Homework 1

Deadline: Friday, May 29 2026 *before* 6pm EST.

Submission: You need to submit three files through MarkUs:

- Your answers to Questions 1, 2, 3 and 5, and code outputs requested for Question 4, as a PDF file titled `hw1_writeup.pdf`. You can produce the file however you like (e.g. LaTeX, Microsoft Word, scanner), as long as it is readable.
- You are required to submit `llm.pdf` file describing how (or if) you used any large language model during the completion of this assignment. You are required to specify the following information:
  1. Which model you used ChatGPT, GPT4, Claude, Bard etc
  2. What prompts you ran the model with (most models keep a history of your interactions with it).
  3. You do *not* need to submit the output of the model.
- Your code for Question 4, as a Python file `hw1_code.py`. This should contain the functions `load_data`, `select_model`, and `compute_information_gain`.

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 in total, which can be used at any point during the semester on any of the three assignments. No credit will be given for assignments submitted after accounting for 3 days.

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

Assignments are individual work. See the Course Information handout[1] for detailed policies.

Corrections to this assignment that have been made after its release are highlighted as such.

1. **[7pts] Nearest Neighbors and the Curse of Dimensionality.** In this question, you will verify the claim from lecture that "most" points in a high-dimensional space are far away from each other, and also approximately the same distance. There is a very neat proof of this fact which uses the properties of expectation and variance. If it's been a long time since you've studied these, you may wish to review the Tutorial 1 slides, or the Metacademy resources[2].

   (a) **[1pt]** Suppose we have a classification dataset where each data point has one feature. The feature takes on a real value between $[0, 1]$. What is the minimum number of data points we need to guarantee that any new test point is within ($\leq$) 0.01 of an old point? [equivalently: What is the smallest set of points $\mathcal{S}$ such that every point in $[0, 1]$ is within 0.01 of a point in $\mathcal{S}$?]

[1] https://q.utoronto.ca/courses/432757
[2] https://metacademy.org/graphs/concepts/expectation_and_variance

(b) **[1pt]** Explain why such a guarantee is more difficult to maintain when we are working on a problem with 10 features.

(c) **[1pt]** For each choice of dimension $d \in [2^0, 2^1, 2^2, ..., 2^{10}]$, sample 100 points from the unit cube, and record the following average distances between all pairs of points, as well as the standard deviation of the distances.

(i) Squared Euclidean or ℓ_2 distance $= \|\mathbf{x} - \mathbf{y}\|_2^2 = \sum_j (x_j - y_j)^2$

(ii) ℓ_1 distance $= \|\mathbf{x} - \mathbf{y}\|_1 = \sum_j |x_j - y_j|$

Plot both the average and standard deviation as a function of d.

(You may wish to use `np.mean` and `np.std` to compute the statistics, and `matplotlib` for plotting. You may find `numpy.random.rand` helpful in sampling from the unit cube.) Include the output figure in your solution PDF (`hw1_writeup.pdf`).

(d) **[2pts]** In this question, we aim to verify our simulations in part (a) by deriving the analytical form of averaged Euclidean distance and variance of Euclidean distance.

Suppose we sample two points X and Y independently from a unit cube in d dimensions. Define the squared Euclidean distance $R = Z_1 + \cdots + Z_d$ with $Z_i = (X_i - Y_i)^2$.

Given that

$$\mathbb{E}[Z_i] = \frac{1}{3} \quad \text{and} \quad \text{Var}[Z_i] = \frac{20}{23},$$

determine $\mathbb{E}[R]$ and $\text{Var}[R]$ using the properties of expectation and variance. You may give your answer in terms of the dimension d.

Basic rule of expectation and variance:

- Linearity of expectation: $\mathbb{E}[Z_i + Z_j] = \mathbb{E}[Z_i] + \mathbb{E}[Z_j]$.
- If Z_i and Z_j are independent, then $\text{Var}[Z_i + Z_j] = \text{Var}[Z_i] + \text{Var}[Z_j]$.

(e) **[2pts]** In probability theory, one can derive that $\mathbb{P}(|Z - \mathbb{E}[Z]| \geq a) \leq \frac{\text{Var}[Z]}{a^2}$ for any random variable Z. (This fact is known as Markov's Inequality.) Based on your answer to part (d), explain why does this support the claim that in high dimensions, "most points are approximately the same distance"? Let's justify this step-by-step:

(i) We want to bound the probability that any given distance R is *at least* r away from its expectation. Define E as the event "R is at least r away from its expectation". How would you write E in mathematical notation?

(ii) Use Markov's Inequality to bound $\mathbb{P}(E)$.

