Homework 1

Deadline: Wednesday, Sept. 29, at 11:59pm.

Submission: You need to submit three files through MarkUs[1]:

- Your answers to Questions 1, 2, and 3, and outputs requested for Question 2, 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.
- Your code for Question 2, as the 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: 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. **[4pts] Nearest Neighbours 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[3], or the Metacademy resources[4].

   (a) **[2pts]** 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 average squared Euclidean distance between all pairs of points, as well as the standard deviation of the squared Euclidean distances. (Recall that squared Euclidean distance is defined as $\|\mathbf{x} - \mathbf{y}\|^2 = \sum_j (x_j - y_j)^2$.) Plot both the average and the 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`).

   (b) **[2pts]** In this question, we aim to verify our simulations in (a) by deriving the analytical form of averaged distance and variance of distance. Suppose we sample two points X and Y independently from a unit cube in d dimensions and define the squared Euclidean distance $R = Z_1 + \cdots + Z_d$ with $Z_i = (X_i - Y_i)^2$. Given that $\mathbb{E}[Z_i] = \frac{1}{6}$ and $\mathrm{Var}[Z_i] = \frac{7}{180}$, determine $\mathbb{E}[R]$ and $\mathrm{Var}[R]$ using the properties of expectation and variance. You may give your answer in terms of the dimension d.

   Basic rule of expectation and variance: $\mathbb{E}[Z_i + Z_j] = \mathbb{E}[Z_i] + \mathbb{E}[Z_j]$ and if we further know Z_i and Z_j are independent, then $\mathrm{Var}[Z_i + Z_j] = \mathrm{Var}[Z_i] + \mathrm{Var}[Z_j]$.

[1] https://markus.teach.cs.toronto.edu/csc311-2021-09
[2] http://www.cs.toronto.edu/~rgrosse/courses/csc311_f21/syllabus.pdf
[3] https://www.cs.toronto.edu/~rgrosse/courses/csc311_f21/tutorials/tut01.pdf
[4] https://metacademy.org/graphs/concepts/expectation_and_variance

(c) **[Optional, 0pts]** 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 (b), explain why does this support the claim that in high dimensions, "most points are far away, and approximately the same distance"?

2. **[8pts] 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 the course web page, but you do not need to run it. The cleaned-up data are available as `clean_real.txt` and `clean_fake.txt` on the course web page.

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.

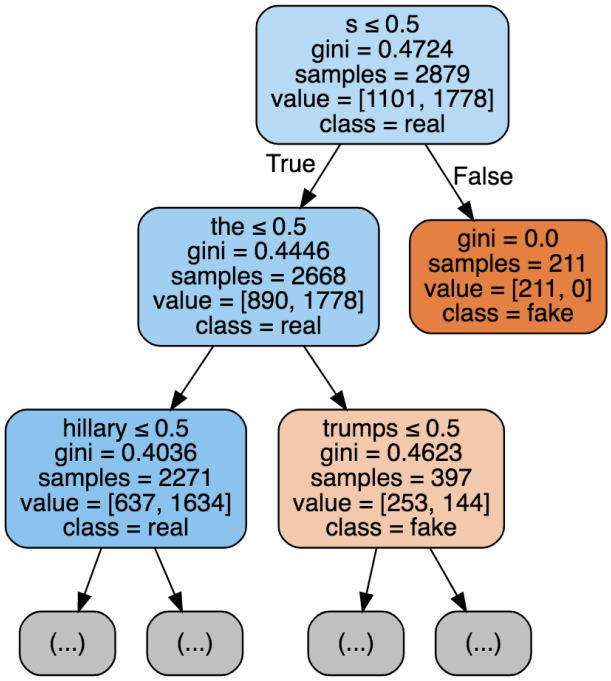
Here's a link to the documentation of `sklearn.tree.DecisionTreeClassifier`: http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

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, preprocesses it using a vectorizer (http://scikit-learn.org/stable/modules/classes.html#module-sklearn.feature_extraction.text), 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 two different split criteria (information gain 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. Include the output of this function in your solution PDF (`hw1_writeup.pdf`).

(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.

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

3. **[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, we saw 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. We can encourage them to stay small by adding a penalty:

$$\mathcal{R}(\mathbf{w}) = \frac{\lambda}{2}\mathbf{w}^\top \mathbf{w} = \frac{\lambda}{2}\sum_{j=1}^{D} w_j^2$$

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

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

where i indexes the data points, $\beta_j \geq 0$ 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 $\beta_j = 0$, 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 $\mathcal{J}_{\text{reg}}^{\beta}$. Your answer should have the form:

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

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

(b) **[3pts]** It's also possible to solve the regularized regression problem 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 design matrix $\mathbf{X}$ with one row per training example.