

Machine learning Part 2

Model selection & validation

Jérôme Dockès & Nikhil Bhagwat

QLS612 course 2022-07-12



Outline

Introduction: cross-validation

Model and hyperparameter selection

Dimensionality reduction

Conclusion: summary of pitfalls

Recap of part 1

Supervised learning

- Regression: least-squares linear regression
- Classification: logistic regression

Linear regression



Logistic regression



Recap of part 1

Supervised learning

- Regression: least-squares linear regression
- Classification: logistic regression

Regularization

- ℓ_2 a.k.a. ridge regularization

Recap of part 1

Supervised learning

- Regression: least-squares linear regression
- Classification: logistic regression

Regularization

- ℓ_2 a.k.a. ridge regularization

Model evaluation and selection

- Out-of-sample generalization; independent test set
- Performance metrics:
 - regression: mean squared error
 - classification: accuracy, ROC curve
- Cross-validation

Notation & vocabulary

Supervised learning framework

$$Y = f(X) + E \quad (1)$$

- $Y \in \mathbb{R}$: output (a.k.a. target, dependent variable) to predict

Notation & vocabulary

Supervised learning framework

$$Y = f(X) + E \quad (1)$$

- $Y \in \mathbb{R}$: output (a.k.a. target, dependent variable) to predict
- $X \in \mathbb{R}^p$: features (a.k.a. inputs, regressors, descriptors, independent variables)

Notation & vocabulary

Supervised learning framework

$$Y = f(X) + E \quad (1)$$

- $Y \in \mathbb{R}$: output (a.k.a. target, dependent variable) to predict
- $X \in \mathbb{R}^p$: features (a.k.a. inputs, regressors, descriptors, independent variables)
- $E \in \mathbb{R}$: unmodelled noise

Notation & vocabulary

Supervised learning framework

$$Y = f(X) + E \quad (1)$$

- $Y \in \mathbb{R}$: output (a.k.a. target, dependent variable) to predict
- $X \in \mathbb{R}^p$: features (a.k.a. inputs, regressors, descriptors, independent variables)
- $E \in \mathbb{R}$: unmodelled noise
- f : the function we try to approximate

Example (Linear regression)

$$Y = \beta_0 + \langle X, \beta \rangle + E \quad (2)$$

$$= \beta_0 + \sum_{j=1}^p X_j \beta_j + E \quad (3)$$

"learning" = choosing $\beta_0 \in \mathbb{R}$ and $\beta \in \mathbb{R}^p$

How to set parameters: Empirical Risk Minimization

- Choose a loss function L measuring how bad is our error.
- Example: squared error $L(Y, \hat{Y}) = (Y - \hat{Y})^2$, where \hat{Y} is the prediction
- We want to minimize the expected error (risk): $\mathbb{E}[L(Y, \hat{Y})]$

How to set parameters: Empirical Risk Minimization

We do not know the risk: estimate it from a sample.

Given n training examples $\mathbf{X} \in \mathbb{R}^{n \times p}$, $\mathbf{y} \in \mathbb{R}^n$, minimize the empirical risk: $\sum_{i=1}^n L(\mathbf{y}_i, \hat{\mathbf{y}}_i)$

For linear regression:

find $\hat{\beta}_0 \in \mathbb{R}$, $\hat{\beta} \in \mathbb{R}^p$ that minimize

$$\|\mathbf{y} - \hat{\mathbf{y}}\|_2^2 = \|\mathbf{y} - \hat{\beta}_0 - \mathbf{X} \hat{\beta}\|_2^2 \quad (4)$$

$$= \sum_{i=1}^n (\mathbf{y}_i - \hat{\beta}_0 - \sum_{j=1}^p \mathbf{x}_{ij} \hat{\beta}_j)^2 \quad (5)$$

"Fitting" the parameters to \mathbf{X}, \mathbf{y} .

Evaluating a model

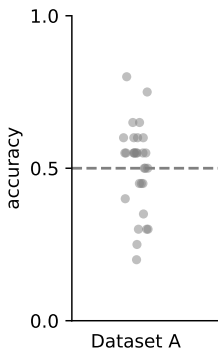
We always want to do 2 distinct things:

- Select a model (set the parameters).
- Evaluate its performance.

We can never do both on the same data!

