Session 13:

Supervised learning, part 2

Andreas Bjerre-Nielsen

Agenda

Vaaaamos

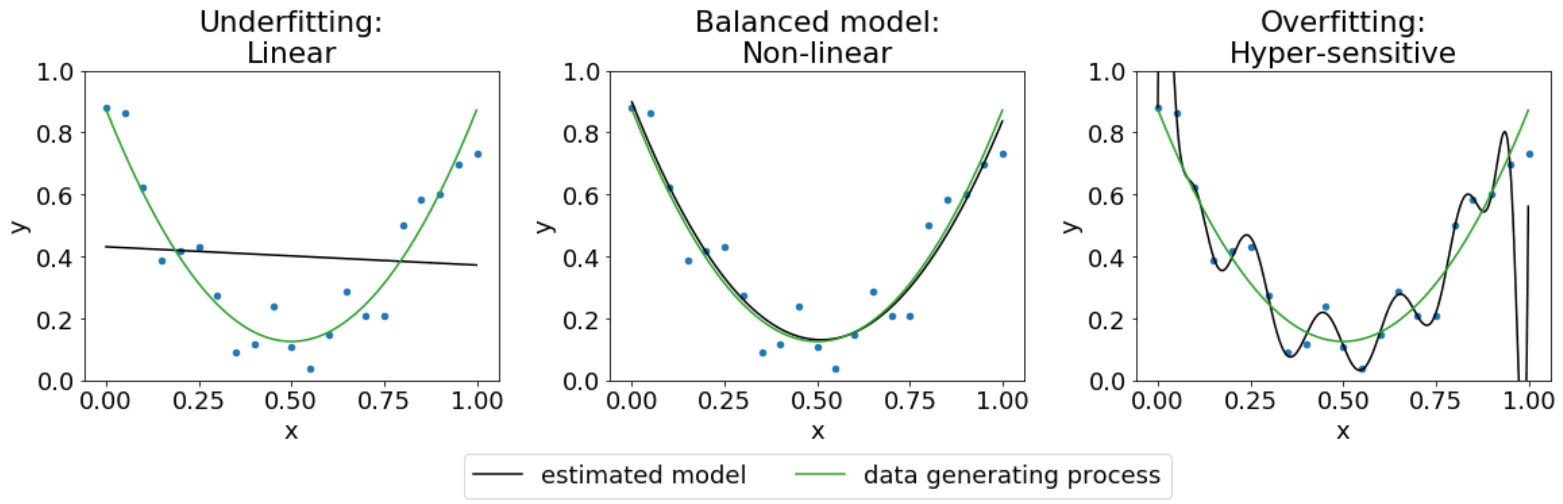
In [2]:
```
import warnings
from sklearn.exceptions import ConvergenceWarning
warnings.filterwarnings(action='ignore', category=ConvergenceWarning)

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

plt.style.use('default') # set style (colors, background, size, gridlines etc.)
plt.rcParams['figure.figsize'] = 10, 4 # set default size of plots
plt.rcParams.update({'font.size': 18})
```

Supervised problems (1)

What is the tradeoff for making supervised regression models?

```
In [3]: f_bias_var['regression'][2]
```

```
Out[3]:
```



Supervised problems (2)

What was a remedy to overfitting in linear models? How do we measure overfitting?

- .
- .

Model building

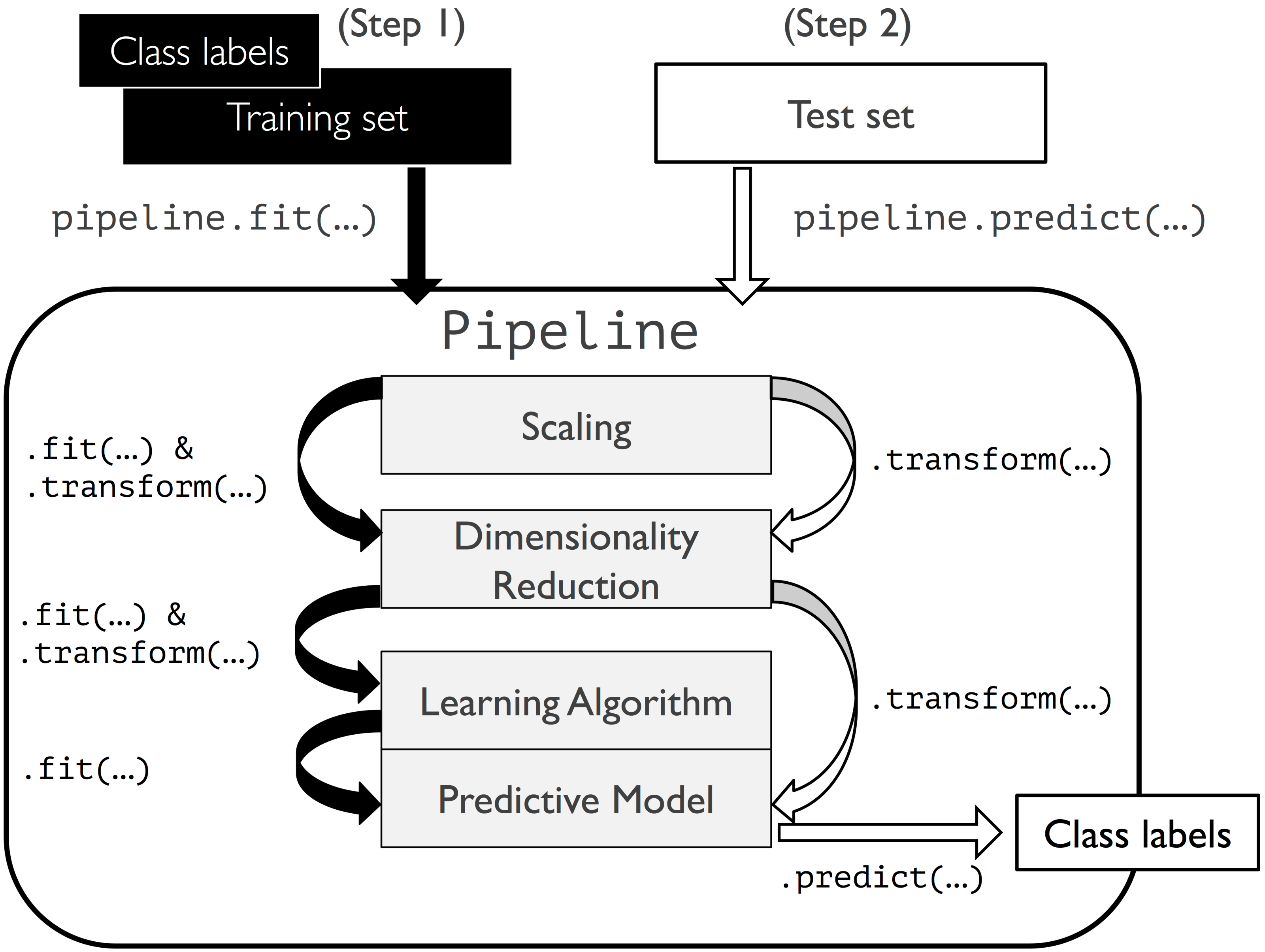
Model pipelines (1)

Is there a smart way to build ML models?

