Introduction to Machine Learning

Lecture 4 - 2DV516: Model Selection and Regularization

Amilcar Soares

amilcar.soares@lnu.se

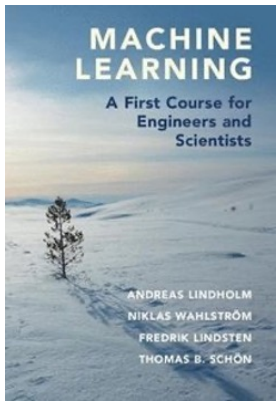
Slides and used datasets are available in Moodle

A big thanks to Dr. Jonas Nordqvist for providing most of the slides for this Lecture.

April 16, 2024

Agenda - Model Selection and Regularization

Reading Instructions



- Lindholm, A., Wahlstrom, N., Lindsten, F., & Schon, T. B. (2022). Machine learning: a first course for engineers and scientists. Cambridge University Press.
  - **Chapter 4:** Understanding, Evaluating, and Improving Performance. **Pages 63 to 90**.
- This lecture and slides focus on mathematics (high-level), concepts, and understanding of concepts.

Model Assessment

Q1. How to evaluate a model?

Q2. After the evaluation, is this a good/bad model?

Generalization error. How well does the method generalize, *i.e.*, what is the error on unseen data?

There are three different types of errors:

- **Training error:** Used to improve the parameters in training.
- **Validation error:** Used to improve hyperparameters or for model selection.
- **Test error:** Used to estimate the generalization error.

For sufficiently large n, the following is a suitable split of the data.

- Training: 60%
- Validation: 20%
- Testing: 20%

How to evaluate models in classification problems?

Confusion matrix

Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class.

A/P	c_1	c_2	...	c_k
c_1				
c_2				
$\vdots$				
c_k				

Table: Confusion matrix for a classification problem with k classes

Confusion Matrix for a Binary Problem

Context:

- In binary classification problems, we aim to categorize instances into one of two classes: positive (e.g., presence of a disease) or negative (e.g., absence of a disease).
- The confusion matrix is a fundamental tool for evaluating the performance of such models.

Example of a Confusion Matrix:

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Definitions:

- **True Positive (TP):** Instances correctly predicted as positive.
- **True Negative (TN):** Instances correctly predicted as negative.
- **False Positive (FP):** Instances incorrectly predicted as positive when they are actually negative.
- **False Negative (FN):** Instances incorrectly predicted as negative

Confusion Matrix for a Binary Problem

Example of a Confusion Matrix: 100 labeled as Disease and 100 labeled as No Disease

	Predicted Disease	**Predicted No Disease**
Actual Disease	63	37
Actual No Disease	28	72

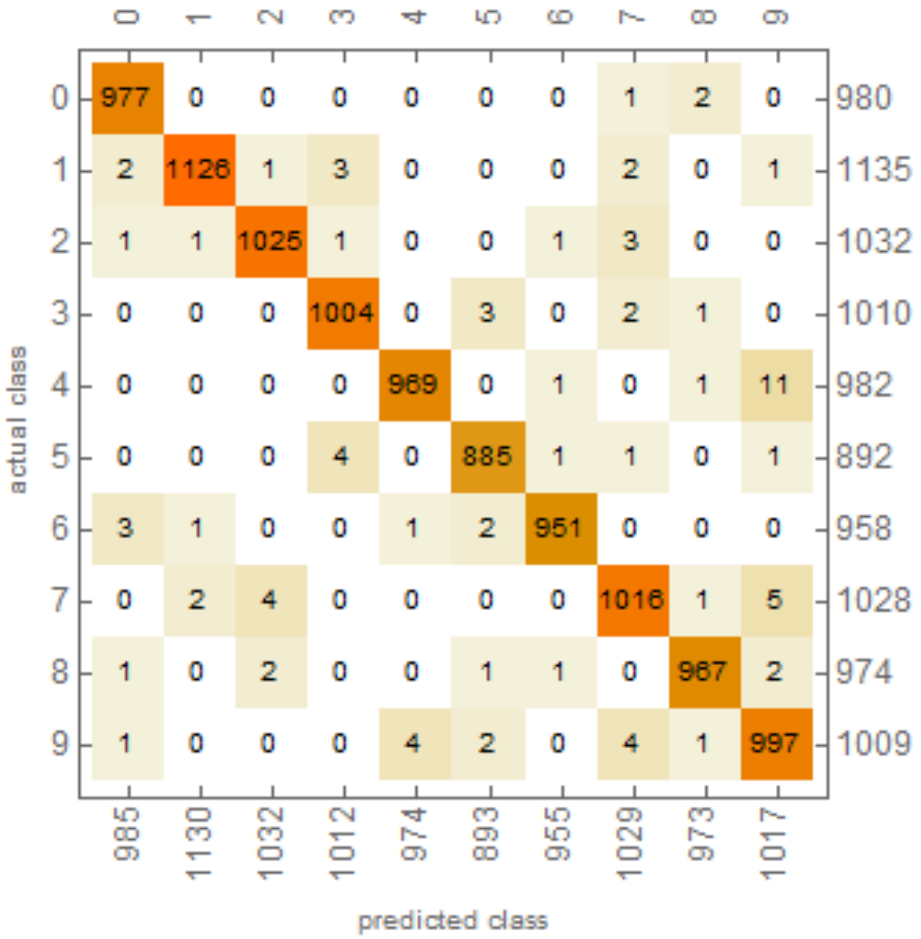
Pseudocode:

```
%pred is a vector of the predictions, y is the vector with
the (0/1) labels
TP = sum((pred+y)==2);
TN = sum((pred+y)==0);
FP = sum((pred-y)==1);
FN = sum((pred-y)==-1);
This code won't work directly, but it will with a few
modifications.
```

Accuracy: $ACC = \frac{TP+TN}{TP+TN+FP+FN}$

Example MNIST

For an MNIST classification model the confusion matrix is the following



Imbalanced data sets

Example

Let's assume that we've trained two ML-models, $\mathcal{M}_1$ and $\mathcal{M}_2$. They are trained to recognize whether a picture contains a cat or not.

