Homework 1

Deadline: Monday, Jan 31, at 9:59am.

Submission: You need to submit the final PDF file through Quercus.
Codes: You need to submit all the codes as an appendix.
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 tutorial.

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

1. **[3pts] Basics about Machine Learning.**

   (a) **[1pts]** Briefly explain the concept of "Overfitting" and list at least 2 different techniques to alleviate the potential overfitting issue.

   (b) **[1pts]** List three pitfalls of K Nearest Neighbors approach.

   (c) **[1pts]** Design an application in your own medical research which can use linear regression or KNN. Also provide brief comments about potential risks of such ML approach.

2. **[2pts] Python Familiarity.**

   Complete all of the questions in the Google Colab Notebook. You can submit a link to your completed notebook for this question. Make sure to run each cell before submitting to verify that your code works.

3. **[3pts] Regularized Linear Regression.** For this problem, we will use the linear regression model from the lecture (for simplicity, we omit the bias term):

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

   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:

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

[1] https://lmp1210-uoft.github.io/2022/assets/misc/syllabus.pdf

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

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

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) **[2pts]** Determine the gradient descent update rules for the regularized cost function $\mathcal{J}^{\beta}_{\text{reg}}$. Your answer should have the form:

$$w_j \leftarrow \cdots$$

This form of regularization is sometimes called "weight decay". Based on this update rule, please comment the general rule of setting β_j if you think the $j-th$ feature (or variable) is less important?

(b) **[1pts]** 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 $\mathcal{J}^{\beta}_{\text{reg}}$:

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

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

4. **[6pts] Classification with Nearest Neighbours.** In this question, you will use the `scikit-learn`'s KNN classifer 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 357 "benign" cells and 212 "malignant" cells from a digitized image of a fine needle aspirate (FNA) of a breast mass. 30 features of each cells are recorded. The data is obtained through https://www.kaggle.com/uciml/breast-cancer-wisconsin-data. We will use the data stored in `HW1_data.csv` on the course website for this assignment.

You will build a KNN classifier to classify benign vs. malignant cells. Instead of coding the KNN yourself, you will do what we normally do in practice — use an existing implementation. You should use the `KNeighborsClassifier` included in `sklearn`. Note that figuring out how to use this implementation, its corresponding attributes and methods is a part of the assignment.

(a) **[1pts]** Write a function `load_data` which loads the data, and splits the entire dataset randomly into approximately 70% training, 10% validation, and 20% test examples. The output of this function will be training data, validation data, and test data.

(b) **[3pts]** Write a function `select_knn_model` that uses a KNN classifer to classify between benign vs. malignant cells. Use a range of k values between 1 to 20 and compute both training and validation accuracies, leaving other arguments to `KNeighborsClassifier` at their default values. Generate a plot showing the training and validation accuracy for each k. Report the generated plot in your write-up. Choose the model with the best validation accuracy and report its accuracy on the test data.

(c) **[2pts]** Repeat part (b), passing argument `metric='cosine'` to the `KNeighborsClassifier`.