Yes, we build a pipeline:

- Preprocessing data
  - Standard: adding polynomials, imputation, rescaling
  - Unsupervised learning (more info..)
- Supervised learning

Model pipelines (2)


(Step 1)
Class labels
Training set
pipeline.fit(...)
(Step 2)
Test set
pipeline.predict(...)
Pipeline
Scaling
Dimensionality Reduction
Learning Algorithm
Predictive Model
.fit(...) & .transform(...)
.fit(...) & .transform(...)
.fit(...)
.transform(...)
.transform(...)
.predict(...)
Class labels

Model pipelines (3)

What are the advantages of using a pipeline?

- Ensures good practice - we only fit on training data.
- Much less code!

Applying a model pipeline (1)

What would this look like in Python?

In [4]:
```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures, StandardScaler

pipe_preproc = make_pipeline(PolynomialFeatures(),
                             StandardScaler())

# pipe_preproc.steps
```

Applying a model pipeline (2)

Let's some load Boston house price data

```
In [49]: print('\n'.join(load_boston()['DESCR'].split('\n')[13:28]))
```

```
    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwi
se)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's
```

Applying a model pipeline (3)

And how do I apply the pipe on the data?

```
In [41]:
```

```python
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_boston
X = load_boston().data
y = load_boston().target

print(load_boston().feature_names)

# splitting into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y)

# apply preproc - fit on train
pipe_preproc.fit(X_train)
X_train_prep = pipe_preproc.transform(X_train)
X_test_prep = pipe_preproc.transform(X_test)
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

Model selection

Measuring the problem

Does machine learning work out of the box?

- In some cases ML works quite well out of the box.
- Often ML requires making careful choices.
    - Note that automated machine learning packages and services exist.

Which choices are to be made?

- We need to pick model building **hyperparameters**.
- E.g. λ for Lasso, Ridge.

Model validation (1)

How do we measure our model's performance for different hyperparameters?

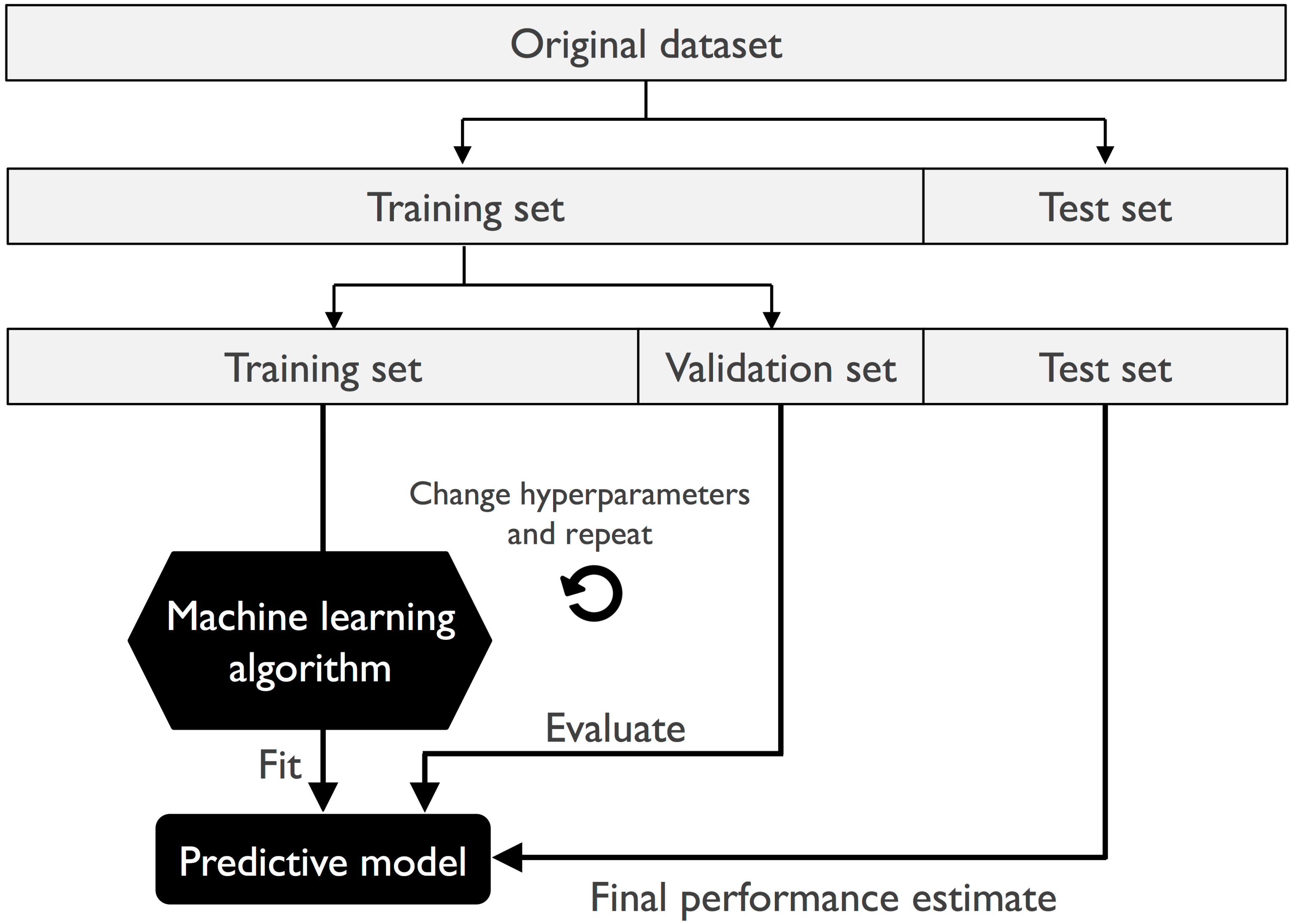
- Remember we cannot use the test set.

Could we somehow mimick what we do with test data?

- Yes, we can split the remaining non-test data into training and validation data:
  - we train model for various hyperparameters on training data;
  - pick the hyperparameters which performs best on validation data.

Model validation (2)

The non-test data is split into training and validation


Original dataset
Training set
Test set
Training set
Validation set
Test set
Change hyperparameters and repeat
Machine learning algorithm
Evaluate
Fit
Predictive model
Final performance estimate

Model validation (3)

What would this look like in Python?

In [6]:
```
# splitting into development (2/3) and test data (1/3)
X_dev, X_test, y_dev, y_test = train_test_split(X, y, test_size=1/3, random_state=1)

# splitting development into train (1/3) and validation (1/3)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=1/2, random_state=1)
```

Model validation (4)

Let's train a linear regression model

In [7]:
```
from sklearn.linear_model import Lasso, LinearRegression
pipe_lr = make_pipeline(PolynomialFeatures(include_bias=False),
                           StandardScaler(),
                           LinearRegression())

pipe_lr.fit(X_dev,y_dev)
```

Out[7]:
```
Pipeline(memory=None,
     steps=[('polynomialfeatures', PolynomialFeatures(degree=2, include_bias=False, i
nteraction_only=False)), ('standardscaler', StandardScaler(copy=True, with_mean=True,
with_std=True)), ('linearregression', LinearRegression(copy_X=True, fit_intercept=Tru
e, n_jobs=1, normalize=False))])
```

Model validation (5)

Let's find the Lasso model which performs best in the validation set

In [8]:

