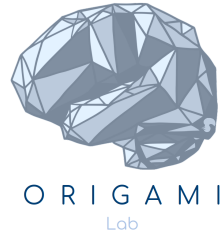
Machine learning Part 2

Model selection & validation

Jérôme Dockès & Nikhil Bhagwat

QLS612 course 2022-07-12



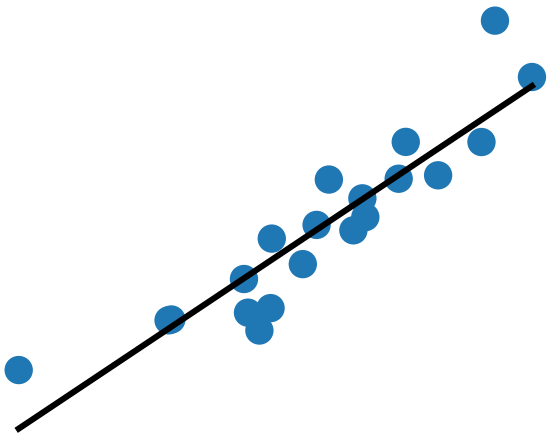
Outline

Recap of part 1

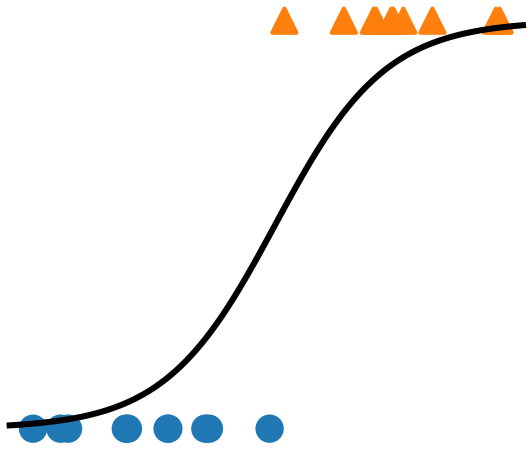
Supervised learning

- 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

$$Y = f(X) + E \tag{1}$$

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

Notation & vocabulary

Supervised learning framework

$$Y = f(X) + E \tag{1}$$

- $Y \in \mathbb{R}$: 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

$$Y = f(X) + E \tag{1}$$

- $Y \in \mathbb{R}$: 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

$$Y = f(X) + E \tag{1}$$

- $Y \in \mathbb{R}$: 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

$$Y = \beta_0 + \langle X, \beta \rangle + E \tag{2}$$

$$= \beta_0 + \sum_{j=1}^{p} X_j \, \beta_j + E \tag{3}$$

"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 $\mathrm{L}(Y, \hat{Y}) = (Y - \hat{Y})^2$, where $\hat{Y}$ is the prediction
- We want to minimize the expected error (risk): $\mathbb{E}[\mathrm{L}(Y, \hat{Y})]$

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

$$\|\mathbf{y} - \hat{\mathbf{y}}\|_2^2 = \|\mathbf{y} - \hat{\beta}_0 - \mathbf{X}\,\hat{\boldsymbol{\beta}}\|_2^2 \tag{4}$$

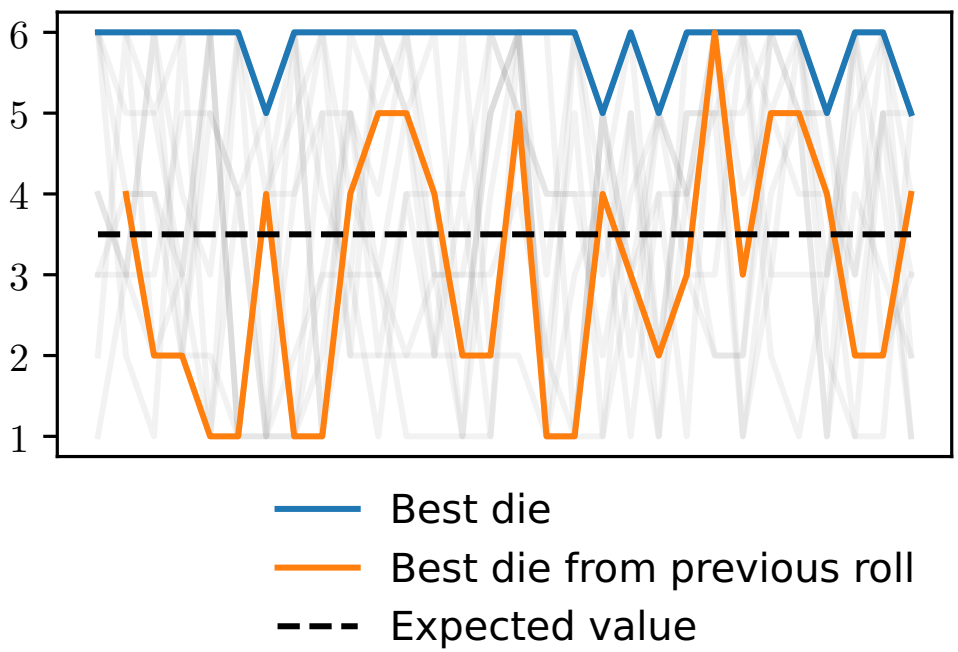
$$= \sum_{i=1}^{n} (\mathbf{y}_i - \hat{\beta}_0 - \sum_{j=1}^{p} \mathbf{X}_{ij}\,\hat{\boldsymbol{\beta}}_j)^2 \tag{5}$$

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

Need for fresh test data

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

- I have 4 dice and want to find one that rolls high numbers
- I roll them all once and select the die that gives the highest number
- The selected die rolled a 5. Is 5 a good estimate of that die's average result? What if I had 1,000 dice?
- I need to roll it again to get an unbiased estimate



