Methods 3: Multilevel Statistical Modeling and Machine Learning

Week 10: *Organising and preprocessing messy data*
November 30, 2021

by: Lau Møller Andersen

REMEMBER: Corona-passport is a requirement now

(I’m not going to check you)

Santa instructions?

https://bornibyen.dk/aarhus/articles/sjov-udendoers-kunst-i-aarhus

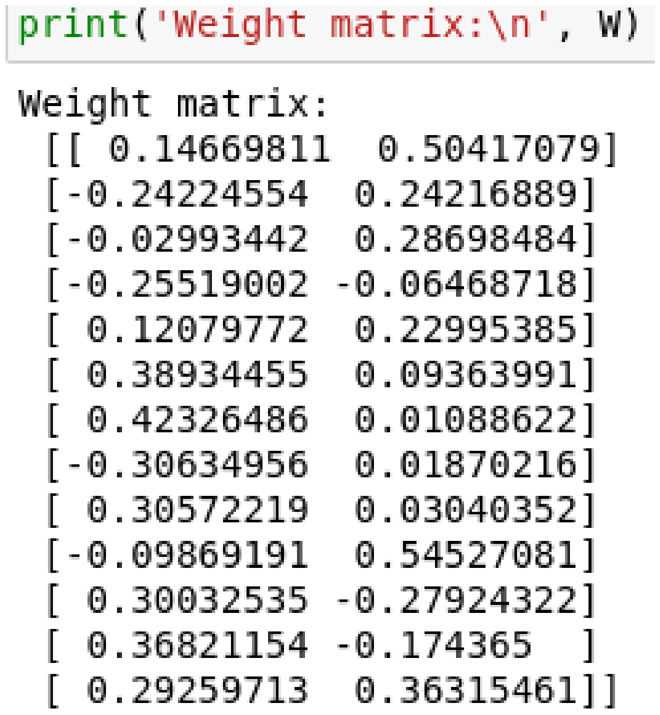
CHUNG MOO
ÅRHUS TAEKWONDO KLUB

December 21st
(11-13)
Too late?

Silkeborgvej 41E, baghuset 1.th, 8000 Aarhus C.

Sign-up list here:
https://cryptpad.fr/pad/#/2/pad/edit/NHEOwwpF2nH8vTGDNPuQZENx/

Follow-up (PCA):
How to interpret ***W***?



```
print('Weight matrix:\n', W)
```

```
Weight matrix:
 [[ 0.14669811  0.50417079]
 [-0.24224554  0.24216889]
 [-0.02993442  0.28698484]
 [-0.25519002 -0.06468718]
 [ 0.12079772  0.22995385]
 [ 0.38934455  0.09363991]
 [ 0.42326486  0.01088622]
 [-0.30634956  0.01870216]
 [ 0.30572219  0.03040352]
 [-0.09869191  0.54527081]
 [ 0.30032535 -0.27924322]
 [ 0.36821154 -0.174365  ]
 [ 0.29259713  0.36315461]]
```

$$\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{W}$$

$\boldsymbol{Z}_{\boldsymbol{n \times k}}$: Underlying generator of data
$\boldsymbol{X}_{\boldsymbol{n \times d}}$: Observed data (mix of the generators)
$\boldsymbol{W}_{\boldsymbol{d \times k}}$: (inverse) weighting matrix (bringing us from data to generator)

$n:$ number of observations
$d:$ dimensions in observed data (n predictor variables)
$k:$ number of dimensions kept
$d-k:$ number of dimensions projected out

Let us derive the **forward** weighting matrix, $\boldsymbol{W}_{dxd}$, which brings us from generator, ***Z***, to observed data, ***X***

$$\boldsymbol{X} = ?$$

$$\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{W}$$

$$\boldsymbol{Z}^{\boldsymbol{T}} = (\boldsymbol{X}\boldsymbol{W})^{T} = \boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{X}^{\boldsymbol{T}}$$

$$(\boldsymbol{W}^{\boldsymbol{T}})^{-1}\boldsymbol{Z}^{\boldsymbol{T}} = \boldsymbol{X}^{\boldsymbol{T}}$$

$$\boldsymbol{W}\boldsymbol{Z}^{\boldsymbol{T}} = \boldsymbol{X}^{\boldsymbol{T}}$$

for orthonormal vectors: $(\boldsymbol{W}^{\boldsymbol{T}})^{-1} = \boldsymbol{W}$

Or expressed based on $\boldsymbol{X}$

$$\boldsymbol{X}^{\boldsymbol{T}} = \boldsymbol{W}\boldsymbol{Z}^{\boldsymbol{T}}$$

$$\boldsymbol{X} = \boldsymbol{X}^{\boldsymbol{TT}} = \left(\boldsymbol{W}\boldsymbol{Z}^{\boldsymbol{T}}\right)^{T}$$

$$\boldsymbol{X} = \boldsymbol{Z}\boldsymbol{W}^{\boldsymbol{T}}$$

$\boldsymbol{W}^{\boldsymbol{T}}:$ forward weighting matrix: bringing us from generator to data

$$\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{W} \qquad \boldsymbol{X} = \boldsymbol{Z}\boldsymbol{W}^{\boldsymbol{T}}$$

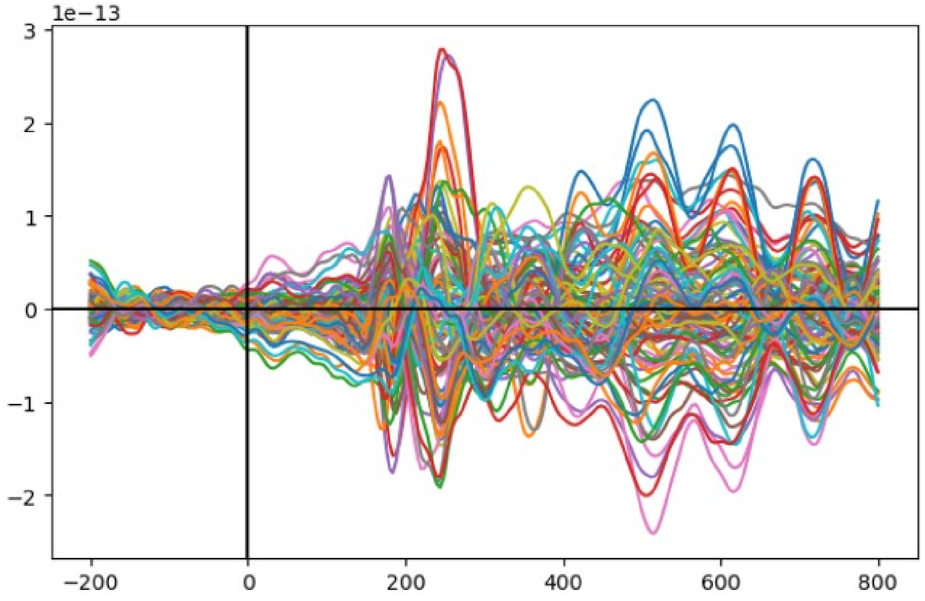
$\boldsymbol{W}^{\boldsymbol{T}}$: forward weighting matrix (or mixing matrix); from generator to data
$\boldsymbol{W}$: inverse weighting matrix (or unmixing matrix); from data to generator

W matrices are involved in many cases where we make measurements, ***X***, on something that is generated by a plethora of sources, ***Z***

e.g magneto- and electroencephalography

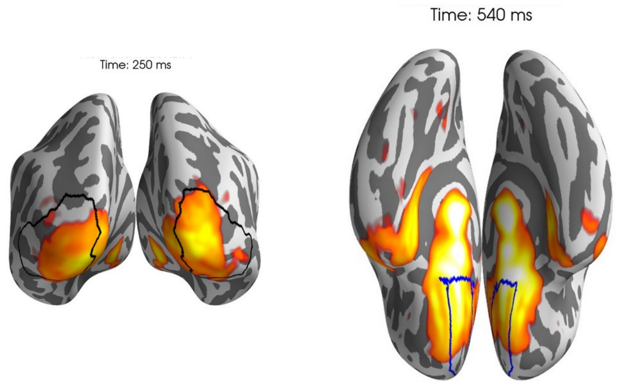
$$X = ZW^T$$

X

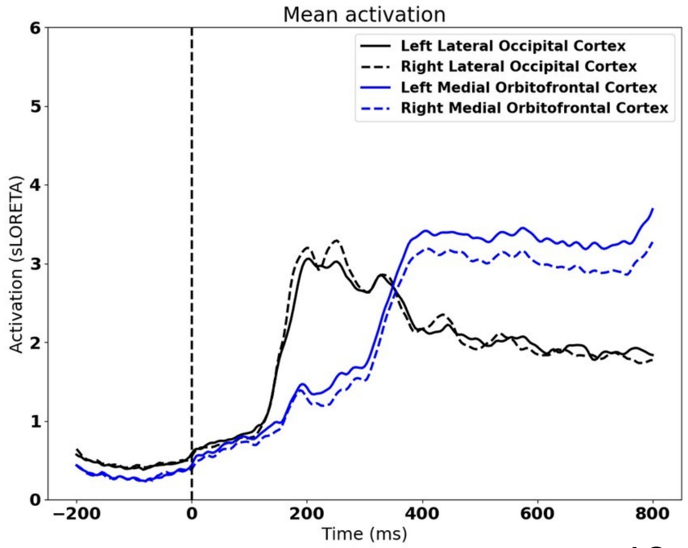


1e−13
3
2
1
0
−1
−2
−200
0
200
400
600
800

Z



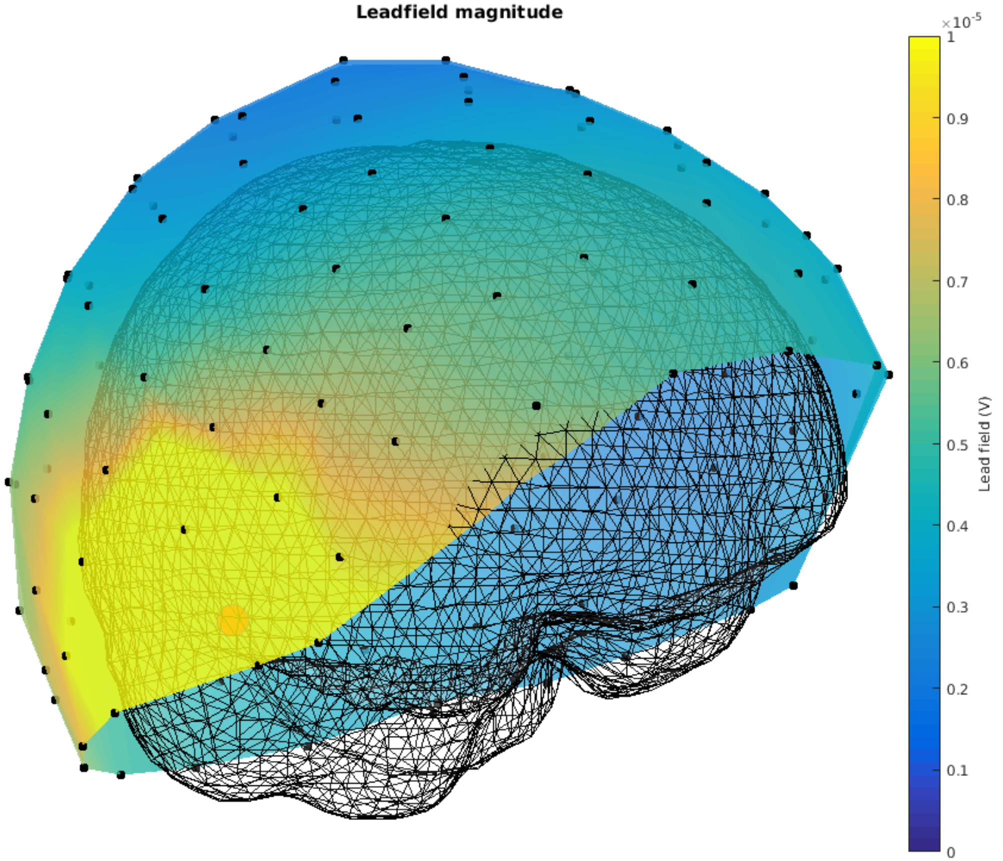
Time: 250 ms
Time: 540 ms



Mean activation
Left Lateral Occipital Cortex
Right Lateral Occipital Cortex
Left Medial Orbitofrontal Cortex
Right Medial Orbitofrontal Cortex
Activation (sLORETA)
Time (ms)
0
1
2
3
4
5
6
−200
0
200
400
600
800

$$\boldsymbol{X} = \boldsymbol{Z}\boldsymbol{W}^{T}$$

$$\boldsymbol{W}^{T}$$



The so-called forward model, $\boldsymbol{W}^{T}$, models how each source, $\boldsymbol{Z}$, is seen by the sensors, $\boldsymbol{X}$, when it is active