```
from sklearn.metrics import mean_squared_error as mse

perform = []
lambdas = np.logspace(-4,4,33)
for lambda_ in lambdas:
    pipe_lasso = make_pipeline(PolynomialFeatures(include_bias=False),
                               StandardScaler(),
                               Lasso(alpha=lambda_, random_state=1))
    pipe_lasso.fit(X_train, y_train)
    perform.append(mse(pipe_lasso.predict(X_val),y_val))

optimal = pd.Series(perform,index=lambdas).nsmallest(1)
print(optimal)
```

Be aware of corner solutions; if so extend range.

Finds the smallest number in each list

Lambda MSE

```
0.017783    12.235154
dtype: float64
```

Model validation (6)

Let's compare the performance of the Lasso vs. Linear Regression

In [9]:
```
pipe_lasso = make_pipeline(PolynomialFeatures(include_bias=False),
                              StandardScaler(),
                              Lasso(alpha=optimal.index[0]))

pipe_lasso.fit(X_dev,y_dev)
print('Lasso', round(mse(pipe_lasso.predict(X_test),y_test), 3))
print('LinReg', round(mse(pipe_lr.predict(X_test),y_test), 3))
```

```
Lasso 11.637
LinReg 17.895
```

Bias and variance (1)

How do we describe the modelling error?

From [Wikipedia Sunday, August 19, 2018 (https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff)](https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff):

- model **bias**: *an error from erroneous assumptions in the learning algorithm*
  - oversimplification of models, cannot approximate all patterns found
- model **variance**: *an error from sensitivity to small fluctuations in the training set*
  - reacts too much to sample errors and thus finds too many spurious relations

Bias and variance (2)

- **over fitting**: low bias / high variance
  - traning our model captures all patterns but we also find some irrelevant
  - examples: Decision Trees, Support Vector Machines or Neural Networks
- **under fitting**: high bias / low variance
  - traning our model captures all patterns but we also find some irrelevant
  - examples: linear and logistic regression (without polynomial expansion)

Bias and variance (3)

Not so fast.. OLS is unbiased, right?

Yes, OLS is unbiased. But ..

- Requires we know the true form of the model.

What happens if we introduce regularization?

- Then model is no longer unbiased.

Smarter validation

Is this approach the smartest way for deciding on choice of hyperparameters?

NO

Our model choice depends a lot on which sample we pick. Could we use more of the data?

Cross validation

The holdout method

How do we got the more out of the data?

We reuse the data in the development set repeatedly

- We test on all the data
- Rotate which parts of data is used for test and train.

Leave-one-out CV

How do we got the most of the data?

The most robust approach

- Each single observation in the training data we use the remaining data to train.
- Makes number of models equal to the number of observations
- Very computing intensive - does not scale! LOOCV

K fold method (1)

How do balance computing time vs. overfitting?

We split the sample into K even sized test bins.

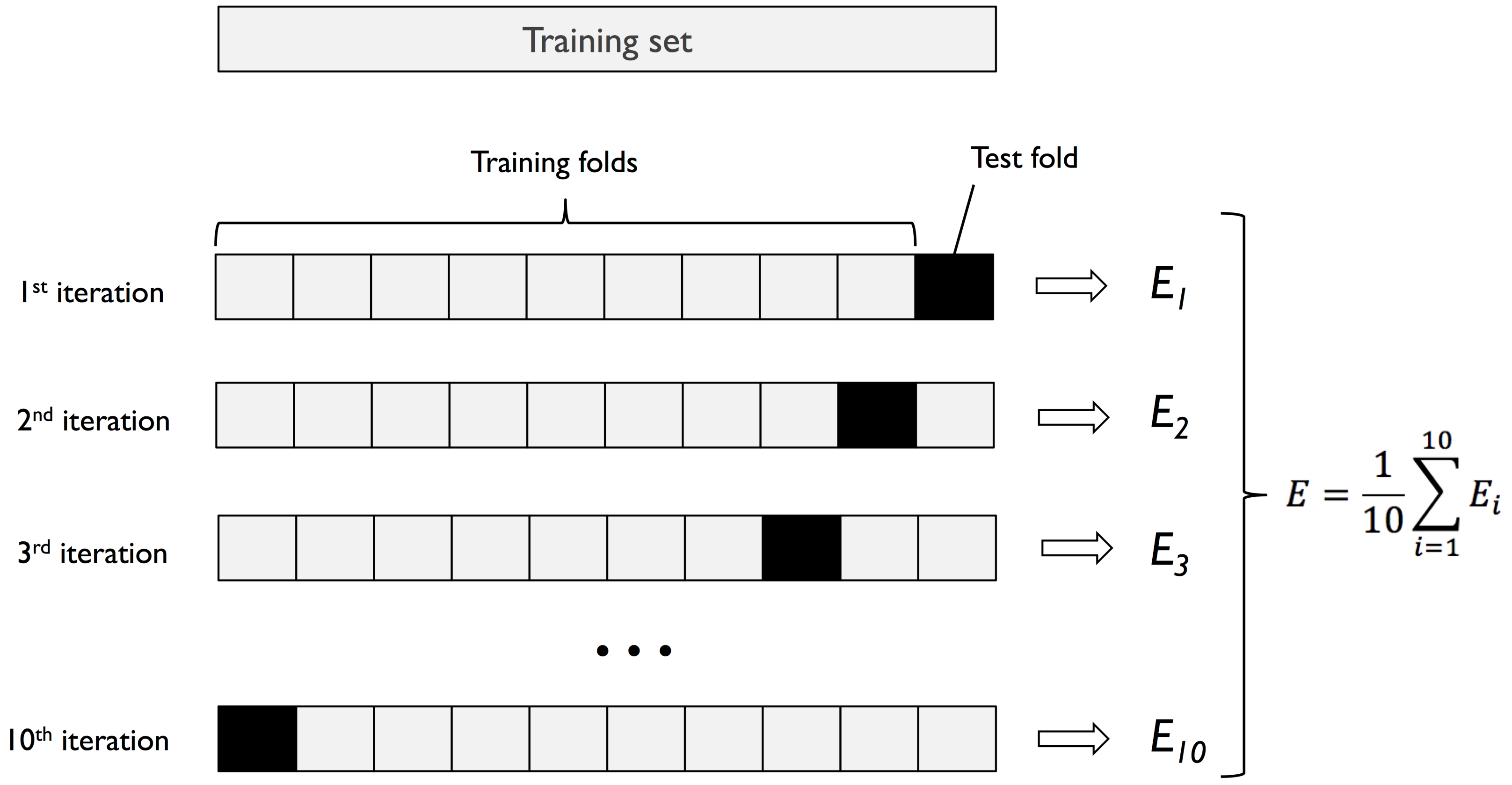
- For each test bin k we use the remaining data for training.

Advantages:

- We use all our data for testing.
- Training is done with 100-(100/K) pct. of the data, i.e. 90 pct. for K=10.

K fold method (2)

In K-fold cross validation we average the errors.



$$E = \frac{1}{10}\sum_{i=1}^{10} E_i$$

K fold method (3)

How would we use K-fold cross validation to select our model?

In [12]:
```
from sklearn.model_selection import KFold
kfolds = KFold(n_splits=10)
mseCV = []

for lambda_ in lambdas:

    pipe_lassoCV = make_pipeline(PolynomialFeatures(degree=3, include_bias=False),
                                 StandardScaler(),
                                 Lasso(alpha=lambda_, random_state=1))
    mseCV_ = []

    for train_idx, val_idx in kfolds.split(X_dev, y_dev):