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`

Evaluating performance with `sklearn.metrics`

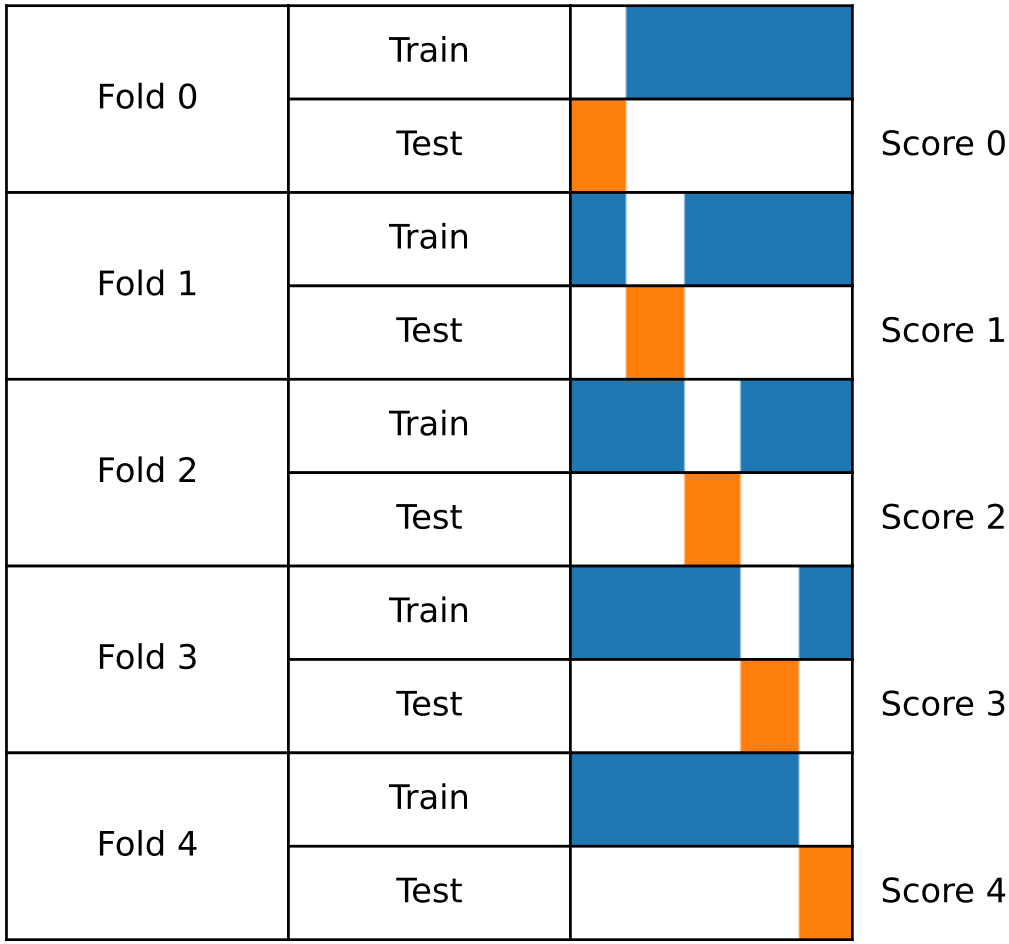
```
estimator = Ridge()
estimator.fit(X_train, y_train)
predictions = estimator.predict(X_test)

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`

Cross-validation



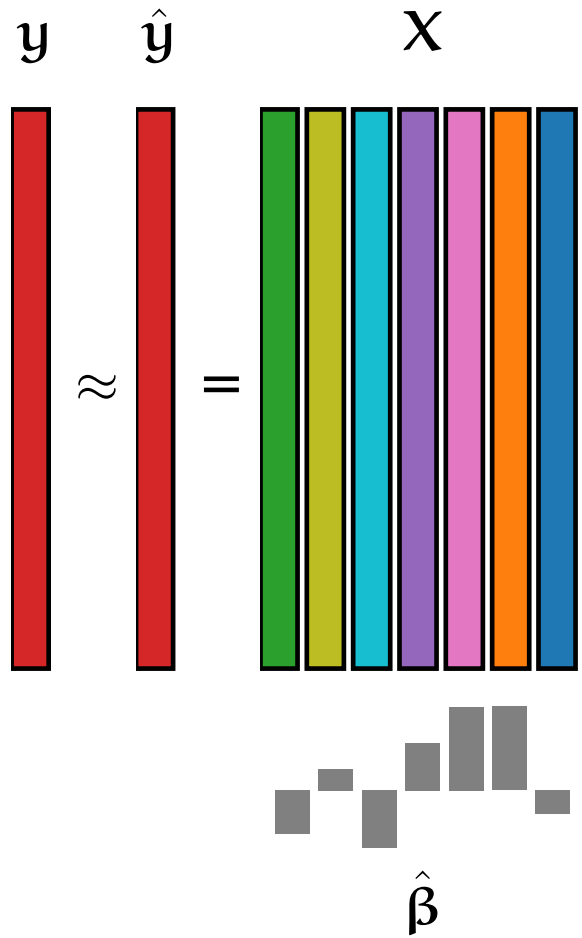
scikit-learn.org/stable/modules/cross_validation.html
sklearn.model_selection.cross_validate
ex_02_cross_validate_questions.py

Outline

Need for regularization

Linear regression: projection on the column space of $\mathbf{X}$

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}} \qquad (6)$$

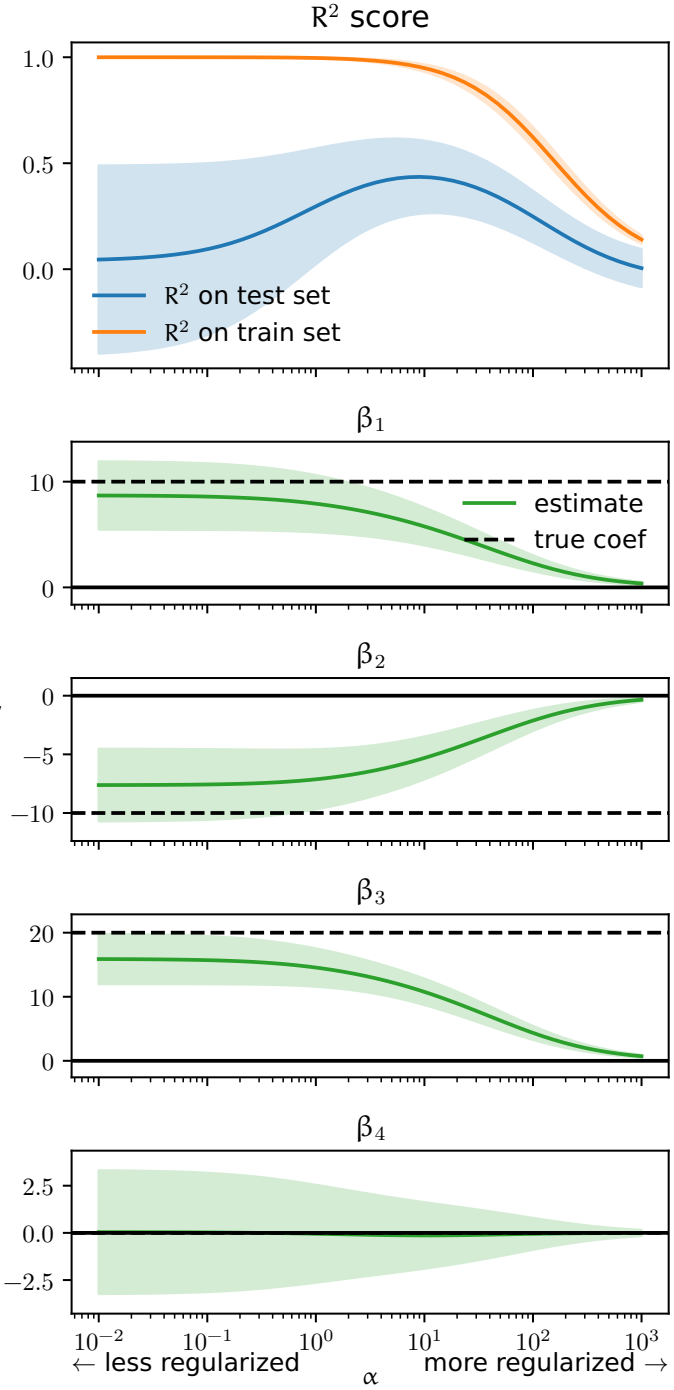


- Too many features: high variance & unstable solution
- Solutions: regularization, dimensionality reduction

Regularization

Example: ridge regression

$$\underset{\boldsymbol{\beta}, \beta_0}{\operatorname{argmin}} \|\mathbf{y} - \beta_0 - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \alpha \|\boldsymbol{\beta}\|_2^2 \tag{7}$$



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

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

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"

