COMPSCI 589
Lecture 4: Overfitting, Regularization, and Crossvalidation

Benjamin M. Marlin

College of Information and Computer Sciences
University of Massachusetts Amherst

Slides by Benjamin M. Marlin (marlin@cs.umass.edu).
Created with support from National Science Foundation Award# IIS-1350522.

Outline

Open Questions

- To date we have introduced five classifiers: KNN, Decision Trees, Naive Bayes, LDA and Logistic Regression.

Open Questions

- To date we have introduced five classifiers: KNN, Decision Trees, Naive Bayes, LDA and Logistic Regression.
- In the case of KNN and Decision Trees, we saw that the results are very sensitive to the number of neighbors, and the depth of the tree.

Open Questions

- To date we have introduced five classifiers: KNN, Decision Trees, Naive Bayes, LDA and Logistic Regression.
- In the case of KNN and Decision Trees, we saw that the results are very sensitive to the number of neighbors, and the depth of the tree.
- In the case of logistic regression, we saw that $P_{\mathbf{w}}(y|\mathbf{x})$ is sensitive to the magnitude of the weights $\mathbf{w}$.

Open Questions

- To date we have introduced five classifiers: KNN, Decision Trees, Naive Bayes, LDA and Logistic Regression.
- In the case of KNN and Decision Trees, we saw that the results are very sensitive to the number of neighbors, and the depth of the tree.
- In the case of logistic regression, we saw that $P_{\mathbf{w}}(y|\mathbf{x})$ is sensitive to the magnitude of the weights $\mathbf{w}$.
- In this lecture, we'll discuss what changes in these models as we vary these parameters and introduce methodology for selecting optimal values for them.

Outline

Model Capacity

- Deterministic classifiers that can represent more complex decision boundaries are said to have higher *capacity* than classifiers that can only represent simpler boundaries.

Model Capacity

- Deterministic classifiers that can represent more complex decision boundaries are said to have higher *capacity* than classifiers that can only represent simpler boundaries.
- Probabilistic classifiers that can represent more complex sets of conditionals $P(Y|X)$ are said to have higher *capacity* than probabilistic classifiers that can only represent simpler sets of conditionals.

Model Capacity

- Deterministic classifiers that can represent more complex decision boundaries are said to have higher *capacity* than classifiers that can only represent simpler boundaries.
- Probabilistic classifiers that can represent more complex sets of conditionals $P(Y|X)$ are said to have higher *capacity* than probabilistic classifiers that can only represent simpler sets of conditionals.
- **Question:** How would you rank the capacity of the classifiers we've seen so far?

Generalization

- **Question:** Why can't we always just use the classifier with the highest possible capacity for every problem?

Generalization

- **Question:** Why can't we always just use the classifier with the highest possible capacity for every problem?
- **Answer:** This would always minimize the error on the training data. However, what we really care about for prediction problems is *generalization*.

Generalization

- **Question:** Why can't we always just use the classifier with the highest possible capacity for every problem?
- **Answer:** This would always minimize the error on the training data. However, what we really care about for prediction problems is *generalization*.
- **Generalization:** The ability of a trained classifier to achieve an error rate on *future, unseen examples* (the generalization error rate) that is comparable to the training error rate.

Generalization

- **Question:** Why can't we always just use the classifier with the highest possible capacity for every problem?

- **Answer:** This would always minimize the error on the training data. However, what we really care about for prediction problems is *generalization*.

- **Generalization:** The ability of a trained classifier to achieve an error rate on *future, unseen examples* (the generalization error rate) that is comparable to the training error rate.

- **Capacity Control:** To achieve optimal generalization performance for a given training set, we often need to control model capacity carefully.

Overfitting and Underfitting

- **Overfitting:** The generalization error for a classifier is much worse than the training error. This usually results from choosing a classifier with too much capacity so that it models the noise in the training data.

- **Underfitting:** Occurs when the capacity of the classifier is too low to capture the actual structure in the training data, leading to both high training error and high generalization error.

Bias-Variance Trade-Off

- **Bias:** A classifier is said to have low *bias* if the true decision boundary or conditionals $P(Y|X)$ can be approximated closely by the model.

Bias-Variance Trade-Off

- **Bias:** A classifier is said to have low *bias* if the true decision boundary or conditionals $P(Y|X)$ can be approximated closely by the model.
- **Variance:** A classifier is said to have low *variance* if the decision boundary or conditionals $P(Y|X)$ it constructs are stable with respect to small changes to the training data.

