

Methods 3: Multilevel Statistical Modeling and Machine Learning

Week 8: Model evaluation and hyperparameter tuning

October 29, 2024

The course plan

Week 1: Introduction

Instructor sessions: *Setting up R and Python and recollection of the general linear model*

Week 2: Multilevel linear regression

Instructor sessions: *Modelling subject level effects – and how do they differ from group level effects?*

Week 3: Link functions and fitting generalised linear multilevel models

Instructor sessions: *What to do when the response variable is not continuous?*

Week 4: Evaluating Generalised linear mixed models

Instructor sessions: *How do we assess how models compare to one another?*

Week 5: Explanation and Prediction

Instructor sessions: *Code review*

Week 6: Mid-way evaluation and Machine Learning Intro

Instructor sessions: *Getting Python Running*

Week 7: Linear and logistic regression revisited (machine learning)

Instructor sessions: *Categorizing responses based on informed guesses*

Week 8: Model evaluation and hyperparameter tuning

Instructor sessions: *Code review*

Week 9: Dimensionality Reduction, Principled Component Analysis (PCA)

Instructor sessions: *What to do with very rich data?*

Week 10: Outlook, unsupervised classification and neural networks

Instructor sessions: *Data with no labels and networks*

Week 11: Organising and preprocessing messy data

Instructor sessions: *Code review*

Week 12: Final evaluation and wrap-up of course

Instructor sessions: *Ask anything!*

Recap

- We saw how gradient descent provides a common framework for fitting several kinds of models
- We saw a need to test the effects of hyperparameters in a consistent way
- We were introduced to Support Vector Machines

Recap

STEPS OF MACHINE LEARNING ANALYSIS

1. Selection of features. **Next time, PCA**
2. Choosing a performance metric.
3. Choosing a classifier and optimization algorithm.
4. Evaluating the performance of the model. **Today**
5. Tuning the algorithm. **Today**


(p. 50: Raschka, 2015)

Learning goals and outline

Model evaluation and hyperparameter tuning

- 1) Understanding how cross-validation can improve validity
- 2) Understanding and being able to apply *Pipeline* from *scikit-learn*
- 3) Being able to run an analysis optimising the hyperparameters

Breast Cancer Wisconsin



Breast Cancer Wisconsin (Diagnostic)

Donated on 10/31/1995

Diagnostic Wisconsin Breast Cancer Database.

Dataset Characteristics	Subject Area	Associated Tasks
Multivariate	Health and Medicine	Classification
Feature Type	# Instances	# Features
Real	569	30

Dataset Information

Additional Information

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. A few of the images can be found at <http://www.cs.wisc.edu/~street/images/>

...

SHOW MORE ▾

Has Missing Values?

No

<https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic>

The data

```
### TERMINAL COMMAND

## pip install ucimlrepo

### CODE IMPORT

from ucimlrepo import fetch_ucirepo

# fetch dataset
breast_cancer_wisconsin_diagnostic = fetch_ucirepo(id=17)

# data (as pandas dataframes)
X_df = breast_cancer_wisconsin_diagnostic.data.features
y_df = breast_cancer_wisconsin_diagnostic.data.targets

# metadata
print(breast_cancer_wisconsin_diagnostic.metadata)

# variable information
print(breast_cancer_wisconsin_diagnostic.variables)
```

```
In [79]: X_df
Out[79]:
```

	radius1	texture1	...	symmetry3	fractal_dimension3
0	17.99	10.38	...	0.4601	0.11890
1	20.57	17.77	...	0.2750	0.08902
2	19.69	21.25	...	0.3613	0.08758
3	11.42	20.38	...	0.6638	0.17300
4	20.29	14.34	...	0.2364	0.07678
..
564	21.56	22.39	...	0.2060	0.07115
565	20.13	28.25	...	0.2572	0.06637
566	16.60	28.08	...	0.2218	0.07820
567	20.60	29.33	...	0.4087	0.12400
568	7.76	24.54	...	0.2871	0.07039

[569 rows x 30 columns]

```
In [80]: y_df
Out[80]:
```

	Diagnosis
0	M
1	M
2	M
3	M
4	M
..	...
564	M
565	M
566	M
567	M
568	B

[569 rows x 1 columns]