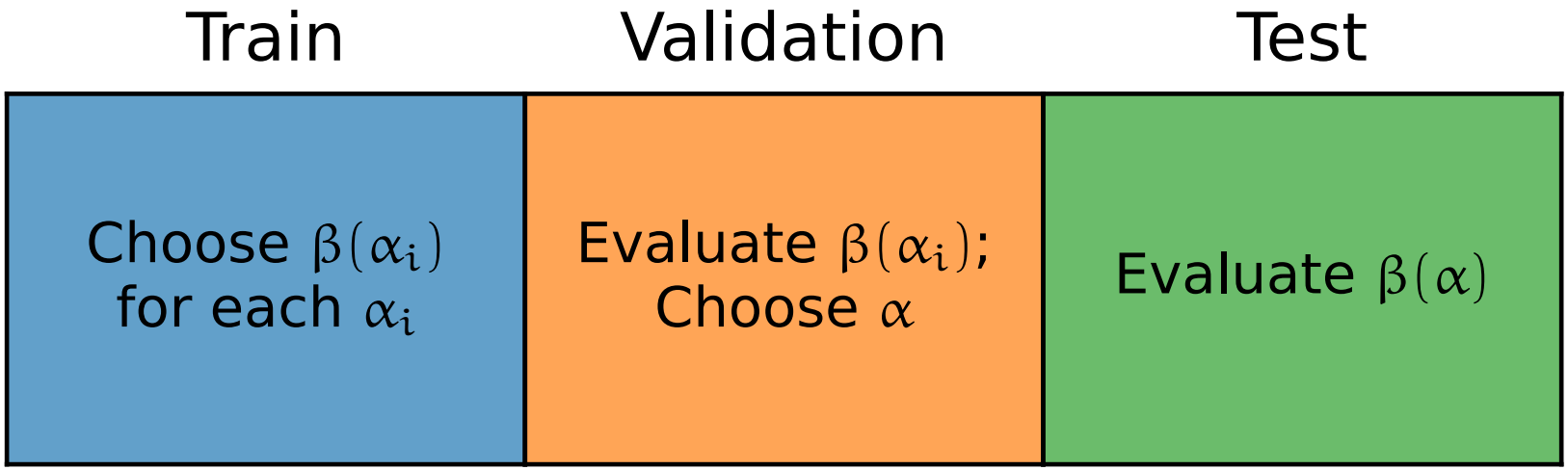
Setting the parameters

- **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.

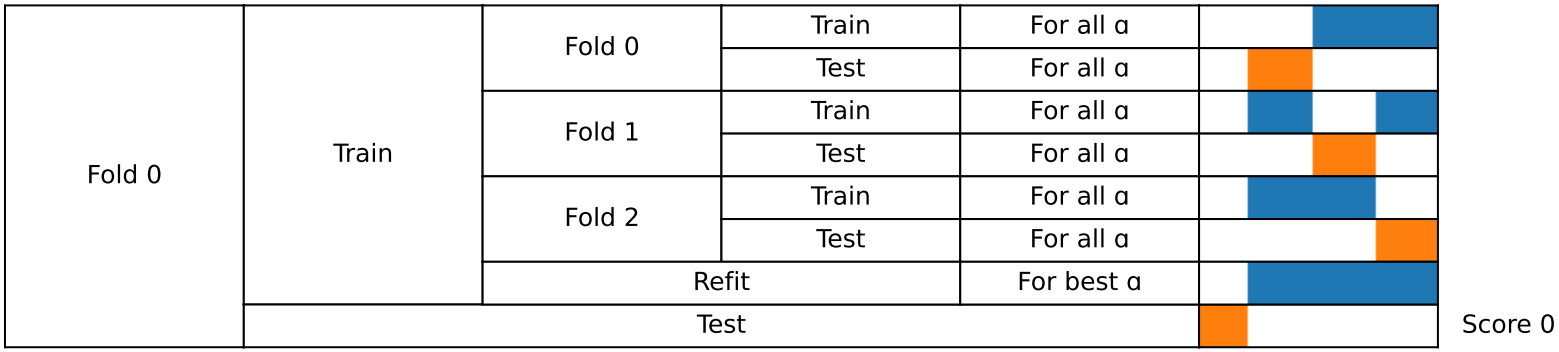
Setting the hyperparameters

- 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


Train
Validation
Test
Choose $\beta(\alpha_i)$ for each α_i
Evaluate $\beta(\alpha_i)$; Choose α
Evaluate $\beta(\alpha)$

Nested cross-validation



Nested cross-validation with scikit-learn

- In general: `GridSearchCV` (User Guide)

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

- 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

Dimensionality reduction

$$\hat{\mathbf{y}} = \hat{\beta}_0 + \hat{\boldsymbol{\beta}}_1 \mathbf{X}_{:,1} + \hat{\boldsymbol{\beta}}_2 \mathbf{X}_{:,1} + \cdots + \hat{\boldsymbol{\beta}}_p \mathbf{X}_{:,p} \tag{8}$$

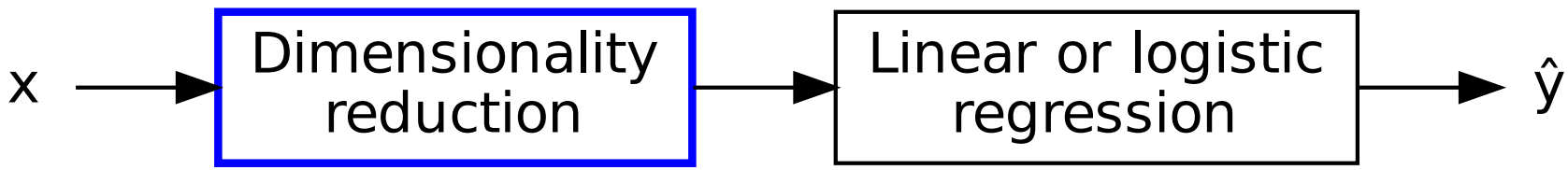
Problems when the number of features p becomes large

- Bigger errors on test data (larger variance of predictions)
- Numerical stability issues
- Computational cost and memory usage

Dimensionality reduction

Until now

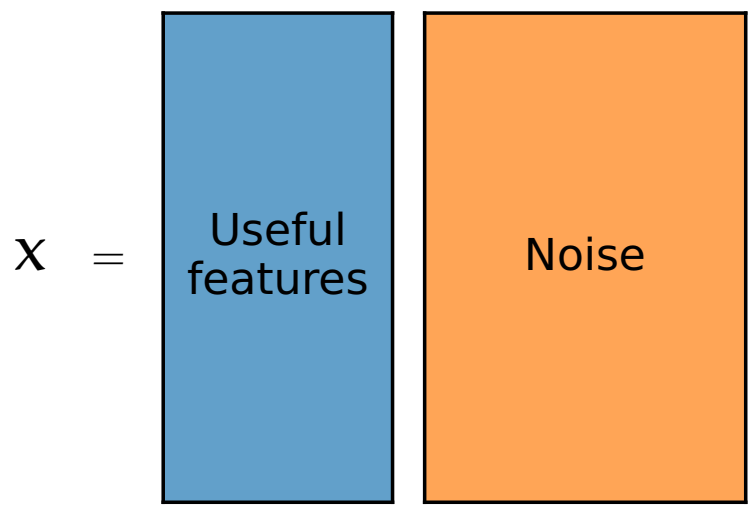
Add a step in the pipeline: simplifying the inputs