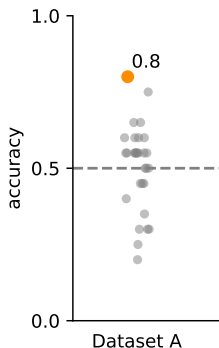
Training error is a biased estimator of the risk

- 30 different models (eg 30 possible values for $\hat{\beta}_0, \hat{\beta}$)
- All have a risk (expected accuracy) of 0.5
- Evaluate on a first dataset



Training error is a biased estimator of the risk

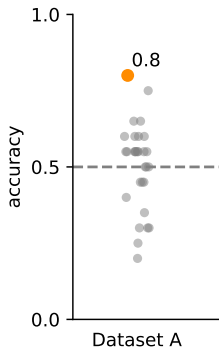
- 30 different models (eg 30 possible values for $\hat{\beta}_0, \hat{\beta}$)
- All have a risk (expected accuracy) of 0.5
- Evaluate on a first dataset **and select the best model**



Training error is a biased estimator of the risk

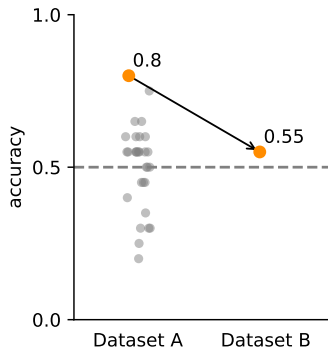
On a new dataset, the selected model will perform on average:

1. Better than on dataset A?
2. Worse than on dataset A?
3. The same?



Training error is a biased estimator of the risk

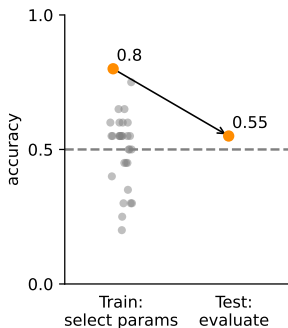
The selected model is more likely to perform worse on average than on dataset A.



Training error is a biased estimator of the risk

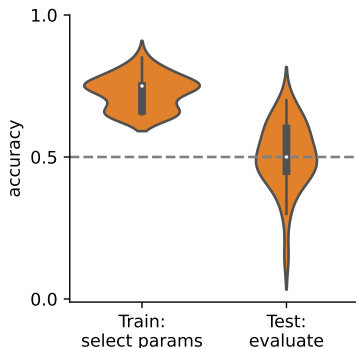
The selected model is more likely to perform worse on average than on the dataset used to select it:

To estimate its risk we need a new dataset.



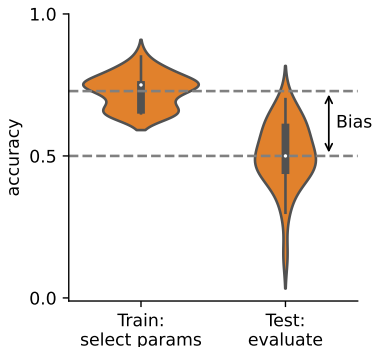
Training error is a biased estimator of the risk

Distribution of train and test errors across 30 repetitions:



Training error is a biased estimator of the risk

- The systematic difference is the bias.
- It is why we cannot use the training error to estimate model performance.



Estimating prediction performance

When you hear "best", "maximum", "select", ... think "bias"

Setting the parameters

- **Select** β that gives the **best** prediction on training data
- The prediction score for $\hat{\beta}$ is biased: compute a new score on unseen test data.

scikit-learn "estimator API": fit; predict

```
estimator = Ridge()  
estimator.fit(X_train, y_train)  
predictions = estimator.predict(X_test)
```

Scikit-learn user guide
[sklearn.linear_model.Ridge](#)

("API": "Application Programming Interface" – the specific way in which the library exposes its behaviour to user code: method names & signatures, etc.)