The former model has 5% test error, and the latter has 6%. Which is the better model?

Assume that the test set contains 10,000 pictures, of which 500 are cats. Also, the confusion matrices of $\mathcal{M}_1$ and $\mathcal{M}_2$ are given by

a/p	cat	no cat
cat	0	500
no cat	0	9500

a/p	cat	no cat
cat	500	0
no cat	600	8900

Precision and recall

Precision is defined by

$$\text{precision} = \frac{TP}{TP + FP}.$$

Precision is also known as positive predictive value. Measures if all the positives were correct.

Recall is defined by

$$\text{recall} = \frac{TP}{TP + FN}.$$

Recall is also known as sensitivity. Measures if all the positives were found.

F-score is the harmonic mean of the two, *i.e.*

$$\text{F-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

Example

What is the precision and recall for $\mathcal{M}_1$ and $\mathcal{M}_2$?

a/p	cat	no cat
cat	0	500
no cat	0	9500

a/p	cat	no cat
cat	500	0
no cat	600	8900

Example

What is the precision and recall for $\mathcal{M}_1$ and $\mathcal{M}_2$?

a/p	cat	no cat
cat	0	500
no cat	0	9500

a/p	cat	no cat
cat	500	0
no cat	600	8900

Precision (P):

$$P_{M_1} = \frac{TP}{TP + FP} = \frac{0}{0 + 0} = 0(\text{undefined}) \; P_{M_2} = \frac{500}{500 + 600} = .4545$$

Recall (R):

$$R_{M_1} = \frac{TP}{TP + FN} = \frac{0}{0 + 500} = 0 \; R_{M_2} = \frac{500}{500 + 0} = 1$$

F-score (F1):

$$F1_{M_1} = 2 \times \frac{P_{M_1} \times R_{M_1}}{P_{M_1} + R_{M_1}} = \frac{2 \times 0 \times 0}{0 + 0} = \text{undefined} \; F1_{M_2} = \frac{2 \times 1 \times .4545}{1 + .4545} = 0.625$$

When precision or recall is zero (undefined) (resulting in an undefined F-score), the classifier's performance is extremely poor in one or both metrics. For instance, a precision of zero indicates that the classifier never correctly identified positive instances. In contrast, a recall of zero indicates that the classifier failed to identify any positive instances at all.

Error Analysis

Where did we go wrong? In machine learning models, errors can occur in various ways, which we can characterize as follows:

1. **False Negatives:** These occur when instances that are actually positive are incorrectly classified as negative by the model. For example:
   - In medical diagnosis, a disease patient is incorrectly identified as disease-free.
   - In credit card fraud detection, a fraudulent transaction is mistakenly classified as legitimate.
2. **False Positives:** These occur when instances that are actually negative are incorrectly classified as positive by the model. For example:
   - In email spam detection, a legitimate email is incorrectly flagged as spam.
   - In facial recognition systems, a person not in the database is mistakenly identified as someone else.

Understanding the errors the model makes is crucial for improving its performance.
Depending on the problem domain, the focus may be reducing the number of errors or specifically targeting false positives.

- In medical diagnosis, false negatives could have serious consequences, so reducing these errors is paramount.
- In spam email detection, false positives can inconvenience users, so minimizing these errors might be a priority.

Receiver operating characteristic (ROC)

If we consider a binary classification problem with 11 data points $\{x_i, y_i\}$, and we arrange our predictions according to some score (e.g., the probability threshold used in a logistic regression).

Data point	x_7	x_{10}	x_1	x_4	x_3	x_6	x_8	x_2	x_9	x_{11}	x_5
Score	.1	.2	.22	0.3	.4	.5	.6	.7	.8	.9	1
Label	0	0	1	0	1	0	1	1	1	1	1

Table: Data points sorted by some scoring function

We will obtain different values for precision and recall depending on where we set the decision boundary (DB).

DB	Precision	Recall	F-score
lower than x_7	$\frac{7}{11} \approx 0.636$	$\frac{7}{7+0} = 1$	0.778
between x_{10} and x_1	$\frac{7}{9} \approx 0.77$	$\frac{7}{7+0} = 1$	0.875
between x_4 and x_3	$\frac{6}{7} \approx 0.85$	$\frac{6}{7} \approx 0.85$	0.85
between x_6 and x_8	$\frac{5}{5+0} = 1$	$\frac{5}{7} \approx 0.71$	0.831
above x_5	1	0	0

Table: Precision, Recall, and F-score for different decision boundaries

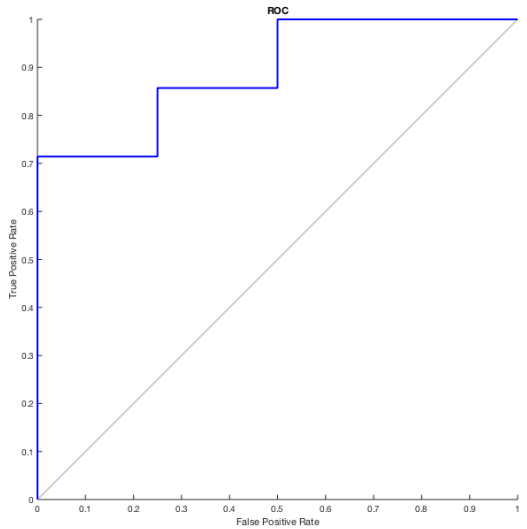
In conclusion, you may want to play with the decision boundaries to find a balance between precision and recall.

Receiver operating characteristic (ROC)

A variant of the metrics precision and recall are the *true positive rate* (TPR)[1] and *false positive rate* (FPR), where the latter is defined by

$$\text{FPR} = \frac{FP}{FP + TN}.$$

If we plot the FPR versus the TPR, we obtain the so-called ROC curve, and for the previous problem, it looks like the following graph



[1] which is in fact recall $= TP/(TP + FN)$

AUC-ROC

A way to compare classifiers is using the so-called *Area under the curve* (AUC) defined by

$$\text{AUC} = \int_0^1 \text{ROC}(t)dt,$$

where $\text{ROC}(t)$ is the ROC curve.