Label encoding

```
import numpy as np
from sklearn.preprocessing import LabelEncoder
X = X_df.values
y = y_df.values
print(np.unique(y))      ['B' 'M']
print(y.shape)           (569, 1)
y = np.squeeze(y) ## removing the singleton dimension
print(y.shape)           (569,)
le = LabelEncoder()
y = le.fit_transform(y)
print(np.unique(y))      [0 1]
```


Training and test data split

ALSO CALLED HOLDOUT METHOD

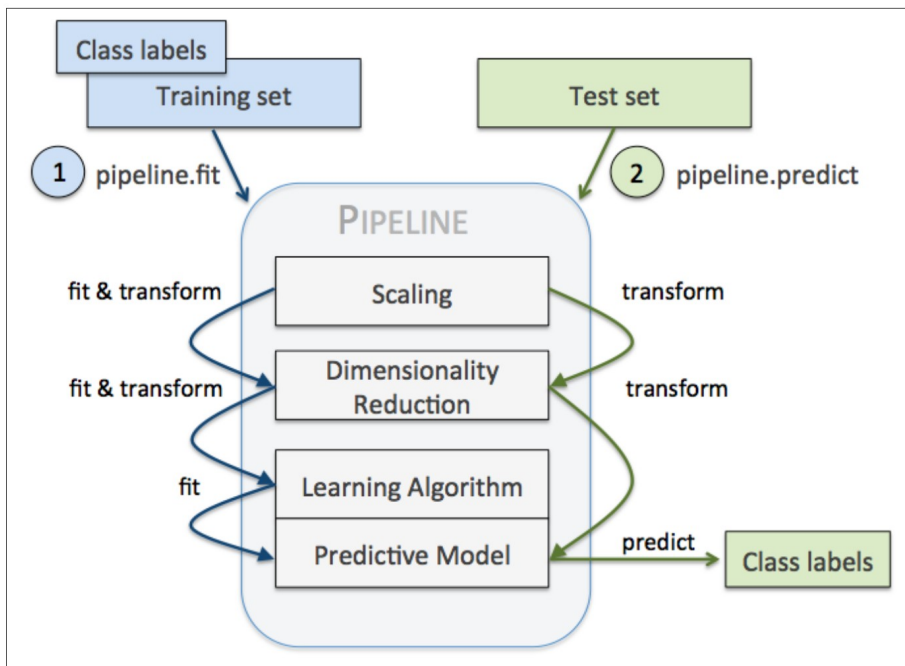
```
## SPLIT DATA

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.20, random_state=1)

print(y_train.shape)      (455,)
print(y_test.shape)       (114,)
```

Pipeline



(p. 172: Raschka, 2015)

```
### PIPELINE WAY
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
```

```
pipe_lr = Pipeline(
    [
        ('sc1', StandardScaler()),
        ('pca', PCA(n_components=2)),
        ('clf', LogisticRegression(random_state=1))
    ]
)
```

```
pipe_lr.fit(X_train, y_train)
pipe_lr.score(X_test, y_test)
```

What does the pipeline do?

SCALING

```
#%% scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
## why transform below and not fit_transform??
X_test_std = sc.transform(X_test)
```