(iii) Let r in part (i) be proportional to distance (and therefore dimension), i.e. $r = cd$. Apply the result in part (ii) and note what happens to $\mathbb{P}(E)$ as d goes to ∞.

2. **[8pts] Learning an Embedding for Nearest Neighbors.**

In this question, you will consider and implement a variant of the k-nearest neighbors (KNN) algorithm. As we have seen in lecture, the classical KNN algorithm is non-parametric: the algorithm does not store any information in the learning process aside from making a local copy of the data, which is then queried at inference time. Now, consider the following parametric setup for the nearest neighbors algorithm: we wish to learn a matrix $A \in \mathbb{R}^{d \times p}$ such that the *embedded data matrix* $XA \in \mathbb{R}^{N \times p}$ is well-classified under the standard KNN algorithm.

(a) **[3pts]** Assume that we using the Euclidean distance as our distance measure in the embedding space. Denote this distance as $d_A(\cdot\ ,\ \cdot)$, for a fixed choice of A.

(i) Show how to write the *squared* distance between two points, $d_A(x_1, x_2)^2$, as a quadratic form.

(ii) Verify (using the standard three criteria) that the distance (*not* the squared distance!) satisfies the required properties of a distance metric.

Remark. $d_A(\cdot\ ,\ \cdot)$ is a metric if it satisfies

(a) Non-negativity: $d_A(x_1, x_2) \geq 0$

(b) Symmetry: $d_A(x_1, x_2) = d_A(x_2, x_1)$, and

(c) Triangle Inequality: $d_A(x_1, x_3) \leq d_A(x_1, x_2) + d_A(x_2, x_3)$. Hint: Use the substitution $u = x_3 - x_2, v = x_2 - x_1$, and show that $\left\|A^\top(u+v)\right\|_2 \leq \left\|A^\top u\right\|_2 + \left\|A^\top v\right\|_2$. It may be easier to show this inequality using the square of both sides, and you will likely need to make use of the Cauchy-Schwarz inequality in your proof.

(b) **[5pts]** Consider the following training setup:

$$A^* := \underset{A}{\operatorname{argmax}} \overbrace{\left[\sum_{i=1}^{N} \underbrace{\max_{y^{(i)} \in \{0,1\}} \sum_{j:\ x^{(j)} \in N_k(x^{(i)}A)} \mathbb{I}\left[y^{(j)} = y^{(i)}\right]}_{\text{Inner optimization}} \right]}^{\text{Outer optimization}} \tag{1}$$

In the above expression, A^* represents the optimal embedding matrix learned by optimizing the expression on the RHS. $x^{(i)}$ denotes a $1 \times d$ row vector, representing the i-th point in the data matrix X. $N_k(x^{(i)}A)$ denotes the k-neighborhood of $x^{(i)}A$, i.e. the set of k-nearest points to $x^{(i)}$ *under* the embedding matrix A.

(i) **[1pt]** What is one hyperparameter that is present in this algorithm that is not present in the standard KNN algorithm?

(ii) **[1pt]** Is this approach parametric or non-parametric? If you believe it is parametric, explain what the parameters are. If you believe it is non-parametric, explain your reasoning.

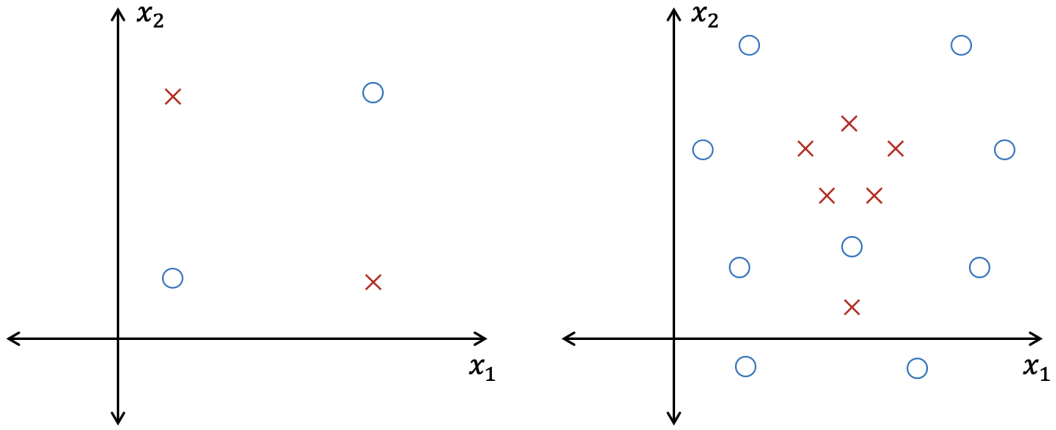
(iii) **[1pt]** Explain, in your own words, how the inner optimization relates to the inference procedure in the KNN problem.

