ECE 469/ECE 568 Machine Learning

Textbook:
Machine Learning: a Probabilistic Perspective by Kevin Patrick Murphy

Southern Illinois University

September 13, 2024

A recap for the last lecture

- In the last lecture, we discussed the followings:
  - Introduction to regularization to mitigate overfitting
  - Types of regularization techniques for ML
  - Intuitions behind mathematical formulations for regularizers

Hyperparameters in machine learning models

- Recall that we learned about model parameters and hyperparameters:
  - Model training –> model parameters
  - Hyperparameter tuning –> best hyperparameters
- The behavior of the learning algorithm can be controlled by using several settings, which are termed "hyperparameters".
- These hyperparameters are not typically learned by the learning algorithm itself.

Hyperparameters in machine learning models

- Examples for hyperparameters:
  1. In our polynomial regression example, there is a single hyperparameter: the degree of the polynomial (M) –> This acts as a "capacity" hyperparameter.
  2. You have also learned that in regularization, the value of λ can be used to control the strength of weight decay. Hence, λ is another example of a hyperparameter.
- Sometimes, we may choose a learning architecture/framework setting to be a hyperparameter (that the learning algorithm does not learn) because this particular setting is difficult to optimize.
- Typically, we do not try to learn hyperparameters because it is not appropriate to learn that hyperparameter on the training set.

Validation set

- Recall that a test set consists examples coming from the same distribution as the training set. But data in test set and training set are independent.
- Note that the test set is not used in any way to learn the model, including its hyperparameters.
- A validation set can be used to choose appropriate values for hyperparameters.
- Data from the test set may not be used in the validation set.

Validation set

Validation set

- We may split the training data into two disjoint subsets: -> One subset is used to learn the model parameters: –> Training set.
- While the other subset is the validation set, which is used to estimate the generalization error during or after training, allowing for the hyperparameters to be updated accordingly.
- The subset of data used to guide the selection of hyperparameters is called the validation set.

Validation set

- Validation set can be used to tune/fine-tune hyperparameters.
- Validation data can be dedicated or shared with training data.
- For example, the training data can be split into a 80% training set and a 20% validation set.
- In cross-validation techniques, training and validation may be performed on the same set of training data.
- The training set is used to train the machine learning model, while the validation set is used to select the hyperparameters for the final model.
- The test set is typically independent from both training and validation data.
- Test data-set is used to estimate the final performance metrics such as the test error after all hyperparameter selections have been completed.

Training, validation and testing

- Recall the splitting of data-sets into training, validation, and test sets.

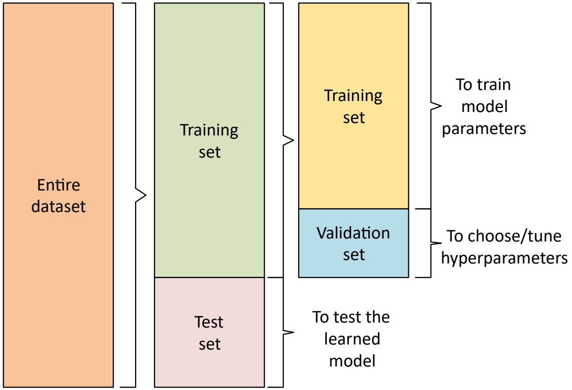

Entire dataset
Training set
Test set
Training set
Validation set
To train model parameters
To choose/tune hyperparameters
To test the learned model

Figure: Splitting a data-set into training, validating, and testing sets

Regularization in machine learning

- Here, we formally define regularized loss function.

$$\underbrace{\tilde{J}(\mathbf{w})}_{\text{regularized loss}} = \underbrace{J(\mathbf{w})}_{\text{original loss function}} + \underbrace{\lambda}_{\text{regularization hyperparameter}} \times \underbrace{R(\mathbf{w})}_{\text{regularizer}}$$

- The balance between the regularized loss and original loss function is The balance is controlled by the regularization parameter λ.
- The most commonly used regularization is the l_2 regularization.

$$R_{\mathbf{w}} = \frac{1}{N}||\mathbf{w}||_2^2 = \frac{1}{N}\mathbf{w}^T\mathbf{w},$$

  where N is the size of the data-set.
- $||\mathbf{w}||_2$ is l_2-norm defined as $||\mathbf{w}|| = \sqrt{\mathbf{w}^T\mathbf{w}} = \sqrt{w_0^2 + w_1^2 + \cdots + w_n^2}$.
- It encourages the optimizer ot find a model with the smallest l_2 norm.
- In deep learning, the most commonly used regularizer is l_2 regularization (a.k.a. weight-decay regularizer).