COURSE EVALUATION

you should have received an email

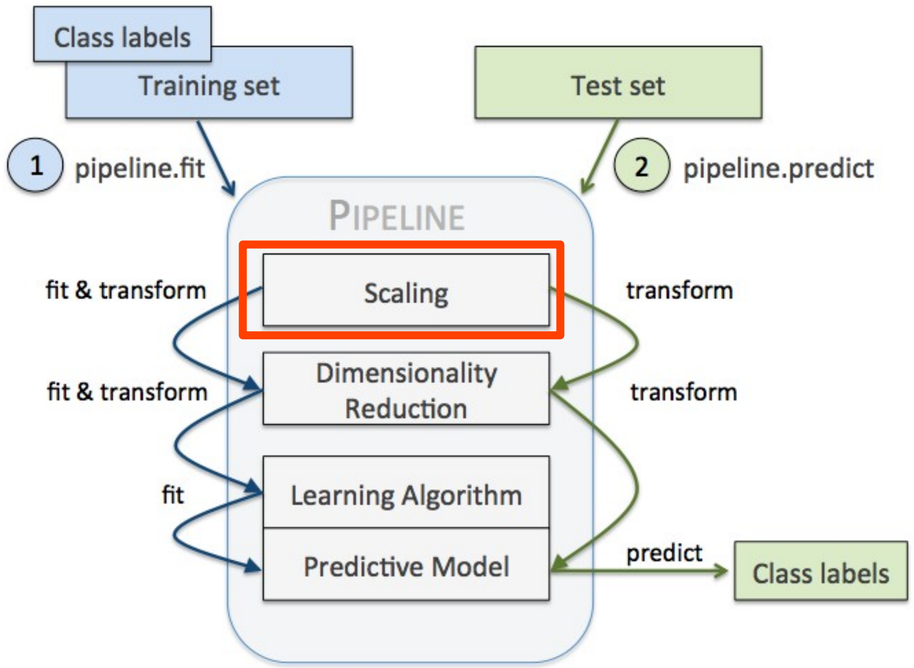
we’ll use ~15 minutes on it

Learning goals

Organising and preprocessing messy data

1) Learning why data should be scaled
2) Using feature selection to avoid overfitting
3) Understanding the basic steps of a machine learning pipeline

Pipeline example



(p. 172: Raschka, 2015)

Scaling

- Normalisation

$$x_{norm}^{(i)} = \frac{x^{(i)} - x_{min}}{x_{max} - x_{min}}$$

x_{max}: maximum value for the feature: x

x_{min}: minimum value for the feature: x

Brings everything on to the scale: [0, 1]

- Standardisation

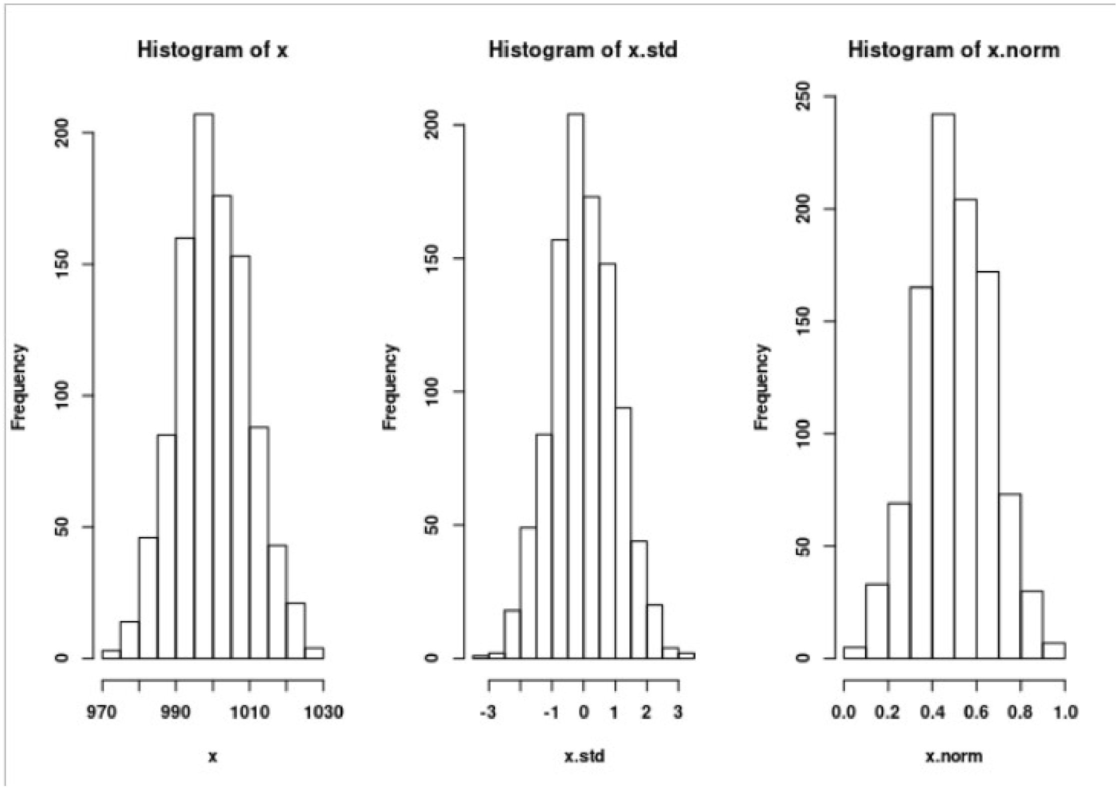
$$x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x}$$

μ_x: sample mean for the feature: x

σ_x: sample standard deviation for the feature: x

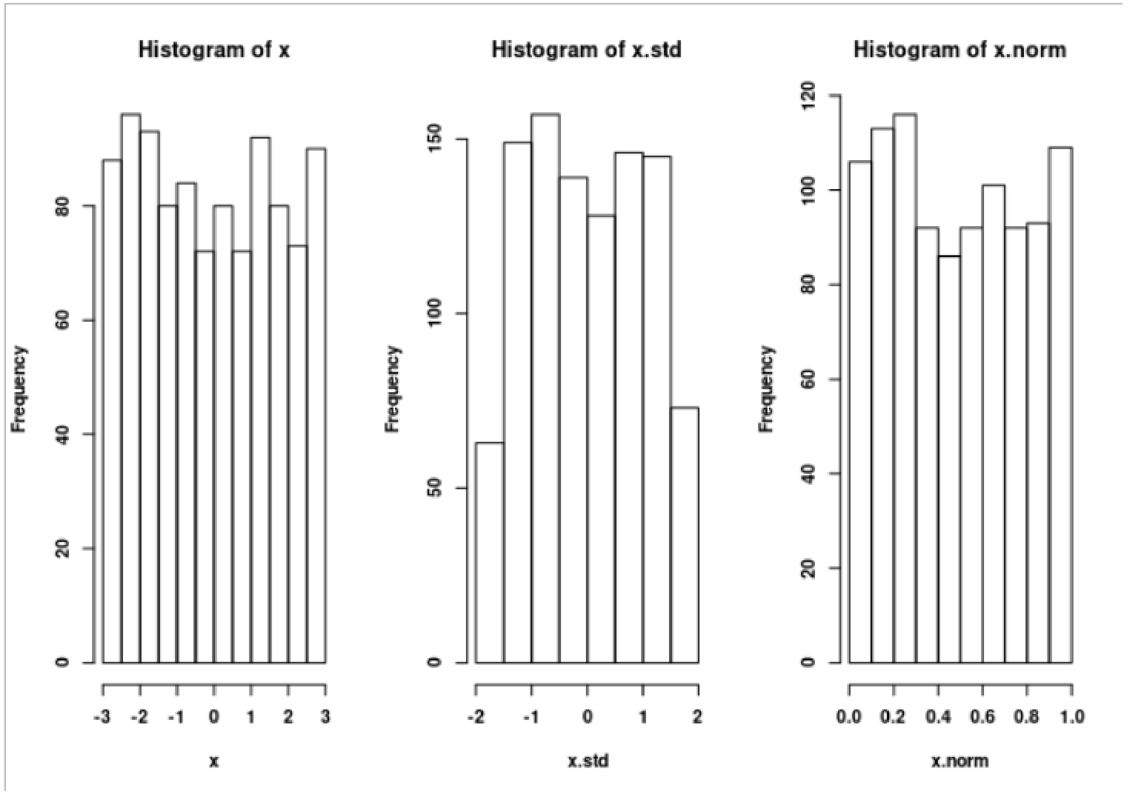
Brings data onto a normal distribution with with μ=0 and σ=1

Normal data

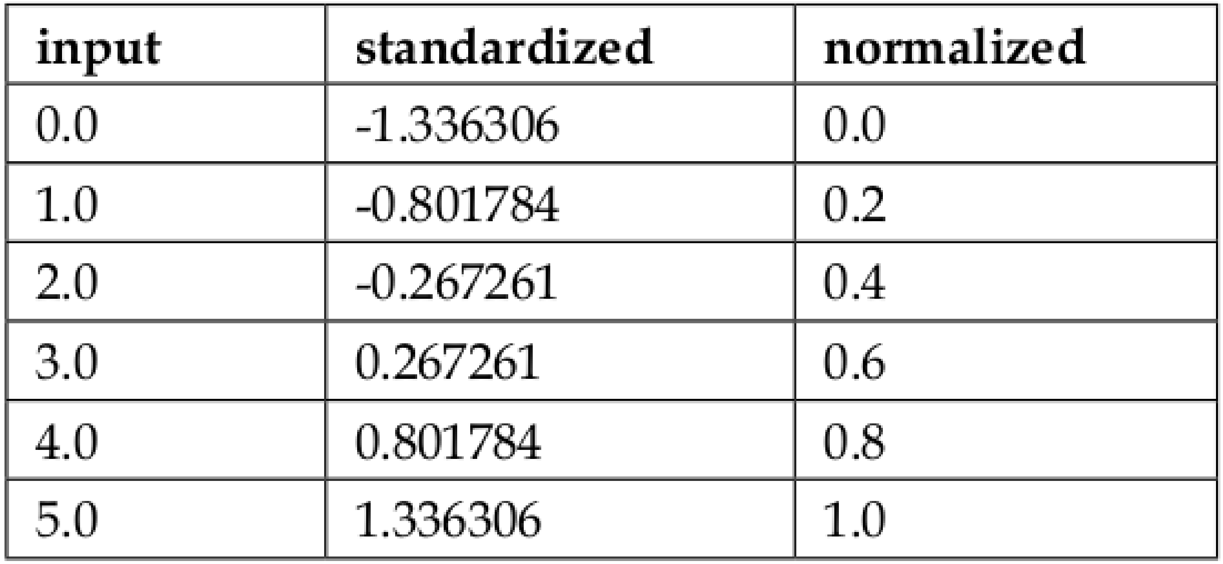


$\mu = 0; \sigma = 1$

Non-Normal data

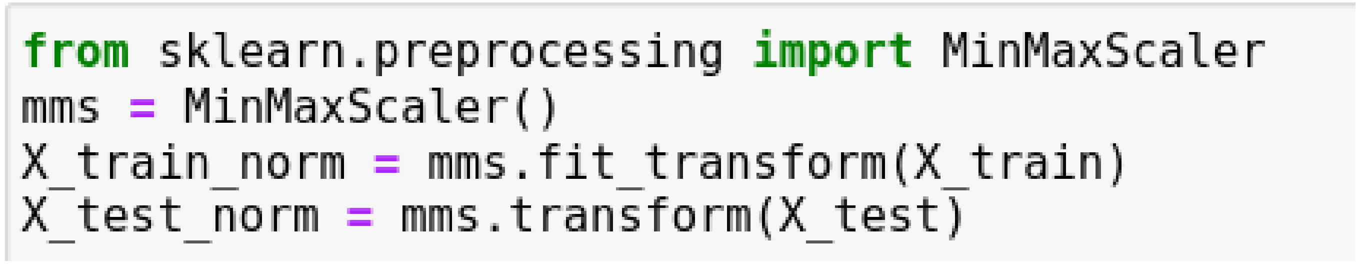


$\mu = 0; \sigma = 1$

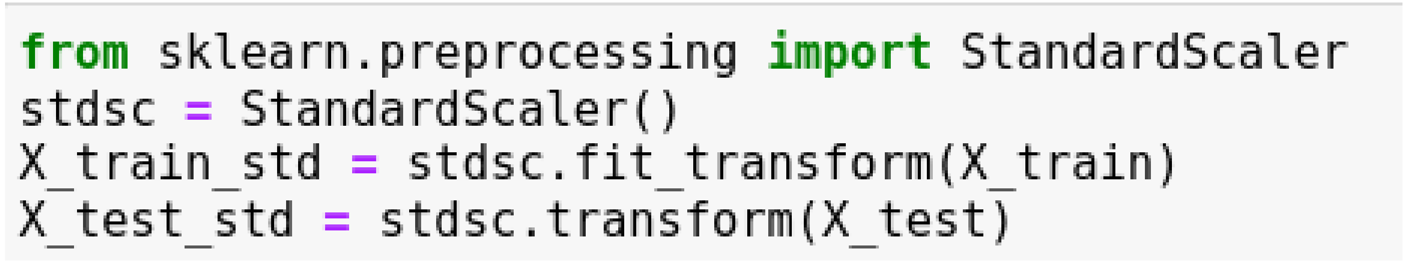


input	standardized	normalized
0.0	-1.336306	0.0
1.0	-0.801784	0.2
2.0	-0.267261	0.4
3.0	0.267261	0.6
4.0	0.801784	0.8
5.0	1.336306	1.0

