CS 189/289A Introduction to Machine Learning
Spring 2025 Jonathan Shewchuk HW7

Due: Wednesday, May 7 at 11:59 pm

Deliverables:

1. Submit a PDF of your homework, **with an appendix listing all your code**, to the Gradescope assignment entitled "Homework 7 Write-Up". In addition, please include, as your solutions to each coding problem, the specific subset of code relevant to that part of the problem. You may typeset your homework in LaTeX or Word (submit PDF format, **not** .doc/.docx format) or submit neatly handwritten and scanned solutions. **Please start each question on a new page.** If there are graphs, include those graphs in the correct sections. **Do not** put them in an appendix. We need each solution to be self-contained on pages of its own.

   - In your write-up, please state with whom you worked on the homework.
   - In your write-up, please copy the following statement and sign your signature next to it. (Mac Preview and FoxIt PDF Reader, among others, have tools to let you sign a PDF file.) We want to make it *extra* clear so that no one inadvertently cheats.

     "I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted."

2. Submit all the code needed to reproduce your results to the Gradescope assignment entitled "Homework 7 Code". Yes, you must submit your code twice: once in your PDF write-up following the directions as described above so the readers can easily read it, and once in compilable/interpretable form so the readers can easily run it. Do **NOT** include any data files we provided. Please include a short file named README listing your name, student ID, and instructions on how to reproduce your results. Please take care that your code doesn't take up inordinate amounts of time or memory to run. If your code cannot be executed, your solution cannot be verified.

1 Honor Code

Declare and sign the following statement:

"I certify that all solutions in this document are entirely my own and that I have not looked at anyone else's solution. I have given credit to all external sources I consulted."

Signature : _____

While discussions are encouraged, *everything* in your solution must be your (and only your) creation. Furthermore, all external material (i.e., *anything* outside lectures and assigned readings, including figures and pictures) should be cited properly. We wish to remind you that consequences of academic misconduct are *particularly severe*!

2 The Training Error of AdaBoost

Recall that in AdaBoost, our input is an $n \times d$ design matrix X with n labels $y_i = \pm 1$, and at the end of iteration T the importance of each sample is reweighted as

$$w_i^{(T+1)} = w_i^{(T)} \exp(-\beta_T \, y_i \, G_T(X_i)), \quad \text{where} \quad \beta_T = \frac{1}{2} \ln\left(\frac{1 - \text{err}_T}{\text{err}_T}\right) \quad \text{and} \quad \text{err}_T = \frac{\sum_{y_i \neq G_T(X_i)} w_i^{(T)}}{\sum_{i=1}^n w_i^{(T)}}.$$

Note that err_T is the weighted error rate of the classifier G_T. Recall that $G_T(z)$ is ± 1 for all points z, but the metalearner has a non-binary decision function $M(z) = \sum_{t=1}^T \beta_t G_t(z)$. To classify a test point z, we calculate $M(z)$ and return its sign.

In this problem we will prove that if every learner G_t achieves 51% accuracy (that is, only slightly above random), AdaBoost will converge to zero training error. (If you get stuck on one part, move on; all five parts below can be done without solving the other parts, and parts (c) and (e) are the easiest.)

(a) We want to change the update rule to "normalize" the weights so that each iteration's weights sum to 1; that is, $\sum_{i=1}^n w_i^{(T+1)} = 1$. That way, we can treat the weights as a discrete probability distribution over the sample points. Hence we rewrite the update rule in the form

$$w_i^{(T+1)} = \frac{w_i^{(T)} \exp(-\beta_T \, y_i \, G_T(X_i))}{Z_T} \tag{1}$$

for some scalar Z_T. Show that if $\sum_{i=1}^n w_i^{(T)} = 1$ and $\sum_{i=1}^n w_i^{(T+1)} = 1$, then

$$Z_T = 2\sqrt{\text{err}_T(1 - \text{err}_T)}. \tag{2}$$

Hint: sum over both sides of (1), then split the right summation into misclassified points and correctly classified points.

(b) The initial weights are $w_1^{(1)} = w_2^{(1)} = \cdots = w_n^{(1)} = \frac{1}{n}$. Show that

$$w_i^{(T+1)} = \frac{1}{n \prod_{t=1}^T Z_t} e^{-y_i M(X_i)}. \tag{3}$$

(c) Let B (for "bad") be the number of sample points out of n that the metalearner classifies incorrectly. Show that

$$\sum_{i=1}^n e^{-y_i M(X_i)} \geq B. \tag{4}$$

Hint: split the summation into misclassified points and correctly classified points.