Gradient descent with l_2 regularization

- Recall that the Ridge (l_2) regularized loss is

$$\tilde{J}(\mathbf{w}) = J(\mathbf{w}) + \frac{\lambda}{N}\mathbf{w}^T\mathbf{w}$$

- The weight update in gradient descent with l_2 regularization is

$$\mathbf{w} := \mathbf{w} - \alpha\left(\nabla\tilde{J}(\mathbf{w})\right)$$

- The gradient of the regularized loss is given by

$$\nabla\tilde{J}(\mathbf{w}) = \nabla J(\mathbf{w}) + \frac{2\lambda}{N}\mathbf{w}$$

- Then, the weight update is given by

$$\mathbf{w} := \mathbf{w} - \alpha\left(\nabla J(\mathbf{w}) + \frac{2\lambda}{N}\mathbf{w}\right) = \left(1 - \frac{2\alpha\lambda}{N}\right)\mathbf{w} - \alpha\nabla J(\mathbf{w})$$

- The term $(1 - 2\alpha\lambda/N)$ in-front of $\mathbf{w}$ corresponds to a weight decay.
- This is because the gradient descent with learning rate α on the regularized loss $R(\mathbf{w})$ is now equivalent to decaying $\mathbf{w}$ by a scalar factor of $(1 - \alpha\lambda)$ and then applying the standard gradient descent.

Other regularizers

- Another regularizer can be chosen as $R(\mathbf{w}) = ||\mathbf{w}||_1/N = \frac{1}{N}\sum_{i=1}^{M}|w_i|$, where N is the size of the training data-set.
- This is termed as the least absolute shrinkage and selection operator (LASSO).
- In deep learning, other common regularizers include dropout, data augmentation, regularizing the spectral norm of the weight matrices, etc.
- Regularization in deep learning is an active research area.

Explicit vs. implicit regularization

- In explicit regularization, we explicitly add a term to the optimization objective function.
- Implicit regularization is all other forms of regularization such as early stopping, using a robust loss function, and discarding outliers.

Analytical solution for the optimal weights with Ridge regression

- Let us consider a linear regression model: $f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T\mathbf{x}$.
- The modified loss function with Ridge regularizer is given by

$$J(\mathbf{w}) = \frac{1}{N}\sum_{m=1}^{M}\left[y^{(m)} - \mathbf{w}^T\mathbf{x}^{(m)}\right]^2 + \frac{\lambda}{N}\sum_{i=0}^{n} w_i^2$$

- This loss function can also be written as

$$J(\mathbf{w}) = \frac{1}{N}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^T\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right) + \frac{\lambda}{N}\mathbf{w}^T\mathbf{w}$$

where $\mathbf{X} = [\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots, \mathbf{x}^{(N)}]^T$ and $\mathbf{y} = [y^{(1)}, y^{(2)}, \cdots, y^{(N)}]^T$