        X_train, y_train, = X_dev[train_idx], y_dev[train_idx]
        X_val, y_val = X_dev[val_idx], y_dev[val_idx]

        pipe_lassoCV.fit(X_train, y_train)

        mseCV_.append(mse(pipe_lassoCV.predict(X_val), y_val))
    mseCV.append(mseCV_)
```

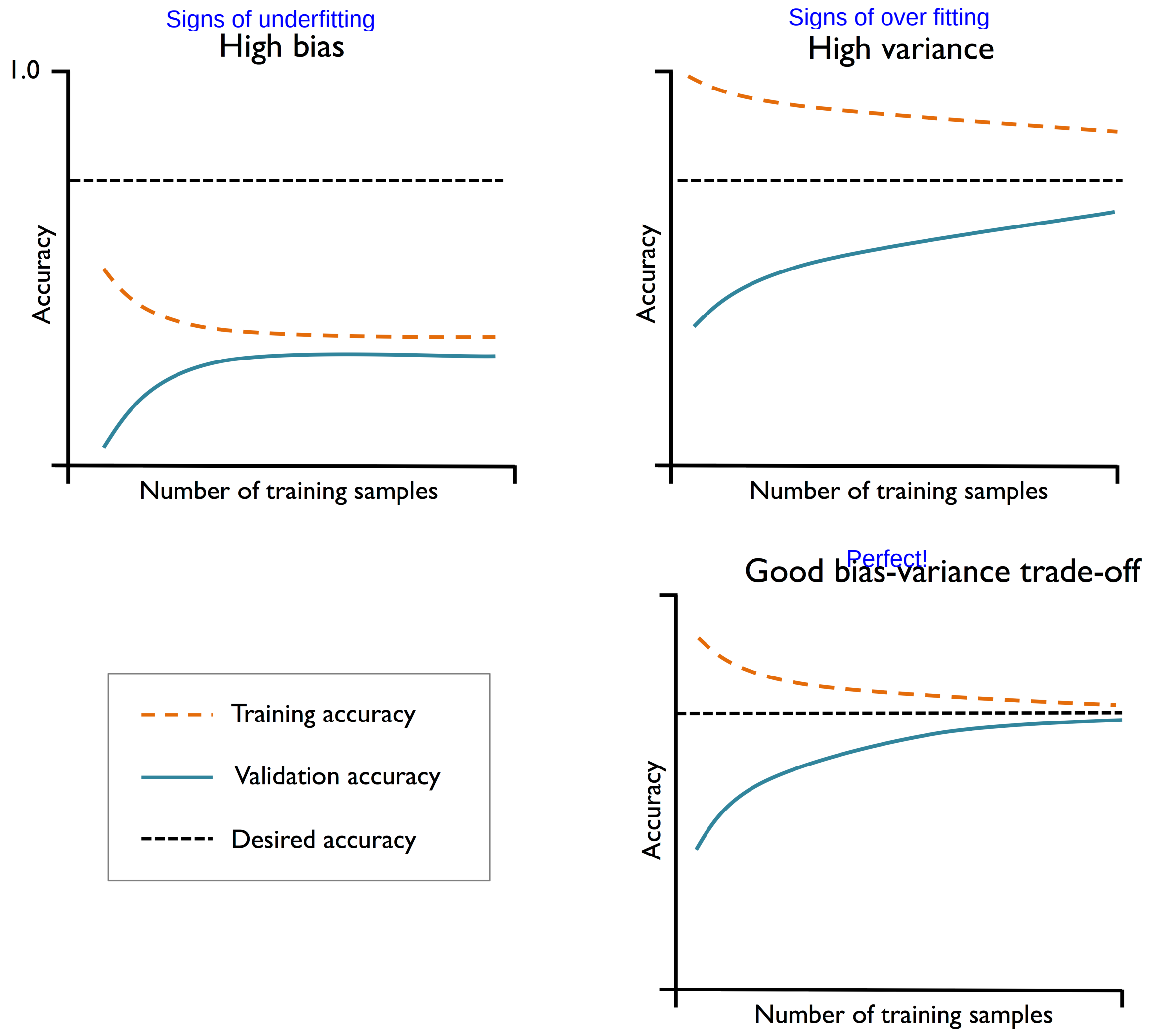
K fold method (4)

```
In [13]: optimalCV = pd.DataFrame(mseCV, index=lambdas).mean(axis=1).nsmallest(1)

         pipe_lassoCV = make_pipeline(PolynomialFeatures(include_bias=False),
                                      StandardScaler(),
                                      Lasso(alpha=optimalCV.index[0]))

         pipe_lassoCV.fit(X_dev,y_dev)

         model_pipes = ('Lasso', pipe_lasso), ('Lasso CV', pipe_lassoCV),('LinReg', pipe_lr)

         for model_name, model_pipe in model_pipes:
             score = mse(model_pipe.predict(X_test),y_test)
             print(model_name, round(score, 1))
```