(d) Use the formulas (2), (3), and (4) to show that if $\text{err}_t \leq 0.49$ for every learner G_t, then $B \to 0$ as $T \to \infty$. Hint: (2) implies that every $Z_t < 0.9998$. How can you combine this fact with (3) and (4)?

(e) Explain briefly why AdaBoost with short decision trees is a form of subset selection when the number of features is large.

3 IM2SPAIN: Nearest Neighbors for Geo-location

For this problem, we will use nearest neighbors (NN or k-NN) to predict latitude and longitude coordinates of images from their CLIP embeddings. You'll be modifying starter code in the provided `im2spain` directory.

We are using a dataset of images scraped from Flickr with geo-tagged locations within Spain. Each image has been processed with OpenAI's CLIP image model (`https://github.com/openai/CLIP`) to produce features that can be used with k-NN.

The CLIP model was not explicitly trained to predict coordinates from images, but from task-agnostic pre-training on a large web-crawl dataset of captioned images has learned a generally useful mapping from images to embedding vectors. These feature vectors turn out to encode various pieces of information about the image content such as object categories, textures, 3D shapes, etc. In fact, these very same features were used to filter out indoor images from outdoor images in the construction of this dataset.

Note 1: Throughout the problem we use MDE which stands for Mean Displacement Error (in miles). Displacement is the (technically spherical) distance between the predicted coordinates and ground truth coordinates. Since all our images are located within a relatively small region of the globe, we can approximate spherical distances with Euclidean distances by treating latitude/longitude as cartesian coordinates. Assume 1 degree latitude is equal to 69 miles and 1 degree longitude is 52 miles in this problem.

Note 2: You can use `sklearn.neighbors.NearestNeighbors` as shown in the code, but do not use `sklearn.neighbors.KNeighborsRegressor` or `sklearn.model_selection.GridSearchCV` – you will need to implement your own regression and grid search functions.

Deliverables: Include your modified `im2spain_starter.py` script in your submission. Your submitted file should include all modifications requested in this problem.

(a) Let's visualize the data. Using the code already provided, plot the image locations and the first two PCA dimensions of the features, colored by longitude coordinate (east-west position). Does using two principal components capture any information about the longitude of an image? Are there any clusters or outliers that might correspond to particular landscapes or cities?

(b) Modify the starter code in `im2spain_starter.py` to find the three nearest neighbors in the training set of the test image file `53633239060.jpg`. Include those three image files (as images) in order from nearest to 3rd nearest in your submission. Report the coordinates of the test image and the nearest neighbors. How many of the 3 nearest neighbors are "correct"?

(c) Before we begin with our k-NN model, let's first establish a naive constant baseline of simply predicting the training set centroid (coordinate-wise average) location for every test image. Modify the code in `im2spain_starter.py` to implement the constant baseline. What is its MDE in miles?

(d) The main hyperparameter of a k-nearest neighbor classifier is k itself. Use a 1-D grid search in `im2spain_starter.py` to create a plot of the MDE (in miles) of k-NN regression versus k, where k is

the number of neighbors. Include your plot in your write-up. What is the best value of k? What is the lowest error?

(e) Explain your plot in (d) in terms of bias and variance. (In the definitions of *bias* and *variance*, you should think of the ground truth function g and the predicted hypothesis h as both being in the form of a longitude and a latitude, and you should assume that we integrate the bias-squared and the variance over the probability distribution of Spain travel photos that people might take.) In particular, given n training points, how is the bias different for $k = 1$ versus $k = n$? How is the variance different for $k = 1$ versus $k = n$? What happens for intermediate values of k?

(f) We do not need to weight every neighbor equally: closer neighbors may be more relevant. For this problem, weight each neighbor by the inverse of its distance (in feature space) to the test point by modifying `im2spain_starter.py`. Plot the error of k-NN regression with distance weighting vs. k, where k is the number of neighbors. What is the best value of k? What is the MDE in miles? How does performance compare to part (e)?

Note: When computing the inverse distance, add a small value (e.g. 10^{-8}) to the denominator to avoid division by zero.

(g) k-NN yields a *non-parametric* model which means its complexity can grow without bound as we increase the amount of training data. This is in contrast to *parametric* models such as linear regression that assume a fixed number of parameters, so the complexity of the model is bounded even if trained with infinite data. (We typically think of modern deep neural nets as functionally non-parametric, though they technically have a finite parameter size, because when we have more data we usually add more parameters.)

Let's compare the performance of k-NN with linear regression at different sizes of training datasets to get a sense of their respective "scaling curves". Plot the test error of both k-NN and linear regression for various percentages of training data. Which method would you expect to continue improving with twice as much training data?

Note: use the optimal value of k at each training dataset size by running grid search.