HOMEWORK 1 - V1

CSC311 Fall 2019

University of Toronto

Version history: V0 → V1: make notation consistent with slides & clarify Q3.2 answers to the previous version still acceptable.

- **Deadline:** Wed, Oct. 2, at 23:59.
- **Submission:** You need to submit one pdf file through MarkUs including all your answers, plots, and your code. You can produce the file however you like (e.g. LaTeX, Microsoft Word, etc), as long as it is readable. Points will be deducted if we have a hard time reading your solutions or understanding the structure of your code. For each question, you should append your code right after your answers to that question.

1. Nearest Neighbors and the Curse of Dimensionality - 30 pts. 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.

(a) *[10 pts]* First, consider two independent univariate random variables X and Y sampled uniformly from the unit interval $[0, 1]$. Determine the expectation and variance of the random variable Z, defined as the squared distance $Z = (X - Y)^2$.

(b) *[10 pts]* Now suppose we sample two points independently from a unit cube in d dimensions. Observe that each coordinate is sampled independently from $[0, 1]$, i.e. we can view this as sampling random variables $X_1, \ldots, X_d, Y_1, \ldots, Y_d$ independently from $[0, 1]$. The squared Euclidean distance can be written as $R = Z_1 + \cdots + Z_d$, where $Z_i = (X_i - Y_i)^2$. Using the properties of expectation and variance, determine $\mathbb{E}[R]$ and $\mathrm{Var}[R]$. You may give your answer in terms of the dimension d, and $\mathbb{E}[Z]$ and $\mathrm{Var}[Z]$ (the answers from part (a)).

(c) *[10 pts]* Based on your answer to part (b), compare the mean and standard deviation of R to the maximum possible squared Euclidean distance (i.e. the distance between opposite corners of the cube). Why does this support the claim that in high dimensions, "most points are far away, and approximately the same distance"?

2. Decision Trees - 30 pts. 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 titles not part of the headlines, removing special characters and restricting real news headlines after October 2016 using the word "trump". The cleaned data are available as `clean_real.txt` and `clean_fake.txt` on the course webpage. It is expected that you use these cleaned data sources for this assignment.

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, its corresponding attributes and methods is a part of the assignment.

All code should be included in your pdf submission file.

(a) *[6 pts]* 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, we suggest you use `CountVectorizer` as it is the simplest in nature), and splits the entire dataset randomly into 70% training, 15% validation, and 15% test examples.

(b) *[6 pts]* 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.

(c) *[6 pts]* 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 does not have to be an image: it is perfectly fine to display text. It may also be hand-drawn. Include your visualization in your solution pdf.

(d) *[12 pts]* 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. Regression - 40 pts. In this question, you will derive certain properties of linear regression, and experiment with cross validation to tune its regularization parameter.

3.1. *Linear regression - 10 pts.* Suppose that $\mathbf{X} \in \mathbb{R}^{n \times m}$ with $n \geq m$ and $\mathbf{t} \in \mathbb{R}^n$, and that $\mathbf{t}|(\mathbf{X}, \mathbf{w}) \sim \mathcal{N}(\mathbf{X}\mathbf{w}, \sigma^2\mathbf{I})$. We know that the maximum likelihood estimate $\hat{\mathbf{w}}$ of $\mathbf{w}$ is given by

$$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{t}.$$

(a) Write the log-likelihood implied by the model above, and compute its gradient w.r.t. $\mathbf{w}$. By setting it equal to 0, derive the above estimator $\hat{\mathbf{w}}$.

(b) Find the distribution of $\hat{\mathbf{w}}$, its expectation and covariance matrix. Hint: Read the property of multivariate Gaussian random vectors in preliminaries.pdf on course webpage.

3.2. *Ridge regression and MAP - 10 pts.* Suppose that we have $\mathbf{t}|(\mathbf{X}, \mathbf{w}) \sim \mathcal{N}(\mathbf{X}\mathbf{w}, \sigma^2\mathbf{I})$ and we place a normal prior on $\mathbf{w}|\mathbf{X}$, i.e., $\mathbf{w} \sim \mathcal{N}(0, \tau^2\mathbf{I})$. Recall from the first tutorial (also in preliminaries.pdf) that MAP estimate of $\mathbf{w}$ is given as the maximum of the posterior density

$$\hat{\mathbf{w}}_{MAP} = \underset{\mathbf{w}}{\text{argmax}}\{p(\mathbf{w}|\mathbf{X}, \mathbf{t}) \propto p(\mathbf{t}|\mathbf{X}, \mathbf{w})p(\mathbf{w}|\mathbf{X})\}.$$

Here, $\propto$ notation means *proportional to*, and is used since we dropped the term $p(\mathbf{t}|\mathbf{X})$ in the denominator as it doesn't have $\mathbf{w}$ in it, thus it doesn't contribute to the maximization problem.

Show that the MAP estimate of $\mathbf{w}$ given $(\mathbf{t}, \mathbf{X})$ in this context is

$$\hat{\mathbf{w}}_{MAP} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{t} \tag{3.1}$$

where $\lambda = \sigma^2/\tau^2$.