Evaluating performance with `sklearn.metrics`

```
estimator = Ridge()  
estimator.fit(X_train, y_train)  
predictions = estimator.predict(X_test)  
  
mse = metrics.mean_squared_error(y_test, predictions)
```

[sklearn.linear_model.Ridge](#)
[sklearn.metrics](#)
[User guide on model evaluation](#)

[ex_01_fit_predict_questions.py](#)

Some possible metrics for regression

R^2 score (coefficient of determination): `r2_score`

$$R^2(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}, \quad (6)$$

where $\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$











Mean Squared Error (MSE): `mean_squared_error`

$$\text{MSE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (7)$$

Mean Absolute Error (MAE): `mean_absolute_error`

$$\text{MAE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i| \quad (8)$$

Cross-validation

Fold 0	Train		Score 0
	Test		
Fold 1	Train		Score 1
	Test		
Fold 2	Train		Score 2
	Test		
Fold 3	Train		Score 3
	Test		
Fold 4	Train		Score 4
	Test		

User guide on cross-validation

`sklearn.model_selection.cross_validate`

`sklearn.model_selection.cross_val_score`

`ex_02_cross_validate_questions.py`

Outline

Introduction: cross-validation

Model and hyperparameter selection

Dimensionality reduction

Conclusion: summary of pitfalls

Need for regularization

Linear regression: projection on the column space of X

$$\hat{y} = X \hat{\beta} \quad (9)$$

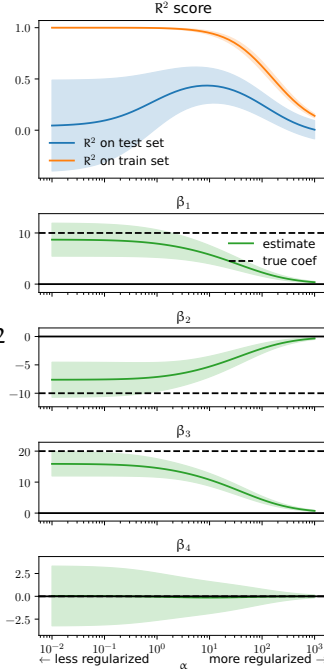


- Too many features: high variance & unstable solution
- Solutions: **regularization**, dimensionality reduction

Regularization

Example (Ridge regression)

$$\operatorname{argmin}_{\boldsymbol{\beta}, \beta_0} \|\mathbf{y} - \beta_0 - \mathbf{X} \boldsymbol{\beta}\|_2^2 + \alpha \|\boldsymbol{\beta}\|_2^2 \quad (10)$$



$$\text{Var}(\hat{\beta}_i) = \mathbb{E}(\hat{\beta}_i - \mathbb{E}(\hat{\beta}_i))^2$$

$$\text{Bias}(\hat{\beta}_i) = \mathbb{E}(\hat{\beta}_i) - \beta_i$$

Setting hyperparameters

How can we choose the ridge hyperparameter α ?

Try a few and pick the best one...

But measure its performance on separate data!

Nested cross-validation

When you hear "best", "maximum", "select", ... think "bias"

Nested cross-validation

When you hear "best", "maximum", "select", ... think "bias"

Setting the parameters

- **Select** β that gives the **best** prediction on training data
- The prediction score for $\hat{\beta}$ is biased: compute a new score on unseen test data.

Nested cross-validation

When you hear "best", "maximum", "select", ... think "bias"

Setting the parameters









- **Select** β that gives the **best** prediction on training data
- The prediction score for $\hat{\beta}$ is biased: compute a new score on unseen test data.

























Setting the hyperparameters

























- Repeat step 1 for a few values of α , fitting and testing several models
- **Select** the hyperparameter that obtains the **best** prediction on test data
- The prediction score of that model on *test* data is biased: evaluate it again on unseen data

































One split













Fold 0	Train	Fold 0	Train	For all α	
			Test	For all α	
		Fold 1	Train	For all α	
			Test	For all α	
		Fold 2	Train	For all α	
			Test	For all α	
		Refit		For best α	
	Test				

Fold 1	Train	Fold 0	Train	For all α		
			Test	For all α		 
		Fold 1	Train	For all α	 	 
			Test	For all α	 	
		Fold 2	Train	For all α	 	
			Test	For all α	 	
		Refit		For best α	 	
	Test				 	

Fold 2	Train	Fold 0	Train	For all α			
			Test	For all α			
		Fold 1	Train	For all α			
			Test	For all α			
		Fold 2	Train	For all α			
			Test	For all α			
		Refit		For best α			
	Test						

Fold 3	Train	Fold 0	Train	For all α				
			Test	For all α				
		Fold 1	Train	For all α				
			Test	For all α				
		Fold 2	Train	For all α				
			Test	For all α				
		Refit		For best α				
	Test							

Fold 4	Train	Fold 0	Train	For all α	
			Test	For all α	
		Fold 1	Train	For all α	  
			Test	For all α	
		Fold 2	Train	For all α	
			Test	For all α	
		Refit		For best α	
	Test				

Nested cross-validation with scikit-learn

- In general: [GridSearchCV](#) ([User Guide](#))

```
model = GridSearchCV(  
    Ridge(), {"alpha": [.1, 1., 10.]})  
scores = cross_val_score(model, X, y)
```

- Use [CV estimators](#) when possible: [RidgeCV](#), [LassoCV](#), ...