For the example of the previous slide, we have

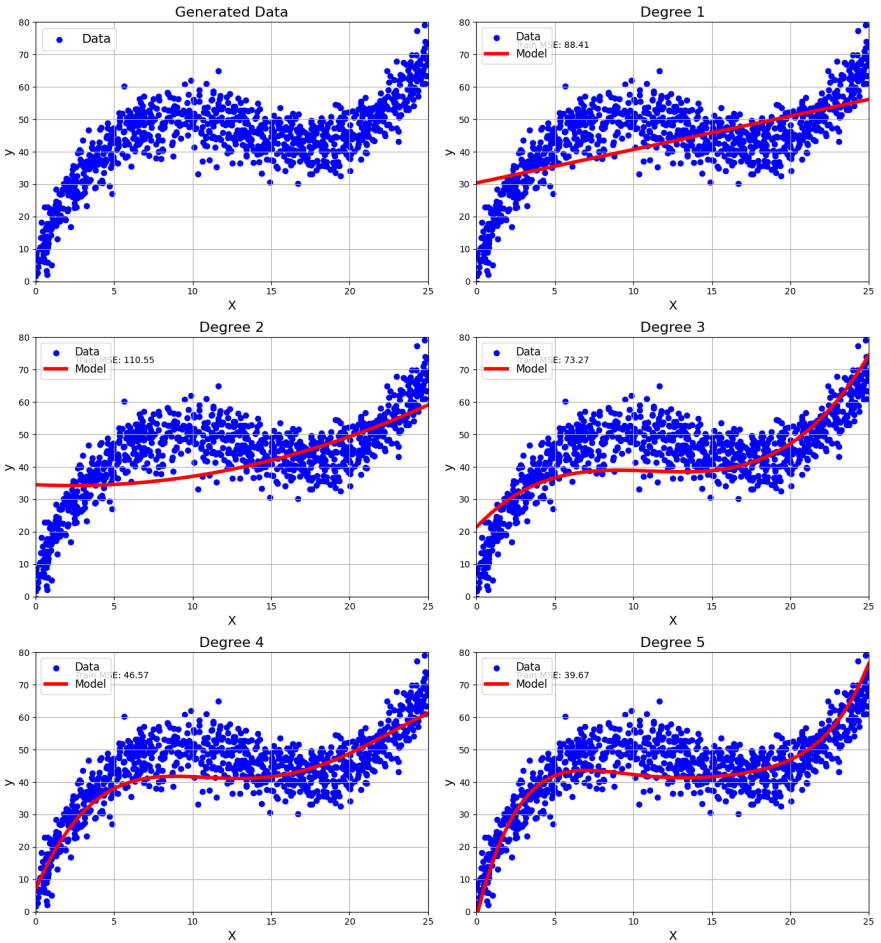
$$\text{AUC} = 0.8929.$$

Note that $\text{AUC} = 1$ is the optimal result, and $\text{AUC} = 0.5$ is just as bad as random guessing.

Bias-Variance trade-off

Even though we have the best possible model to our training data, there might be better models to minimize the generalization error.

