Homework 1

Deadline: Thursday, Feb 2, 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: 0.5 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) **[0.5pts]** Briefly explain the concept of "Overfitting" and list at least 2 different techniques to alleviate the potential overfitting issue.

   (b) **[1pts]** Describe how the magnitude of K impact's modelling in KNN algorithm and how to choose it for a problem.

   (c) **[1.5pts]** Design two applications in your own medical research, one in regression category and one in classification category. Also provide brief comments about potential risks of such ML approach on medical samples.

2. **[2.5pts] 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/2023/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. **[3pts] Classification with Nearest Neighbours.** In this question, you will use the `scikit-learn`'s KNN classifier to classify patients into liver patient (liver disease) or not (no disease). 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 416 liver patient records and 167 non liver patient records collected from North East of Andhra Pradesh, India. 10 variables for each patient are recorded, and the true label is in column **Dataset**. The data is obtained through https://www.kaggle.com/datasets/uciml/indian-liver-patient-records. 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 patients into liver patient (liver disease) or not (no disease). 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. Make sure using `df.fillna(0)` before splitting the data, which is for filling the missing values of some patient variables with 0.

(b) **[1pts]** Write a function `select_knn_model` that uses a KNN classifier to classify between liver disease or no disease. 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) **[1pts]** Repeat part (b), passing argument `metric='cosine'` to the `KNeighborsClassifier`.

5. **[2.5pts] Classification with Decision Tree.** In this question, you will solve the exact same problem as Q4 using a different machine learning algorithm: Decision Tree.

(a) **[1.5pts]** Write a function `train_decision_tree` to classify between liver disease or no disease. Use the `DecisionTreeClassifier` from `sklearn.tree`, and set the following parameters as its input: `criterion='gini', splitter='best', min_samples_leaf=1`. Report train, validation and test accuracies. Is the model overfitted (training accuracy is very close to 1 while test accuracy is not)? If yes, try training with `min_samples_leaf=2` and `min_samples_leaf=3` arguments and report all the accuracies for the new models.

(b) **[1pts]** In the previous question, we set the criterion to create the tree using Gini index. Use the following code implementation to visualize the tree and each node Gini values. Share a screenshot of the first and second layers from the tree's root here. Replace your `model` and `feature_name` in your code. The generated image will be saved in the same folder as you code.

```
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from graphviz import Source

graph = Source(export_graphviz(model, out_file=None, feature_names=feature_names))
graph.format = 'png'
graph.render('dt', view=True)
```

6. **[1pts] Classification with Logistic Regression.** Train a model on the same exact problem this time using `LogisticRegression` from `sklearn.linear_model`. Report train, val, and test accuracies.