COMPSCI 689
Lecture 9: Advanced Supervised Learning Experiment Designs

Benjamin M. Marlin

College of Information and Computer Sciences
University of Massachusetts Amherst

Slides by Benjamin M. Marlin (marlin@cs.umass.edu).

Review

- So far, we have two basic machine learning experiment designs: the train-test experiment and the train-validation-test experiment.
- In this lecture, we'll introduce additional more complex experiment types.

Experiment Design Principles

- Standard machine learning experiments aim to estimate generalization performance (performance on future data).
- Essentially all machine learning models include model complexity (regularization) hyper-parameters that require some form of validation-set method to select.
- More generally, we are often interested in comparing a proposed method to several baseline or state-of-the-art methods to determine which method has the best generalization performance.
- You **must** apply hyper-parameter selection methods to **all** methods that you compare when assessing generalization performance. It is never acceptable to compare to a baseline method using its default regularization hyper-parameter values on a novel task.

Performance Measures for Classification

- Classification Accuracy (A): Number of correctly classified instances over the data set size.
- Classification Error (E): Number of incorrectly classified instances over the data set size.
- Precision (P): The fraction of true positives to the total of true positives and false positives.
- Recall (R): The fraction of true positives to the total of true positives and false negatives.
- F1 Score: $2(P \cdot R)/(P + R)$
- Run Time (train and/or test)

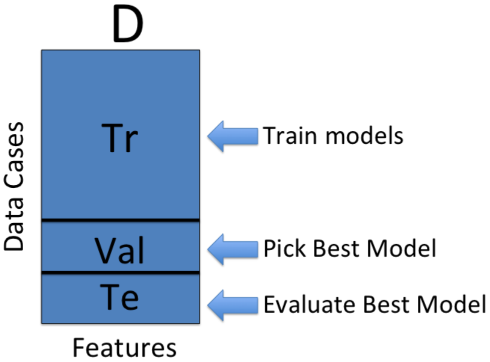
Performance Measures for Regression

- Mean Squared Error (MSE): $\frac{1}{N}\sum_{n=1}^{N}(y_n - f_\theta(\mathbf{x}_n))^2$
- Root Mean Squared Error (RMSE): $\sqrt{\frac{1}{N}\sum_{n=1}^{N}(y_n - f_\theta(\mathbf{x}_n))^2}$
- Mean Absolute Error (MAE): $\frac{1}{N}\sum_{n=1}^{N}|y_n - f_\theta(\mathbf{x}_n)|$
- Run Time (train and/or test)

Experiment Recipe 1: Train-Validation-Test

- Given a data set D, we randomly partition the data cases into a training set (Tr), a validation set (V), and a test set (Te). Typical splits are 60/20/20, 80/10/10, etc.
- Models M_i are learned on Tr for each choice of hyperparameters H_i
- The validation performance Val_i of each model M_i is evaluated on V.
- The hyperparameters H_* with the lowest value of Val_i are selected.
- The model can then be re-trained using these hyperparameters on $Tr + V$, yielding a final model M_*
- Generalization performance is estimated by evaluating the performance of M_* on the test data Te.

Example: Train-Validation-Test



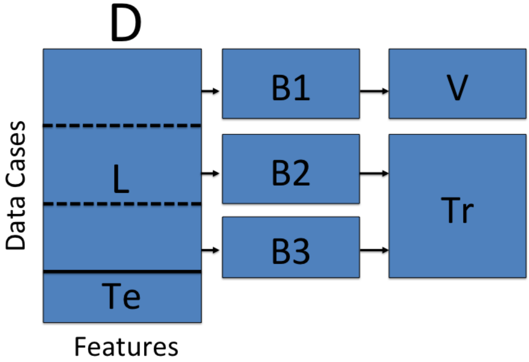
Note that the order of the data cases needs to be randomly shuffled before partitioning D.

Experiment Recipe 2: Crossvalidation-Test

- Randomly partition D into a learning set L and a test set Te (typically 50/50, 80/20, etc).
- We next randomly partition L into a set of K blocks $B_1, ..., B_K$.
- For each crossvalidation fold $k = 1, ..., K$:
  - Let $V = B_k$ and $Tr = L \setminus B_k$ (the remaining $K - 1$ blocks).
  - Learn M_{ik} on Tr for each choice of hyperparameters H_i.
  - Compute performance Val_{ik} of M_{ik} on V.
- Select hyperparameters H_* minimizing $\frac{1}{K}\sum_{k=1}^{K} Val_{ik}$.
- Re-train model on L using these hyperparameters, yielding final model M_*.
- Estimate generalization performance by evaluating error/accuracy of M_* on Te.