Bias-Variance Trade-Off

- **Bias:** A classifier is said to have low *bias* if the true decision boundary or conditionals $P(Y|X)$ can be approximated closely by the model.

- **Variance:** A classifier is said to have low *variance* if the decision boundary or conditionals $P(Y|X)$ it constructs are stable with respect to small changes to the training data.

- **Bias-Variance Dilemma:** To achieve low generalization error, we need classifiers that are low-bias and low-variance, but this isn't always possible.

Bias-Variance Trade-Off

- **Bias:** A classifier is said to have low *bias* if the true decision boundary or conditionals $P(Y|X)$ can be approximated closely by the model.

- **Variance:** A classifier is said to have low *variance* if the decision boundary or conditionals $P(Y|X)$ it constructs are stable with respect to small changes to the training data.

- **Bias-Variance Dilemma:** To achieve low generalization error, we need classifiers that are low-bias and low-variance, but this isn't always possible.

- **Bias-Variance and Capacity:** On complex data, models with low capacity have low variance, but high bias; while models with high capacity have low bias, but high variance.

Outline

Hyperparameters

- In order to control the capacity of a classifier, it needs to have capacity control parameters.

Hyperparameters

- In order to control the capacity of a classifier, it needs to have capacity control parameters.
- Because capacity control parameters can not be chosen based on training error, they are often called hyperparameters.

Hyperparameters

- In order to control the capacity of a classifier, it needs to have capacity control parameters.
- Because capacity control parameters can not be chosen based on training error, they are often called hyperparameters.
- **Question:** What are the capacity control parameters for KNN and decision trees?

Hyperparameters

- In order to control the capacity of a classifier, it needs to have capacity control parameters.
- Because capacity control parameters can not be chosen based on training error, they are often called hyperparameters.
- **Question:** What are the capacity control parameters for KNN and decision trees?
- **Question:** What are the capacity control parameters for naive Bayes and logistic regression?

Capacity, Smoothness and Regularization

- **Capacity and Smoothness:** In the case of probabilistic classifiers like NB, LDA, and LR, we can think of capacity in terms of the *smoothness* of $P(Y|X)$.

Capacity, Smoothness and Regularization

- **Capacity and Smoothness:** In the case of probabilistic classifiers like NB, LDA, and LR, we can think of capacity in terms of the *smoothness* of $P(Y|X)$.
- **Regularization:** We can control the smoothness of $P(Y|X)$ using a technique called *regularization* that penalizes parameters that result in overly complex $P(Y|X)$ during learning.

Capacity, Smoothness and Regularization

- **Capacity and Smoothness:** In the case of probabilistic classifiers like NB, LDA, and LR, we can think of capacity in terms of the *smoothness* of $P(Y|X)$.
- **Regularization:** We can control the smoothness of $P(Y|X)$ using a technique called *regularization* that penalizes parameters that result in overly complex $P(Y|X)$ during learning.
- **Laplace Smoothing:** Laplace smoothing is a simple method for smoothing the parameter estimates of a categorical distribution:

$$P_\theta(X = v) = \theta_v = \frac{\alpha + \sum_{i=1}^{n}[x_i = v]}{V\alpha + n}, \quad v = 1..V$$

Capacity, Smoothness and Regularization

- **Capacity and Smoothness:** In the case of probabilistic classifiers like NB, LDA, and LR, we can think of capacity in terms of the *smoothness* of $P(Y|X)$.
- **Regularization:** We can control the smoothness of $P(Y|X)$ using a technique called *regularization* that penalizes parameters that result in overly complex $P(Y|X)$ during learning.
- **Laplace Smoothing:** Laplace smoothing is a simple method for smoothing the parameter estimates of a categorical distribution:

$$P_\theta(X = v) = \theta_v = \frac{\alpha + \sum_{i=1}^{n}[x_i = v]}{V\alpha + n}, \quad v = 1..V$$

- As the regularization hyperparameter α increases, our estimate of the distribution smooths out toward being uniform. This provides capacity control for NB with binary or categorical features.

Regularization For Logistic Regression

- In the case of logistic regression, the smoothness of $P_{\mathbf{w}}(Y|X)$ is determined by the magnitude of $\mathbf{w}$.

Regularization For Logistic Regression

- In the case of logistic regression, the smoothness of $P_{\mathbf{w}}(Y|X)$ is determined by the magnitude of $\mathbf{w}$.
- We can control the magnitude of $\mathbf{w}$ by introducing a regularization term into the conditional log likelihood learning criteria that penalizes the norm of $\mathbf{w}$.