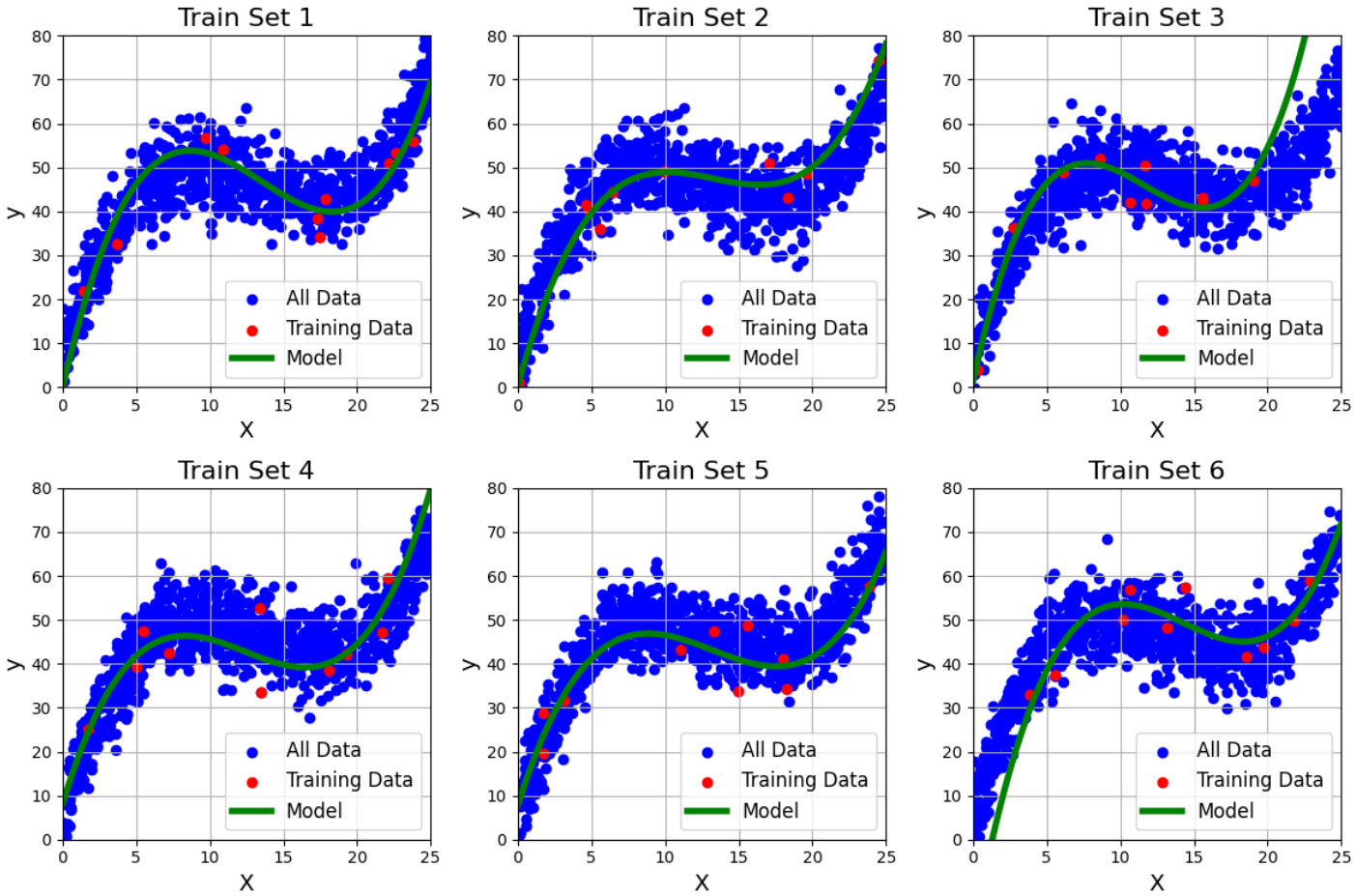
Bias



Bias-Variance trade-off

Even though we have the best possible model to our training data, there might be better models to minimize the generalization error.

Variance



Bias-Variance trade-off: An example

	Training error	**Validation error**	**Comment**
Model 1	10 %	11%	This model seems to suffer from *high bias*
Model 2	1%	6%	This model seems to suffer from *high variance*
Model 3	10%	20%	This model seems to suffer from both high variance and high bias.

Model selection

How to compare two different models, trying to estimate the same phenomena using the same data?

- Validation set
- Cross-validation
- Akaike information criterion (AIC)
- (There are several others, BIC, AICc, Adjusted R^2, C_p, etc.)

The size of the validation set compared to the training set is of course up for discussion and different for different scenarios. However, as a rule of thumb use 4:1 in favor for the training data.

Model selection vs. Model assessment

Note the difference between *model selection* and *model assessment*. The former estimates the performance of different models in order to choose the best. The latter is having chosen a final model, estimating its prediction performance, *i.e.* the generalization error.

Hence,

- the methods of model selection is not always applicable for model assessment, *e.g.* AIC
- and you can *never* reapply the same method used for model selection in order to do model assessment, *e.g.* validation set

Eating the cake and having it too

Cross-validation is a technique to estimate the performance of a prediction model.

There are several different flavours of this technique and we will discuss two versions namely:

- leave one out cross-validation
- k-fold cross-validation.

The essential idea is to estimate the validation error but still being able to use all training examples in the training of the model – eating the cake and having it too!

Cross-validation is particularly useful when the number of training examples n is small.

Leave one out cross-validation

The setup is as follows

- Dataset (X, y), where X is $n \times p$
- We want to fit a model $\mathcal{M}$ to the data
- We have no validation set, but still want to estimate the validation error

Leave one out cross-validation

The setup is as follows

- Dataset (X, y), where X is $n \times p$
- We want to fit a model $\mathcal{M}$ to the data
- We have no validation set, but still want to estimate the validation error

The algorithm for leave one out cross-validation is as follows.

Algorithm: Leave one out cross-validation

1. For $j = 1, \ldots, n$
   1.1 Train a model $\mathcal{M}_j$ using all training examples $X \setminus \{x_j\}$
   1.2 Classify the instance x_j using $\mathcal{M}_j$ and store its prediction in say `prd`
2. Obtain your cross-validation error by comparing the predictions in `prd` with the labels y. Use MSE for regression and error rate for classification.

This is a simple and efficient method to estimate the test error of f. However, as n grows, training that number of models might be cumbersome.

Leave one out cross-validation

The setup is as follows

- Dataset (X, y), where X is $n \times p$
- We want to fit a model $\mathcal{M}$ to the data
- We have no validation set, but still want to estimate the validation error

The algorithm for leave one out cross-validation is as follows.

Algorithm: Leave one out cross-validation

1. For $j = 1, \ldots, n$
   1.1 Train a model $\mathcal{M}_j$ using all training examples $X \setminus \{x_j\}$
   1.2 Classify the instance x_j using $\mathcal{M}_j$ and store its prediction in say `prd`
2. Obtain your cross-validation error by comparing the predictions in `prd` with the labels y. Use MSE for regression and error rate for classification.

This is a simple and efficient method to estimate the test error of f.
However, as n grows, training that number of models might be cumbersome.
Enter k-fold cross-validation!

k-fold cross-validation

This procedure is similar to that of leave one out cross-validation, but instead of partitioning the data into two parts of size 1 and $n-1$ for testing and training, we separate the data as two parts of size n/k and $n-n/k$ and repeat the procedure.

Algorithm: k-fold cross validation

1. Select parameter k

k-fold cross-validation

This procedure is similar to that of leave one out cross-validation, but instead of partitioning the data into two parts of size 1 and $n-1$ for testing and training, we separate the data as two parts of size n/k and $n-n/k$ and repeat the procedure.

Algorithm: k-fold cross validation

1. Select parameter k
2. Randomly permute the data, and partition it into k equally (or roughly equally) sized subsets. Denote these subsets by W_j for $j = 1, \ldots, k$.

k-fold cross-validation

This procedure is similar to that of leave one out cross-validation, but instead of partitioning the data into two parts of size 1 and $n-1$ for testing and training, we separate the data as two parts of size n/k and $n-n/k$ and repeat the procedure.

Algorithm: k-fold cross validation

1. Select parameter k
2. Randomly permute the data, and partition it into k equally (or roughly equally) sized subsets. Denote these subsets by W_j for $j = 1, \ldots, k$.
3. For $j = 1, \ldots, k$

k-fold cross-validation

This procedure is similar to that of leave one out cross-validation, but instead of partitioning the data into two parts of size 1 and $n-1$ for testing and training, we separate the data as two parts of size n/k and $n - n/k$ and repeat the procedure.