```
Lasso 11.6
Lasso CV 12.4
LinReg 17.9
```

Learning curves (1)

What does a balanced model look like?

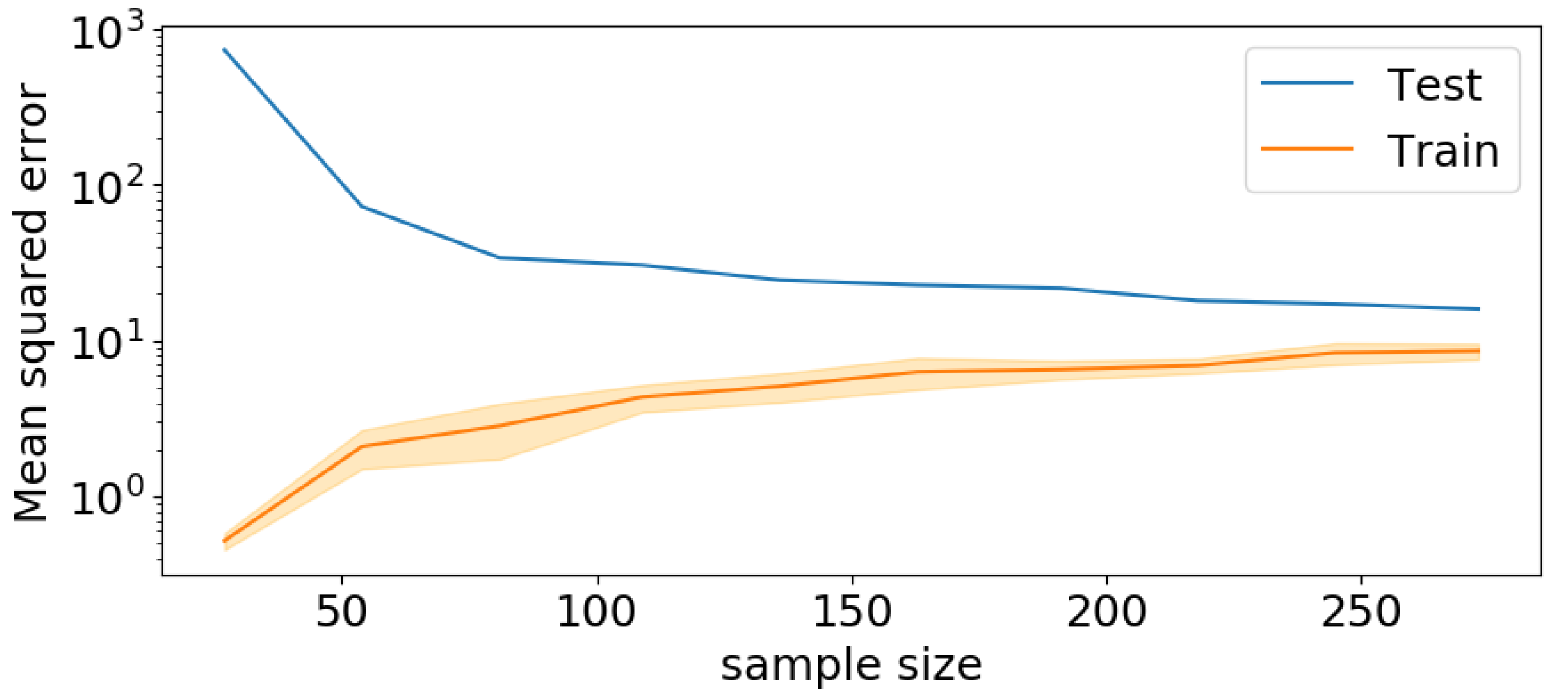


Learning curves (2)

```
In [37]: from sklearn.model_selection import learning_curve

         train_sizes, train_scores, test_scores = \
             learning_curve(estimator=pipe_lasso,
                            X=X_train,
                            y=y_train,
                            train_sizes=np.linspace(0.1, 1.0, 10),
                            scoring='neg_mean_squared_error',
                            cv=10)

         mse_ = pd.DataFrame({'Train':-train_scores.mean(axis=1),
                              'Test':-test_scores.mean(axis=1)})\
                 .set_index(pd.Index(train_sizes,name='sample size'))
```

Learning curves (3)

```
In [38]: f_learn, ax = plt.subplots(figsize=(10,4))
         mse_.plot(ax=ax, logy=True)
         ax.fill_between(train_sizes,
                         -train_scores.mean(1) + train_scores.std(1)*1.96,
                         -train_scores.mean(1) - train_scores.std(1)*1.96,
                         alpha=0.25,
                         color='orange')
         ax.set_ylabel('Mean squared error')

Out[38]: Text(0,0.5,'Mean squared error')
```



Tools for model selection

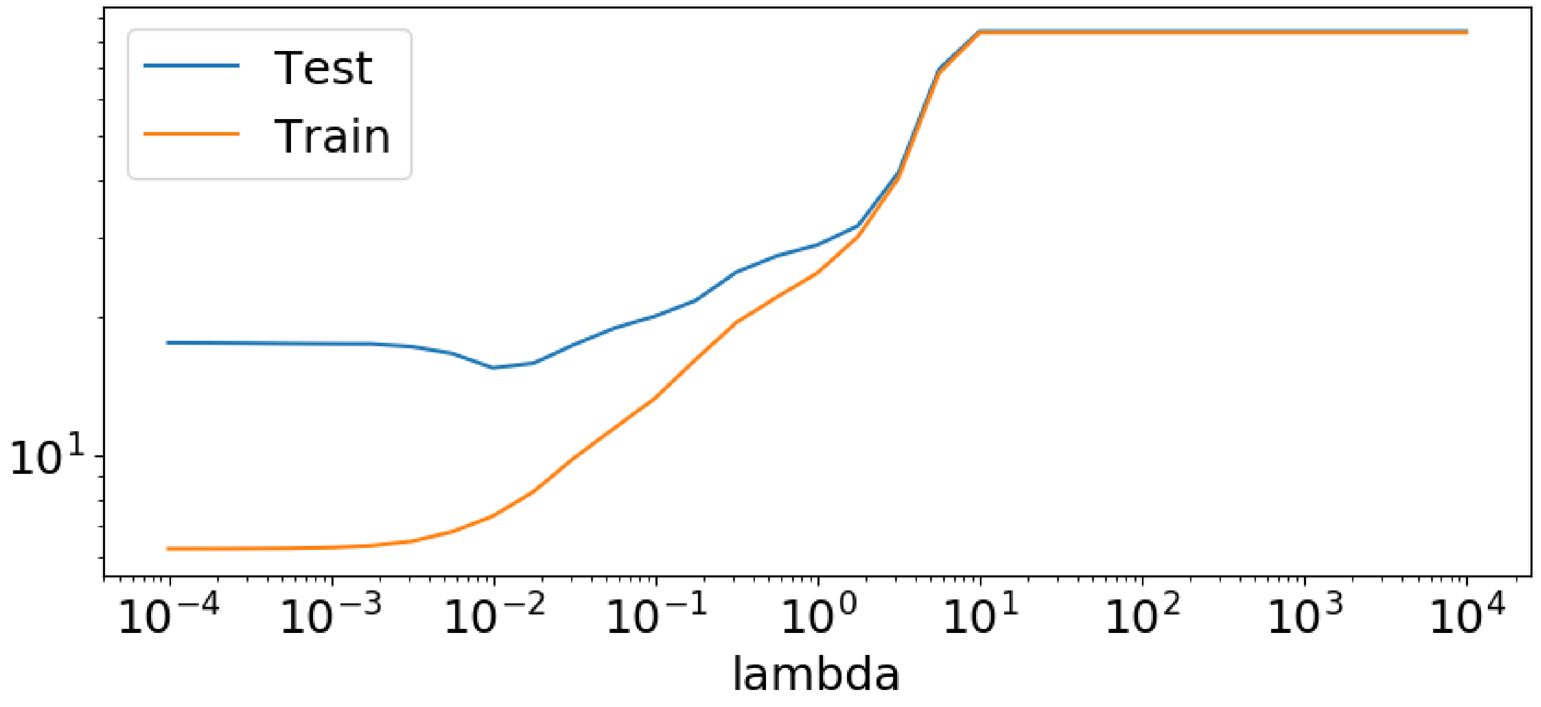
Validation curves (1)

```
In [18]: from sklearn.model_selection import validation_curve

         train_scores, test_scores = \
             validation_curve(estimator=pipe_lasso,
                              X=X_train,
                              y=y_train,
                              param_name='lasso__alpha',
                              param_range=lambdas,
                              scoring='neg_mean_squared_error',
                              cv=5)

         mse_score = pd.DataFrame({'Train':-train_scores.mean(axis=1),
                                   'Test':-test_scores.mean(axis=1),
                                   'lambda':lambdas})\
                       .set_index('lambda')
         print(mse_score.Test.nsmallest(1))
```

```
lambda
0.01    15.511598
Name: Test, dtype: float64
```

Validation curves (2)

```
In [19]: mse_score.plot(logx=True, logy=True)

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x2910be29160>
```



Grid search (1)

How do we search for two or more optimal parameters?

- Goal: find the optimal parameter combination:
  $$\lambda_1^*, \lambda_2^* = \arg\min_{\lambda_1, \lambda_2} MSE^{CV}(X_{train}, y_{train})$$
- Option 1: We can loop over the joint grid of parameters.
  - One level for each parameter.
  - Caveats:
- Option 2: sklearn has `GridSearchCV` has a tool which tests all parameter combinations.

Grid search (2)

How does this look in Python?