Regularization For Logistic Regression

- In the case of logistic regression, the smoothness of $P_{\mathbf{w}}(Y|X)$ is determined by the magnitude of **w**.
- We can control the magnitude of **w** by introducing a regularization term into the conditional log likelihood learning criteria that penalizes the norm of **w**.

$$\theta_* = \arg\max_{\theta} \sum_{i=1}^{n} \log P(Y = y_i|\mathbf{X} = \mathbf{x}_i) - \lambda||\mathbf{w}||_2^2$$

Regularization For Logistic Regression

- In the case of logistic regression, the smoothness of $P_{\mathbf{w}}(Y|X)$ is determined by the magnitude of $\mathbf{w}$.
- We can control the magnitude of $\mathbf{w}$ by introducing a regularization term into the conditional log likelihood learning criteria that penalizes the norm of $\mathbf{w}$.

$$\theta_* = \arg\max_{\theta} \sum_{i=1}^{n} \log P(Y = y_i|\mathbf{X} = \mathbf{x}_i) - \lambda||\mathbf{w}||_2^2$$

- Some formulations of this problem up-weight the contribution from the data instead:

$$\theta_* = \arg\max_{\theta} C\sum_{i=1}^{n} \log P(Y = y_i|\mathbf{X} = \mathbf{x}_i) - ||\mathbf{w}||_2^2$$

Regularization For Logistic Regression

- This problem can also be solved using other norms, including the ℓ_1 norm. This has the advantage of creating sparse weight vectors:

Regularization For Logistic Regression

- This problem can also be solved using other norms, including the ℓ_1 norm. This has the advantage of creating sparse weight vectors:

$$\theta_* = \arg\max_{\theta} \sum_{i=1}^{n} \log P(Y = y_i|\mathbf{X} = \mathbf{x}_i) - \lambda||\mathbf{w}||_1$$

$$\theta_* = \arg\max_{\theta} C\sum_{i=1}^{n} \log P(Y = y_i|\mathbf{X} = \mathbf{x}_i) - ||\mathbf{w}||_1$$

Regularization For Logistic Regression

- This problem can also be solved using other norms, including the ℓ_1 norm. This has the advantage of creating sparse weight vectors:

$$\theta_* = \arg\max_{\theta} \sum_{i=1}^{n} \log P(Y = y_i|\mathbf{X} = \mathbf{x}_i) - \lambda||\mathbf{w}||_1$$

$$\theta_* = \arg\max_{\theta} C\sum_{i=1}^{n} \log P(Y = y_i|\mathbf{X} = \mathbf{x}_i) - ||\mathbf{w}||_1$$

- The regularization hyperparameters are either λ or C. As λ increases, the learned $P(Y|X)$ will smooth out. As C decreases, the learned $P(Y|X)$ will smooth out.

Outline

Model Selection and Evaluation

- We've identified the parameters that control the capacity of our models, we need a way to choose optimal values for these parameters.

Model Selection and Evaluation

- We've identified the parameters that control the capacity of our models, we need a way to choose optimal values for these parameters.
- In addition, we will want an estimate of the generalization error that the selected parameters achieve.

Model Selection and Evaluation

- We've identified the parameters that control the capacity of our models, we need a way to choose optimal values for these parameters.
- In addition, we will want an estimate of the generalization error that the selected parameters achieve.
- To obtain, valid results, we need to use appropriate methodology and construct learning experiments carefully.

Model Selection and Evaluation

- We've identified the parameters that control the capacity of our models, we need a way to choose optimal values for these parameters.
- In addition, we will want an estimate of the generalization error that the selected parameters achieve.
- To obtain, valid results, we need to use appropriate methodology and construct learning experiments carefully.
- **Guiding Principle:** Data used to estimate generalization error can not be used for any other purpose (ie: model training, hyperparameter selection, feature selection, etc.) or the results of the evaluation will be **biased**.

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.

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

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 error Val_i of each model M_i is evaluated on V.