Algorithm: k-fold cross validation

1. Select parameter k
2. Randomly permute the data, and partition it into k equally (or roughly equally) sized subsets. Denote these subsets by W_j for $j = 1, \ldots, k$.
3. For $j = 1, \ldots, k$
   3.1 Train a model $\mathcal{M}_j$ using all training examples $X \setminus \{W_j\}$

k-fold cross-validation

This procedure is similar to that of leave one out cross-validation, but instead of partitioning the data into two parts of size 1 and $n-1$ for testing and training, we separate the data as two parts of size n/k and $n - n/k$ and repeat the procedure.

Algorithm: k-fold cross validation

1. Select parameter k
2. Randomly permute the data, and partition it into k equally (or roughly equally) sized subsets. Denote these subsets by W_j for $j = 1, \ldots, k$.
3. For $j = 1, \ldots, k$
   - 3.1 Train a model $\mathcal{M}_j$ using all training examples $X \setminus \{W_j\}$
   - 3.2 Classify the instances in W_j using $\mathcal{M}_j$ and store their prediction in the vector `prd`

k-fold cross-validation

This procedure is similar to that of leave one out cross-validation, but instead of partitioning the data into two parts of size 1 and $n-1$ for testing and training, we separate the data as two parts of size n/k and $n - n/k$ and repeat the procedure.

Algorithm: k-fold cross validation

1. Select parameter k
2. Randomly permute the data, and partition it into k equally (or roughly equally) sized subsets. Denote these subsets by W_j for $j = 1, \ldots, k$.
3. For $j = 1, \ldots, k$
   - 3.1 Train a model $\mathcal{M}_j$ using all training examples $X \setminus \{W_j\}$
   - 3.2 Classify the instances in W_j using $\mathcal{M}_j$ and store their prediction in the vector `prd`
4. Compute the cross-validation error by comparing the prediction in `prd` with the labels y. Use MSE for regression and error rate for classification.

k-fold cross-validation

This procedure is similar to that of leave one out cross-validation, but instead of partitioning the data into two parts of size 1 and $n-1$ for testing and training, we separate the data as two parts of size n/k and $n - n/k$ and repeat the procedure.

Algorithm: k-fold cross validation

1. Select parameter k
2. Randomly permute the data, and partition it into k equally (or roughly equally) sized subsets. Denote these subsets by W_j for $j = 1, \ldots, k$.
3. For $j = 1, \ldots, k$
   - 3.1 Train a model $\mathcal{M}_j$ using all training examples $X \setminus \{W_j\}$
   - 3.2 Classify the instances in W_j using $\mathcal{M}_j$ and store their prediction in the vector `prd`
4. Compute the cross-validation error by comparing the prediction in `prd` with the labels y. Use MSE for regression and error rate for classification.

k-fold cross-validation

This procedure is similar to that of leave one out cross-validation, but instead of partitioning the data into two parts of size 1 and $n-1$ for testing and training, we separate the data as two parts of size n/k and $n - n/k$ and repeat the procedure.

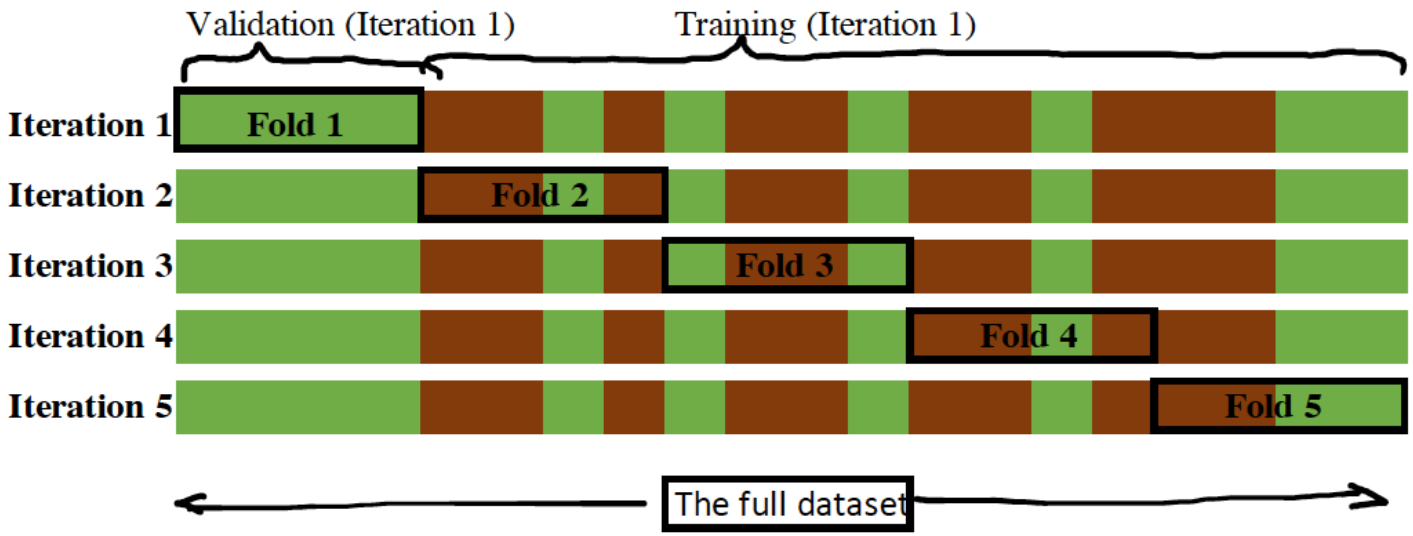
Algorithm: k-fold cross validation

1. Select parameter k
2. Randomly permute the data, and partition it into k equally (or roughly equally) sized subsets. Denote these subsets by W_j for $j = 1, \ldots, k$.
3. For $j = 1, \ldots, k$
   - 3.1 Train a model $\mathcal{M}_j$ using all training examples $X \setminus \{W_j\}$
   - 3.2 Classify the instances in W_j using $\mathcal{M}_j$ and store their prediction in the vector `prd`
4. Compute the cross-validation error by comparing the prediction in `prd` with the labels y. Use MSE for regression and error rate for classification.

Once going through this procedure, all training examples will have been used for testing precisely once and for training $k-1$ times.