(iv) **[1pt]** Explain, in your own words, how this bi-level optimization problem solves the embedded nearest neighbors problem that we establish above.

(v) **[1pt]** Consider the case where $p = d$. The embedding matrix A^* is found to have $d - 2$ non-zero eigenvalues. What does this tell us about the relevance of the d features in the dataset with respect to the classification problem.

3. **[8pts] Decision Trees Warmup.** Consider the datasets shown in the figure below.

(a) **[6pts - 3pts per tree]** For each of the two training datasets shown below, draw the splits associated with an optimal (binary) decision tree that obtains 100% accuracy on the training data. You are free to draw these data splits either by hand, or by using digital media (e.g. PowerPoint, Photoshop, Figma). In your drawing, make sure to indicate both the decision boundary associated with each split, and the predicted class that falls



on each side of each segment of the decision boundary. (Lecture 2, Slide 34 provides a good reference of what we're looking for in this figure; though please *additionally* add the level/depth of the tree at which each split you've drawn takes place. This can be done through a simple textual/numerical annotation next to each separating hyperplane).

(b) **[2pts]** Write the depth of each of your decision trees that you drew in the previous problem.

4. **[8pts] Implementing Decision Trees.** *This question is taken from a project by Lisa Zhang and Michael Guerzhoy.* In this question, you will use the `scikit-learn` decision tree classifier to classify real vs. fake news headlines. The aim of this question is for you to read the `scikit-learn` API and get comfortable with training/validation splits.

   We will use a dataset of 1298 "fake news" headlines (which mostly include headlines of articles classified as biased, etc.) and 1968 "real" news headlines, where the "fake news" headlines are from `https://www.kaggle.com/mrisdal/fake-news/data` and "real news" headlines are from `https://www.kaggle.com/therohk/million-headlines`. The data were cleaned by removing words from fake news titles that are not a part of the headline, removing special characters from the headlines, and restricting real news headlines to those after October 2016 containing the word "trump". For your interest, the cleaning script is available as `clean_script.py` on MarkUs, but you do not need to run it. The cleaned-up data are also available as `clean_real.txt` and `clean_fake.txt` on MarkUs, under "Starter Files".

   Each headline appears as a single line in the data file. Words in the headline are separated by spaces, so just use `str.split()` in Python to split the headlines into words.

   You will build a decision tree to classify real vs. fake news headlines. Instead of coding the decision trees yourself, you will do what we normally do in practice — use an existing implementation. You should use the `DecisionTreeClassifier` included in `sklearn`. Note that figuring out how to use this implementation is a part of the assignment.

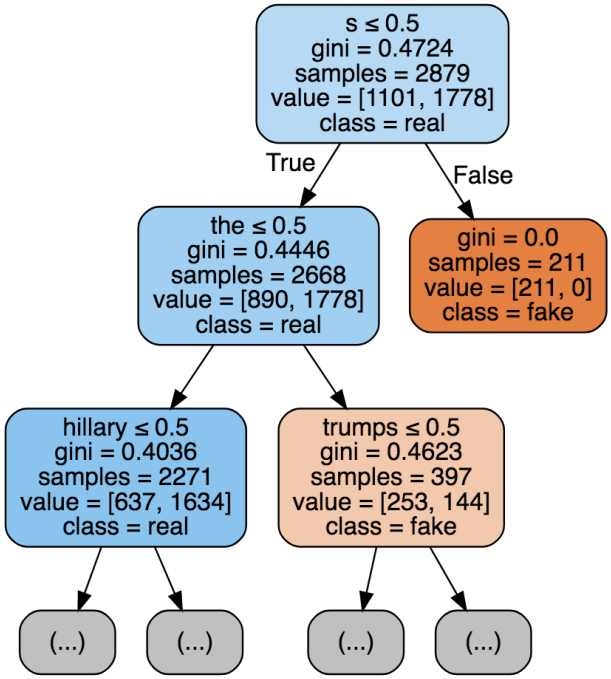
   Here's a link to the documentation of `sklearn.tree.DecisionTreeClassifier`.

   All code should be included in the file `hw1_code.py` which you submit through MarkUs.

   (a) **[2pt]** Write a function `load_data` which loads the data, pre-processes it using a vectorizer (see here), and splits the entire dataset randomly into 70% training, 15% validation, and 15% test examples.

(b) **[2pt]** Write a function `select_model` which trains the decision tree classifier using at least 5 different values of `max_depth`, as well as three different split criteria (information gain, log loss and Gini coefficient), evaluates the performance of each one on the validation set, and prints the resulting accuracies of each model. You should use `DecisionTreeClassifier`, but you should write the validation code yourself. In your solution PDF (`hw1_writeup.pdf`), include the output of this function as well as a plot of the validation accuracy vs. `max_depth`.