```
In [21]: from sklearn.model_selection import GridSearchCV

         gs = GridSearchCV(estimator=pipe_lasso,
                           param_grid=[{'lasso__alpha':lambdas}],
                           scoring='neg_mean_squared_error',
                           cv=10,
                           n_jobs=-1)

         gs = gs.fit(X_train, y_train)
         gs.best_params_
```

```
Out[21]: {'lasso__alpha': 0.01}
```

- Notation: double underscore between estimator and hyperparameter, e.g. 'est__hyperparam'
- Scoring: negative MSE as we're maximizing the score ~ minimize MSE.

Grid search (3)

What if we have 10,000 parameter combinations?

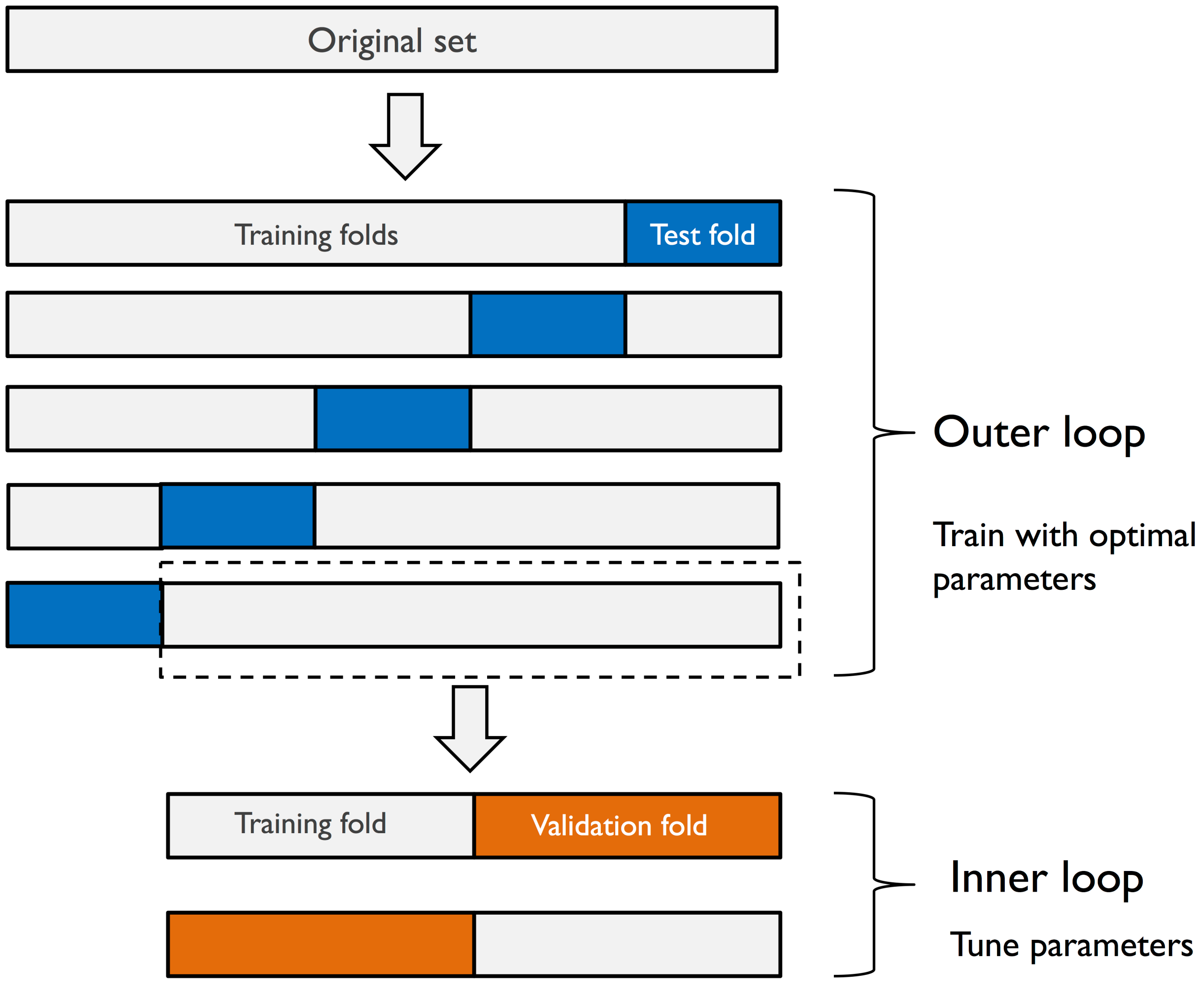
- Option 1: you buy a cluster on Amazon, learn how to parallelize across computers. Intro in last lecture.
- Option 2: you drop some of the parameter values
- Option 3: `RandomizedSearchCV` searches a subset of the combinations.

Enhancing cross validation (1)

- Model validation does not consider that we are also tuning hyperparameters:
  - Leads too overfitting (Varma & Simon 2006; Cawley, Talbot 2010).
- Solution is **nested cross validation**.
  - Validation step should not be modelled as 1) train; 2) test.
  - Better way is 1) model selection: train, validate; 2) test.
  - Implement as pp 204-205 in Python for Machine Learning:
    - first inner loop: `GridSearchCV`
    - second outer loop: `cross_val_score`

Enhancing cross validation (1)

Cross-val. suffers from the fact that it models test-train


Original set
Training folds
Test fold
Outer loop
Train with optimal parameters
Training fold
Validation fold
Inner loop
Tune parameters

Dimensionality reduction