There is a computational gain compared to the leave-one-out version.

Cross-validation



Cross validation in `sklearn`

There are different implementations of cross-validation in `sklearn`

1. If you simply want the scores of the model you can use `sklearn.model_selection.cross_val_score`

   `cross_val_score(mdl, X, y, cv=`**parameter** k`, scoring=`**your choice of metric**`)`

   Returns a k-dimensional vector of scores per fold
2. If you want the actual predictions you can instead use `sklearn.model_selection.cross_val_predict`

   `cross_val_predict(mdl, X, y, cv=`**parameter** k`)`

   Returns the prediction `y_pred`

Akaike information criterion (AIC)

AIC is another way to determine which model is better than another

$$\mathrm{AIC} := 2d - 2\ln(\hat{L}).$$

Akaike information criterion (AIC)

AIC is another way to determine which model is better than another

$$\text{AIC} := 2d - 2\ln(\hat{L}).$$

The following formulation of AIC is valid for linear models. For non-linear models, we would have to change the criterion, which is given by

$$\text{AIC} = n\ln(\text{MSE}) + 2d + nC,$$

where n is the number of training examples, d is the number of parameters (including intercept) and $C = \ln(2\pi) + 1$ is a constant. The important part to notice is that AIC increases as d increases.

Akaike information criterion (AIC)

AIC is another way to determine which model is better than another

$$\text{AIC} := 2d - 2\ln(\hat{L}).$$

The following formulation of AIC is valid for linear models. For non-linear models, we would have to change the criterion, which is given by

$$\text{AIC} = n\ln(\text{MSE}) + 2d + nC,$$

where n is the number of training examples, d is the number of parameters (including intercept) and $C = \ln(2\pi) + 1$ is a constant. The important part to notice is that AIC increases as d increases.

This measure of model quality is *relative*. This means that should only compare models which model the same phenomena. In general you should choose the model with the lowest AIC.

Other examples of model criterion are C_p and Bayesian information criterion (BIC).

AIC example

Example

A linear regression model f with 4 features was trained using 10 examples. After fitting f we obtain MSE $= 10$. Thus,

$$\text{AIC}_f = 10 \ln(10) + 10 + 10C \approx 61.4$$

Removing two features in f give us a new model g which after fitting turns out to have MSE $= 12$. Hence,

$$\text{AIC}_g = 10 \ln(12) + 6 + 10C \approx 59.2.$$

The latter, less complex model, should be chosen as the added complexity in f compared to g didn't decrease the MSE enough to compensate for the more complex model.

For small n there is an adjustment to AIC, called AICc which can be used instead.

What to Do If the Results Are Bad?

To address the problem of **high bias**, where the model is too simple to capture the underlying patterns in the data, consider the following techniques:

- **Adding Features:** Introducing additional relevant features to the model can increase its expressiveness, allowing it to capture more complex relationships in the data. This can help reduce underfitting and improve the model's ability to fit the training data.
- **Adding Transformed Features:** Transforming existing features or adding derived features, such as powers of features (e.g., in polynomial regression), can enable the model to capture nonlinear relationships between the input variables and the target variable.

To address the problem of **high variance**, where the model is overly sensitive to the training data and fails to generalize well to unseen data, consider the following techniques:

- **Getting More Training Examples:** Increasing the training dataset size helps the model capture a more representative sample of the underlying data distribution. This can reduce overfitting by providing more diverse examples for the model to learn from.
- **Trying Smaller Sets of Features:** By reducing the number of features used for training, the model's complexity is decreased, making it less prone to overfitting. This can lead to better generalization performance on unseen data.
- **Use Regularization:** This technique adds a penalty term to the model's loss function, encouraging simpler models or constraining the weights to avoid excessive complexity.

Occam's razor

All else being equal, the simplest explanation is the best one

Reducing the number of features by selecting a smaller subset may have two positive effects: increased **interpretability** and **robustness**.

Why select a 'simpler' model?

- Easier to pitch
- Easier to interpret (!)
- Get rid of unnecessary data gathering
- It might be less sensitive to variations in the data, *i.e.* better at predicting unseen data

Feature selection

Problem: Find a smaller subset of features by the principle of Occam's razor

Proposed solution: Best subset selection

The idea is applicable to any supervised ML-method and both for regression and classification, and setup is as follows:

- Dataset (X, y) where X is $n \times p$ matrix
- Note that there are 2^p possible combinations of the p features

The algorithm is as follows.

Feature selection

Problem: Find a smaller subset of features by the principle of Occam's razor

Proposed solution: Best subset selection

The idea is applicable to any supervised ML-method and both for regression and classification, and setup is as follows:

- Dataset (X, y) where X is $n \times p$ matrix
- Note that there are 2^p possible combinations of the p features

The algorithm is as follows.

Algorithm: Best subset selection

1. Train all 2^p possible models
2. Select the *best* model among $\mathcal{M}_0, \ldots, \mathcal{M}_{2^p-1}$ using some model selection criterion.

Feature selection

Problem: Find a smaller subset of features by the principle of Occam's razor

Proposed solution: Best subset selection

The idea is applicable to any supervised ML-method and both for regression and classification, and setup is as follows:

- Dataset (X, y) where X is $n \times p$ matrix
- Note that there are 2^p possible combinations of the p features

The algorithm is as follows.

Algorithm: Best subset selection

1. Train all 2^p possible models
2. Select the *best* model among $\mathcal{M}_0, \ldots, \mathcal{M}_{2^p-1}$ using some model selection criterion.

The obvious downside with best subset selection is that is scales poorly. For example, for $p = 30$ there are $1,073,741,824$ models to train and compare.

Forward selection algorithm

Updated proposed solution: Forward subset selection
Same setup as before

- Dataset (X, y), where X is an $n \times p$ matrix
- There are in total 2^p possible models that can be fit for any combination of subsets of our p features

We will instead use an algorithm called *forward subset selection* to reduce the number of models we have to train to $\binom{p+1}{2} = \frac{(p+1)p}{2}$. In the previous example, that would mean $\binom{31}{2} = 465$ models.

