Lab#7: Bias-variance Tradeoff

Prepared by: Teeradaj Racharak (r.teeradaj@gmail.com)

An important concept in machine learning is the bias-variance tradeoff. Models with high bias are not complex enough for the data and tend to underfit. In contrast, models with high variance overfit to the training data.

Today, you will practice 'learning curves' by plotting training and test errors to diagnose bias-variance problem.

Learning curves with linear regression[1]

Problem 1.1 [Python from scratch]. In this exercise, you will code to generate the learning curves for diagnosing your learning algorithms. Recall that a learning curves plots training and cross validation error as a function of training set size. Your task is to complete the function `learningCurve` so that it returns a vector of errors for the training set and cross validation set.

To plot the learning curves, we need a training and cross validation set error for different training set size. To obtain different training set sizes, you should use different subsets of the original training set `X`. Specifically, for a training set size of `i`, you should use the first `i` examples (*i.e.* `X[:i, :]` and `y[:i, :]`).

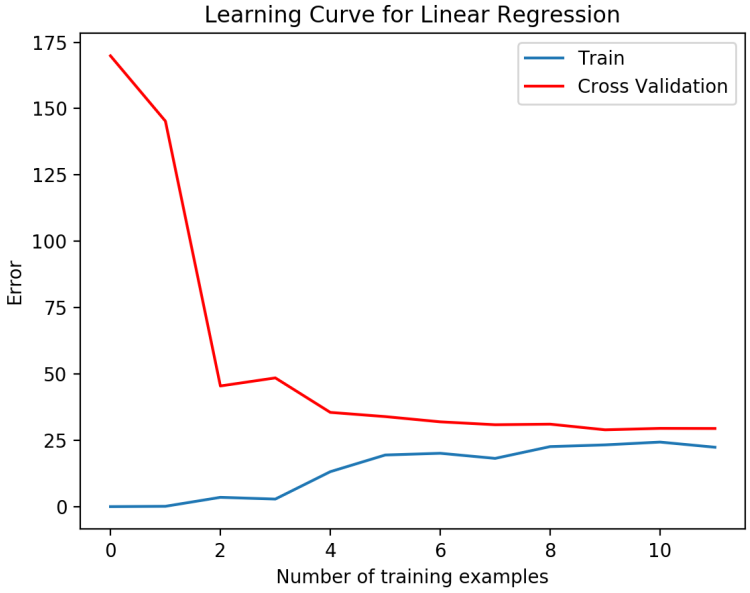

Figure 1 Learning curves for linear regression

[1] This exercise is taken from and revised from Andrew Ng's machine learning assignments (Coursera).

You can use the function gradientDescent to find the optimal $\boldsymbol{\theta}$. Note that λ is passed as a parameter to the function learningCurve. After learning parameter $\boldsymbol{\theta}$, you should compute the error on the training and cross validation sets. When you are computing the training set error, make sure you compute it on the training subset (*i.e.* X[:i, :] and y[:i, i]) (instead of the entire training set). However, for the cross validation error, you should compute it over the entire cross validation set. You should store the computed errors in the vectors J_train and J_cv.

When you have finished your implementation, you should obtain a plot as above. From Figure 1, you may observe that both the train error and cross validation error are high when the number of training examples is increased. This reflects a 'high bias' problem in the model *i.e.* the linear regression model is too simple and is unable to fit our dataset well. In the next exercise, you will implement polynomial regression to fit a better model for this dataset.

Learning curves with polynomial regression

The problem with our linear model was that it was too simple for the data and resulted in underfitting (high bias). To address this problem, we add more features and our hypothesis becomes as follows:

$$\begin{aligned} h_\theta(x) &= \theta_0 + \theta_1 * (\text{feature}) + \theta_2 * (\text{feature})^2 + \ldots + \theta_p * (\text{feature})^p \\ &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_p x_p \end{aligned}$$

Noted that by defining $x_1 = (\text{feature})$, $x_2 = (\text{feature})^2$, …, $x_p = (\text{feature})^p$, we will obtain a linear regression model where the features are the various powers of the original value.