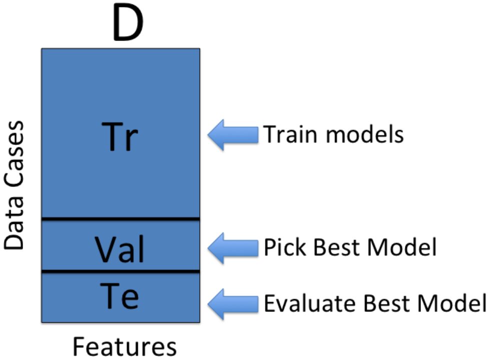
Recipe 1: Train-Validation-Test

- 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 error Val_i of each model M_i is evaluated on V.
- The hyperparameters H_* with the lowest value of Val_i are selected and the classifier is re-trained using these hyperparameters on $Tr + V$, yielding a final model M_*

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 error Val_i of each model M_i is evaluated on V.
- The hyperparameters H_* with the lowest value of Val_i are selected and the classifier is re-trained using these hyperparameters on $Tr + V$, yielding a final model M_*
- Generalization performance is estimated by evaluating error/accuracy 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.

Recipe 2: Crossvalidation-Test

- Randomly partition D into a learning set L and a test set Te (typically 50/50, 80/20, etc).

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 $B_1, ..., B_K$.

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 $k = 1, ..., K$:

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 $Tr = L/B_k$ (the remaining $K - 1$ blocks).

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 $Tr = L/B_k$ (the remaining $K - 1$ blocks).
  - Learn M_{ik} on Tr for each choice of hyperparameters H_i.

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 $Tr = L/B_k$ (the remaining $K - 1$ blocks).
  - Learn M_{ik} on Tr for each choice of hyperparameters H_i.
  - Compute Val_{ik} of M_{ik} on V.

Recipe 2: Crossvalidation-Test

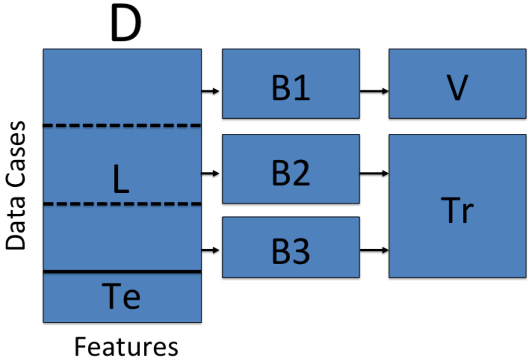
- Randomly partition D into a learning set L and a test set Te (typically 50/50, 80/20, etc).
- We next randomly partition L into a set of K blocks $B_1, ..., B_K$.
- For each crossvalidation fold $k = 1, ..., K$:
  - Let $V = B_k$ and $Tr = L/B_k$ (the remaining $K - 1$ blocks).
  - Learn M_{ik} on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

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 $Tr = L/B_k$ (the remaining $K - 1$ blocks).
  - Learn M_{ik} on Tr for each choice of hyperparameters H_i.
  - Compute Val_{ik} of M_{ik} on V.
- Select hyperparameters H_* minimizing $\frac{1}{K}\sum_{k=1}^{K} Val_{ik}$ and 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

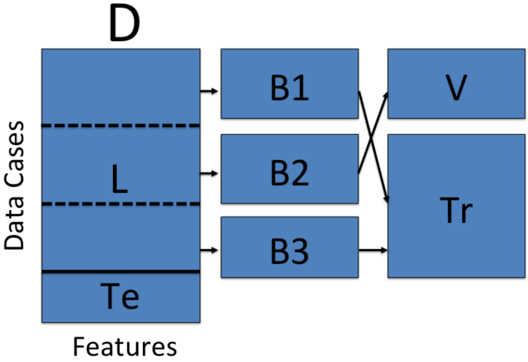
First Cross Validation Fold



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

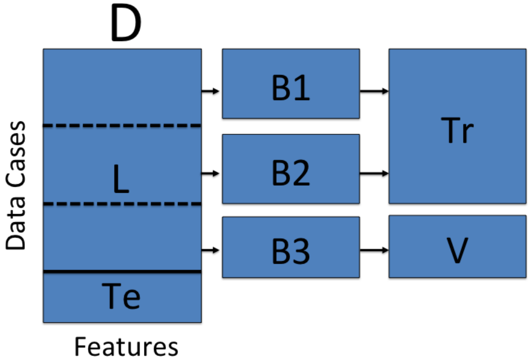
Second Cross Validation Fold



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.

Recipe 3: Random Resampling Validation-Test

- Randomly partition the data cases into a learning set L and a test set Te (typically 50/50, 80/20, etc).