Simulated data for linear regression

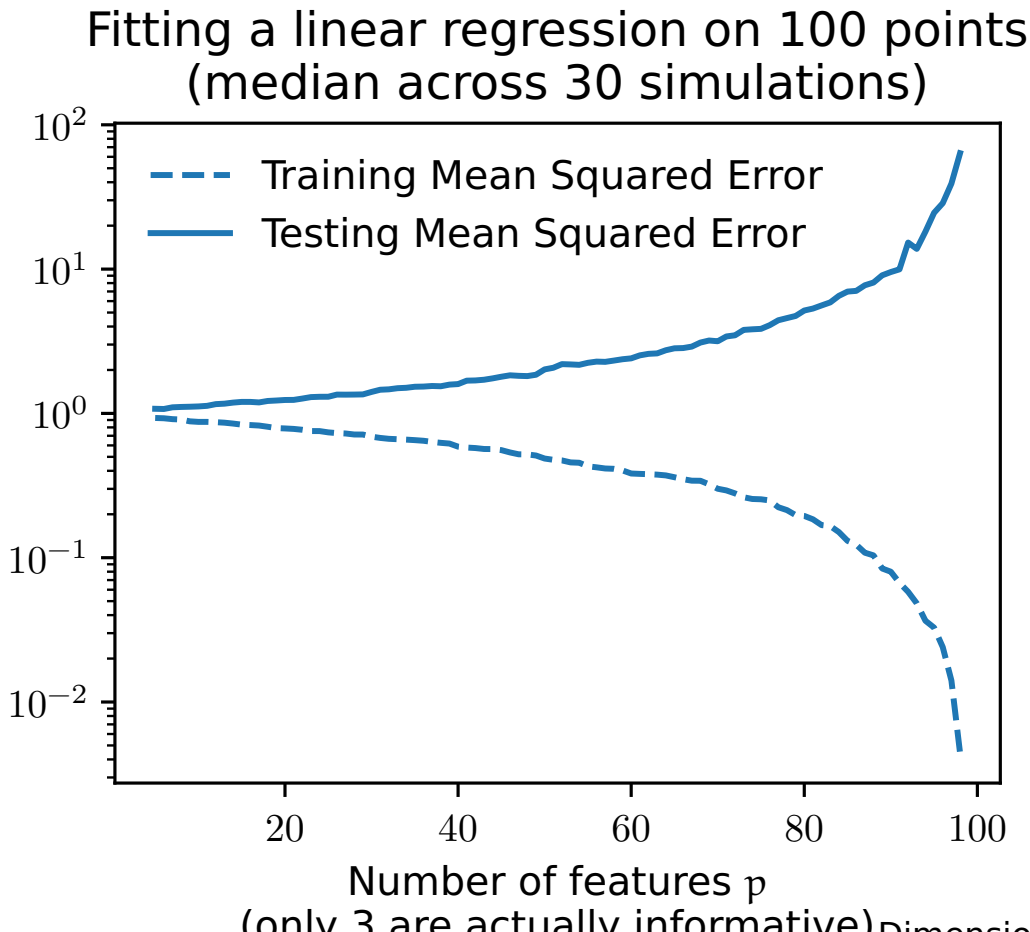
- Generate $\mathbf{X} \in \mathbb{R}^{n \times 3}$, $\boldsymbol{\beta} \in \mathbb{R}^3$, $\boldsymbol{e} \in \mathbb{R}^n$ and $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{e} \in \mathrm{R}^n$
- Append columns containing random noise to $\mathbf{X}$
- Now $\mathbf{X} \in \mathbb{R}^{n \times p}$, with $p \geqslant 3$, 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 $\mathbf{X}$) one by one for association with the output $\mathbf{y}$
- Keep only a fixed number or percentage of the features

Simple (linear) association criteria

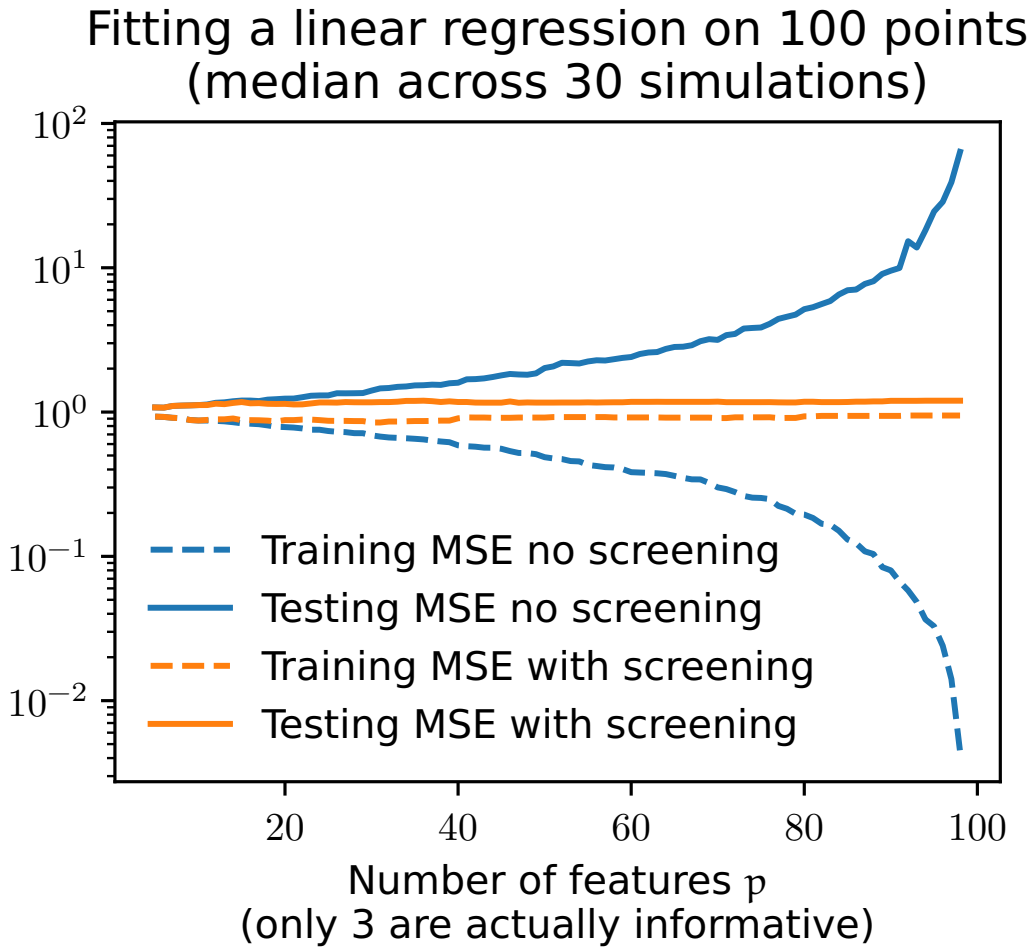
- for regression: correlation
- for classification: ANalysis Of VAriance

Read more in the scikit-learn user guide

scikit-learn feature selection

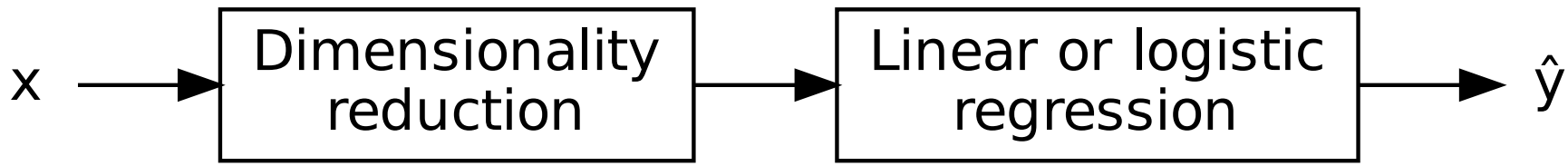
Univariate feature selection

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

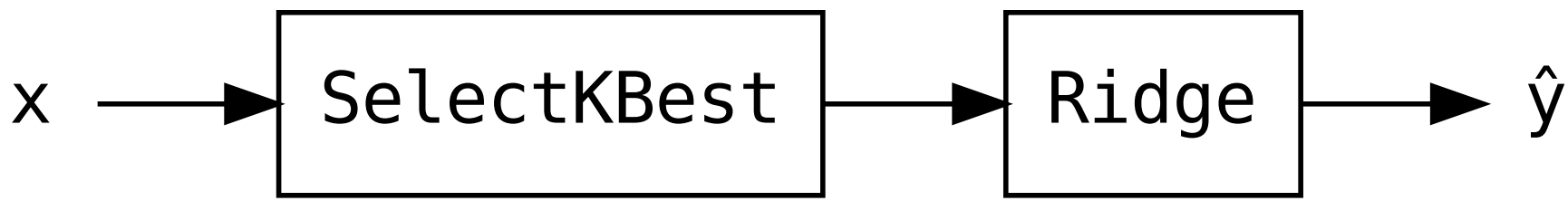


Dataset transformations

Typical pipeline



Example



scikit-learn "transformer API": `fit; transform`

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

can also be written:

```
transformer = SelectKBest()
transformed_train = transformer.fit_transform(X_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)

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`

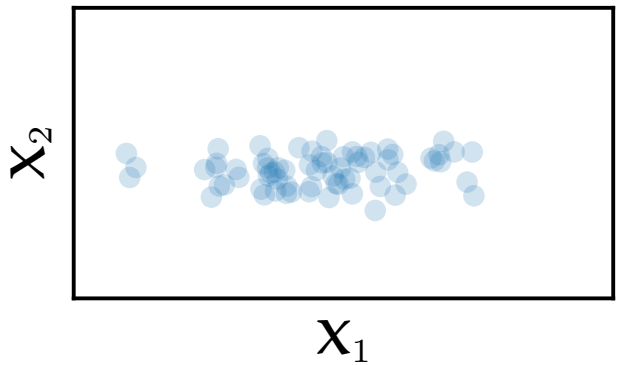
```
model = make_pipeline(
    SelectKBest(), LogisticRegression())
```

`ex_04_feature_selection_questions.py`

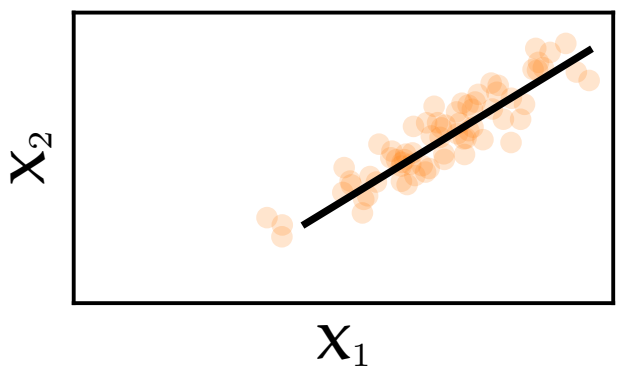
Linear decomposition methods

Another approach to dimensionality reduction

Maybe OK to drop X_2:



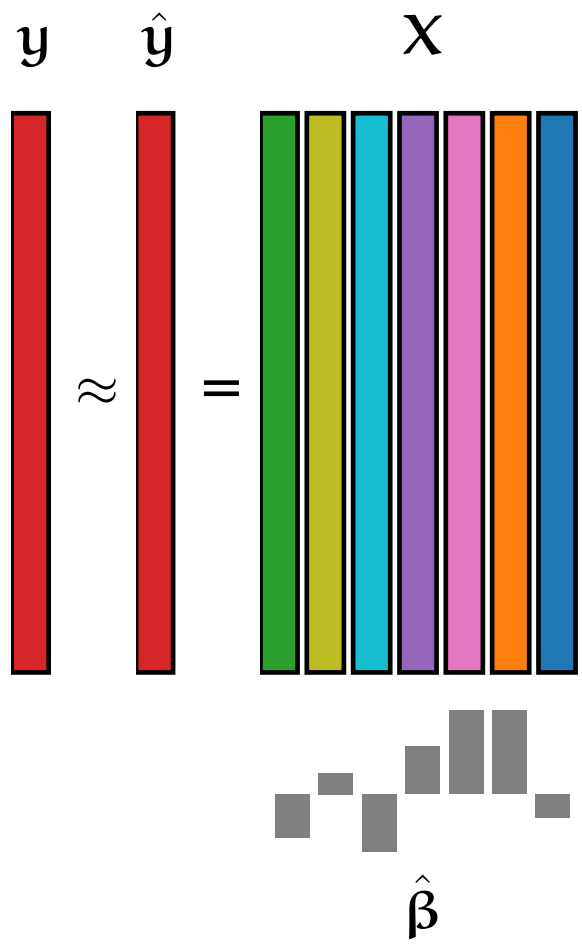
Data low-dimensional but no feature can be dropped:



Find a better referential in which to represent the data

Linear regression: projection on the column space of X

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

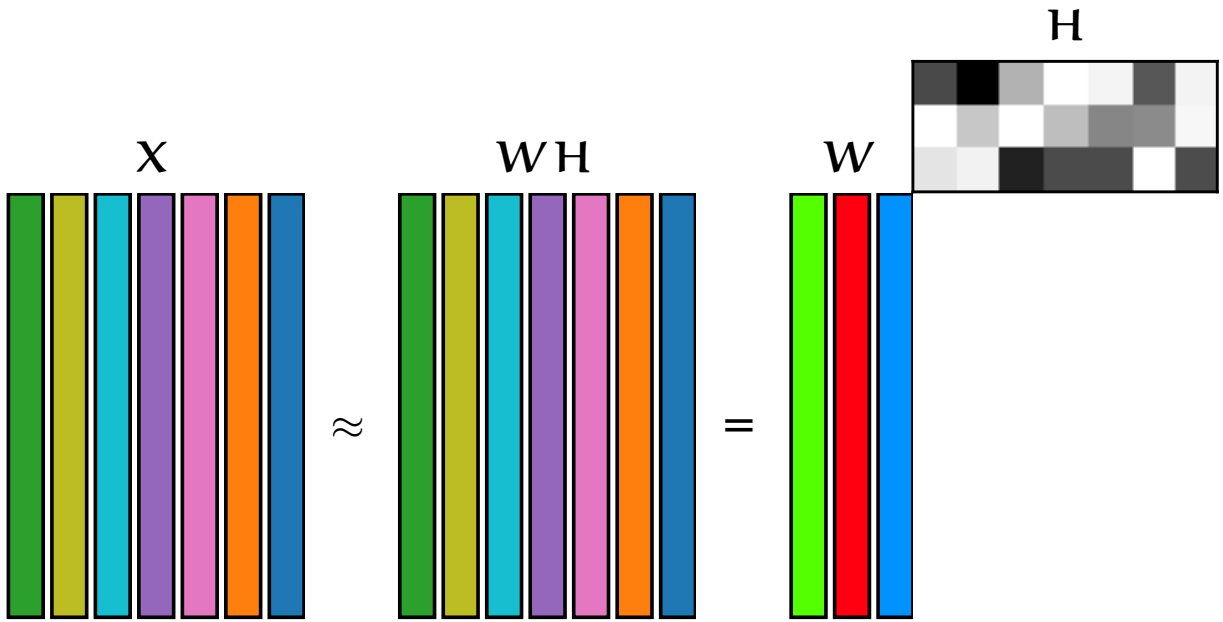


- Too many features: high variance & unstable solution
- Feature selection: drop some columns of X
- Other ways to build a family of k vectors on which to regress y?

Linear decomposition: low-rank approximation of X

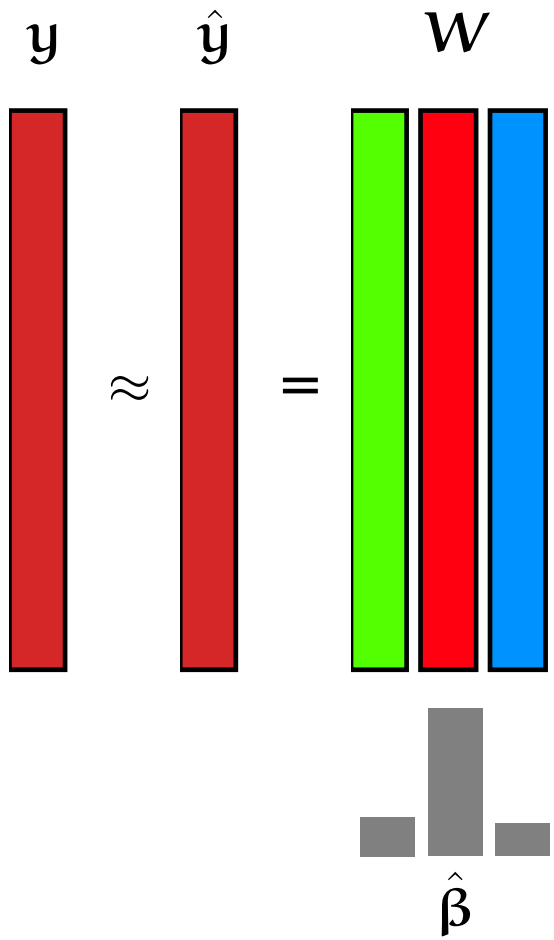
Minimize

$$\|X - WH\|_F^2 = \sum_{i,j} (X_{i,j} - (WH)_{i,j})^2 \tag{10}$$



Linear regression after dimensionality reduction

$$\hat{\mathbf{y}} = \boldsymbol{W}\hat{\boldsymbol{\beta}} \tag{11}$$



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

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

$$\boldsymbol{x} \in \mathbb{R}^p \to \text{projection} \to \boldsymbol{w} \in \mathbb{R}^k \to \langle \cdot, \hat{\boldsymbol{\beta}} \rangle \to \hat{y} \in \mathbb{R} \tag{12}$$

Principal Component Analysis

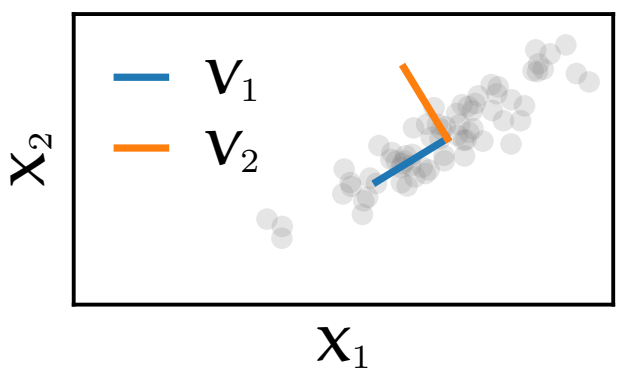
- Singular Value Decomposition of $\mathbf{X}$:

$$\mathbf{X} = \mathbf{U}\,\mathbf{S}\,\mathbf{V}^{\mathsf{T}} \tag{13}$$

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

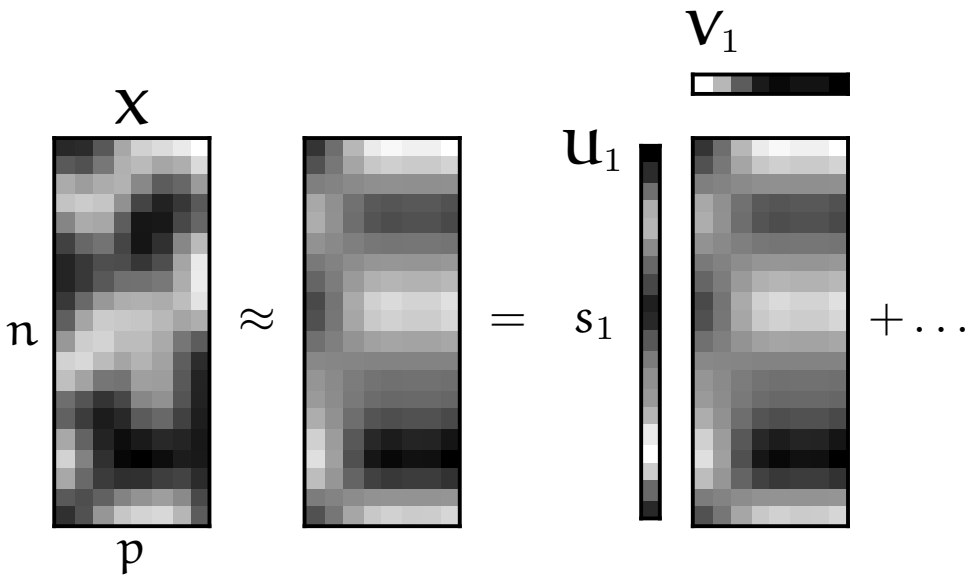
- $r = \min(n, p)$
- $\mathbf{S} \succeq 0$ diagonal with decreasing values s_j along the diagonal
- $\mathbf{U}^{\mathsf{T}}\mathbf{U} = \mathrm{I}_r$
- $\mathbf{V}^{\mathsf{T}}\mathbf{V} = \mathrm{I}_r$