Analytical solution for Ridge regression

- This loss function can be expanded as

$$
\begin{aligned}
J(\mathbf{w}) &= \frac{1}{N}\left(\mathbf{X}\mathbf{w}-\mathbf{y}\right)^T\left(\mathbf{X}\mathbf{w}-\mathbf{y}\right)+\frac{\lambda}{N}\mathbf{w}^T\mathbf{w} \\
&= \frac{1}{N}\left((\mathbf{X}\mathbf{w})^T-\mathbf{y}^T\right)\left(\mathbf{X}\mathbf{w}-\mathbf{y}\right)+\frac{\lambda}{N}\mathbf{w}^T\mathbf{w} \\
&= \frac{1}{N}\left(\mathbf{w}^T\mathbf{X}^T-\mathbf{y}^T\right)\left(\mathbf{X}\mathbf{w}-\mathbf{y}\right)+\frac{\lambda}{N}\mathbf{w}^T\mathbf{w} \\
&= \frac{1}{N}\left(\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}-\mathbf{w}^T\mathbf{X}^T\mathbf{y}-\mathbf{y}^T\mathbf{X}\mathbf{w}+\mathbf{y}^T\mathbf{y}\right)+\frac{\lambda}{N}\mathbf{w}^T\mathbf{w}
\end{aligned}
$$

- Useful identities in multivariate calculus:

$f(\mathbf{w})$	$\frac{\partial f}{\partial \mathbf{w}}$
$\mathbf{w}^\top\mathbf{x}$	$\mathbf{x}$
$\mathbf{x}^\top\mathbf{w}$	$\mathbf{x}$
$\mathbf{w}^\top\mathbf{w}$	$2\mathbf{w}$
$\mathbf{w}^\top\mathbf{C}\mathbf{w}$	$2\mathbf{C}\mathbf{w}$

Analytical solution for Ridge regression

- The gradient vector of this loss function can be computed as

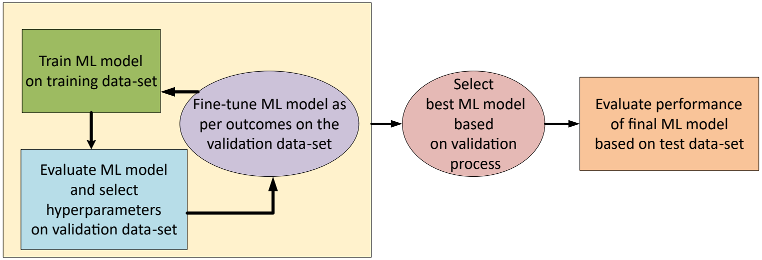
$$\begin{aligned}\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{bmatrix} \frac{\partial J(\mathbf{w})}{\partial w_0} \\ \frac{\partial J(\mathbf{w})}{\partial w_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial w_n} \end{bmatrix} &= \frac{1}{N} \nabla_{\mathbf{w}} \left(\mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} - 2\mathbf{w}^T \mathbf{X}^T \mathbf{y} + \mathbf{y}^T \mathbf{y} + \frac{\lambda}{N} \mathbf{w}^T \mathbf{w} \right) \\ &= \frac{2}{N} \mathbf{X}^T \mathbf{X} \mathbf{w} - \frac{2}{N} \mathbf{X}^T \mathbf{y} + \frac{2\lambda}{N} \mathbf{w}\end{aligned}$$

- Next, the optimal weight vector that minimizes the loss function $J(\mathbf{w})$ can be computed as

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \frac{2}{N} \mathbf{X}^T \mathbf{X} \mathbf{w} - \frac{2}{N} \mathbf{X}^T \mathbf{y} + \frac{2\lambda}{N} \mathbf{w} = 0$$

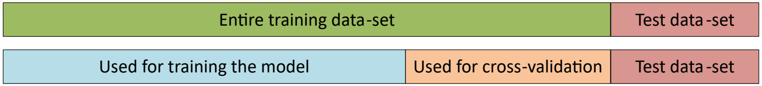
$$\mathbf{w}^* = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

Validation: Model selection via cross-validation


Train ML model on training data-set
Fine-tune ML model as per outcomes on the validation data-set
Evaluate ML model and select hyperparameters on validation data-set
Select best ML model based on validation process
Evaluate performance of final ML model based on test data-set

- Next, we are going to discuss validation techniques to finalize hyperparameters and choosing the final model.