SCALING

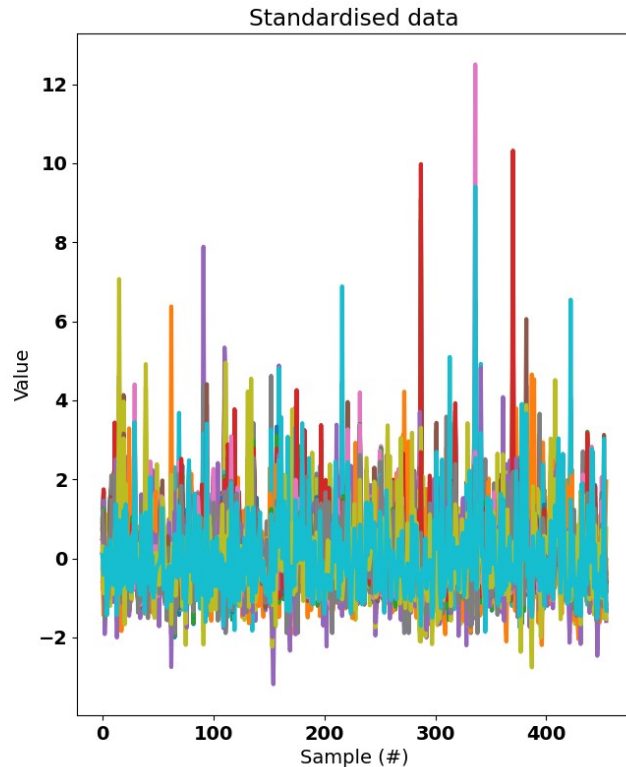
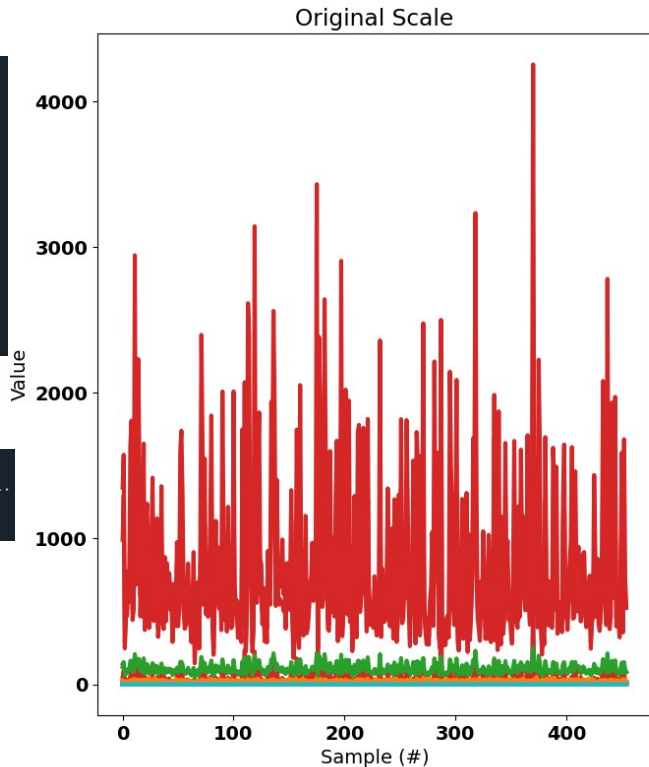
Each line is a feature