Principal components analysis (1)

How can we reducing the number of features?

One solution is finding the **principal components**.

- essence: we get **fewer features** of **greater importance**.
- the new features are:
    - *uncorrelated* (i.e. linearly independent, orthogonal)
    - ordered so decreasing in how much variation of the feature data they explain

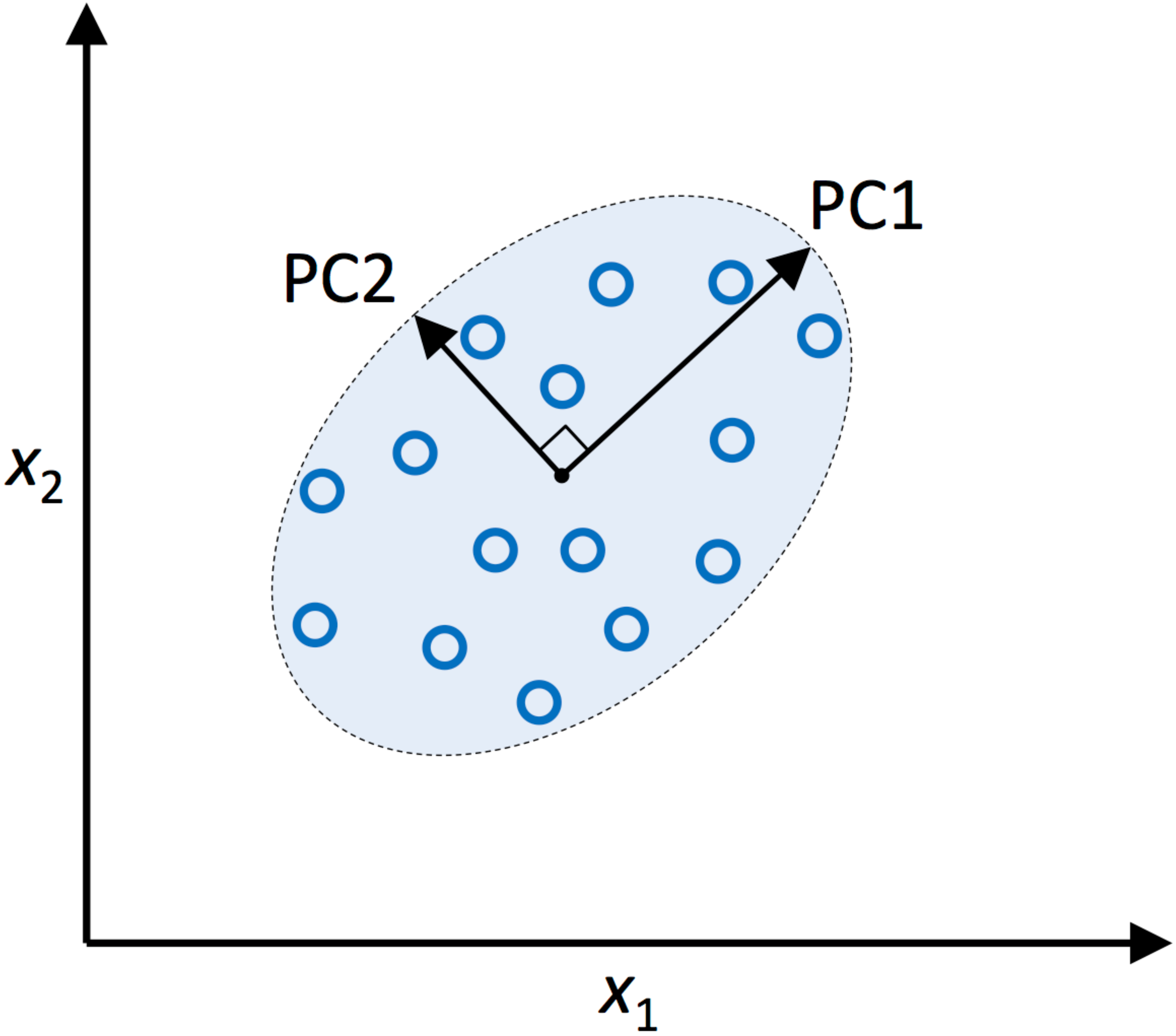
The method is called **principal components analysis**

- corresponds to eigen decomposition of matrix into
    - principal eigenvectors (factors)
    - principal eigenvalues (factor importance)

Principal components analysis (2)

Finding principal components for two features. Notice:

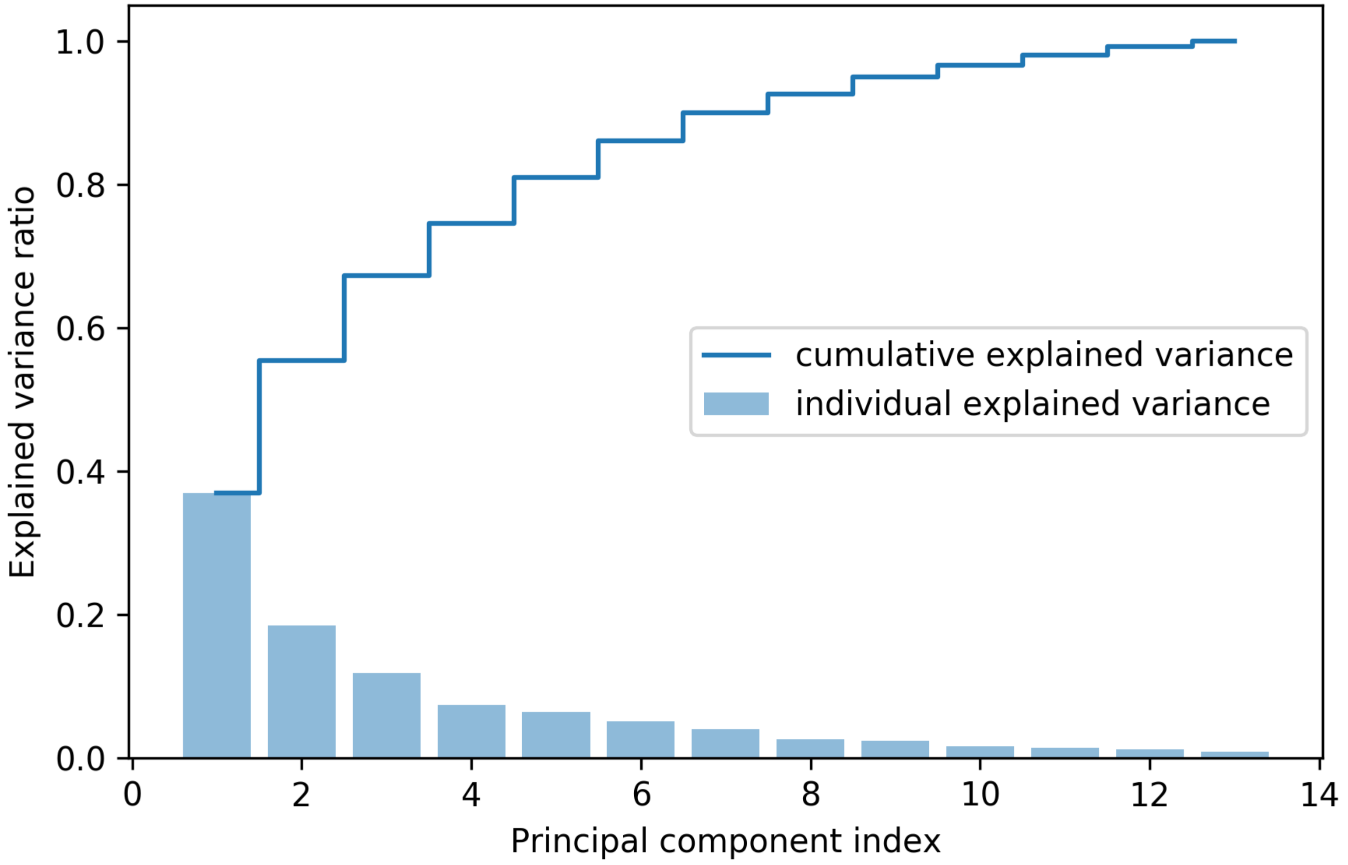
- The factors are orthogonal
- The first factor explains more variation, |PC1| > |PC2|


PC1
PC2
x_2
x_1

Principal components analysis (3)

We can plot the explained variation against the component indices, often called scree plot.

Normally when do principal component analysis, after the third featuer we see an "eblow": thieres a jump in how much is explained. Normally you would eyebsall it; around 3 and 4 is where we want to ...
Which is different than ML. As it is part of how we workw with our preprocessing and therefore we should throw it into our model building.



Principal components analysis (4)

How do we choose the number of components?

- Standard is to look for an *elbow* in the previous scree plot.

What might go wrong about this approach??

- The number of feautures should be a hyperparameter in the model building!!!

Principal components analysis (5)

How does this look in Python?