Truncating the SVD to keep only the first k components gives the best rank-k approximation of $\mathbf{X}$



Singular Value Decomposition

$$X = U S V^T \tag{14}$$



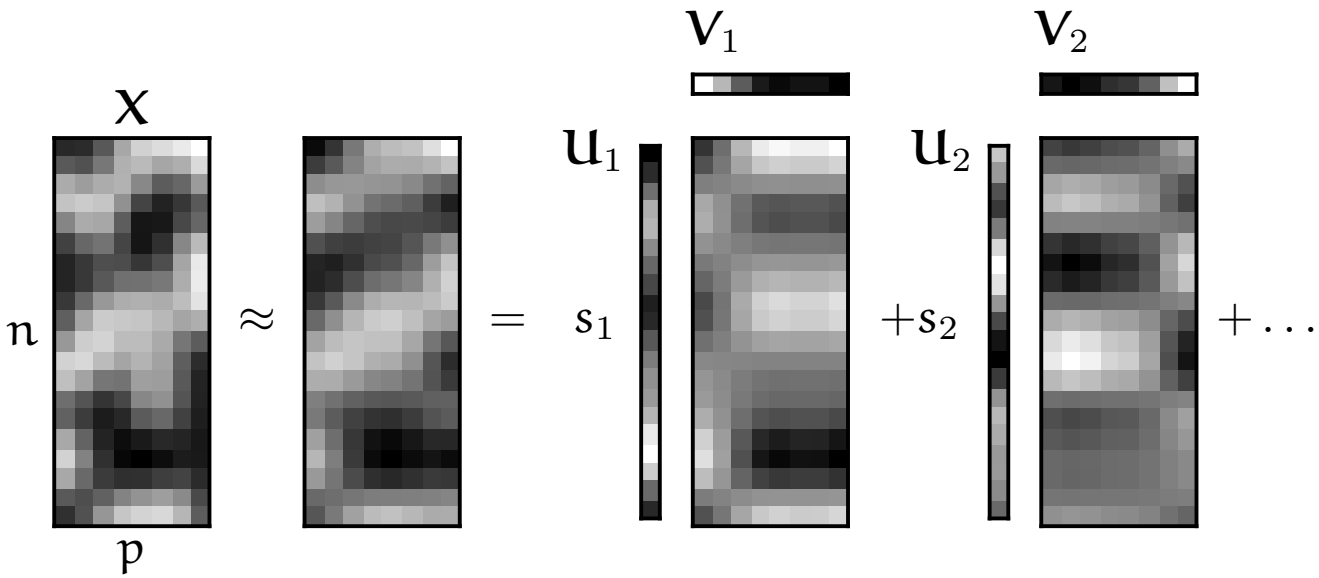
Explained variance: 0.53

$$U^T U = I_p \tag{15}$$

$$V^T V = I_p \tag{16}$$

Singular Value Decomposition

$$X = U S V^T \tag{17}$$



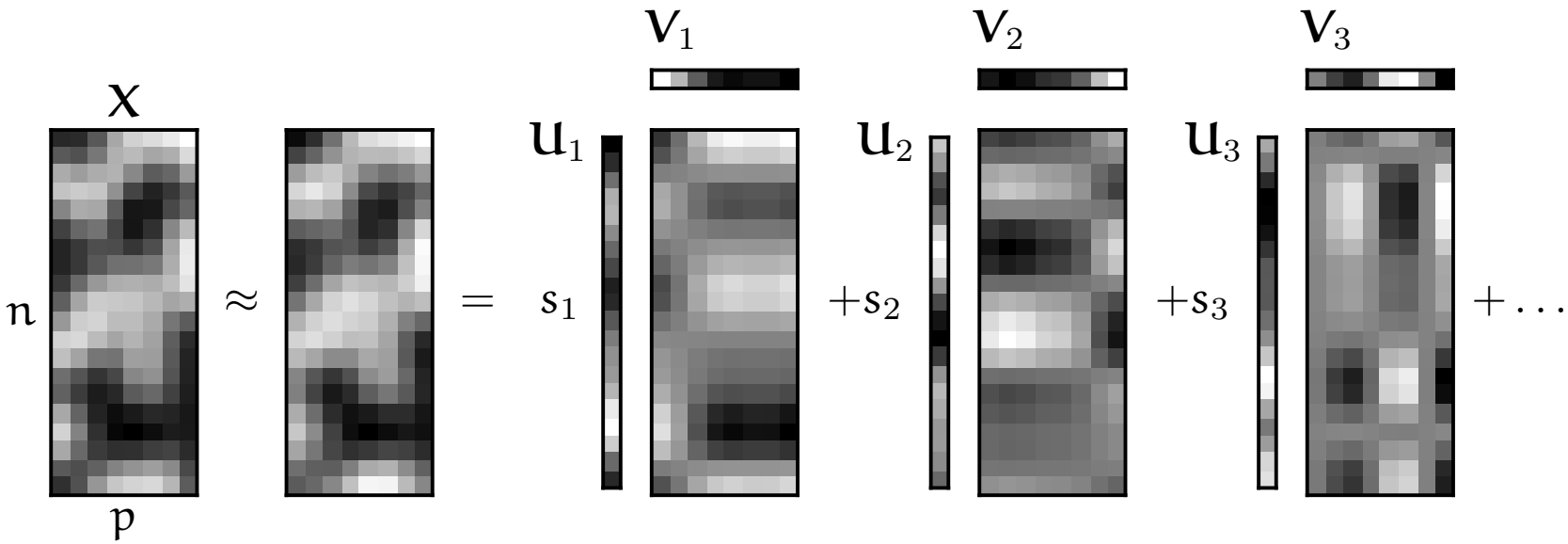
Explained variance: 0.84

$$U^T U = I_p \tag{18}$$

$$V^T V = I_p \tag{19}$$

Singular Value Decomposition

$$X = U S V^T \tag{20}$$



Explained variance: 0.97

$$U^T U = I_p \tag{21}$$

$$V^T V = I_p \tag{22}$$

Other decomposition methods

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

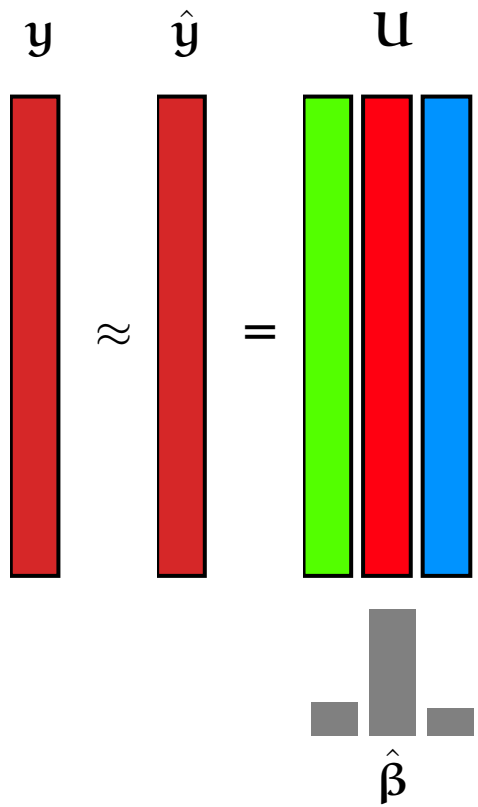
- Dictionary Learning
- Non-negative Matrix Factorization