[ex_03_grid_search_regression_questions.py](#)

Implementing nested CV

`ex_04_nested_cross_validation_questions.py`

Outline

Introduction: cross-validation

Model and hyperparameter selection

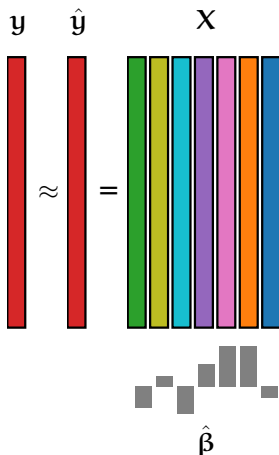
Dimensionality reduction

Conclusion: summary of pitfalls

Dimensionality reduction

Linear regression: projection on the column space of X

$$\hat{y} = X\hat{\beta} \quad (11)$$



- Too many features: high variance & unstable solution
- Solutions: regularization, **dimensionality reduction**

Dimensionality reduction

Until now



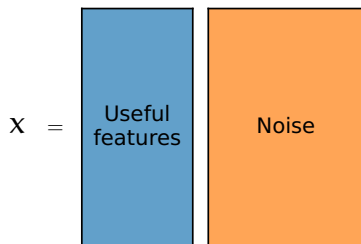
Add a step in the pipeline: simplifying the inputs



Simulated data for linear regression

- Generate $\mathbf{X} \in \mathbb{R}^{n \times 3}$, $\boldsymbol{\beta} \in \mathbb{R}^3$, $\mathbf{e} \in \mathbb{R}^n$ and $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{e} \in \mathbb{R}^n$
- Append columns containing random noise to \mathbf{X}
- Now $\mathbf{X} \in \mathbb{R}^{n \times p}$, with $p \geq 3$, but only the first 3 columns are linked with \mathbf{y}
- Split into training and testing tests and evaluate a linear regression model: what happens when p becomes large?

See [sklearn.datasets.make_regression](#) for generating data



Model complexity: overfitting

- Model complexity increases with dimension.
- Example: a linear model in dimension p can fit exactly (0 training error) any set of $p + 1$ points.
- Risk of overfitting: fitting exactly training data but failing on test data



Univariate feature selection

- a.k.a. feature screening, filtering ...
- Check features (columns of X) one by one for association with the output y
- Keep only a fixed number or percentage of the features

Simple (linear) association criteria

- for regression: correlation
- for classification: ANalysis Of VAriance

Read more in the scikit-learn user guide
[scikit-learn feature selection](#)

