National University of Singapore
School of Computing
CS3244: Machine Learning
Week 06 - Tutorial 04-b Solution

Regularization and Validation

Note : Tutorial solution explanation is recorded here.

1. **Cross Validation.** Consider a learning model that takes $m^2 \log m$ seconds of training when using m examples.

   (a) What is the total time (in seconds) needed when running leave one out cross validation(LOOCV) on 30 such models with different parameters to get the final hypothesis?

   When using LOOCV, for each model, we partition the data to m parts, taking $m-1$ for training and 1 for validation orderly. Hence, we need to train with $m-1$ examples for m times for each model. Therefore, the total time we need is:

   $$30 \times m \times (m-1)^2 \log(m-1)$$

   (b) What is the total amount time (in seconds) needed when running 10-fold cross validation on 30 such models with different parameters to get the final hypothesis?

   When using 10-fold cross validation, we partition the data to 10 parts, taking 9 parts for training and 1 part for validation. Hence, we need to train with $\frac{9}{10}m$ examples for 10 times, validate using the last part, and repeat this process 10 times, rotating the part used for validation. Therefore, the time we use to select the best model is:

   $$30 \times 10 \times \left(\frac{9}{10}m\right)^2 \log\left(\frac{9}{10}m\right)$$

   To get to the final hypothesis, we use all the m examples to train on the model we selected, which takes $m^2 \log m$ seconds. Hence, the total time we need is:

   $$\left(30 \times 10 \times \left(\frac{9}{10}m\right)^2 \log\left(\frac{9}{10}m\right)\right) + m^2 \log m$$

   Where do the 30 models come from? This is for model selection. They can come from using the same learning algorithm $\mathcal{A}$ and using different hyperparameters for testing their efficacy (e.g., different levels of regularization). They can also come from using different learners $\mathcal{A}_{1\ldots|A|}$ (e.g. linear regression, decision tree, etc.).
   Note the tradeoff in doing k-fold where k is small or large on training performance: more folds linearly scales up the training time. In particular, for large datasets where m is big (e.g., most deep learning scenarios), any form of cross-validation can be prohibitive.
   Note that we typically don't retrain a LOOCV system for all m examples, because m is trivially close to $m-1$, so any of the h^- hypotheses would be adequate. Also note that the testing time is not included in the analysis.

2. **Solving for Regularisation.** In this problem, we are working on a linear regression problem with regularisation on points in a 2D space. The figure below shows contour plots for the linear regression problem with different regulariser. The ellipsis contours represent the squared error term. The circle and diamond contours represent the regularisation penalty term for $\lambda = 5$.

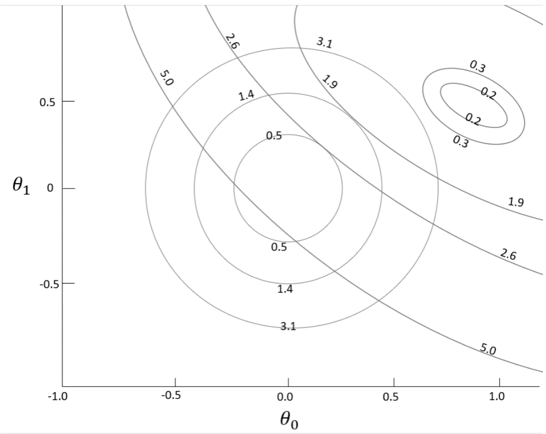

Figure 1: Contour plots for the linear regression problem with L2 regularisation

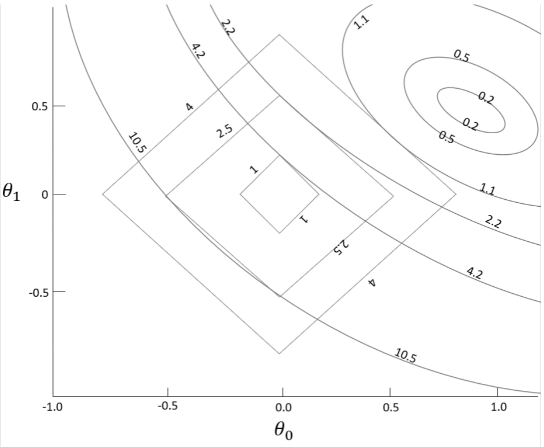

Figure 2: Contour plots for the linear regression problem with L1 regularisation

For each of the following cases, determine the rough values of θ_0 and θ_1 using the figures as reference.

(a) No regularisation.

(b) L2 regularisation with $\lambda = 5$.

(c) L1 regularisation with $\lambda = 5$.

You have to check for different values of regularisation cost and squared error term to see which gives the smallest sum. Also note that the contours (as given in the problem description), already figure in $\lambda = 5$, so there's no need to multiply by λ when computing the sum.

(a) $\theta_0 = 0.9$, $\theta_1 = 0.5$. Cost: approximately 0: no MSE loss and no regularization penalty.

(b) $\theta_0 = 0.2$, $\theta_1 = 0.25$. Cost: 3.1 = 2.6 (MSE) + 0.5 (L2 penalty).

(c) $\theta_0 = 0.0$, $\theta_1 = 0.55$. Cost: 4.7 = 2.2 (MSE) + 2.5 (L1 penalty).