Recipe 3: Random Resampling Validation-Test

- Randomly partition the data cases into a learning set L and a test set Te (typically 50/50, 80/20, etc).
- For sample $s = 1, ..., S$:

Recipe 3: Random Resampling Validation-Test

- Randomly partition the data cases into a learning set L and a test set Te (typically 50/50, 80/20, etc).
- For sample $s = 1, ..., S$:
  - Randomly partition L into Tr and V (again 50/50, 80/20, etc).

Recipe 3: Random Resampling Validation-Test

- Randomly partition the data cases into a learning set L and a test set Te (typically 50/50, 80/20, etc).
- For sample $s = 1, ..., S$:
  - Randomly partition L into Tr and V (again 50/50, 80/20, etc).
  - Learn M_{is} on Tr for each choice of hyperparameters H_i.

Recipe 3: Random Resampling Validation-Test

- Randomly partition the data cases into a learning set L and a test set Te (typically 50/50, 80/20, etc).
- For sample $s = 1, ..., S$:
  - Randomly partition L into Tr and V (again 50/50, 80/20, etc).
  - Learn M_{is} on Tr for each choice of hyperparameters H_i.
  - Compute Val_{is} of M_{is} on V.

Recipe 3: Random Resampling Validation-Test

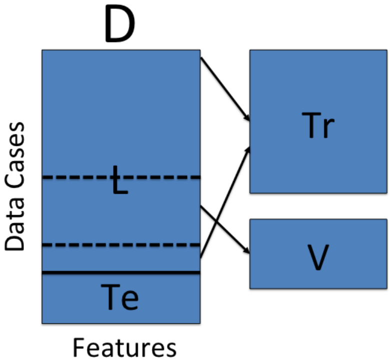
- Randomly partition the data cases into a learning set L and a test set Te (typically 50/50, 80/20, etc).
- For sample $s = 1, ..., S$:
  - Randomly partition L into Tr and V (again 50/50, 80/20, etc).
  - Learn M_{is} on Tr for each choice of hyperparameters H_i.
  - Compute Val_{is} of M_{is} on V.
- Select hyperparameters H_* minimizing $\frac{1}{S}\sum_{s=1}^{S} Val_{is}$ and re-train model on L using these hyperparameters, yielding final model M_*.

Recipe 3: Random Resampling Validation-Test

- Randomly partition the data cases into a learning set L and a test set Te (typically 50/50, 80/20, etc).
- For sample $s = 1, ..., S$:
  - Randomly partition L into Tr and V (again 50/50, 80/20, etc).
  - Learn M_{is} on Tr for each choice of hyperparameters H_i.
  - Compute Val_{is} of M_{is} on V.
- Select hyperparameters H_* minimizing $\frac{1}{S}\sum_{s=1}^{S} Val_{is}$ and 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-Sample Random Resampling and Test

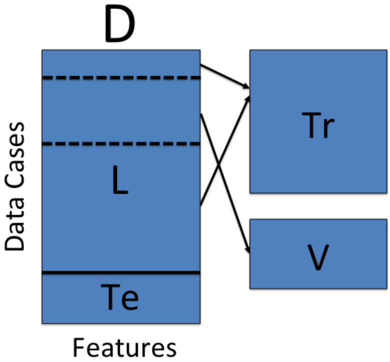
First Sample



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

Example: 3-Sample Random Resampling and Test

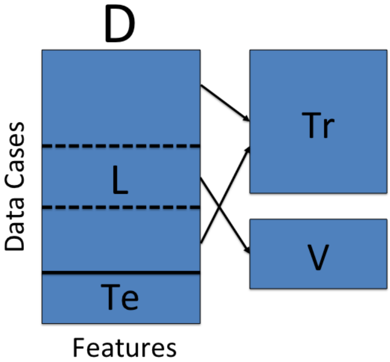
Second Sample



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

Example: 3-Sample Random Resampling and Test

Third Sample



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

Recipe 4: Crossvalidation-Crossvalidation

- Randomly partition data set D into a set of J blocks $C_1, ..., C_J$.

Recipe 4: Crossvalidation-Crossvalidation

- Randomly partition data set D into a set of J blocks $C_1, ..., C_J$.
- For $j = 1, ..., J$:

Recipe 4: 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 $L_j = D/C_j$

Recipe 4: Crossvalidation-Crossvalidation

- Randomly partition data set D into a set of J blocks $C_1, ..., C_J$.
- For $j = 1, ..., J$:
  - Let $Te_j = C_j$ and $L_j = D/C_j$
  - Partition L_j into a set of K blocks $B_1, ..., B_K$.