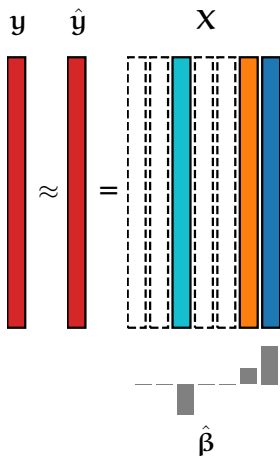
Original regression problem

$$\hat{y} = X\hat{\beta} \quad (12)$$



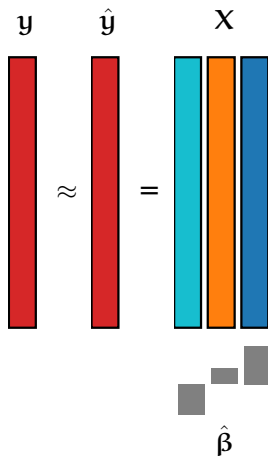
After univariate feature selection

$$\hat{\mathbf{y}} = \mathbf{X} \hat{\boldsymbol{\beta}} \quad (13)$$



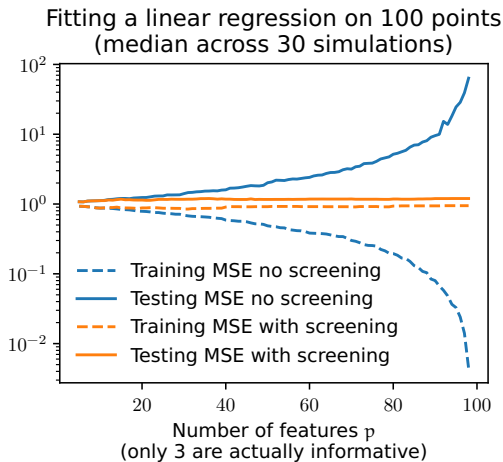
After univariate feature selection

$$\hat{\mathbf{y}} = \mathbf{X} \hat{\boldsymbol{\beta}} \quad (14)$$



Univariate feature selection

Keeping only the 10 best features (most correlated with y)



Dataset transformations

Typical pipeline



Example



scikit-learn "transformer API": fit; transform

```
transformer = SelectKBest()  
transformer.fit(X_train, y_train)  
transformed_train = transformer.transform(X_train)
```

can also be written:

```
transformer = SelectKBest()  
transformed_train = transformer.fit_transform(  
    X_train, y_train)
```

scikit-learn feature selection
scikit-learn Transformer API

`feature_selection.SelectKBest`

`fit:`

- compute ANOVA or correlation for each column of X
- Remember the indices of the k columns with highest scores

`transform:`

- Index input to keep only the k selected columns

`sklearn.feature_selection.SelectKBest`

Fit the transformer only on train data!

```
transformer = SelectKBest()  
transformed_train = transformer.fit_transform(  
    X_train, y_train)  
  
transformed_test = transformer.transform(X_test)
```

Pipelines

To chain transformations and an estimator, use

`sklearn.pipeline.Pipeline`

- can be used to properly cross-validate whole pipeline
- can be combined with `cross_validate`, `GridSearchCV`, ...
- easily created with `sklearn.pipeline.make_pipeline`

```
model = make_pipeline(SelectKBest(), Ridge())
```

`ex_05_feature_selection_questions.py`

Linear decomposition methods

Another approach to dimensionality reduction

Maybe OK to drop X_2 :



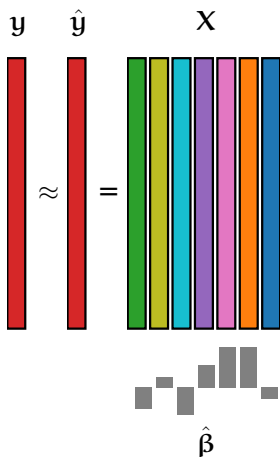
Data low-dimensional but no feature can be dropped:



Find a better referential in which to represent the data

Linear regression: projection on the column space of X

$$\hat{y} = X\hat{\beta} \quad (15)$$

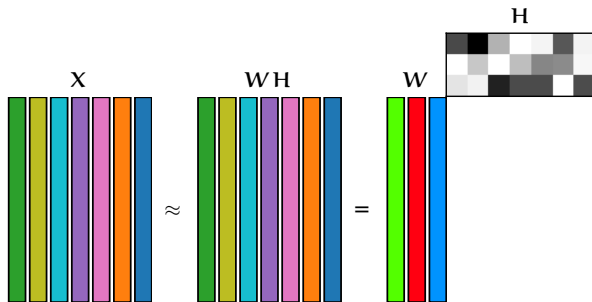


- Too many features: high variance & unstable solution
- Feature selection: drop some columns of X
- Other ways to build a family of k vectors on which to regress y ?

Linear decomposition: low-rank approximation of X

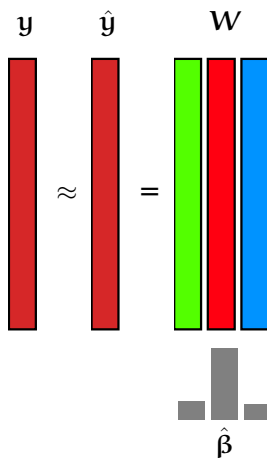
Minimize

$$\|X - WH\|_F^2 = \sum_{i,j} (X_{i,j} - (WH)_{i,j})^2 \quad (16)$$



Linear regression after dimensionality reduction

$$\hat{y} = W \hat{\beta} \quad (17)$$



Prediction for a new data point $\mathbf{x} \in \mathbb{R}^p$

- Find the combination of rows of \mathbf{H} that is closest to \mathbf{x} :
regress \mathbf{x} on \mathbf{H}^T
- Multiply by $\hat{\boldsymbol{\beta}}$

$$\mathbf{x} \in \mathbb{R}^p \rightarrow \text{projection} \rightarrow \mathbf{w} \in \mathbb{R}^k \rightarrow \langle \cdot, \hat{\boldsymbol{\beta}} \rangle \rightarrow \hat{y} \in \mathbb{R} \quad (18)$$