(p. 111: Raschka, 2015)

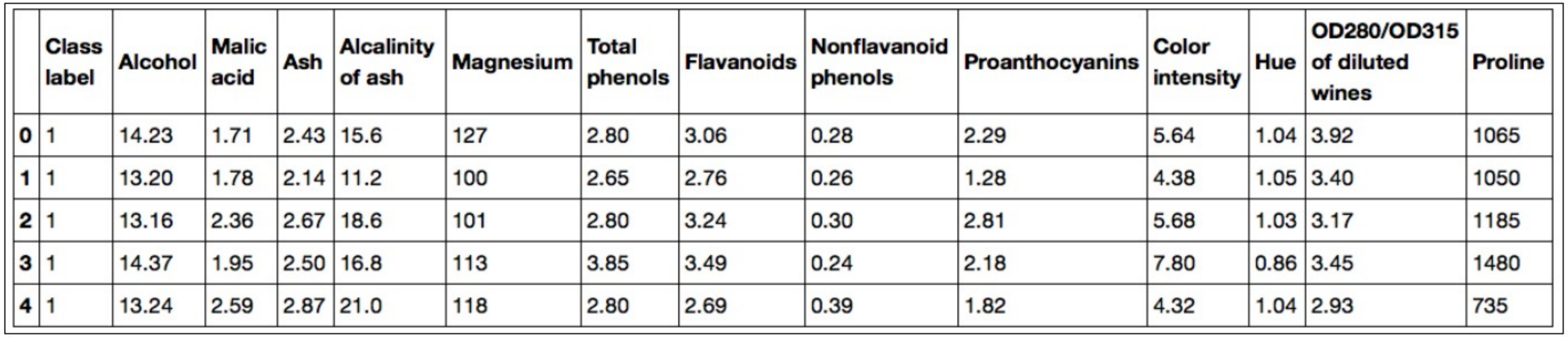


```python
from sklearn.preprocessing import MinMaxScaler
mms = MinMaxScaler()
X_train_norm = mms.fit_transform(X_train)
X_test_norm = mms.transform(X_test)
```



```python
from sklearn.preprocessing import StandardScaler
stdsc = StandardScaler()
X_train_std = stdsc.fit_transform(X_train)
X_test_std = stdsc.transform(X_test)
```

Wine dataset



	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wines	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

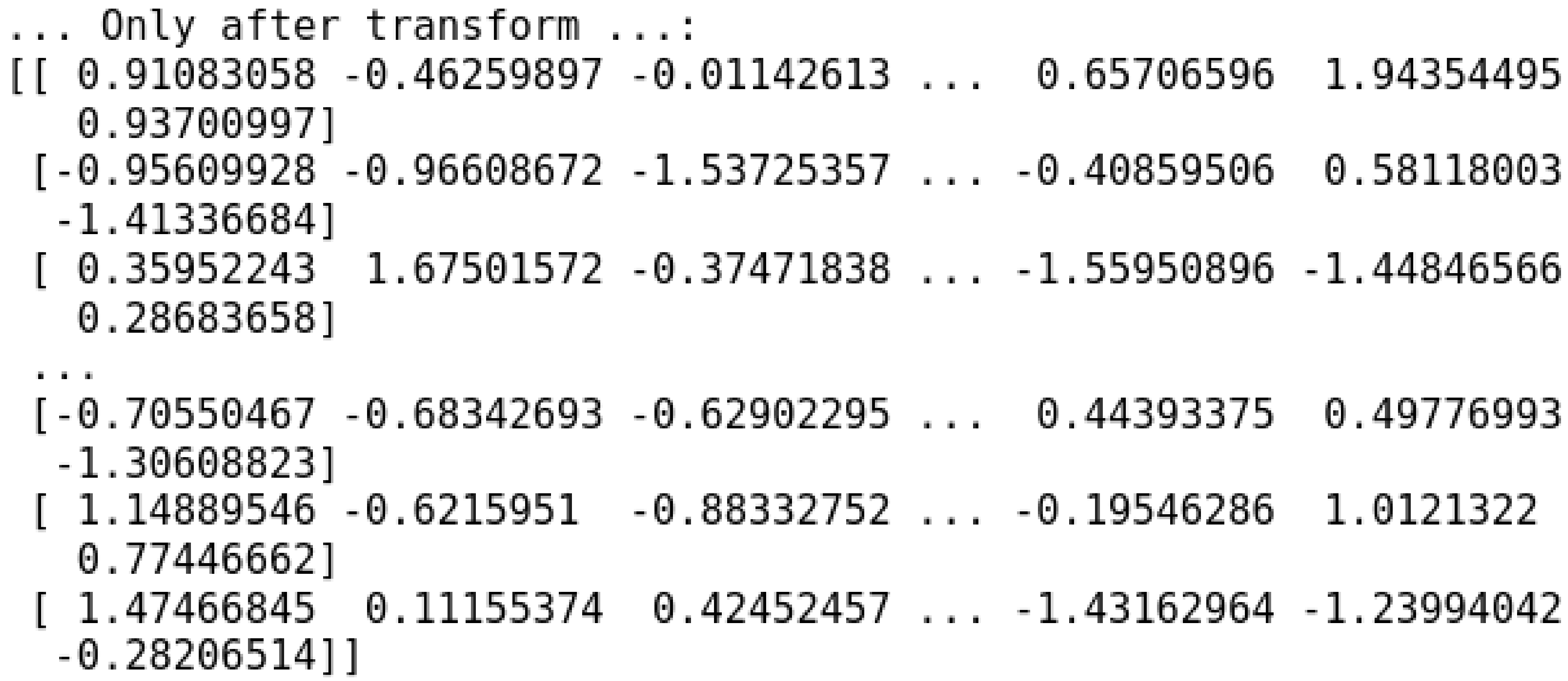
(p. 109: Raschka, 2015)

```
Original data:
[[1.371e+01 1.860e+00 2.360e+00 ... 1.110e+00 4.000e+00 1.035e+03]
 [1.222e+01 1.290e+00 1.940e+00 ... 8.600e-01 3.020e+00 3.120e+02]
 [1.327e+01 4.280e+00 2.260e+00 ... 5.900e-01 1.560e+00 8.350e+02]
 ...
 [1.242e+01 1.610e+00 2.190e+00 ... 1.060e+00 2.960e+00 3.450e+02]
 [1.390e+01 1.680e+00 2.120e+00 ... 9.100e-01 3.330e+00 9.850e+02]
 [1.416e+01 2.510e+00 2.480e+00 ... 6.200e-01 1.710e+00 6.600e+02]]
```

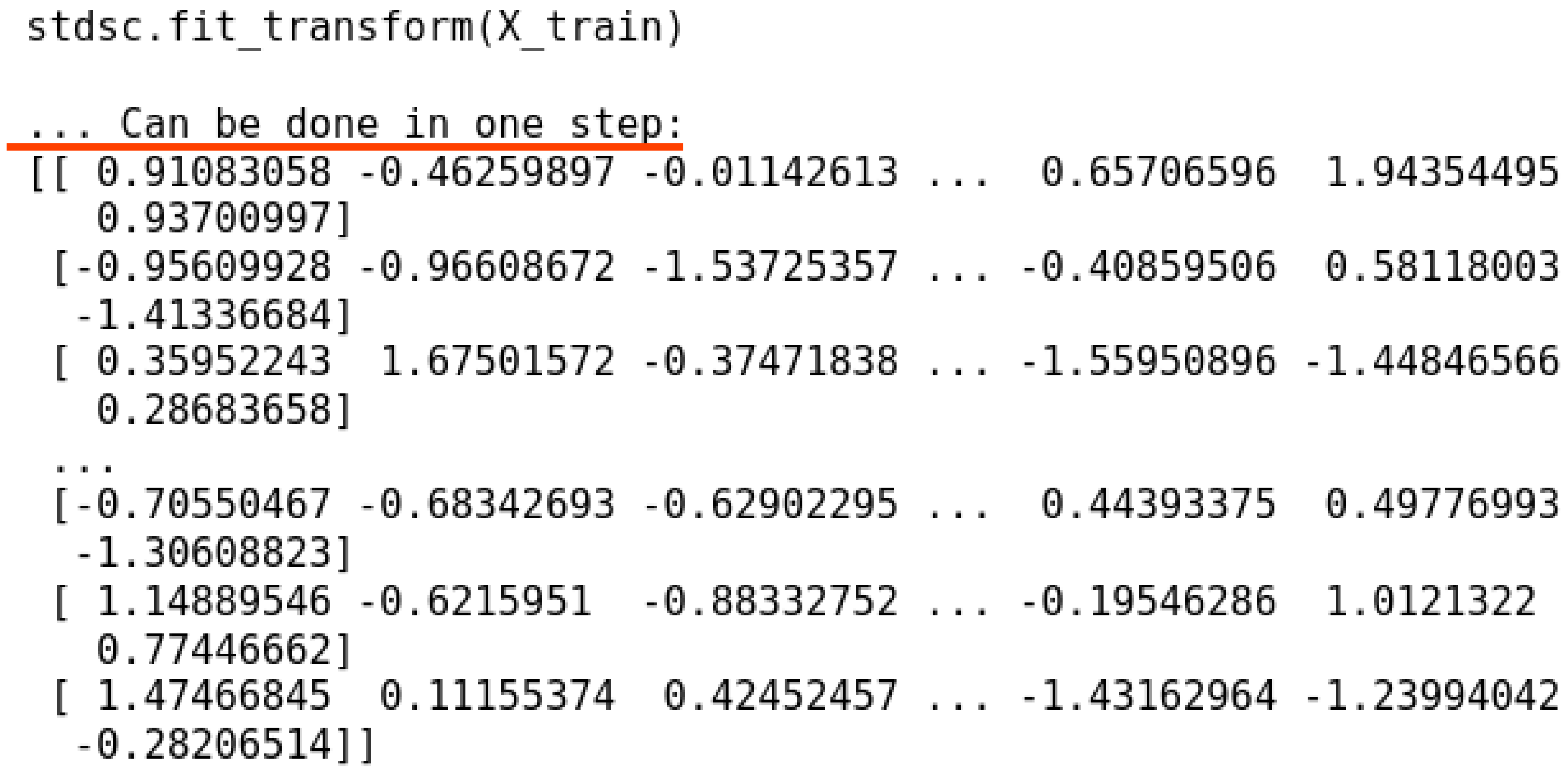
```
stdsc.fit(X_train)
```

```
Fitted mean:
[1.29830645e+01 2.38370968e+00 2.36314516e+00 1.95258065e+01
 1.00088710e+02 2.25838710e+00 1.96951613e+00 3.64274194e-01
 1.61250000e+00 4.99991935e+00 9.55854839e-01 2.60193548e+00
 7.46766129e+02]

Fitted variance:
[6.36966415e-01 1.28165721e+00 7.57683338e-02 1.27099792e+01
 2.13161485e+02 3.57332882e-01 9.46920734e-01 1.53099571e-02
 3.55007460e-01 5.48412330e+00 5.50353660e-02 5.17446254e-01
 9.46240663e+04]

Nothing happens to data after fit...:
[[1.371e+01 1.860e+00 2.360e+00 ... 1.110e+00 4.000e+00 1.035e+03]
 [1.222e+01 1.290e+00 1.940e+00 ... 8.600e-01 3.020e+00 3.120e+02]
 [1.327e+01 4.280e+00 2.260e+00 ... 5.900e-01 1.560e+00 8.350e+02]
 ...
 [1.242e+01 1.610e+00 2.190e+00 ... 1.060e+00 2.960e+00 3.450e+02]
 [1.390e+01 1.680e+00 2.120e+00 ... 9.100e-01 3.330e+00 9.850e+02]
 [1.416e+01 2.510e+00 2.480e+00 ... 6.200e-01 1.710e+00 6.600e+02]]
```

stdsc.transform(X_train)