Recipe 4: Crossvalidation-Crossvalidation

- Randomly partition data set D into a set of J blocks $C_1, ..., C_J$.
- For $j = 1, ..., J$:
  - Let $Te_j = C_j$ and $L_j = D/C_j$
  - Partition L_j into a set of K blocks $B_1, ..., B_K$.
  - For $k = 1, ..., K$:

Recipe 4: Crossvalidation-Crossvalidation

- Randomly partition data set D into a set of J blocks $C_1, ..., C_J$.
- For $j = 1, ..., J$:
  - Let $Te_j = C_j$ and $L_j = D/C_j$
  - Partition L_j into a set of K blocks $B_1, ..., B_K$.
  - For $k = 1, ..., K$:
    - Let $V = B_k$ and $Tr = L_j/B_k$.

Recipe 4: Crossvalidation-Crossvalidation

- Randomly partition data set D into a set of J blocks $C_1, ..., C_J$.
- For $j = 1, ..., J$:
  - Let $Te_j = C_j$ and $L_j = D/C_j$
  - Partition L_j into a set of K blocks $B_1, ..., B_K$.
  - For $k = 1, ..., K$:
    - Let $V = B_k$ and $Tr = L_j/B_k$.
    - Learn M_{ik} on Tr for each choice of hyperparameters H_i.

Recipe 4: 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 $L_j = D/C_j$
  - Partition L_j into a set of K blocks $B_1, \ldots, B_K$.
  - For $k = 1, \ldots, K$:
    - Let $V = B_k$ and $Tr = L_j/B_k$.
    - Learn M_{ik} on Tr for each choice of hyperparameters H_i.
    - Compute error Val_{ik} of M_{ik} on V.

Recipe 4: Crossvalidation-Crossvalidation

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

Recipe 4: Crossvalidation-Crossvalidation

- Randomly partition data set D into a set of J blocks $C_1, ..., C_J$.
- For $j = 1, ..., J$:
  - Let $Te_j = C_j$ and $L_j = D/C_j$
  - Partition L_j into a set of K blocks $B_1, ..., B_K$.
  - For $k = 1, ..., K$:
    - Let $V = B_k$ and $Tr = L_j/B_k$.
    - Learn M_{ik} on Tr for each choice of hyperparameters H_i.
    - Compute error Val_{ik} of M_{ik} on V.
  - Select hyperparameters H_* 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.

Recipe 4: Crossvalidation-Crossvalidation

- Randomly partition data set D into a set of J blocks $C_1, ..., C_J$.
- For $j = 1, ..., J$:
  - Let $Te_j = C_j$ and $L_j = D/C_j$
  - Partition L_j into a set of K blocks $B_1, ..., B_K$.
  - For $k = 1, ..., K$:
    - Let $V = B_k$ and $Tr = L_j/B_k$.
    - Learn M_{ik} on Tr for each choice of hyperparameters H_i.
    - Compute error Val_{ik} of M_{ik} on V.
  - Select hyperparameters H_* 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 $\frac{1}{J}\sum_{j=1}^{J} Err_j$

Recipe 4: 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 $L_j = D/C_j$
  - Partition L_j into a set of K blocks $B_1, \ldots, B_K$.
  - For $k = 1, \ldots, K$:
    - Let $V = B_k$ and $Tr = L_j/B_k$.
    - Learn M_{ik} on Tr for each choice of hyperparameters H_i.
    - Compute error Val_{ik} of M_{ik} on V.
  - Select hyperparameters H_* 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 $\frac{1}{J}\sum_{j=1}^{J} Err_j$
- We can define a similar nested random resampling validation procedure.

Trade-Offs

- In cases where the data has a benchmark split into a training set and a test set, we can use Recipes 1-3 by preserving the given test set and splitting the given training set into train and validation sets as needed.

Trade-Offs

- In cases where the data has a benchmark split into a training set and a test set, we can use Recipes 1-3 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 bias. In these cases, Recipe 4 often provides a better estimate of generalization error, but has much higher computational cost.

Trade-Offs

- In cases where the data has a benchmark split into a training set and a test set, we can use Recipes 1-3 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 bias. In these cases, Recipe 4 often provides a better estimate of generalization error, but has much higher computational cost.
- Choosing larger K in cross validation will reduce bias. Choosing larger S in random re-sampling validation will reduce variance and bias. However, both increase 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.