Assignment 1 Solutions

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.
   solution.
   Overfitting is when the model fits to unimportant features in the training set (i.e. noise) that do not improve predictive performance on the held out data. Some techniques to prevent overfitting include the following:
   - K-fold cross-validation
   - Regularization
   - Reducing model complexity
   - Priors (for bayesian approaches)
   - Dropout (for deep learning approaches)

   (b) **[1pts]** List three pitfalls of K Nearest Neighbors approach.
   solution.
   KNN is a simple (and surprisingly effective!) approach but has a number of limitations:
   - Requires access to the training data to make predictions
   - Slow when the size of training data is very large
   - Assumes that all dimensions of data are equally important, this is likely not true

   (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.
   solution.
   Open-ended question, these techniques are very general. Some common risks:
   - Not enough data
   - Data is biased
   - Distribution shift (environment in which the model is deployed is significantly different from training)

2. **[2pts] Python Familiarity.**
   A version of the colab book with the answers can be found here.

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:

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

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:

$$w_j \leftarrow \cdots$$

solution.
The gradient descent update rules for the regularized cost function $\mathcal{J}_{\text{reg}}^{\beta}$ will be of the form:

$$w_j \leftarrow w_j - \alpha\frac{\partial \mathcal{J}_{\text{reg}}^{\beta}}{\partial w_j}$$

where α is the learning rate. Now for the weights w_j, we have:

$$\begin{aligned}\frac{\partial \mathcal{J}_{\text{reg}}^{\beta}}{\partial w_j} &= \frac{\partial \mathcal{J}}{\partial w_j} + \frac{\partial \mathcal{R}}{\partial w_j}\\ &= \frac{1}{N}\sum_{i=1}^{N} x_j(y^{(i)} - t^{(i)}) + \frac{\partial}{\partial w_j}\left(\frac{1}{2}\sum_{j}\beta_j w_j^2\right)\\ &= \frac{1}{N}\sum_{i=1}^{N} x_j(y^{(i)} - t^{(i)}) + \beta_j w_j\end{aligned}$$

Thus, we have:

$$\begin{aligned}w_j &\leftarrow w_j - \alpha\frac{\partial \mathcal{J}}{\partial w_j} - \alpha\beta_j w_j\\ &= (1 - \alpha\beta_j)w_j - \alpha\frac{\partial \mathcal{J}}{\partial w_j}\end{aligned}$$

This form of regularization is sometimes called "weight decay". At each step of gradient descent, weights are pulled towards zero. The further the weights are towards zero, the stronger the force of this "pull". The constant λ controls the strength of the pull.
Less important features should have β_j close to 0.

- **[-1]** Mistakes in subscripts (e.g. i or j).

- **[-1]** Forget to propagate the $1/N$ term in the derivation.
- **[-1]** Calculus errors when calculating the gradients.

(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 $\mathcal{J}_{\text{reg}}^{\beta}$:

$$\frac{\partial \mathcal{J}_{\text{reg}}^{\beta}}{\partial \mathbf{w}_j} = \sum_{j'=1}^{D} \mathbf{A}_{jj'} \mathbf{w}_{j'} - \mathbf{c}_j = 0.$$

Determine formulas for $\mathbf{A}_{jj'}$ and $\mathbf{c}_j$.
solution.
Observe that:

$$\begin{aligned}
\frac{\partial \mathcal{J}_{\text{reg}}^{\beta}}{\partial w_j} &= \frac{\partial \mathcal{J}}{\partial w_j} + \beta_j w_j \\
&= \frac{1}{N} \sum_{j'=1}^{D} \left(\sum_{i=1}^{N} x_j^{(i)} x_{j'}^{(i)} \right) w_{j'} - \frac{1}{N} \sum_{i=1}^{N} x_j^{(i)} t^{(i)} + \beta_j w_j \\
&= \sum_{j'=1}^{D} \left[\left(\frac{1}{N} \sum_{i=1}^{N} x_j^{(i)} x_{j'}^{(i)} \right) + \delta_{j,j'} \beta_j \right] w_{j'} - \frac{1}{N} \sum_{i=1}^{N} x_j^{(i)} t^{(i)},
\end{aligned}$$

where

$$\delta_{j,j'} = \begin{cases} 1, & \text{if } j = j' \\ 0, & \text{otherwise} \end{cases}$$

Now, the values of $\mathbf{A}_{jj'}$ and $\mathbf{c}_j$ are:

$$\begin{aligned}
\mathbf{A}_{jj'} &= \frac{1}{N} \left(\sum_{i=1}^{N} x_j^{(i)} x_{j'}^{(i)} \right) + \delta_{j,j'} \beta_j \\
\mathbf{c}_j &= \frac{1}{N} \sum_{i=1}^{N} x_j^{(i)} t^{(i)}
\end{aligned}$$

- **[-1]** Minor calculus mistakes.
- **[-1]** Major mistake in final solution.
- **[-1]** Not factoring $\beta_j w_j$ in $A_{jj'}$.

4. **[4pts] Classification with Nearest Neighbours.** The aim of this question is for you to read the `scikit-learn` API and get comfortable with KNN and 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.

solution.

Check the colab notebook.

- [**-1**] Incorrect code for writing a function.

(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.

solution.

Check the colab notebook. The best k and corresponding test accuracy may vary depending on the randomness of the split.

- [**-1**] Incorrect code for writing a function.
- [**-1**] Incorrect figures.
- [**-1**] Missing best k and its accuracy.

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

solution.

Check the colab notebook. Repeating part (b) with `metric=‘cosine’`, the best k and corresponding test accuracy may vary but in general larger ks perform better.

- [**-1**] Incorrect figures.
- [**-1**] Not reporting k and its accuracy.