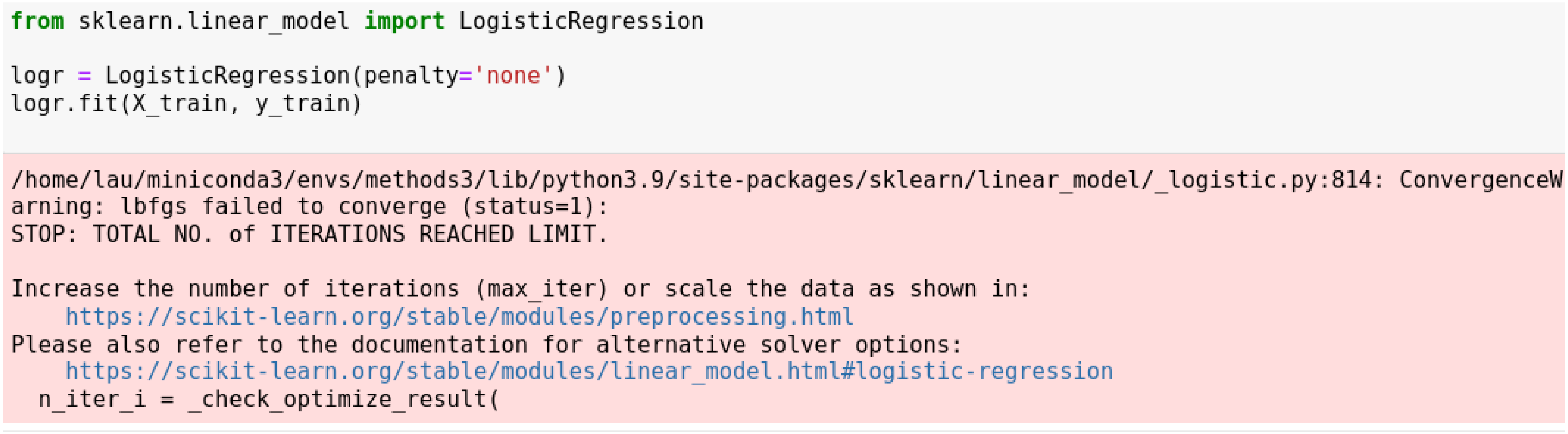


```
... Only after transform ...:
[[ 0.91083058 -0.46259897 -0.01142613 ...  0.65706596  1.94354495
   0.93700997]
 [-0.95609928 -0.96608672 -1.53725357 ... -0.40859506  0.58118003
  -1.41336684]
 [ 0.35952243  1.67501572 -0.37471838 ... -1.55950896 -1.44846566
   0.28683658]
 ...
 [-0.70550467 -0.68342693 -0.62902295 ...  0.44393375  0.49776993
  -1.30608823]
 [ 1.14889546 -0.6215951  -0.88332752 ... -0.19546286  1.0121322
   0.77446662]
 [ 1.47466845  0.11155374  0.42452457 ... -1.43162964 -1.23994042
  -0.28206514]]
```



```
stdsc.fit_transform(X_train)

... Can be done in one step:
[[ 0.91083058 -0.46259897 -0.01142613 ...  0.65706596  1.94354495
   0.93700997]
 [-0.95609928 -0.96608672 -1.53725357 ... -0.40859506  0.58118003
  -1.41336684]
 [ 0.35952243  1.67501572 -0.37471838 ... -1.55950896 -1.44846566
   0.28683658]
 ...
 [-0.70550467 -0.68342693 -0.62902295 ...  0.44393375  0.49776993
  -1.30608823]
 [ 1.14889546 -0.6215951  -0.88332752 ... -0.19546286  1.0121322
   0.77446662]
 [ 1.47466845  0.11155374  0.42452457 ... -1.43162964 -1.23994042
  -0.28206514]]
```

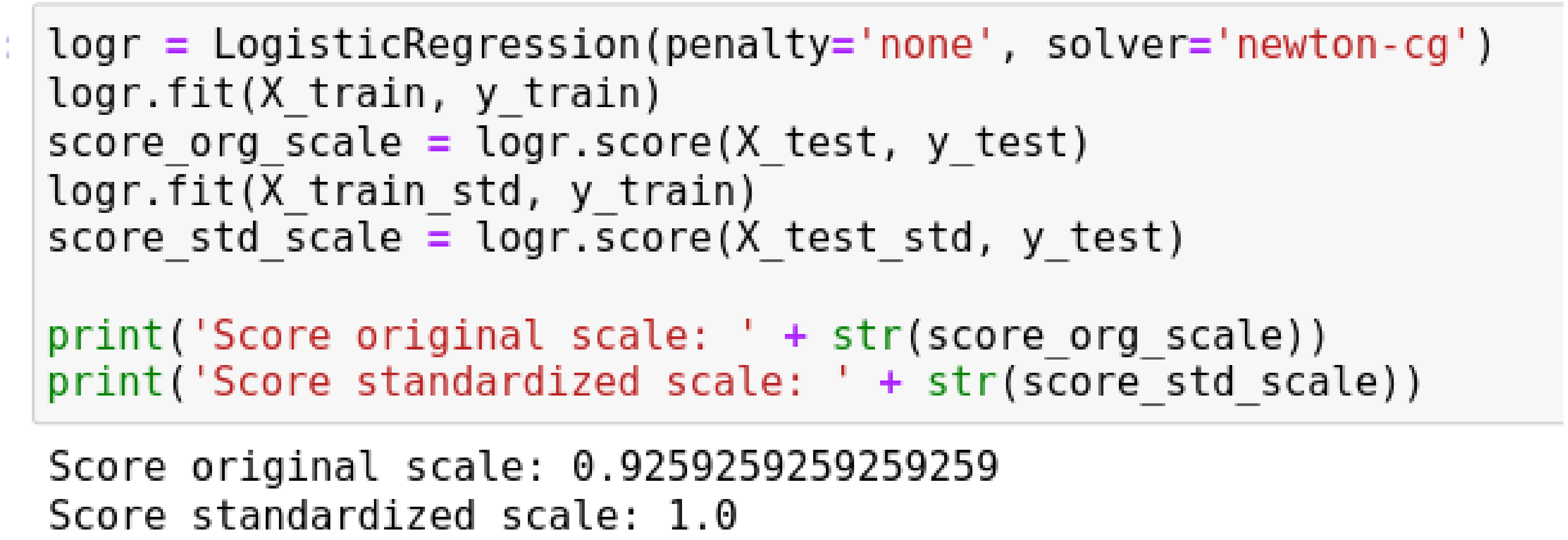
Why scaling - convergence



```
from sklearn.linear_model import LogisticRegression

logr = LogisticRegression(penalty='none')
logr.fit(X_train, y_train)
```

```
/home/lau/miniconda3/envs/methods3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: ConvergenceW
arning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

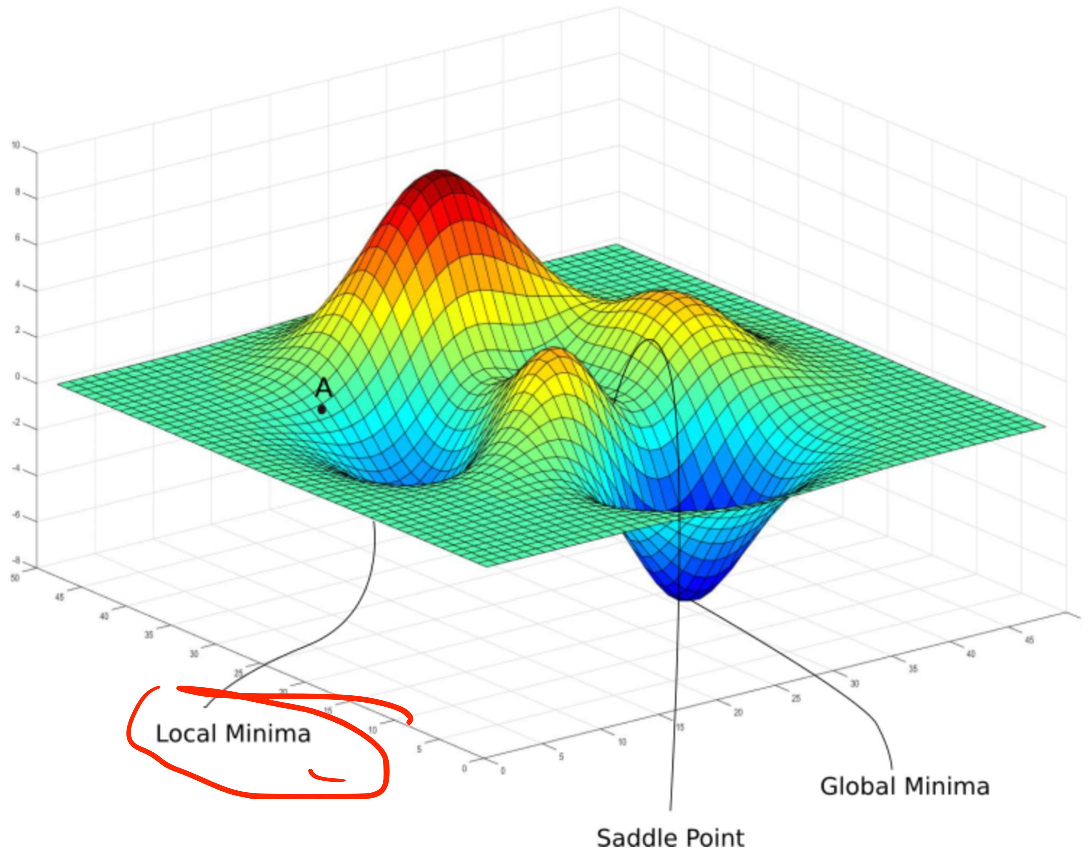
Why scaling - accuracy



```
logr = LogisticRegression(penalty='none', solver='newton-cg')
logr.fit(X_train, y_train)
score_org_scale = logr.score(X_test, y_test)
logr.fit(X_train_std, y_train)
score_std_scale = logr.score(X_test_std, y_test)

print('Score original scale: ' + str(score_org_scale))
print('Score standardized scale: ' + str(score_std_scale))
```