Hold-out (simple) cross-validation



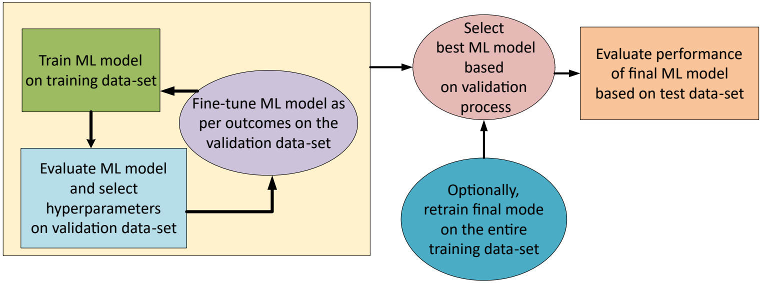
- Let us assume that we have a finite set of models denoted by

$$\mathcal{M} = \{M_1, \cdots, M_D\},$$

where model M_i is the ith degree polynomial regression model.

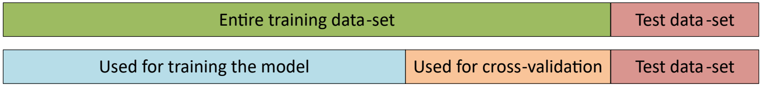
- The hold-out (simple) cross-validation works as follows:
  1. Randomly split the training data set $\mathcal{S}$ into two sets, namely $\mathcal{S}_{\text{train}}$ (say, 80% of the training data) and $\mathcal{S}_{\text{cv}}$ (the remaining 20%), where $\mathcal{S}_{\text{cv}}$ is called the hold-out cross-validation set.
  2. Train each model M_i on $\mathcal{S}_{\text{train}}$ only, to find a set of hypothesis h_i.
  3. Select and output the hypothesis h_i that results in the smallest error $E_{\mathcal{S}_{\text{cv}}}(h_i)$ using the hold-out cross-validation set $\mathcal{S}_{\text{cv}}$.
- Note that $E_{\mathcal{S}_{\text{cv}}}(h_i)$ denotes the mean square error (MSE) of h_i on the cross-validation set $\mathcal{S}_{\text{cv}}$, and it is explicitly referred to as the validation error.

Model selection via cross-validation


Train ML model on training data-set
Evaluate ML model and select hyperparameters on validation data-set
Fine-tune ML model as per outcomes on the validation data-set
Select best ML model based on validation process
Optionally, retrain final mode on the entire training data-set
Evaluate performance of final ML model based on test data-set

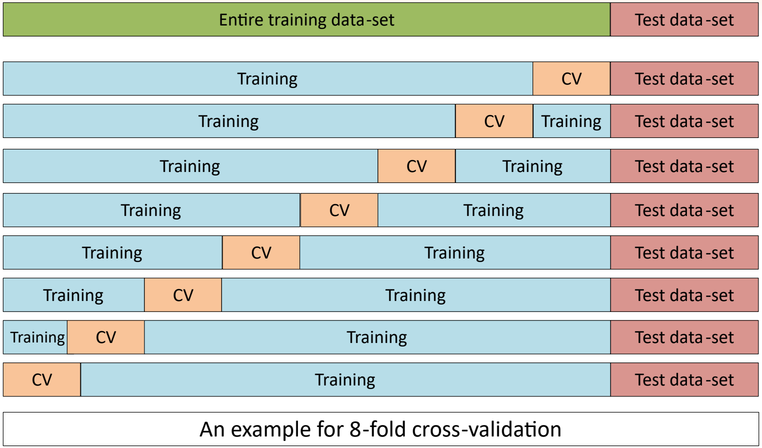
- Optionally, the above third step may also be replaced with selecting the model M_i according to $\arg\min_i E_{\mathcal{S}_{\text{cv}}}(h_i)$, and then retraining M_i on the entire training set $\mathcal{S}$.