Problem 2.1 [Python with scikit-learn]. Now, you will add more features using the higher powers of the existing feature x in the dataset. Your task is to complete the code in Step#4 (of the guide code) so that it maps the original training set X of size $m \times 1$ to the polynomial of degree 8.

It is worth mentioning that, even though we have polynomial terms in our feature vector, we are solving a 'linear' regression optimization problem. The polynomial terms have simply turned into features that we can use for linear regression. And, we are still using the same cost function and gradient computation.

Problem 2.2 [Python with scikit-learn]. Notice that if you train the data with polynomial features directly, it will not run well as the features would be badly scaled (*e.g.* an example with $x = 40$ will have a feature $x_8 = 40^8 \approx 6.5 \times 10^{12}$). So, you are asked to complete feature normalization in Step#5 (of the guide code).

Problem 2.3 [Python from scratch]. Now, you are ready to learn the parameters $\boldsymbol{\theta}$ and employ the learning curves technique. After you have made a plot, you would see that the training error is low but the cross validation error is high (*cf.* Figure 2). This indicates a high variance problem (overfitting).

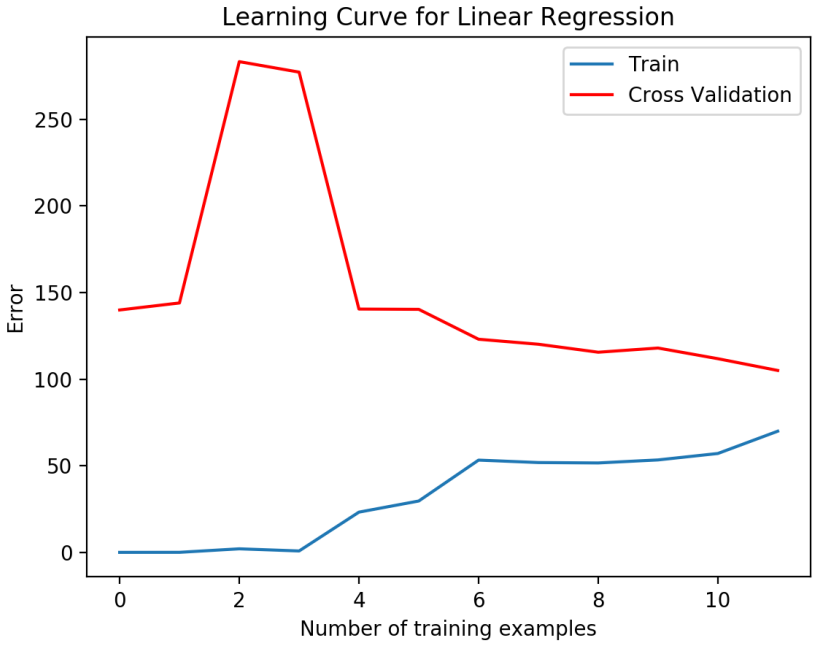

Figure 2 Learning curves for polynomial regression

Problem 2.4 [Python from scratch]. One way to address overfitting is to add regularization to the model. In this part of exercise, you will learn how the regularization parameter λ affects the bias-variance of the regression.

If you try to set $\lambda = 1$, you should see that both the cross validation and training error converge to a relatively low value. This implies that $\lambda = 1$ regularized polynomial regression model does not have a high bias or high variance problem. That is, $\lambda = 1$ achieves a good trade-off between bias and variance.

If you try to set $\lambda = 100$, you should see that the model is unable to fit the training data. We illustrate these experiments in Figure 3.