```
Score original scale: 0.9259259259259259
Score standardized scale: 1.0
```

Why scaling - accuracy



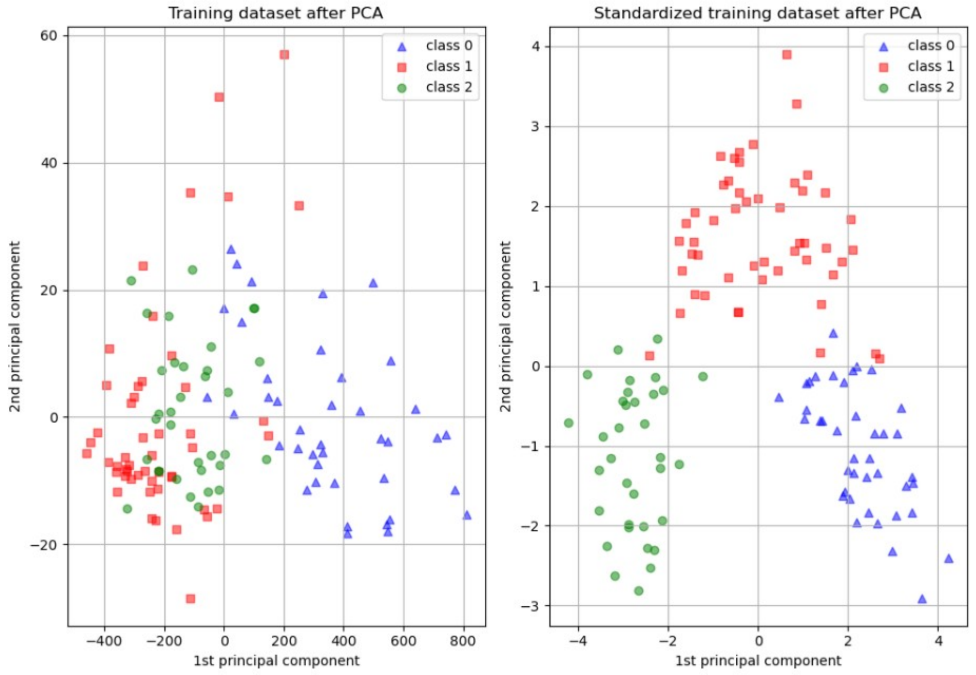
if not scaled, variables with high values will dominate, and we might end in local minima

https://wngaw.github.io/linear-regression/

Why scaling – *lmer* and *glmer*

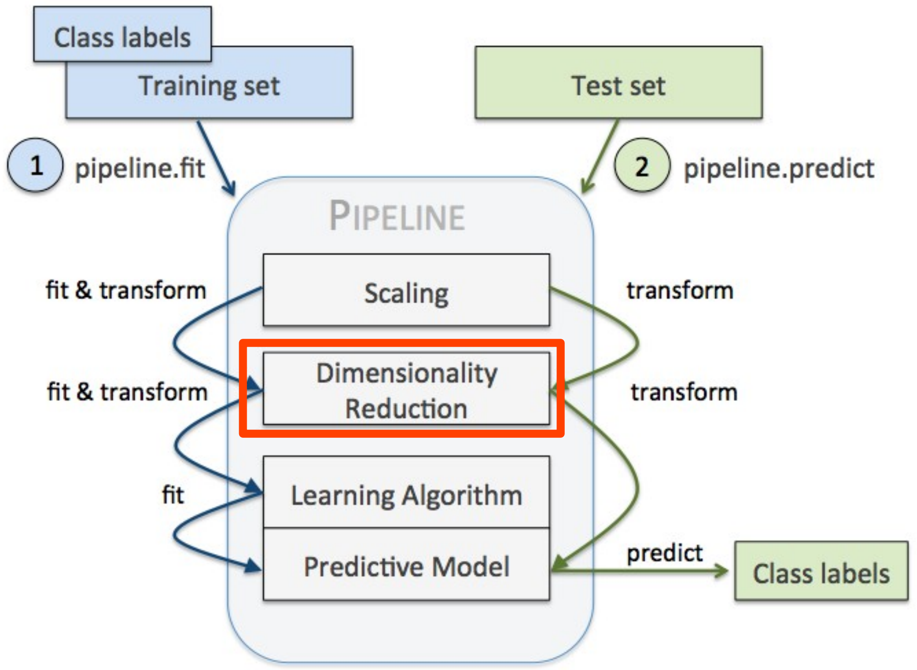
Can also help convergence (also fits via gradient descent)

Also important for PCA



https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html#sphx-glr-auto-examples-preprocessing-plot-scaling-importance-py

Pipeline example



(p. 172: Raschka, 2015)

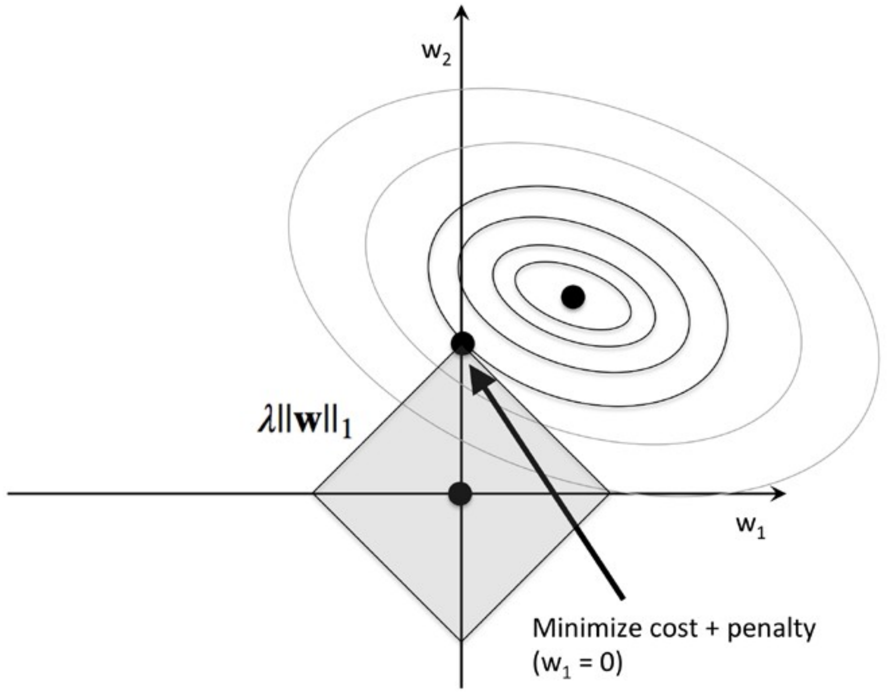
Dimensionality reduction

- Feature selection
  - Choose a subset of the original features
  - L1 regularisation is an example

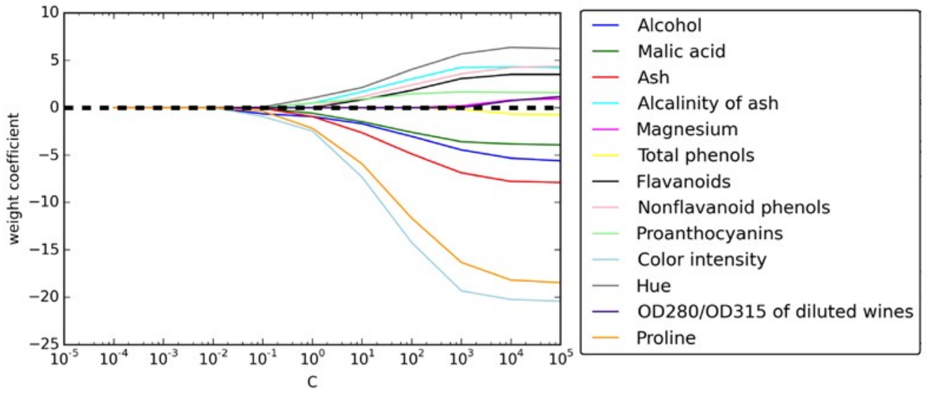
- Feature extraction
  - Create a new feature subspace based on derived information
  - Principal component analysis is an example

Feature selection example

L1 REGULARISATION



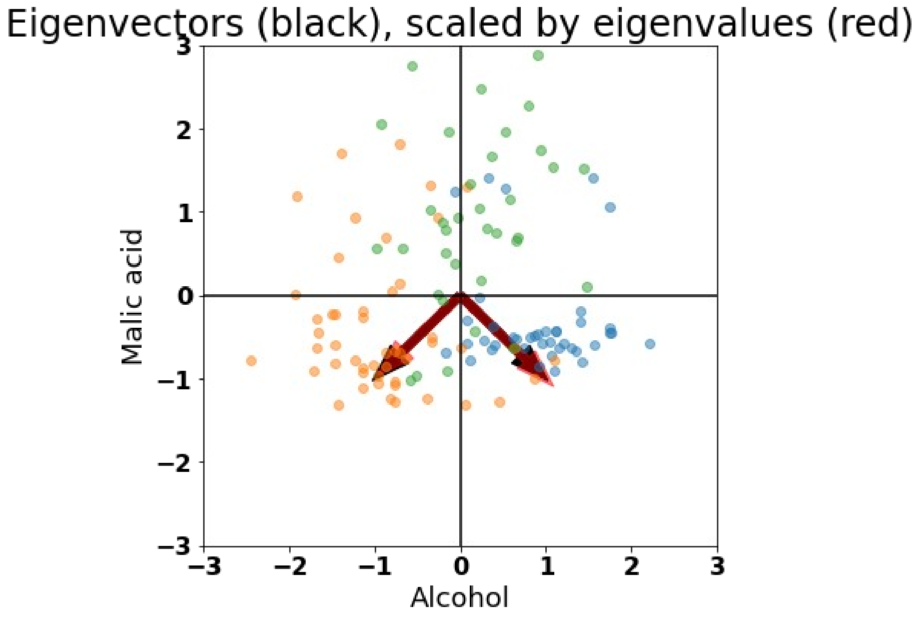
(p. 115: Raschka, 2015)

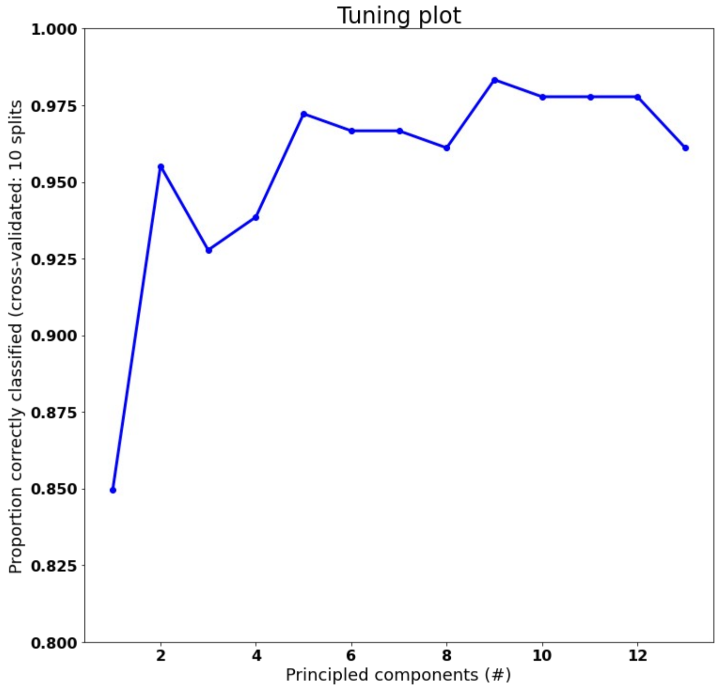


(p. 118: Raschka, 2015)

Feature extraction example

PRINCIPLED COMPONENT ANALYSIS



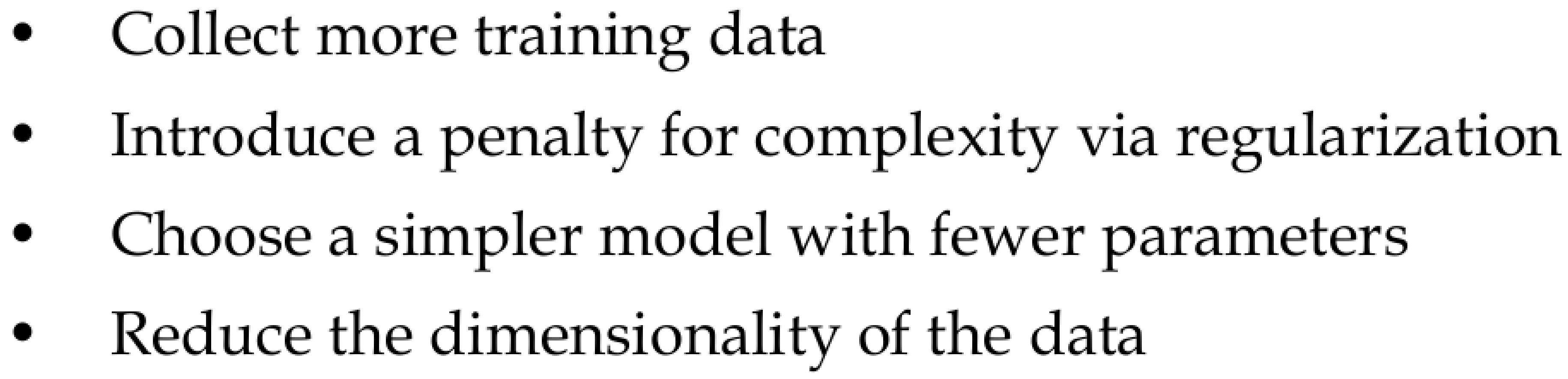


Summary so far:

Scaling makes sure that we get reliable fits and treat each variable equally

Dimensionality reduction makes sure that we are not overfitting

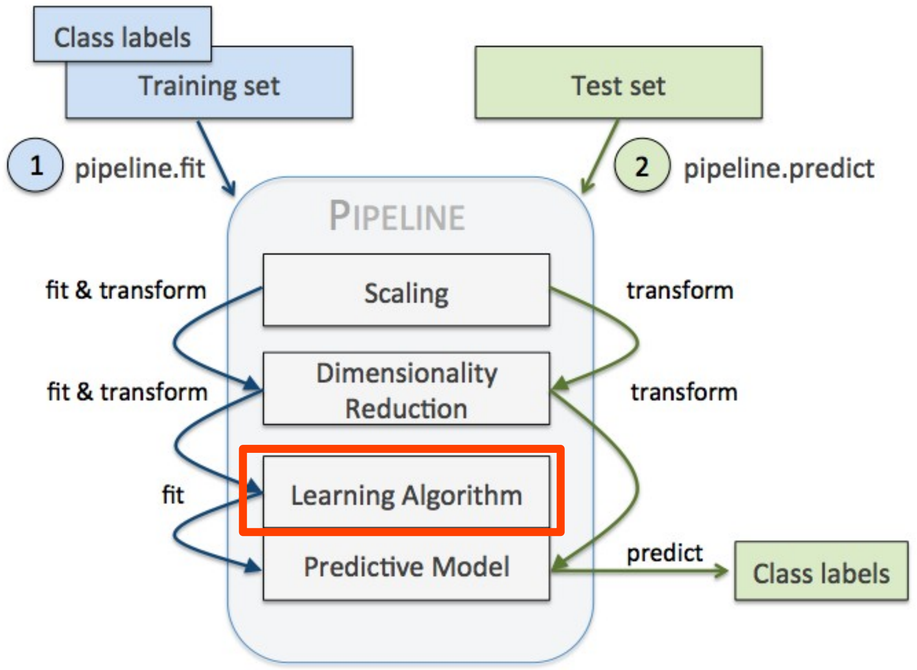
Ways of preventing overfitting



- Collect more training data
- Introduce a penalty for complexity via regularization
- Choose a simpler model with fewer parameters
- Reduce the dimensionality of the data

(p. 112: Raschka, 2015)

Pipeline example



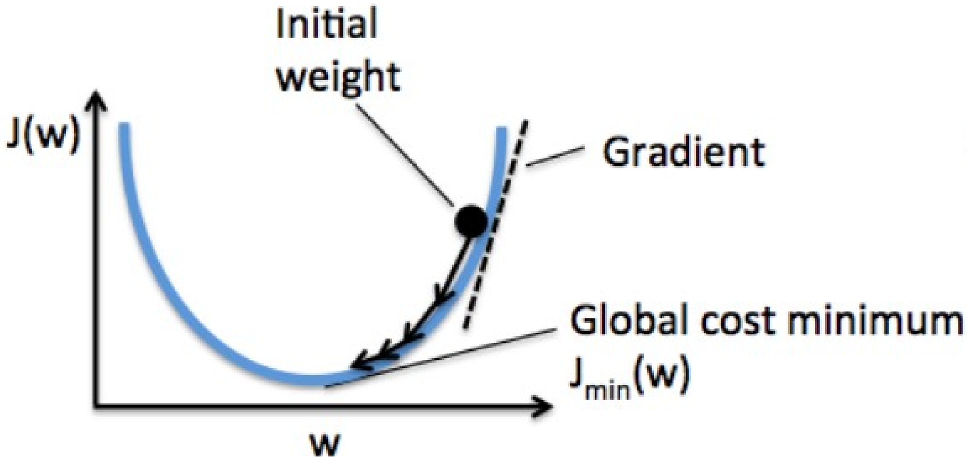
(p. 172: Raschka, 2015)

Learning algorithm examples

GRADIENT DESCENT

$$\Delta \boldsymbol{w} = -\eta \nabla J(\boldsymbol{w})$$

A general formulation for linear models: $\Delta w_j = \eta \sum_i^n \left(y^{(i)} - \phi\left(z^{(i)}\right)\right) x_j^{(i)}$



(p. 40: Raschka, 2015)

Learning algorithm examples

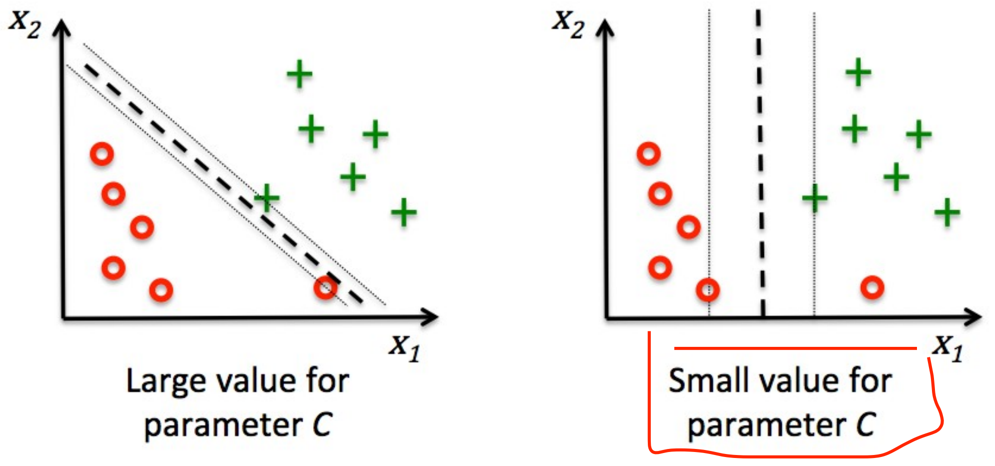
SUPPORT VECTOR MACHINE

minimize: $\frac{1}{2}\|\boldsymbol{w}\|^2 + C(\sum_{i}^{n} \xi^{(i)})$

under the constraints:

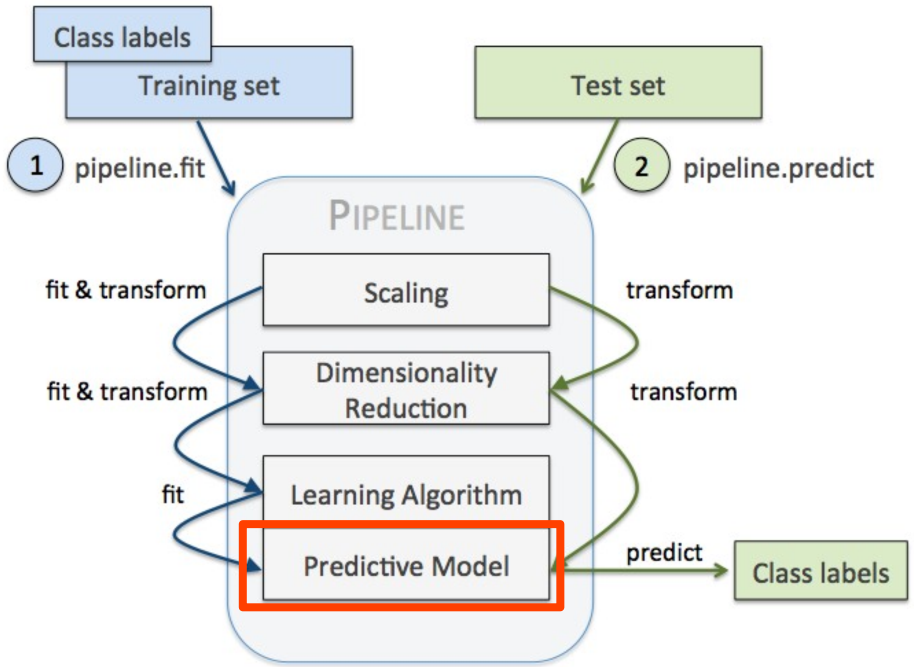
$$w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} \geq 1 \; if \; y^{(i)} = 1$$
$$w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} < -1 \; if \; y^{(i)} = -1$$

better classification of data



(p. 72: Raschka, 2015)

Pipeline example



(p. 172: Raschka, 2015)

Predictive model

LOGISTIC REGRESSION

