





Deep learning for agronomy

3rd year engineer

Ammouri Bilel

 : ESAM
 : ammouri-bilel
 : bilelammouri
 : Ammouri-Bilel
 : 0000-0002-5491-5172

Plan

- 1 Cross Validation
- 2 Regularization
- 3 Hyperparameters
- 4 QUIZ

Choosing Complexity

Choosing the model solely based on the lowest training error could lead to favoring overfitting models.

Consequently, opting for the model with the lowest test error might seem like the logical alternative, given that it approximates the true error.

- ① This is almost right. However, the test set has been tampered, thus is no longer is an unbiased estimate of the true error.
- ② Although we didn't technically train the model on the test set, our model selection was influenced by its performance (the test set).
 - The test set can no longer serve as a surrogate for the unknown, as it has been repeatedly probed to determine the optimal model.

It is crucial to refrain from any interference with the test set until the completion of the training process, using it only once to assess the performance of the final chosen model.

Choosing Complexity

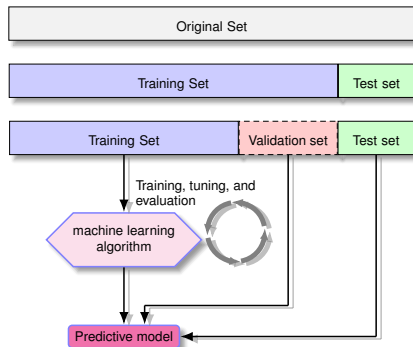
Let's discuss two methods for determining the model's complexity without compromising our test set integrity:

- employing a validation set
- utilizing cross-validation

It's worth noting that while these approaches are effective in selecting the ideal model complexity, they are also instrumental in refining hyperparameters.

Validation Set

So far we have divided our dataset into train and test
We can't use Test to choose our model complexity, so instead, break up Train into ANOTHER dataset
We will pick the model that does best on validation. Note that this now makes the validation error of the "best" model a biased estimate of true error. The test error will be an unbiased estimate though since we never looked at it!



Code validation python

```
train, validation, test = random_split(dataset)
for each model complexity p:
    model = train_model(model_p, train)
    val_err = error(model_p, validation)
    keep track of p with smallest val_err
return best p + error(model_best_p, test)
```

Validation Set

Pros

- Only requires training a model and predicting on the validation set for each complexity of interest
- Easy to implement
- Very fast, and is widely used nowadays in Deep Learning

Cons

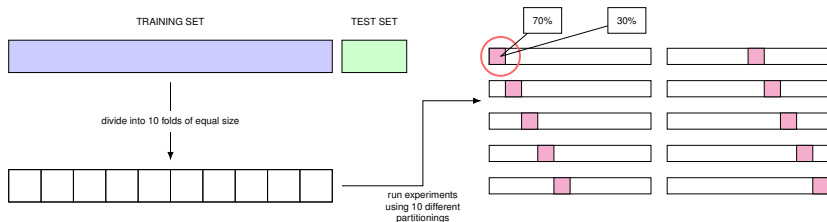
- Have to sacrifice even more training data
- Chance of overfitting is less than when training on the full training set, but there's still a possibility of overfitting

Cross-Validation

Clever idea: Use many small validation sets without losing too much training data.

Still need to break off our test set like before. After doing so, break the training set into k chunks.

For a given model complexity, train it k times. Each time use all but one chunk and use that left out chunk to determine the validation error.



Cross-Validation

Principle

- Split the dataset D in K sets D_k of almost equals size.
- For $k \in 1, \dots, K$:
 - Learn \hat{f}^{-k} from the dataset D minus the set D_k .
 - Compute the empirical error:

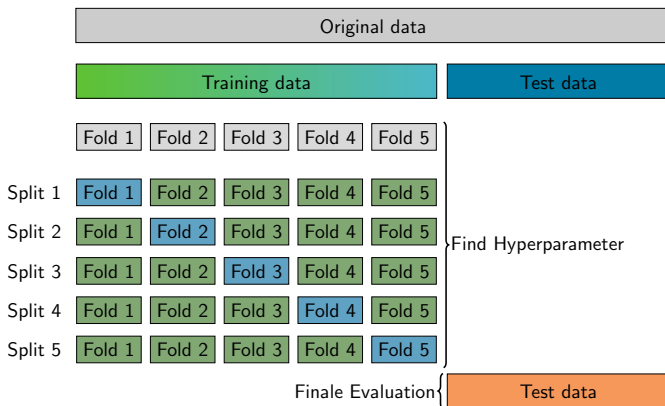
$$R_n^{-v}(\hat{f}^{-k}) = \frac{1}{n_v} \sum_{(X_i, Y_i) \in D_k} l(Y_i, \hat{f}^{-k}(X_i))$$

- Compute the average empirical error:

$$R_n^{CV}(\hat{f}) = \frac{1}{K} \sum_{k=1}^K R_n^{-k}(\hat{f}^{-k})$$

Cross-Validation

For each model complexity / a set of hyperparameters, perform Cross Validation with k iterations and get an average validation error.



Cross-Validation

Code validation python

```
train_set, test_set = random_split(dataset)
# randomly shuffle train_set
# choose a specific k and split it into k groups
for each model complexity p:
    for i in [1, k]:
        model = train_model(model_p, chunks - i)
        val_err = error(model, chunk_i)
        avg_val_err = average val_err over chunks
        keep track of p with smallest avg_val_err
# retrain the model with best complexity p on the full training set
return the error of that model on the test set
```

Cross-Validation

Pros

- Prevent overfitting: By training the model on multiple folds instead of only 1 training set, this allow the model with the best generalization capabilities.
- Don't have to actually get rid of any training data!