- K-means clustering
- ...

What about $\mathbf{y}$?

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

Ridge regression and PCA

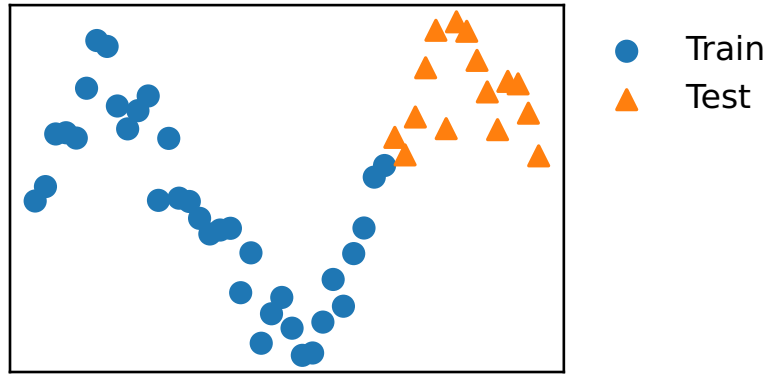
- Both ridge regression and PC regression compute the coordinates of $\mathbf{y}$ in the basis given by the SVD of $\mathbf{X}$
- Ridge shrinks the coordinate along $\mathbf{u}_j$ by a factor $s_j^2/(s_j^2+\alpha)$
- PC regression sets the coordinates to 0 except for those corresponding to the k largest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

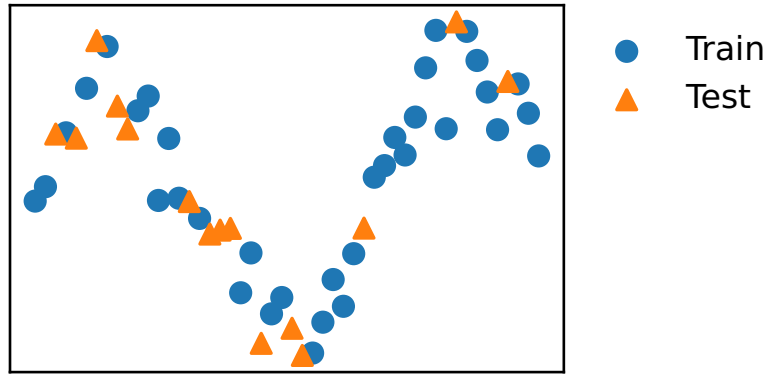


Outline

Split choice example: time series

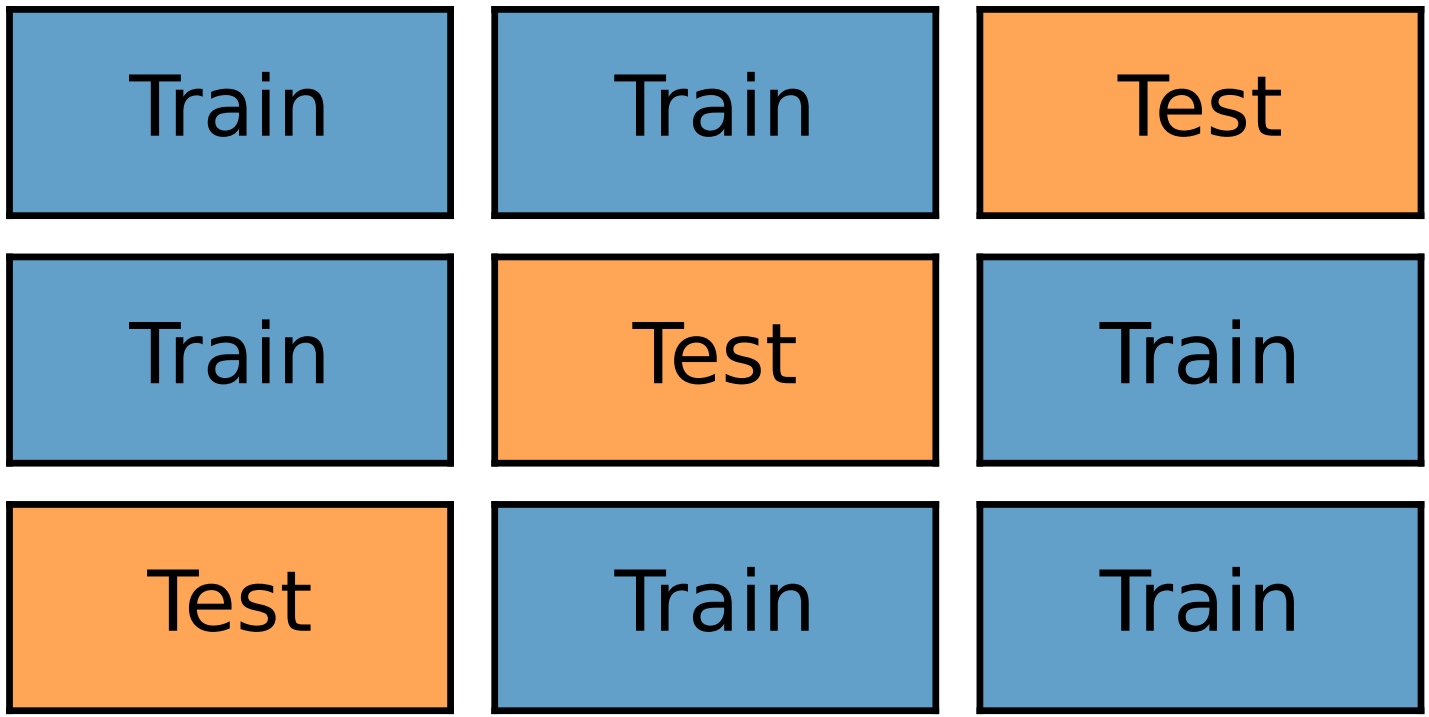
Don’t ignore dependencies between samples: which is easier?





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