In contrast to L1 norm regularization, L2 norm regularization heavily penalizes larger weights for any feature. L2 thus attempts to pull **all** *weights towards small values. In contrast, L1's linear constraint, may use up its soft-constrained "budget" C on any set of weights, regardless of its offset from zero.*

As shown in the diagram above, optimizing θ for minimal L_{train} can often bump into a corner of the L1 norm, causing the weight for the corresponding axis to be set to 0. In L2, since it is a sphere, all points on the L2 budget surface has an equal chance of minimizing L_{train}. Just for your information, the idea of l^p norms generalizes the concept of norms

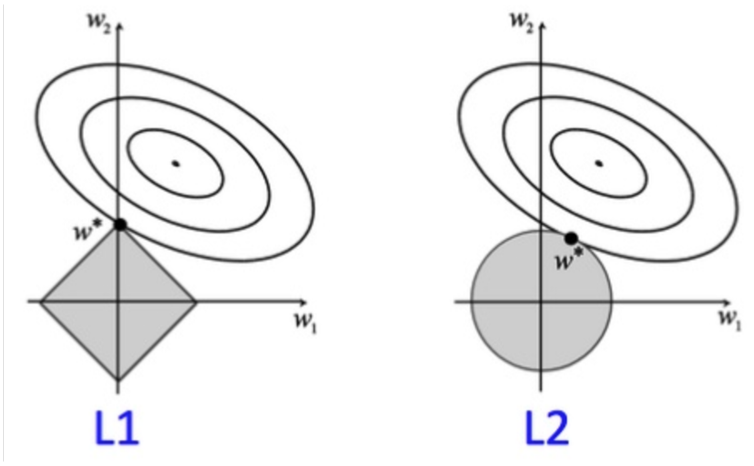

Figure 3: L1 versus L2 norms choosing the best $\theta = \theta^*$. Note that due to the topology of the L1 norm, the "points" on the L1 norm provide a larger opportunity to set a component of θ to zero.

(and spaces) to encompass both L1 and L2 as part of a more general model where the l^p norm is given by $||x||_p = (|x_1|^p + |x_2|^p + ... + |x_n|^p)^{\frac{1}{p}}$. When $p = 0$, strictly speaking l^0 isn't well defined, but can be taken to mean the number of non-zero entries in θ. In this sense L0 "norm" can be thought of the minimizing the number of non-zero weights, similar to the "hard constraint" when we viewed $\mathcal{H}_2$ as a regularized version of $\mathcal{H}_{10}$.

It's also worthwhile knowing that L1 and L2 norm regularization also go by different names in different fields. When studying regression from a statistics perspective, L1 regularization is also known as LASSO by statisticians. L2 regularization, in the same context is known as ridge regression. You can employ more than only form of regularization simultaneously, and when one combines both L1 and L2, this is sometimes referred to as elastic net regularization.

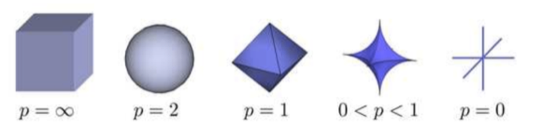

Figure 4: The space of possible regularization outcomes with fixed budget C in under various l^p norms in $\mathbb{R}^3$. Source: https://www.quora.com/What-is-the-difference-between-L1-and-L2-regularization

3. **Variants of Validation.** In class, we have focused on validation on a set of data that is not time-based. Consider a recommender system where new shows are recommended for a consumer to watch based off their past selections and habits. Assume we have features of both the users and items. We also have time series data on the user's actions in such a case, in the form of tuples such as <user u watched show s at time t>.

   (a) Suppose we want to estimate the generalisation error of such a recommender system. What is a shortcoming of the usual validation technique used in class in this case?

The fact that the data also depends on time means that an arbitrary split does not make sense and constitutes data snooping. For example, if the training set is all the even days, and the test set is all the odd days, then we can easily interpolate from the training set to get the value of the test set and obtain very good accuracy. Instead, we want to simulate a real-life scenario where one would have all the past data and need to make a prediction about the future.

(b) Propose how you would modify the usual validation technique to fix this shortcoming.
The change to the usual validation strategy is to divide the data into consecutive blocks. The earliest set of data will be used for training, the next set for validation, and the last set chronologically for testing. For example, if given data from 2013 to 2018, the data from 2013-2014 would be used for training, data from 2015-2016 used for validation and lastly 2017-2018 used as the test set.

(c) Now we consider cross validation on time series data. Suppose we want to perform leave one out cross validation on this time series set of data [1, 2, 3, 4].
Describe how you would perform cross validation, with which sets of data you would use to train and which set you would use as the testing data for each fold.
Similar to how the data cannot be randomly split for validation on time series data. With four segments of data, we can accomplish three-fold validation. The cross validation method would have the following folds:

i. *Training: {1}, Validation: {2}*
ii. *Training: {2}, Validation: {3}*
iii. *Training: {3}, Validation: {4}*

There are certain alternative methods that we can think of but let's explore why these are not favored. Here's one alternative:

i. *Training: {1}, Validation: {2}*
ii. *Training: {1,2}, Validation: {3}*
iii. *Training: {1,2,3}, Validation: {4}*

Having more data for the third line is good, and we would expect to obtain better performance as it has access to more contextual data as input. So we would expect a model trained on $1, 2, 3$ *to outperform one trained on* 1. *But the problem is that the models obtained in the different lines would be of different quality, and are not comparable.*
Here's another alternative. Why would this not be a good idea?

i. *Training: {1}, Validation: {2,3,4}*
ii. *Training: {2}, Validation: {3,4}*
iii. *Training: {3}, Validation: {4}*

This case uses more data for validation, which is normally a good thing because we can lower the variance of our estimate of L_{test}. *However, further future time steps* *are generally harder to predict, such that in the first line, the accuracy of predicting the outcome for Segment 2 would be easier that for Segment 4. Averaging the loss over all of the different segments would not be comparable.*

The key is to ensure that we have comparable training and testing data so that the averaging across performance across folds is comparable. Also note that there is no testing in this case, as we are performing cross-validation.