x}^{(i)}\}_{i=1}^N$ that corresponds to the assignments in one of the optimal configurations of the K-Means algorithm. Taking into account empty clusters, there may be many different configurations that correspond to each optimal clustering.

Recall that the K-Means algorithm is initialized by randomly initializing the cluster centers and assigning points to the closest center. One thing we did not discuss in class is what happens if a cluster center is not assigned any points, i.e., an empty cluster. In this case, we skip the refitting step for any empty cluster, and otherwise leave the K-Means algorithm unchanged.

You will be asked to list the possible optimal clusterings of the K-means algorithm for a specific choice of K and distance metric on the following dataset with 3 points:

$x_1^{(i)}$	$x_2^{(i)}$
0	1
0	-1
4	0

Your answers should be a list of partitions of the three data points above (not configurations of the centers and assignments), although your justification can refer to an underlying configuration. Do not forget the possibility of empty clusters!

(a) **[2 pts]** For the dataset above, list all of the optimal clusterings of the K-Means algorithm with $K = 1$ and the Euclidean norm. Justify your answer.

(b) **[4 pts]** For the dataset above, list all of the optimal clusterings of the K-Means algorithm with $K = 2$ and the Euclidean norm. Justify your answer.

(c) **[4 pts]** For the dataset above, list all of the optimal clusterings of the K-Means algorithm with $K = 2$ and the following distance metric:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\left(\frac{x_1 - y_1}{4}\right)^2 + (x_2 - y_2)^2}$$

Justify your answer.

3. **[10 pts] Principal Component Analysis.** In this problem, you will gain intuition on how PCA works by implementing the algorithm on the same digits dataset. You will complete

the provided code in `pca.py` and experiment with the completed code. Carefully read the provided code in `pca.py`. You should understand the code instead of using it as a black box. You will apply the PCA algorithm to the 600×256 digit images (computing all 256 eigenvalues and eigenvectors).

(a) **[3 pts]** Implement the function `pca` located at `pca.py`. While implementing the function, remember to vectorize the operations; you should not write any `for-loops`. Include your code in the report. You may optionally visualize the eigenvectors with the provided function `show_eigenvectors`.

(b) **[4 pts]** For each image in the validation set, subtract the mean of training data and project it into the low-dimensional space spanned by the first K principal components of training data. After projection, use a 1-NN classifier on K dimensional features (the code vectors) to classify the digit in the low-dimensional space.

You need to implement the classifier yourself in function `pca_classify`. You will do the classification under different K values to see the effect of K. Choose $K = \{2, 5, 10, 20, 30\}$ and, under each K, classify the validation digits using 1-NN. Plot and report results, where the plot should show the curve of validation set classification accuracy versus number of eigenvectors you keep, i.e., K. Include the code in your report as well.

(c) **[2 pts]** If you wanted to choose a particular model from your experiment as the best, which model (number of eigenvectors) would you select? Why?

(d) **[1 pts]** Report the classification accuracy of your final classifier over the test data. How does the model final performance compare to that of logistic regression?