Cons of hold-out (simple) cross-validation and remedies



- Cross-validation techniques should allows us to make more efficient use of the data that is available to us.
- One disadvantage of using hold-out cross-validation is that it wastes a pre-defined portion of (say 20% of the training data).
- This is crucial in learning problems in which data is scarce and costly.
- The remedy is to use k-fold cross-validation, which holds out less data each time.

k-fold cross-validation



An example for 8-fold cross-validation

- k-fold cross-validation involves randomly dividing the data-set into k-folds/groups of approximately equal size.
- The first fold is treated as a validation set, and the model is fit on the remaining $k-1$ folds.

k-fold cross validation

- The mean square error (MSE) evaluated on the first held-out fold is denoted by $\mathrm{MSE_1}$.
- This process is repeated k times, each time, a different fold is treated as a validation set.
- For each hold-out fold, the MSE is calculated as $\mathrm{MSE_1}, \mathrm{MSE_2}, \cdots, \mathrm{MSE_k}$.
- The final k-fold CV MSE is computed by averaging these K errors as

$$\mathrm{MSE_{CV}} = \frac{1}{k}\sum_{i=1}^{k} \mathrm{MSE_i}$$

k-fold cross-validation

- k-fold cross-validation algorithm is given below:

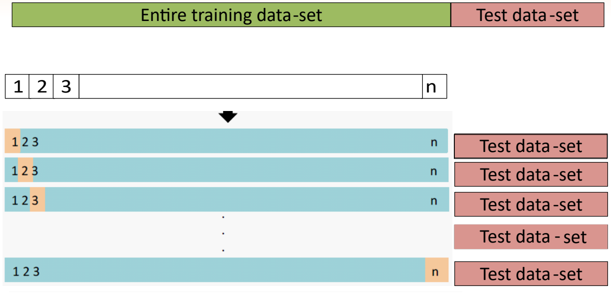
An algorithm for k-fold cross-validation

1. Randomly split the training set $\mathcal{S}$ into k disjoint subsets of N/k training examples each, namely $\{\mathcal{S}_1, \cdots, \mathcal{S}_k\}$. Here $N = |\mathcal{S}|$ is the size of the training set.
2. For each model M_i, evaluate the followings:
    - For $j = 1, \cdots, k$, first train the model M_i on $\mathcal{S}_1 \cup \mathcal{S}_2 \cup \cdots \mathcal{S}_{j-1} \cup \mathcal{S}_{j+1} \cup \cdots \cup \mathcal{S}_k$ (i.e., train on all the data except $\mathcal{S}_j$) to obtain some hypothesis h_{ij}.
    - Then, test the hypothesis h_{ij} on $\mathcal{S}_j$, to compute $E_{\mathcal{S}_j}(h_{ij})$.
    - Finally, the estimated cross-validation error of model M_i is calculated as the average of the $E_{\mathcal{S}_j}(h_{ij})$ (averaged over j).
3. Pick the model M_i with the lowest estimated cross-validation error, and retrain that model on the entire training set $\mathcal{S}$.

k-fold cross-validation

- The resulting hypothesis is then output as the final learning model.
- An independent test data-set is used to evaluate the performance of this final model.
- A typical choice for the number of folds is $k = 10$.
- k-fold cross validation is computationally more expensive than hold-out cross-validation, since it needs train to each model k times.

Leave-one-out cross-validation (LOOCV)



- When data is really scarce, the extreme choice of $k = n$, the size of the training data-set, can be considered to leave out as little data as possible each time.
- Since we are going to hold out one training example at a time, this method is called leave-one-out cross-validation.
- In this setting, we repeatedly train on all but one of the training examples in $\mathcal{S}$, and validate on that held-out example.

Leave-one-out cross-validation (LOOCV)

- Then, the resulting $n = k$ errors can be averaged together to obtain the estimate of the cross-validation error for a learning model.
- The final LOOCV mean square error is computed by averaging all n errors as

$$\mathrm{MSE_{CV}} = \frac{1}{n}\sum_{i=1}^{n}\mathrm{MSE_i}$$

- LOOCV has a two of key advantages over the simple hold-out validation set technique.
- First, LOOCV has far less bias because it repeatedly fits the model on training sets that contain $n - 1$ examples, which is almost as many as are in the entire training data set.
- Consequently, LOOCV would not to overestimate the validation error as much as the hold-out simple validation set technique does.