Cons

- Very slow. For each model selection, we have to train k times
- Very computationally expensive

The choice of k also follows bias-variance tradeoff.

- Lower k : High bias / Higher k : High variance
- In practice, people use $k = 10$ because it achieves a fine medium balance

Before, we used the quality metric that minimized loss:

$$\hat{w} = \arg \min_w L(w)$$

Change quality metric to balance loss with measure of overfitting

- $L(w)$ is the measure of fit
- $R(w)$ measures the magnitude of coefficients

$$\hat{w} = \arg \min_w L(w) + \lambda R(w)$$

λ : regularization parameter

How do we actually measure the magnitude of coefficients?

Magnitude

Sum

$$R(w) = w_0 + w_1 + \dots + w_d$$

Doesn't work because the weights can cancel out (e.g. $w_0 = 1000$, $w_1 = 1000$), which so $R(w)$ doesn't reflect the magnitudes of the weights

Sum of absolute values

$$R(w) = |w_0| + |w_1| + \dots + |w_d| = \|w\|_1$$

It works! We're using L1-norm, for L1-regularization (LASSO)

Sum of squares

$$R(w) = |w_0|^2 + |w_1|^2 + \dots + |w_d|^2 = \|w\|_2^2$$

It works! We're using L2-norm, for L2-regularization (Ridge Regression)

Magnitude

Ridge Regression

Ridge regression is typically used when there is a high correlation between the independent variables. This is because, in the case of multicollinear data, the least square estimates provide unbiased values. However, when collinearity is very high, some bias may occur. To address this issue, a bias matrix is introduced in the equation of Ridge Regression. This powerful regression method helps make the model less susceptible to overfitting.

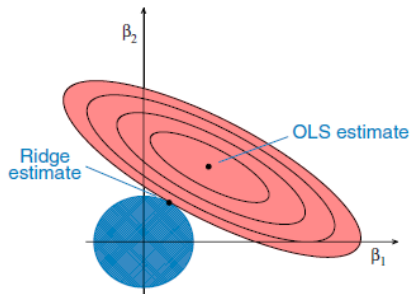


Figure 1: Ridge Regression

$$\beta = (X^tr X + \lambda \times I)^{-1} X^tr Y$$

Magnitude

Lasso Regression

Lasso Regression is a regression technique that combines regularization with feature selection, restricting the absolute size of the regression coefficient.

Consequently, the coefficient value approaches zero, distinguishing it from Ridge Regression.

This feature selection characteristic in Lasso Regression allows for the selection of a specific set of features from the dataset to build the model.

With Lasso Regression, only the necessary features are utilized, while the others are forced to zero.

This helps prevent overfitting in the model.

In instances where the independent variables are highly collinear, Lasso regression selects only one variable and reduces the others to zero.

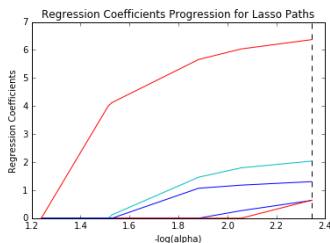


Figura 2: Lasso Regression

$$\frac{1}{N} \sum_{i=1}^N f(x_i, y_i, \alpha, \beta)$$

Ridge Regression

Change quality metric to minimize

$$\hat{w} = \arg \min_w L(w) = \arg \min_w MSE(w) + \lambda \|w\|_2^2$$

λ is a tuning hyperparameter that changes how much the model cares about the regularization term.

Ridge Regression

What if $\lambda = 0$?

$$\hat{w} = \arg \min_w MSE(w) = \hat{w}_{ols}$$

What if $\lambda = \infty$?

- **Case 1:** $w = \vec{0}$, then $\|w\|_2^2 = 0$, which means:

$$L(w) = MSE(\vec{0}) = \text{constant}$$

- **Case 2:** w contains at least 1 $w_i \neq 0$, then $\|w\|_2^2 > 0$:

$$L(w) = MSE(w) + \lambda \|w\|_2^2 \geq \lambda \|w\|_2^2 = \infty$$

Since we're trying to minimize $L(w)$, $\min L(w) = 0$, so $\hat{w} = \vec{0}$.

λ in between?

$$0 \leq \|w\|_2^2 \leq \|w_{ols}\|_2^2$$

There are a number of hyperparameters such as:

- Learning rate (in Gradient Descent)
- Number of iterations (in Gradient Descent)
- Regularization parameter
- Number of folds (in Cross Validation)

A hyperparameter is a parameter whose value is used to control the learning process and is external to the model. By contrast, the values of model parameters are derived via training

Fine-tuning hyperparameters is the process of choosing an optimal set of hyperparameters for the training process. It's painstaking process and there are still a lot of unknowns.

Fix this by normalizing the features so all are on the same scale!

$$\tilde{h}_j(x_i) = \frac{h_j(x_i) - \mu_j(x_1, \dots, x_N)}{\sigma_j(x_1, \dots, x_N)}$$

Where

The mean of feature j :

$$\mu_j(x_1, \dots, x_N) = \frac{1}{N} \sum_i h_j(x_i)$$

The standard deviation of feature j :

$$\sigma_j(x_1, \dots, x_N) = \sqrt{\frac{1}{N} \sum_i (h_j(x_i) - \mu_j(x_1, \dots, x_N))^2}$$

Important: Must scale the test data and all future data using the means and standard deviations of the training set!

Otherwise the units of the model and the units of the data are not comparable!

Regularization parameter

How does λ affect the bias and variance of the model? For each underlined section, select “Low” or “High” appropriately.

- 1 When $\lambda = 0$
The model has (Low / High) Bias and (Low / High) Variance.
- 2 When $\lambda = \infty$
The model has (Low / High) Bias and (Low / High) Variance.