Example: 3-Fold Cross Validation and Test

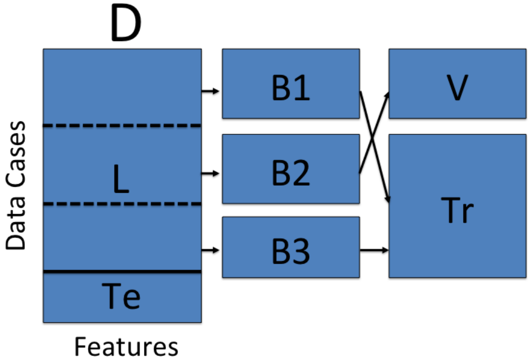
First Cross Validation Fold



Note that the order of the data cases needs to be randomly shuffled before partitioning D into L and Te.

Example: 3-Fold Cross Validation and Test

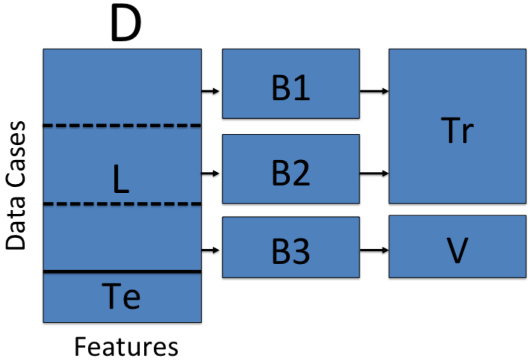
Second Cross Validation Fold



Note that the order of the data cases needs to be randomly shuffled before partitioning D into L and Te.

Example: 3-Fold Cross Validation and Test

Third Cross Validation Fold



Note that the order of the data cases needs to be randomly shuffled before partitioning D into L and Te.

Experiment Recipe 3: Crossvalidation-Crossvalidation

- Randomly partition data set D into a set of J blocks $C_1, \ldots, C_J$.
- For $j = 1, \ldots, J$:
  - Let $Te_j = C_j$ and $L_j = D \setminus C_j$
  - Partition L_j into a set of K blocks $B_1, \ldots, B_K$.
  - For $k = 1, \ldots, K$:
    - Let $V = B_k$ and $Tr = L \setminus B_k$.
    - Learn M_{ik} on Tr for each choice of hyperparameters H_i.
    - Compute error Val_{ik} of M_{ik} on V.
  - Select hyperparameters H_{*j} minimizing $\frac{1}{K}\sum_{k=1}^{K} Val_{ik}$ and re-train model on L_j using these hyperparameters, yielding model M_{*j}.
  - Compute Err_j by evaluating M_{*j} on Te_j.
- Estimate generalization error using $\frac{1}{J}\sum_{j=1}^{J} Err_j$

Experiment Design Trade-Offs

- In cases where the data has a benchmark split into a training set and a test set, we can use Recipes 1 or 2 by preserving the given test set and splitting the given training set into train and validation sets as needed.
- In cases where there is relatively little data, using a single held out test set will have high variance. In these cases, Recipe 3 often provides a better estimate of generalization error, but has much higher computational cost. However, it enables statistical significance testing.
- Choosing larger K in cross validation will reduce variance but increases computational costs. $K = 3, 5, 10$ are common choices for cross validation. $K = N$, also known as Leave-one-out cross validation is also popular when feasible.

Best Practices for Running Experiments

- You should make sure that you fix the random seeds used to partition data and initialize models so all of your results are reproducible (at least by you).
- For models with multiple local optima, you may want to average or max out over initializations to reduce variability.
- You should save the learned model for each fold and hyper-parameter value along with all performance measures of interest in case you need them later.
- You should make sure that the optimal values of all hyper-parameters selected do not fall at the endpoints of the ranges you tested.
- It's a good idea to make plots (for yourself) showing learning curves (train/validation performance vs training iterations) as well as cross-validation plots showing training and validation scores vs hyper-parameter values.