3.3. *Cross validation - 20 pts.* In this problem, you will write a function that performs K-fold cross validation (CV) procedure to tune the penalty parameter λ in Ridge regression. CV procedure is one of the most commonly used methods for tuning hyperparameters. *In this question, you shouldn't use the package* `scikit-learn` to perform CV. *You should implement all of the below functions yourself. You may use* `numpy` *and* `scipy` *for basic math operations such as linear algebra, sampling etc.*

In class we learned *training*, *test*, and *validation* procedures which assumes that you have enough data and you can set aside a validation set and a test set to use it for assessing the performance of your machine learning algorithm. However in practice, this may be problematic since we may not have enough data. A remedy to this issue is K-fold cross- validation which uses a part of the available data to fit the model, and a different part to test it. K-fold CV procedure splits the data into K equal-sized parts; for example, when K = 5, the scenario looks like this:

Fig 1: credit: Elements of Statistical Learning

1. We first set aside a test dataset and never use it until the training and parameter tuning procedures are complete. We will use this data for final evaluation. In this question, test data is provided to you as a separate dataset.
2. CV error estimates the test error of a particular hyperparameter choice. For a particular hyperparameter value, we split the training data into K blocks (See the figure), and for $k = 1, 2, ..., K$ we use the k-th block for validation and the remaining $K - 1$ blocks are for training. Therefore, we train and validate our algorithm K times. Our CV estimate for the test error for that particular hyperparameter choice is given by the average validation error across these K blocks.
3. We repeat the above procedure for several hyperparameter choices and choose the one that provides us with the smalles CV error (which is an estimate for the test error).

Below, we will code the above procedure for tuning the regularization parameter in linear regression which is a hyperparameter. Your `cross_validation` function will rely on 6 short functions which are defined below along with their variables.

- `data` is a variable and refers to a $(\mathbf{t}, \mathbf{X})$ pair (can be test, training, or validation) where $\mathbf{t}$ is the target (response) vector, and $\mathbf{X}$ is the feature matrix.

- `model` is a variable and refers to the coefficients of the trained model, i.e. $\hat{\mathbf{w}}_\lambda$.
- `data_shf = shuffle_data(data)` is a function and takes `data` as an argument and returns its randomly permuted version along the samples. Here, we are considering a uniformly random permutation of the training data. Note that $\mathbf{t}$ and $\mathbf{X}$ need to be permuted the same way preserving the target-feature pairs.
- `data_fold, data_rest = split_data(data, num_folds, fold)` is a function that takes `data`, number of partitions as `num_folds` and the selected partition `fold` as its arguments and returns the selected partition (block) `fold` as `data_fold`, and the remaining data as `data_rest`. If we consider 5-fold cross validation, `num_folds`=5, and your function splits the data into 5 blocks and returns the block `fold` ($\in \{1, 2, 3, 4, 5\}$) as the validation fold and the remaining 4 blocks as `data_rest`. Note that `data_rest` $\cup$ `data_fold` = `data`, and `data_rest` $\cap$ `data_fold` $= \emptyset$.
- `model = train_model(data, lambd)` is a function that takes `data` and `lambd` as its arguments, and returns the coefficients of ridge regression with penalty level λ. For simplicity, you may ignore the intercept and use the expression in equation (3.1).
- `predictions = predict(data, model)` is a function that takes `data` and `model` as its arguments, and returns the `predictions` based on `data` and `model`.
- `error = loss(data, model)` is a function which takes `data` and `model` as its arguments and returns the average squared `error` loss based on `model`. This means if `data` is composed of $\mathbf{t} \in \mathbb{R}^n$ and $\mathbf{X} \in \mathbb{R}^{n \times p}$, and model is $\hat{\mathbf{w}}$, then the return value is $\|\mathbf{t} - \mathbf{X}\hat{\mathbf{w}}\|^2/n$.
- `cv_error = cross_validation(data, num_folds, lambd_seq)` is a function that takes the training `data`, number of folds `num_folds`, and a sequence of λ's as `lambd_seq` as its arguments and returns the cross validation error across all λ's. Take `lambd_seq` as evenly spaced 50 numbers over the interval (0.02, 1.5). This means `cv_error` will be a vector of 50 errors corresponding to the values of `lambd_seq`. Your function will look like:

```
data = shuffle_data(data)
for i = 1,2,...,length(lambd_seq)
       lambd = lambd_seq(i)
       cv_loss_lmd = 0.
       for fold = 1,2, ...,num_folds
            val_cv, train_cv = split_data(data, num_folds, fold)
            model = train_model(train_cv, lambd)
            cv_loss_lmd += loss(val_cv, model)
       cv_error(i) = cv_loss_lmd / num_folds
return cv_error
```

(a) Download the dataset from the course webpage `hw1_data.zip` and place and extract in your working directory, or note its location `file_path`. For example, file path could be `/Users/yourname/Desktop/`

- In `Python`:

```
import numpy as np
data_train = {'X': np.genfromtxt('data_train_X.csv', delimiter=','),
              't': np.genfromtxt('data_train_y.csv', delimiter=',')}
data_test = {'X': np.genfromtxt('data_test_X.csv', delimiter=','),
             't': np.genfromtxt('data_test_y.csv', delimiter=',')}
```

(b) Write the above 6 functions, and identify the correct order and arguments to do cross validation.

(c) Find the training and test errors corresponding to each λ in `lambd_seq`. This part does not use the `cross_validation` function but you may find the other functions helpful.

(d) Plot training error, test error, and 5-fold and 10-fold cross validation errors on the same plot for each value in `lambd_seq`. What is the value of λ proposed by your cross validation procedure? Comment on the shapes of the error curves.