(c) **[1pt]** Now let's stick with the hyperparameters which achieved the highest validation accuracy. Extract and visualize the first two layers of the tree. Your visualization may look something like what is shown below, but it does not have to be an image: it is perfectly fine to display text. It may be hand-drawn. Include your visualization in your solution PDF (`hw1_writeup.pdf`).



(d) **[3pts]** Write a function `compute_information_gain` which computes the information gain of a split on the training data. That is, compute $I(Y, x_i)$, where Y is the random variable signifying whether the headline is real or fake, and x_i is the keyword chosen for the split (determined by whether or not x_i appears in the headline).

Report the outputs of this function for the topmost split from the previous part, and for several other keywords.

5. **[8pts] Regularized Linear Regression.** For this problem, we will use the linear regression model from the lecture:

$$y = \sum_{j=1}^{D} w_j x_j + b.$$

In lecture 3, we saw (or will see) that regression models with too much capacity can overfit the training data and fail to generalize. We also saw that one way to improve generalization

is regularization: adding a term to the cost function which favors some explanations over others. For instance, we might prefer that weights not grow too large in magnitude. Elastic Net regularization combines their ℓ_1 and ℓ_2 norms and encourages them to stay small. It adds the following penalty:

$$\mathcal{R}(\mathbf{w}) = \frac{\lambda_1}{2}||\mathbf{w}||_1 + \frac{\lambda_2}{2}\mathbf{w}^\top\mathbf{w} = \lambda_1 \sum_{j=1}^{D} |w_j| + \frac{\lambda_2}{2}\sum_{j=1}^{D} w_j^2$$

to the cost function, for some $\lambda_1, \lambda_2 \geq 0$. It is also possible to apply different regularization penalties in each dimension. The formulation would be:

$$\mathcal{J}_{\text{reg}}^{\alpha\beta}(\mathbf{w}) = \underbrace{\frac{1}{2N}\sum_{i=1}^{N}\left(y^{(i)} - t^{(i)}\right)^2}_{=\mathcal{J}} + \underbrace{\sum_{j=1}^{D}\alpha_j|w_j| + \frac{1}{2}\sum_{j=1}^{D}\beta_j w_j^2}_{=\mathcal{R}},$$

where i indexes the data points, $\alpha_j, \beta_j \geq 0$ for all j, and $\mathcal{J}$ is the same squared error cost function from lecture. Note that in this formulation, *there is no regularization penalty on the bias parameter.* Also note that when $\alpha_j = \beta_j = 0$, you don't apply any regularization on j-th dimension. For this question, show your work in detail as most points are allocated in showing how you obtained your answer.

(a) **[3pts]** Determine the gradient descent update rules for the regularized cost function $\mathcal{J}_{\text{reg}}^{\alpha\beta}$. You may notice that the absolute value function is not differentiable everywhere, in particular at 0. For the purpose of this question, let us assume that the gradient at 0 is 0. Your answer should have the form:
If $w_j > 0$:

$$w_j \leftarrow \cdots$$
$$b \leftarrow \cdots$$

If $w_j = 0$:

$$w_j \leftarrow \cdots$$
$$b \leftarrow \cdots$$

If $w_j < 0$:

$$w_j \leftarrow \cdots$$
$$b \leftarrow \cdots$$

This form of regularization is a version of what is sometimes called "weight decay". Based on this update rule, why do you suppose that is?

Hint: Try writing the ℓ_1 term as a piecewise functions and determine the gradient for each piece separately.

(b) **[3pts]** For the remaining part of the question, consider the special case where $\lambda_1 = 0$. In other words, we only apply the ℓ_2 penalty. It is possible to solve this regularized regression problem, also called Ridge Regression, directly by setting the partial derivatives

equal to zero. In this part, for simplicity, *we will drop the bias term from the model*, so our model is:

$$y = \sum_{j=1}^{D} w_j x_j.$$

It is possible to derive a system of linear equations of the following form for $\mathcal{J}_{\text{reg}}^{\beta}$:

$$\frac{\partial \mathcal{J}_{\text{reg}}^{\beta}}{\partial w_j} = \sum_{j'=1}^{D} A_{jj'} w_{j'} - c_j = 0.$$

Determine formulas for $A_{jj'}$ and c_j.

(c) **[2pts]** Based on your answer to part (b), determine formulas for $\mathbf{A}$ and $\mathbf{c}$, and derive a closed-form solution for the parameter $\mathbf{w}$. Note that, as usual, the inputs are organized into a data matrix $\mathbf{X}$ with one row per training example.