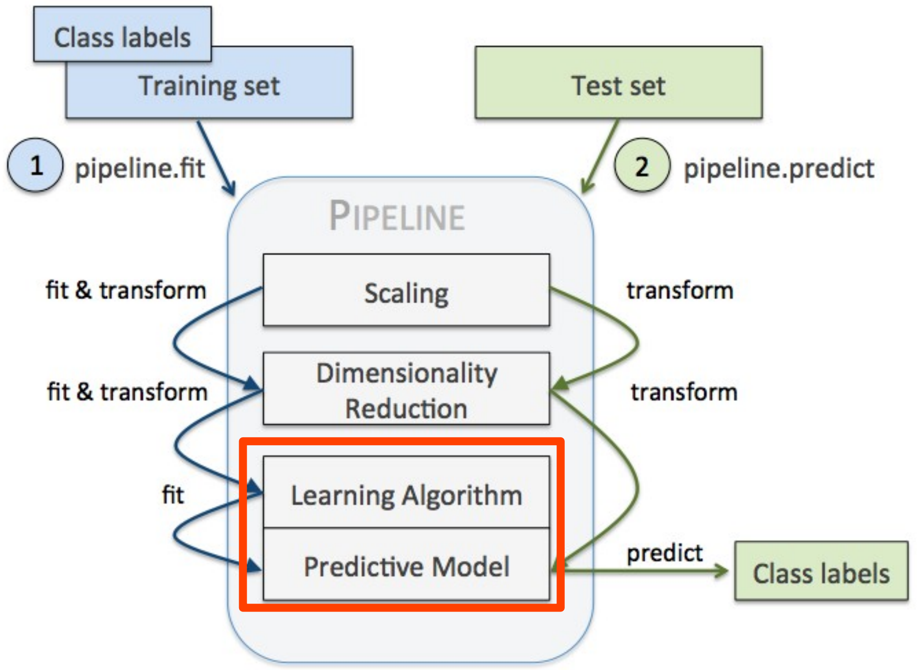
$$\hat{y} = \left\{ \begin{array}{l} 1 \text{ if } z \geq 0.0 \\ 0 \text{ otherwise} \end{array} \right\}$$

Predictive model

LINEAR REGRESSION

$$\hat{\boldsymbol{y}} = \boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x} = w_0 x_0 + w_1 x_1 + \ldots + w_{m-1} x_{m-1} + w_m x_m$$

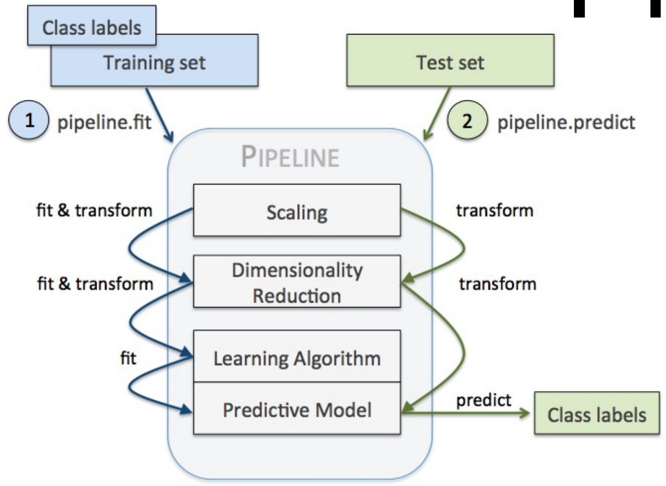
Pipeline example



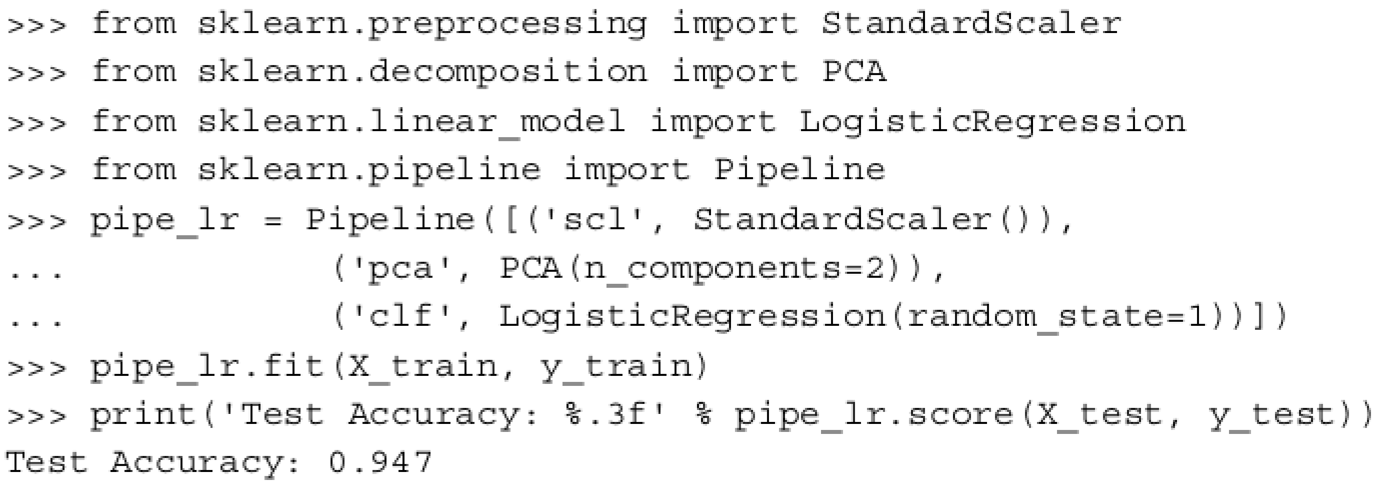
These in practice go together

(p. 172: Raschka, 2015)

Pipeline example



(p. 172: Raschka, 2015)



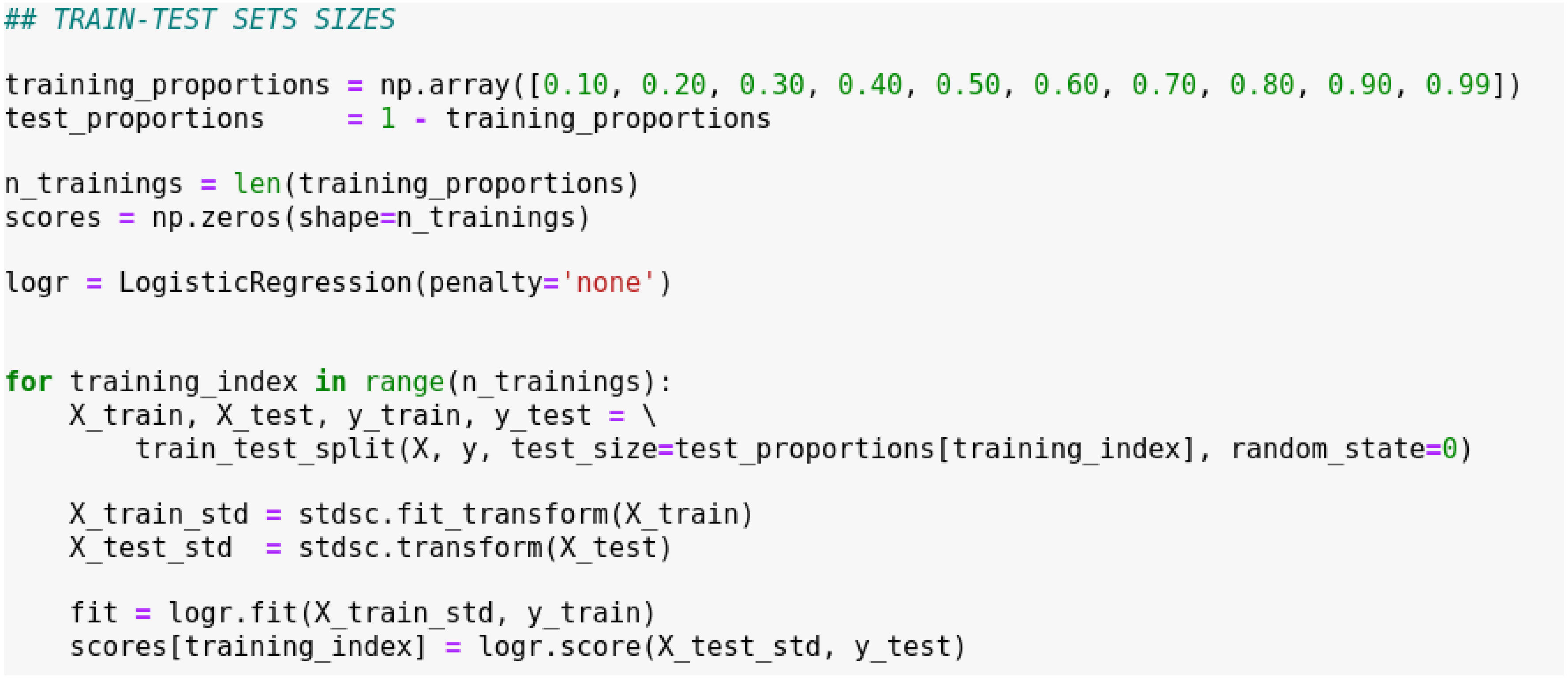
```
>>> from sklearn.preprocessing import StandardScaler
>>> from sklearn.decomposition import PCA
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.pipeline import Pipeline
>>> pipe_lr = Pipeline([('scl', StandardScaler()),
...                     ('pca', PCA(n_components=2)),
...                     ('clf', LogisticRegression(random_state=1))])
>>> pipe_lr.fit(X_train, y_train)
>>> print('Test Accuracy: %.3f' % pipe_lr.score(X_test, y_test))
Test Accuracy: 0.947
```

(p. 171: Raschka, 2015)

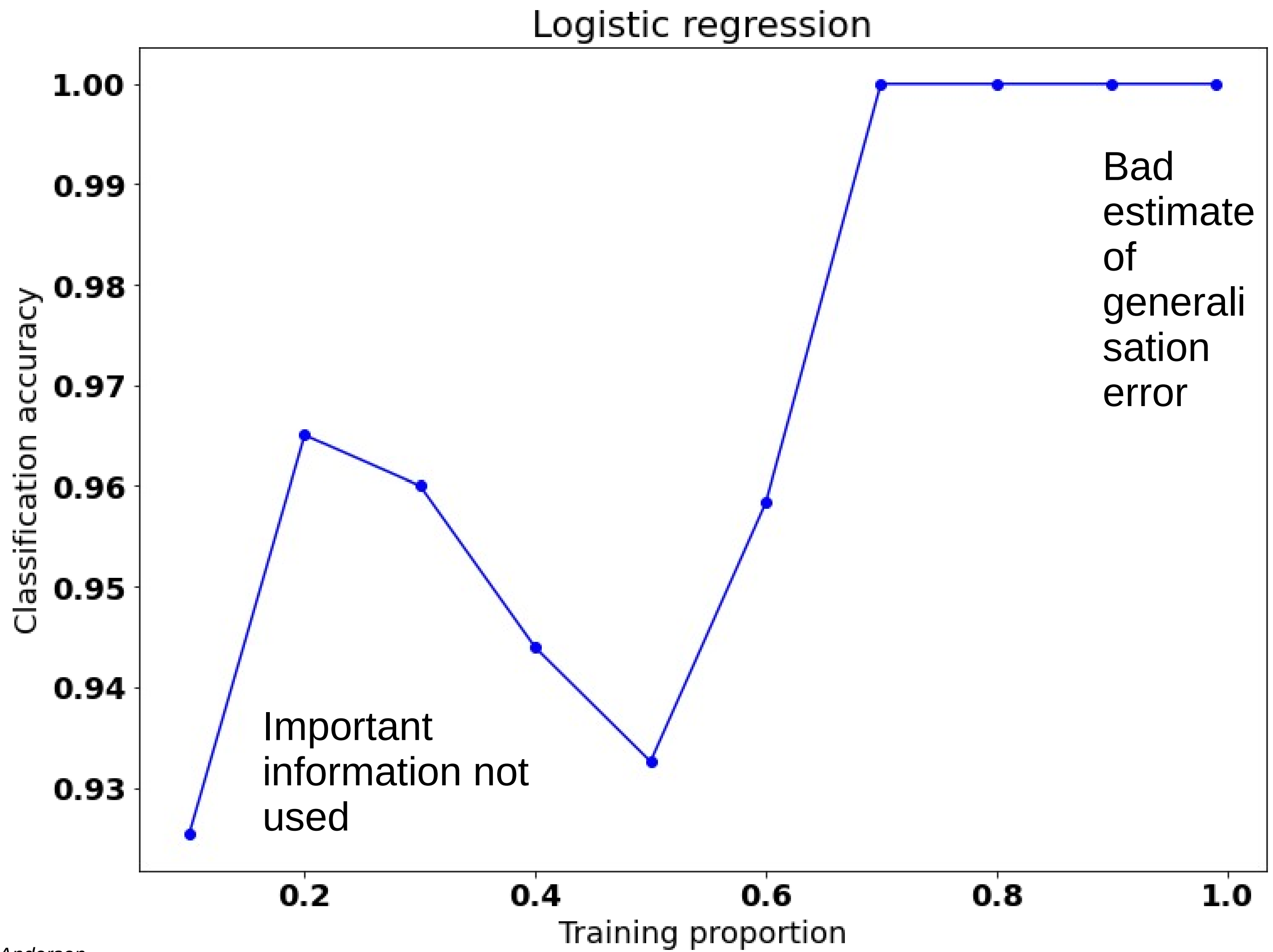
Read more in chapter 6 – you know all the ingredients – now it is just setting the recipe

VALIDATING YOUR MODEL
Dividing into training and test sets

Effects of train-test sizes