Forward selection algorithm

Updated proposed solution: Forward subset selection
Same setup as before

- Dataset (X, y), where X is an $n \times p$ matrix
- There are in total 2^p possible models that can be fit for any combination of subsets of our p features

We will instead use an algorithm called *forward subset selection* to reduce the number of models we have to train to $\binom{p+1}{2} = \frac{(p+1)p}{2}$. In the previous example, that would mean $\binom{31}{2} = 465$ models.

Algorithm: Forward subset selection

1. Start by a *null model* $\mathcal{M}_0$ without any features.
2. For $k = 0, \ldots, p-1$
   - 2.1 Train all $p - k$ models using $\mathcal{M}_k$ with one additional feature.
   - 2.2 Choose the *best* among these $p - k$ models and set it to be $\mathcal{M}_{k+1}$. Here the best means smallest cost.
3. Select the *best* model among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using some model selection criterion.

Why not use the training cost for model selection?

What to Do If the Results Are Bad?

To address the problem of **high bias**, where the model is too simple to capture the underlying patterns in the data, consider the following techniques:

- **Adding Features:** Introducing additional relevant features to the model can increase its expressiveness, allowing it to capture more complex relationships in the data. This can help reduce underfitting and improve the model's ability to fit the training data.
- **Adding Transformed Features:** Transforming existing features or adding derived features, such as powers of features (e.g., in polynomial regression), can enable the model to capture nonlinear relationships between the input variables and the target variable.

To address the problem of **high variance**, where the model is overly sensitive to the training data and fails to generalize well to unseen data, consider the following techniques:

- **Getting More Training Examples:** Increasing the training dataset size helps the model capture a more representative sample of the underlying data distribution. This can reduce overfitting by providing more diverse examples for the model to learn from.
- **Trying Smaller Sets of Features:** By reducing the number of features used for training, the model's complexity is decreased, making it less prone to overfitting. This can lead to better generalization performance on unseen data.
- **Use Regularization:** This technique adds a penalty term to the model's loss function, encouraging simpler models or constraining the weights to avoid excessive complexity.

Regularization

Regularization is a common strategy to prevent overfitting by decreasing its degrees of freedom.

Idea: Constrain the model from getting too complicated, for instance, by constraining the parameters not to grow too large

- The common strategy to 'solve' the ML problems we put forward for solving our problems in the last two lectures was to minimize some cost function $J(\beta)$, optimizing the parameters β for our problem.

Regularization

Regularization is a common strategy to prevent overfitting by decreasing its degrees of freedom.
Idea: Constrain the model from getting too complicated, for instance, by constraining the parameters not to grow too large

- The common strategy to 'solve' the ML problems we put forward for solving our problems in the last two lectures was to minimize some cost function $J(\beta)$, optimizing the parameters β for our problem.
- We've seen that we might overfit the data if our hypothesis f is too flexible.

Regularization

Regularization is a common strategy to prevent overfitting by decreasing its degrees of freedom.
Idea: Constrain the model from getting too complicated, for instance, by constraining the parameters not to grow too large

- The common strategy to 'solve' the ML problems we put forward for solving our problems in the last two lectures was to minimize some cost function $J(\beta)$, optimizing the parameters β for our problem.
- We've seen that we might overfit the data if our hypothesis f is too flexible.
- To avoid overfitting, we may impose a penalty that penalizes too much complexity of f or *shrinks* the coefficients towards zero.

Regularization

Regularization is a common strategy to prevent overfitting by decreasing its degrees of freedom.
Idea: Constrain the model from getting too complicated, for instance, by constraining the parameters not to grow too large

- The common strategy to 'solve' the ML problems we put forward for solving our problems in the last two lectures was to minimize some cost function $J(\beta)$, optimizing the parameters β for our problem.
- We've seen that we might overfit the data if our hypothesis f is too flexible.
- To avoid overfitting, we may impose a penalty that penalizes too much complexity of f or *shrinks* the coefficients towards zero.
- Our 'new' cost function is thus

$$J(\beta) + \lambda R(f).$$

Regularization

Regularization is a common strategy to prevent overfitting by decreasing its degrees of freedom.
Idea: Constrain the model from getting too complicated, for instance, by constraining the parameters not to grow too large

- The common strategy to 'solve' the ML problems we put forward for solving our problems in the last two lectures was to minimize some cost function $J(\beta)$, optimizing the parameters β for our problem.
- We've seen that we might overfit the data if our hypothesis f is too flexible.
- To avoid overfitting, we may impose a penalty that penalizes too much complexity of f or *shrinks* the coefficients towards zero.
- Our 'new' cost function is thus
$$J(\beta) + \lambda R(f).$$
- The parameter $\lambda \geq 0$ controls the importance of the second term $R(f)$, called the *regularization term*, which should decrease the model complexity.

Regularization

Regularization is a common strategy to prevent overfitting by decreasing its degrees of freedom.
Idea: Constrain the model from getting too complicated, for instance, by constraining the parameters not to grow too large

- The common strategy to 'solve' the ML problems we put forward for solving our problems in the last two lectures was to minimize some cost function $J(\beta)$, optimizing the parameters β for our problem.
- We've seen that we might overfit the data if our hypothesis f is too flexible.
- To avoid overfitting, we may impose a penalty that penalizes too much complexity of f or *shrinks* the coefficients towards zero.
- Our 'new' cost function is thus
$$J(\beta) + \lambda R(f).$$
- The parameter $\lambda \geq 0$ controls the importance of the second term $R(f)$, called the *regularization term*, which should decrease the model complexity.
- The two extremes are, of course, the degenerate case $\lambda = 0$, and the limit $\lambda \to \infty$.

LASSO regularization

LASSO regularization was first formulated for linear regression but easily extends to many other methods (such as logistic regression).

The penalty which we infer is given by

$$R(f) = \sum_{j=1}^{p} |\beta_j| = ||\beta||_1$$

- The formula is the sum of the absolute values of the weights (**excluding the bias term**) in the model.
- The bias term (i.e., intercept term) is typically excluded from regularization because it represents the constant offset in the model and is not directly associated with any input feature.
- Besides providing better predictions, this has the advantage that it also yields subset selection by putting $\beta_j = 0$ for some j for λ sufficiently large.