```
In [22]: from sklearn.decomposition import PCA

         pipe_pca_lasso = make_pipeline(PolynomialFeatures(include_bias=False),
                                        StandardScaler(),
                                        PCA(),
                                        Lasso())


         gs = GridSearchCV(estimator=pipe_pca_lasso,
                           param_grid=[{'lasso__alpha':lambdas,
                                        'pca__n_components':range(1, X_train.shape[1]+1)}],
                           scoring='neg_mean_squared_error',
                           cv=10,
                           n_jobs=-1)

         gs = gs.fit(X_train, y_train)
         gs.best_params_
```

```
Out[22]: {'lasso__alpha': 0.1778279410038923, 'pca__n_components': 13}
```

Measures for classification

Breakdown by error type (1)

We measure the accaracy as the rate of true predictions, i.e.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} = \frac{True}{True + False}$$

where our measures are

		Predicted class	
		P	N
Actual class	P	True positives (TP)	False negatives (FN)
	N	False positives (FP)	True negatives (TN)

Breakdown by error type (2)

Some powerful measures:

- Precision: share of predicted positive that are true
  - PRE = $\frac{TP}{TP+FP}$
- Recall: share of actual positive that are true
  - REC = $\frac{TP}{TP+FN} = \frac{TP}{AP}$
  - Also known as True Positive Rate, TPR
- F1: mix recall and precision: $\frac{2 \cdot PRE \cdot REC}{PRE \cdot REC}$
- False Positive Rate: share of actual negatives that are true
  - FPR = $\frac{FP}{FP+TN} = \frac{FP}{AN}$

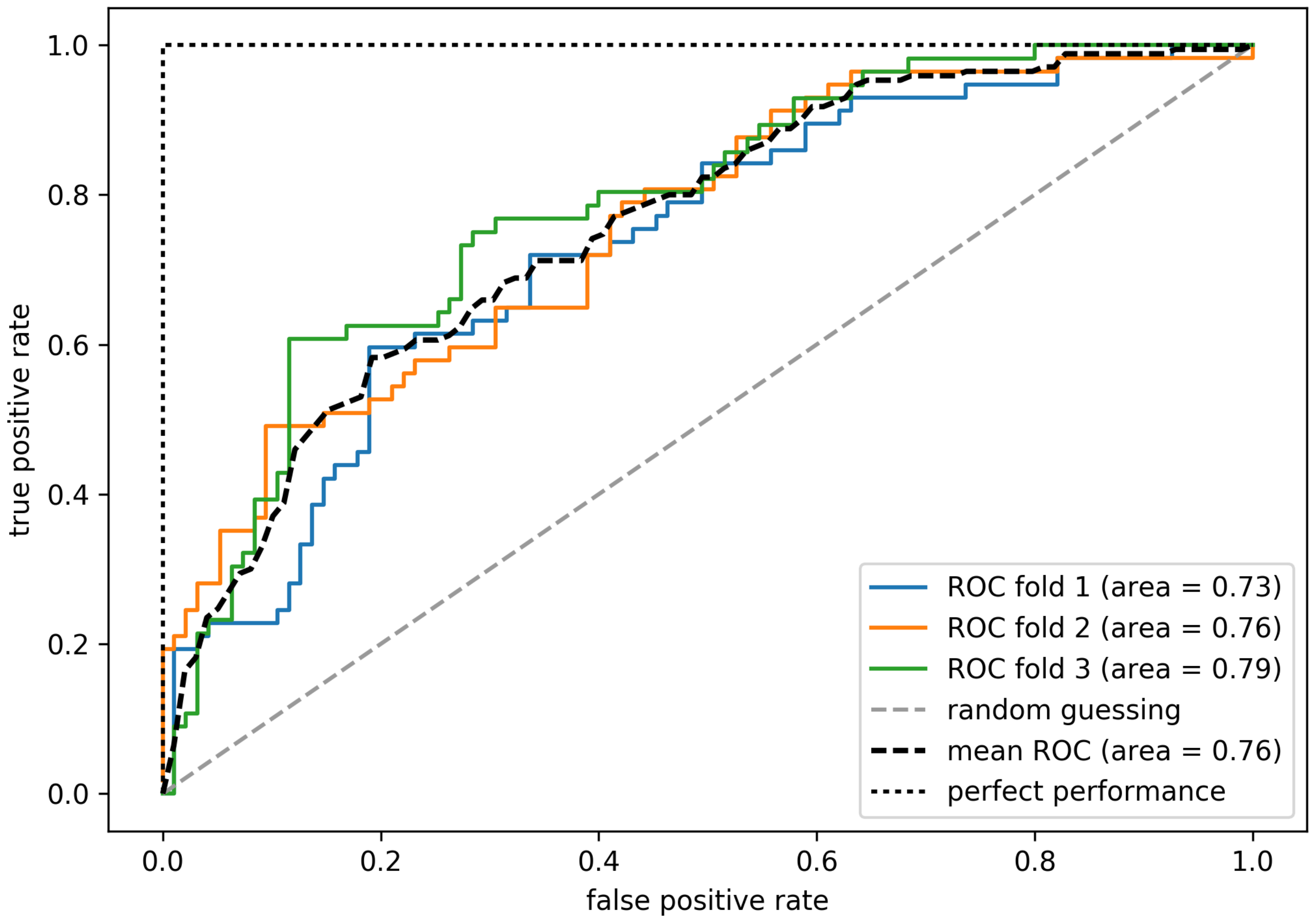
In [ ]:

```
from sklearn.metrics import precision_score, recall_score, f1_score
```

Breakdown by error type (3)

Classification models provide a predicted likelihood of being in the class or not:

- Receiver Operating Characteristic (ROC) curve by varying thresholds for predicted true.
  - ROC is a *theoretical* measure of model performance based on probabilities.
  - AUC: Area Under the (ROC) Curve.



true positive rate
false positive rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC fold 1 (area = 0.73)
ROC fold 2 (area = 0.76)
ROC fold 3 (area = 0.79)
random guessing
mean ROC (area = 0.76)
perfect performance

The end

Return to agenda