```python
## TRAIN-TEST SETS SIZES

training_proportions = np.array([0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 0.99])
test_proportions     = 1 - training_proportions

n_trainings = len(training_proportions)
scores = np.zeros(shape=n_trainings)

logr = LogisticRegression(penalty='none')


for training_index in range(n_trainings):
    X_train, X_test, y_train, y_test = \
        train_test_split(X, y, test_size=test_proportions[training_index], random_state=0)

    X_train_std = stdsc.fit_transform(X_train)
    X_test_std  = stdsc.transform(X_test)

    fit = logr.fit(X_train_std, y_train)
    scores[training_index] = logr.score(X_test_std, y_test)
```

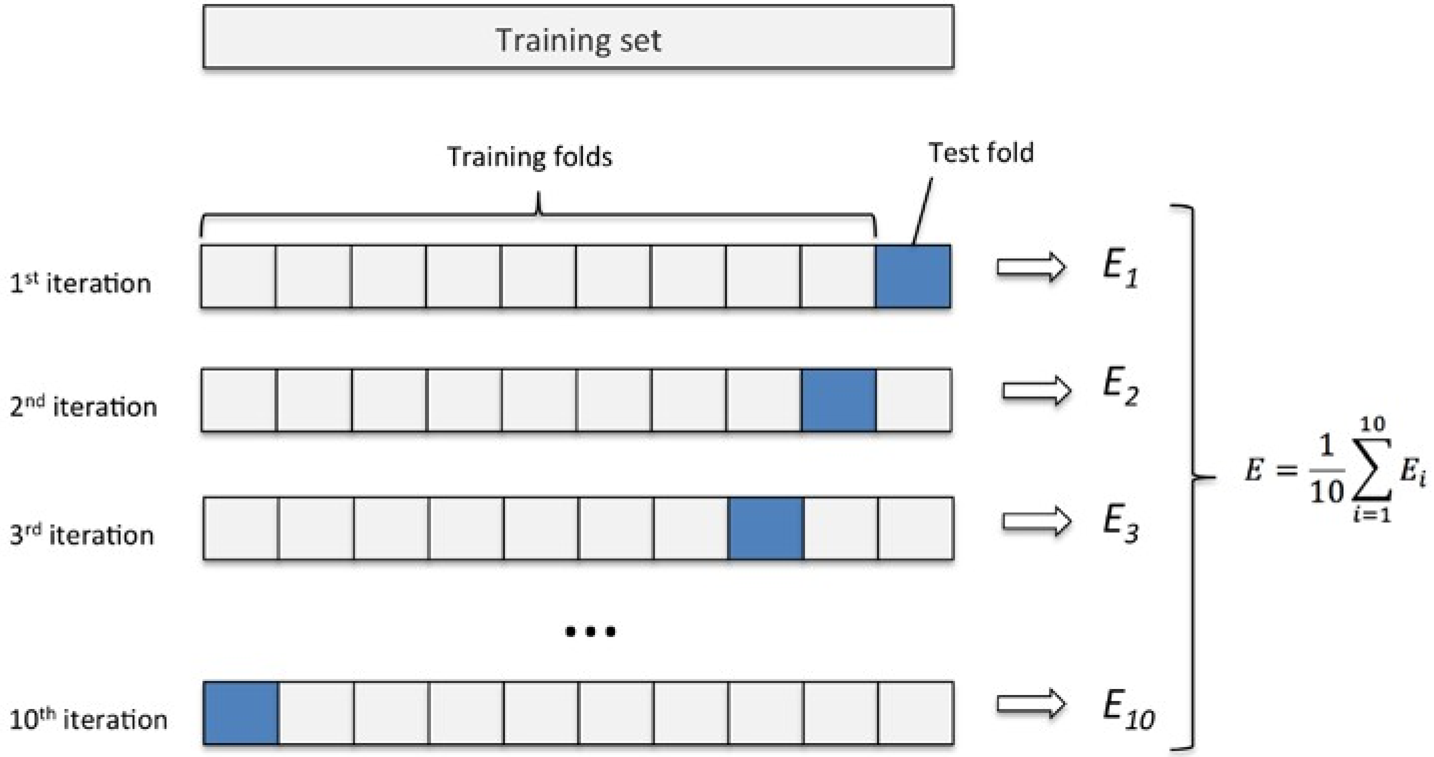


Bad estimate of generalisation error

Important information not used

With large enough datasets, small-proportioned test sets less of a problem

Also less of a problem with *K*-fold cross-validation



(p. 176: Raschka, 2015)

Stratified *K*-fold cross-validation

Makes sure that each fold contains an equal number of labels such that the classifier is not biased towards a set of labels

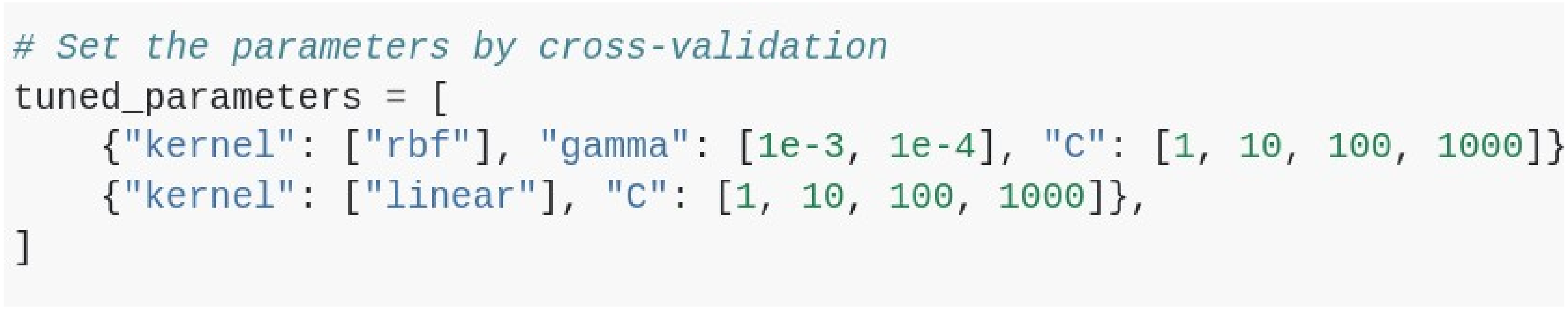
Two types of parameters

- Parameters learnt from data
  - e.g. weights in regression types

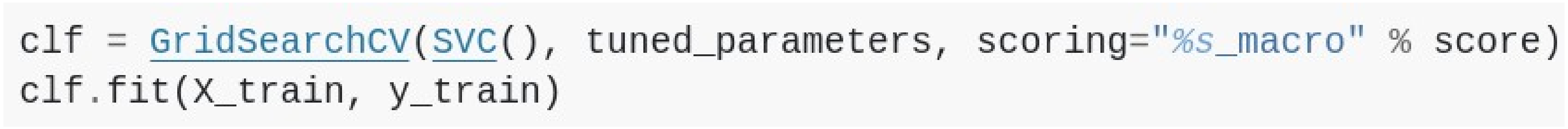
- Tuning parameters
  - or hyperparameters -> determines how weights are set
  - e.g. *C* and *γ*

so tuning parameter/hyper parameters are determining how weights are set (our parameters learnt from data)
Hvis jeg forstår dette rigtigt, så er hyperparameters en måde at justere vores predictive betas (parameters learned from data), i.e en måde at justere vores model (tune den/ finpudse den)

Grid search



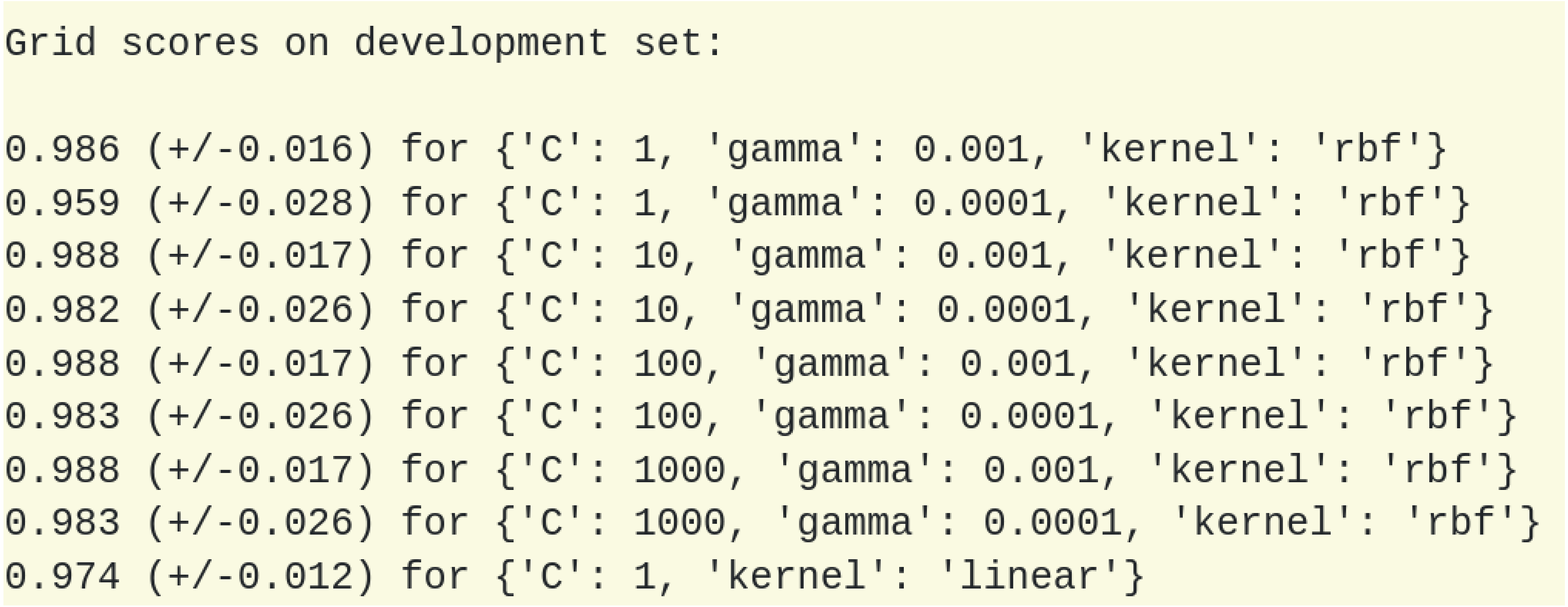
```
# Set the parameters by cross-validation
tuned_parameters = [
    {"kernel": ["rbf"], "gamma": [1e-3, 1e-4], "C": [1, 10, 100, 1000]}
    {"kernel": ["linear"], "C": [1, 10, 100, 1000]},
]
```



```
clf = GridSearchCV(SVC(), tuned_parameters, scoring="%s_macro" % score)
clf.fit(X_train, y_train)
```

https://scikit-learn.org/stable/auto_examples/model_selection/plot_grid_search_digits.html#sphx-glr-auto-examples-model-selection-plot-grid-search-digits-py

Grid search



```
Grid scores on development set:

0.986 (+/-0.016) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.959 (+/-0.028) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.988 (+/-0.017) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.982 (+/-0.026) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.988 (+/-0.017) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.983 (+/-0.026) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.988 (+/-0.017) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.983 (+/-0.026) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.974 (+/-0.012) for {'C': 1, 'kernel': 'linear'}
```

https://scikit-learn.org/stable/auto_examples/model_selection/plot_grid_search_digits.html#sphx-glr-auto-examples-model-selection-plot-grid-search-digits-py

Grid search

Read more in chapter 6 – you know all the ingredients – now it is just setting the recipe

Did you learn?

Organising and preprocessing messy data

1) Learning why data should be scaled
2) Using feature selection to avoid overfitting
3) Understanding the basic steps of a machine learning pipeline

OPTIONAL:
SUPPORT VECTOR MACHINES AND KERNELS

Based on chapter 2:

Abe, S., 2010. Support Vector Machines for Pattern Classification. Springer, London.

Kernels can be used to not explicitly go into higher-dimensional space

$\boldsymbol{x}_{n\times m}$: observations and predictor variables (features)

$\boldsymbol{\phi}(\boldsymbol{x})=(\phi_1(\boldsymbol{x}),\ldots,\phi_l(\boldsymbol{x}))^T$: mapping from m dimensions to l dimensions $(l>m)$

$K(\boldsymbol{x},\boldsymbol{x}')=\boldsymbol{\phi}^{\boldsymbol{T}}(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x}')$: the Kernel function means that we need not explicitly go into higher-dimensional space

Different kernels

Linear kernel

$$K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^{\boldsymbol{T}} \boldsymbol{x}'$$

Polynomial kernel

$$K(\boldsymbol{x}^{\boldsymbol{T}} \boldsymbol{x}' + 1)^d$$

Radial basis function kernel

$$K(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\gamma \|\boldsymbol{x} - \boldsymbol{x}'\|^2\right)$$

K can be rewritten as:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\gamma \|\boldsymbol{x}\|^2) \exp(-\gamma \|\boldsymbol{x}'\|^2) \exp(2\gamma \boldsymbol{x}^{\boldsymbol{T}} \boldsymbol{x}')$$

$$\exp(2\gamma \boldsymbol{x}^{\boldsymbol{T}} \boldsymbol{x}') = 1 + 2\gamma \boldsymbol{x}^{\boldsymbol{T}} \boldsymbol{x}' + 2\gamma^2 (\boldsymbol{x}^{\boldsymbol{T}} \boldsymbol{x}')^2 + \frac{(2\gamma)^3}{3!} (\boldsymbol{x}^{\boldsymbol{T}} \boldsymbol{x}')^3 + \ldots, \text{(an infinite sum)}$$

Thus the Kernel function contains all possible polynomials and can thus represent non-linear problems

Decision functions

$$D(\boldsymbol{x}) = \sum_{i \in S} \alpha_i y_i \exp\left(-\gamma \left\| \boldsymbol{x_i} - \boldsymbol{x} \right\|^2\right) + b$$

$\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_M)^T$ are non-negative Lagrange multipliers
(Lagrange multipliers are used to find local extrema)
b is a bias (a constant)

$$\boldsymbol{x} \in \begin{Bmatrix} \text{Class 1 if } D(\boldsymbol{x}) > 0 \\ \text{Class 2 if } D(\boldsymbol{x}) < 0 \end{Bmatrix}$$

Minimisation problem

minimise $Q(\boldsymbol{w}, b) = \frac{1}{2}\|\boldsymbol{w}\|^2$

subject to $y_i(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}_{\boldsymbol{i}} + b) \geq 1$ for $i = 1, \ldots, M$

unconstrained version

$$Q(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{w} - \sum_{i=1}^{M} \alpha_i\{y_i(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}_{\boldsymbol{i}} + b) - 1\}$$

$\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_M)^T$

$M : m$-dimensional training inputs $\boldsymbol{x}_{\boldsymbol{i}}(i = 1, \ldots, M)$

References

- Abe, S., 2010. Support Vector Machines for Pattern Classification. Springer, London.
- Raschka, S., 2015. Python Machine Learning. Packt Publishing Ltd.