Ridge regularization (or L2 regularization)

More common is this alternative to LASSO regularization, which uses the (squared) Euclidean norm instead of the Taxi cab norm to penalize f, *i.e.*

$$R(f) = \sum_{j=1}^{p} \beta_j^2 = ||\beta||_2^2.$$

- This regularization will penalize all the coefficients and shrink them towards zero but not set any of them equal to zero. Note that the intercept should not be regularized.
- Fortunately, this function is differentiable; thus, we can use it in gradient descent.

Ridge regularization and gradient descent

In using ridge regularization, we need to find a new updated cost function $J^R(\beta)$ to add our regularization term, namely

$$J^R(\beta) = J(\beta) + \frac{\lambda}{2n}\sum_{j=1}^{p}\beta_j^2.$$

The gradient is then given by

$$\nabla(J^R) = \nabla(J) + \frac{\lambda}{n}(0, \beta_1, \ldots, \beta_p),$$

where $J(\beta)$ and ∇J are the original cost and gradient.

In code

The end terms can be written as

```
beta_reg = beta[1:]
J_reg = J + lambda/(2*n)*np.dot(beta_reg, beta_reg)
grad_reg = grad + lambda/n*[0; beta_reg]
```

Other regularization techniques

- **Elastic net**. Let $\lambda_1, \lambda_2 \in \mathbb{R}$. We can combine the properties of the two previously discussed regularization techniques by using *elastic nets*. An elastic net is a linear combination of lasso and ridge regularization

$$J^R(\beta) = J(\beta) + \lambda_1 R_1(f) + \lambda_2 R_2(f).$$

- **Early stopping**. The main idea is to stop the training procedure *before* convergence, and thus, before it overfits the data to much.
- There are also algorithm specific techniques which we'll see later for neural networks

LogisticRegression in `sklearn`

```
Xe = mapFeature(X1,X2,2,Ones=False) # No 1-column!
# C is the inverse of regularization strength; must be a
# positive float.  Smaller values specify stronger
regularization.
# Default is C = 1.0.  tol= 1e-6 ==> reduced tolerance
logreg = LogisticRegression(solver='lbfgs', C=1000.0,
tol=1e-6)
logreg.fit(Xe,y) # fit the model with data
y_pred=logreg.predict(Xe) # predict
errors = np.sum(y_pred!=y) # compare y with y_pred
print('Training errors:  ', errors)
```

Note that `lbfgs` is an optimization method (default option). Many others to choose from. For certain regularizations *e.g* lasso regularization all solvers are not compatible. Hence, study the documentation for further advice.

Regularization in `sklearn`

1. For linear regression
   - 1.1 Ridge regression is given by `sklearn.linear_model.Ridge`
   - 1.2 Lasso regression is given by `sklearn.linear_model.Lasso`
   - 1.3 In both these cases the parameter λ is called `alpha`.
2. For logistic regression
   - 2.1 `sklearn.linear_model.LogisticRegression`
   - 2.2 Regularization is built-in for LogisticRegression, and the instead of λ we use $\texttt{C} = 1/\lambda$
3. Other learning algorithms
   - 3.1 Many of `sklearn` methods have regularization built-in and various regularization parameters are passed when creating the models

Regularization in `sklearn`

1. For linear regression
   - 1.1 Ridge regression is given by `sklearn.linear_model.Ridge`
   - 1.2 Lasso regression is given by `sklearn.linear_model.Lasso`
   - 1.3 In both these cases the parameter λ is called `alpha`.
2. For logistic regression
   - 2.1 `sklearn.linear_model.LogisticRegression`
   - 2.2 Regularization is built-in for LogisticRegression, and the instead of λ we use $\texttt{C} = 1/\lambda$
3. Other learning algorithms
   - 3.1 Many of `sklearn` methods have regularization built-in and various regularization parameters are passed when creating the models

How to find reasonable values for λ to get a good model?

Hyperparameter tuning

Hyperparameters are parameters that are set before the learning process begins.

Some hyperparameters we've seen so far...

- k in kNN
- α, `num_iters` in gradient descent
- λ in regularization
- Choice of model (*e.g.* linear or quadratic)
- 'Choice of dataset'

Grid search is a typical approach for trying to find an optimal set of hyperparameters.

Grid search

Algorithm: Grid search

1. Select hyperparameters to tune: α and λ for gradient descent with regularization
2. Select which values of α and λ should be tested: $\alpha = (0.1, 0.3, 1, 3)$ and $\lambda = (0.1, 1, 10)$.
3. For $i = 1, \ldots, 4$
4. For $j = 1, \ldots, 3$
   4.1 Train a model $\mathcal{M}_{i,j}$ using $\alpha(i)$ and $\lambda(j)$
5. Select the *best* model $\mathcal{M}_{i,j}$ using a validation set or cross-validation.

That is, find the best possible pair in the grid below.

$$\begin{pmatrix} (0.1, 0.1) & (0.1, 1) & (0.1, 10) \\ (0.3, 0.1) & (0.3, 1) & (0.3, 10) \\ (1, 0.1) & (1, 1) & (1, 10) \\ (3, 0.1) & (3, 1) & (3, 10) \end{pmatrix}.$$

This naturally extends to any number of hyperparameters.

Assignment summary

The assignment deliverables from Lecture 4 are divided into two parts.

1. Part 1. You will implement the class ROCAnalysis and ForwardSelection, that will, respectively, implement several stats from the ROC analysis and the Forward selection algorithm (as discussed in class).
2. Part 2. You will use such implementations in a dataset to train models to diagnose people with heart disease.
3. Tips: Many python-like codes were discussed in class and can be changed to reach these implementations.
4. I have added some testing codes for all classes in this assignment. Get the testing zip file in the assignment folder to validate your implementations. A few typos were fixed in the Assignment 2 pdf file. The structure and deliverables did not change.