Principal Component Analysis

- Singular Value Decomposition of X :

$$X = U S V^T \quad (19)$$

with $X \in \mathbb{R}^{n \times p}$, $U \in \mathbb{R}^{n \times r}$, $S \in \mathbb{R}^{r \times r}$, $V \in \mathbb{R}^{r \times p}$

- $r = \min(n, p)$
- $S \succeq 0$ diagonal with decreasing values s_j along the diagonal
- $U^T U = I_r$
- $V^T V = I_r$

Truncating the SVD to keep only the first k components gives the best rank- k approximation of X



Singular Value Decomposition

$$\mathbf{X} = \mathbf{U} \mathbf{S} \mathbf{V}^T \quad (20)$$



Explained variance: 0.53

$$\mathbf{U}^T \mathbf{U} = \mathbf{I}_p \quad (21)$$

$$\mathbf{V}^T \mathbf{V} = \mathbf{I}_p \quad (22)$$

Singular Value Decomposition

$$\mathbf{X} = \mathbf{U} \mathbf{S} \mathbf{V}^T \quad (23)$$



Explained variance: 0.84

$$\mathbf{U}^T \mathbf{U} = \mathbf{I}_p \quad (24)$$

$$\mathbf{V}^T \mathbf{V} = \mathbf{I}_p \quad (25)$$

Singular Value Decomposition

$$\mathbf{X} = \mathbf{U} \mathbf{S} \mathbf{V}^T \quad (26)$$



Explained variance: 0.97

$$\mathbf{U}^T \mathbf{U} = \mathbf{I}_p \quad (27)$$

$$\mathbf{V}^T \mathbf{V} = \mathbf{I}_p \quad (28)$$

Other decomposition methods

Many other methods use the same objective (sum of squared reconstruction errors), but add penalties or constraints on the factors

- Dictionary Learning
- Non-negative Matrix Factorization
- K-means clustering
- ...

What about y ?

- PCA is an example of *unsupervised* learning: it does not use y
- Some other methods take it into account: e.g. Partial Least Squares

Ridge regression and PCA

- Both ridge regression and PC regression compute the coordinates of y in the basis given by the SVD of X
- Ridge shrinks the coordinate along U_j by a factor $s_j^2/(s_j^2 + \alpha)$
- PC regression sets the coordinates to 0 except for those corresponding to the k largest s_j : shrinks by a factor $1_{\{j \leq k\}}$



Outline

Introduction: cross-validation

Model and hyperparameter selection

Dimensionality reduction

Conclusion: summary of pitfalls

(Cross-)validation experiments are simulations

The validation experiments must simulate what will happen when deploying the trained model in production – when starting to use it in real life.

(Cross-)validation experiments are simulations

The validation experiments must simulate what will happen when deploying the trained model in production – when starting to use it in real life.

Example (Deploying a model to a hospital)

A model is trained on research dataset and then shipped and used on a hospital's patients. We cannot:

- Preprocess the patients' data together with the training data.
- Use the patients' data for feature selection.
- Try different models on the patients' data and pick the best.

If we do any of these things in our cross-validation it is not a realistic experiment.

Split choice example: time series

Don't ignore dependencies between samples: which is easier?



Use the appropriate [cross-validation iterator](#)

Remember that CV training sets overlap



So the scores are not independent! Their variance can be underestimated.

Some pitfalls with cross-validation

Overfitting the hyperparameters

- select hyperparameters with nested CV
`sklearn.model_selection.GridSearchCV`

Fitting part of the pipeline on the whole dataset

- use `sklearn.pipeline.Pipeline`

Ignoring dependencies between samples

- e.g. time series: use appropriate `cross-validation iterator`

Ignoring dependencies between CV scores

- Training sets overlap: cross-validation scores of different splits are not independent

Over-interpreting good CV scores