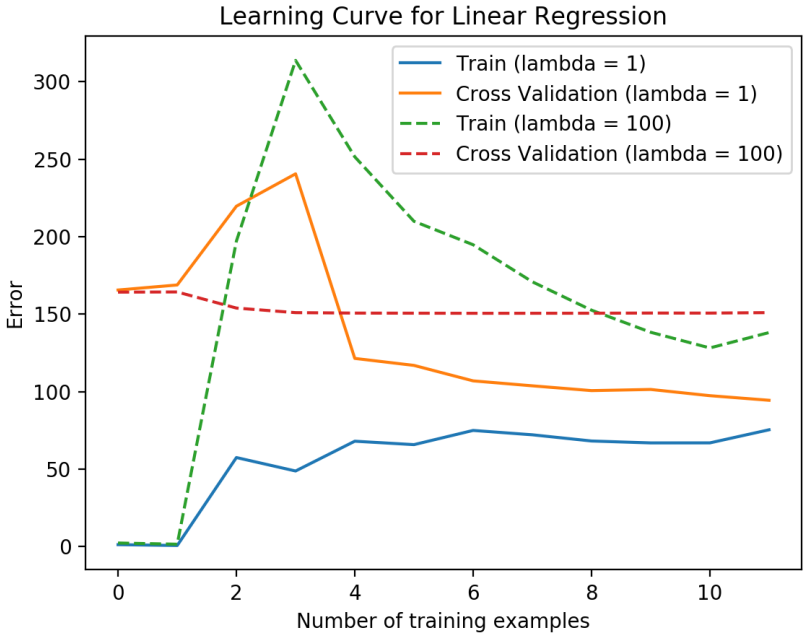

Figure 3 Learning curves when regularized by 1 and 100

Selecting λ using a cross validation set

Now, you observed that the choice of λ can significantly affect the results of regularized polynomial regression on the training and cross validation errors. In particular, a model without regularization (*i.e.* $\lambda = 0$) fits the training set well but does not generalize for new examples. In contrast, a model with too much regularization ($\lambda = 100$) does not fit the training set and testing set well. A good choice of λ (*e.g.* $\lambda = 1$) can provide a good fit to the data.

Problem 3.1. Now, you will implement an automated method to select the parameter λ. To implement this method, you will use a cross validation set to evaluate how good each λ value is. After selecting the best value of λ using the cross validation set, you can then evaluate the model on the test set to estimate how well the model will perform on actual unseen data.

Your task is to complete the code in Step#6 (of the guide code). You may use the function `learningCurve` to train the model using different values of λ and compute the training error and cross validation error; then set the value of λ from the range [0,11]. After you have completed the code, you will see that the best value of λ is around 1 (*cf.* Figure 4).

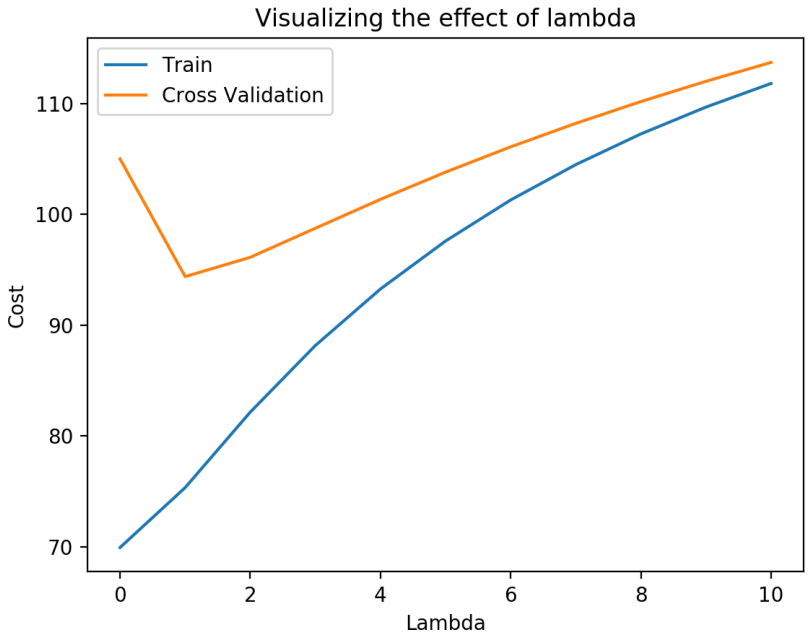

Figure 4 Selecting the best λ from cross validation set

Hopefully, you get the main idea of learning curves ! Though we have practiced only with Python today, we can re-do the same things here using TensorFlow. That is, we write all those information as event log files and use TensorBoard to visual them.