$$\mu \approx 0; \sigma \approx 1$$

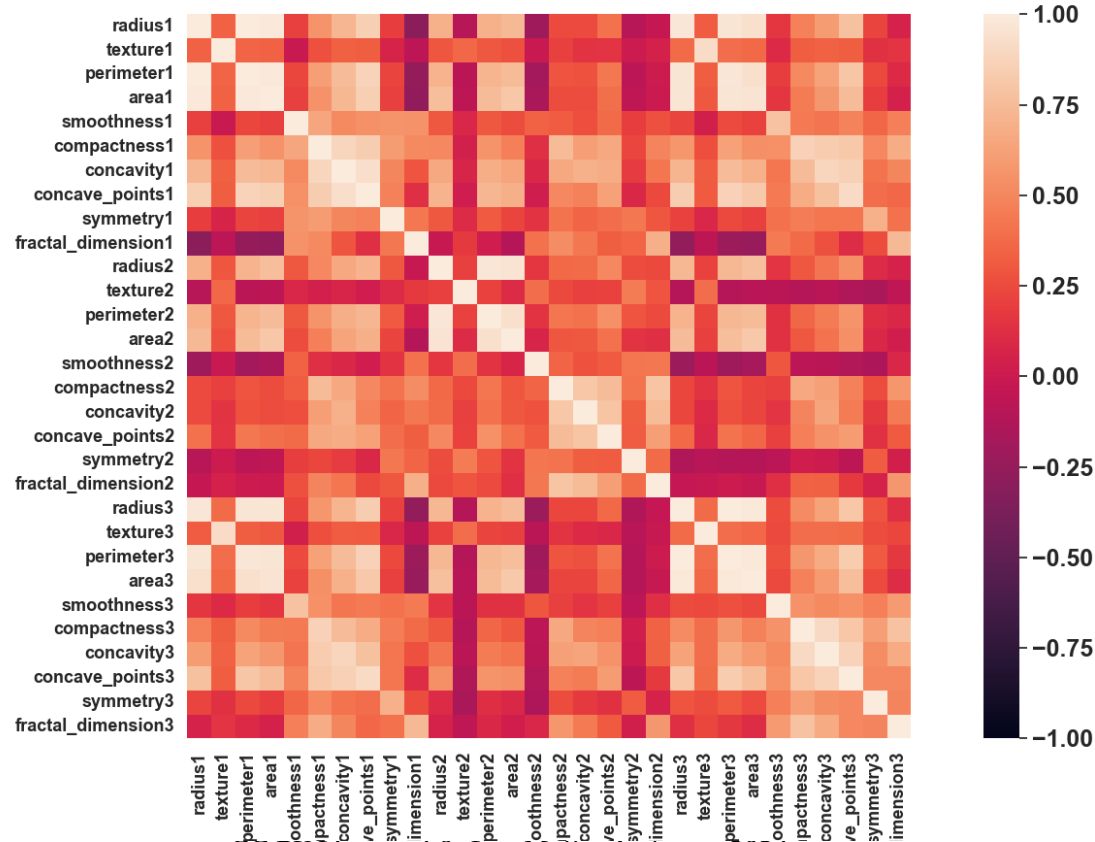
```
In [91]: print(np.mean(X_train_std, axis=0))
```

-9.46251624e-16	9.45641611e-16	4.83276423e-15	1.52259158e-16
-3.93095691e-15	-1.31762733e-17	1.08826257e-16	1.02652929e-15
-4.70685761e-15	-6.16966795e-15	3.64299555e-16	-6.03180509e-16
7.99116573e-16	-4.17980668e-16	-1.72999588e-16	1.53235178e-16
7.90576396e-17	-1.05487963e-15	-4.45065230e-16	8.61581868e-16
3.64787565e-16	2.57193534e-15	-1.02799332e-15	8.93058521e-17
7.92918844e-15	9.87244474e-16	-6.05132550e-16	5.52183452e-16
7.50011556e-16	1.07996640e-15]		

```
In [92]: print(np.mean(X_train_std, axis=0).shape)
(30,)
```

[illegible]

Normalised covariance matrix



CC BY Licence 4.0: Lau Møller Andersen 2024

What does the pipeline do?

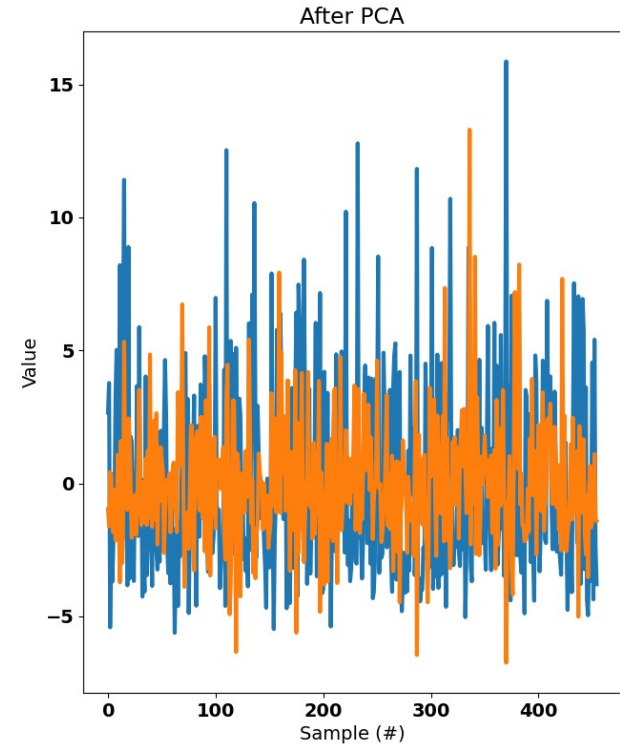
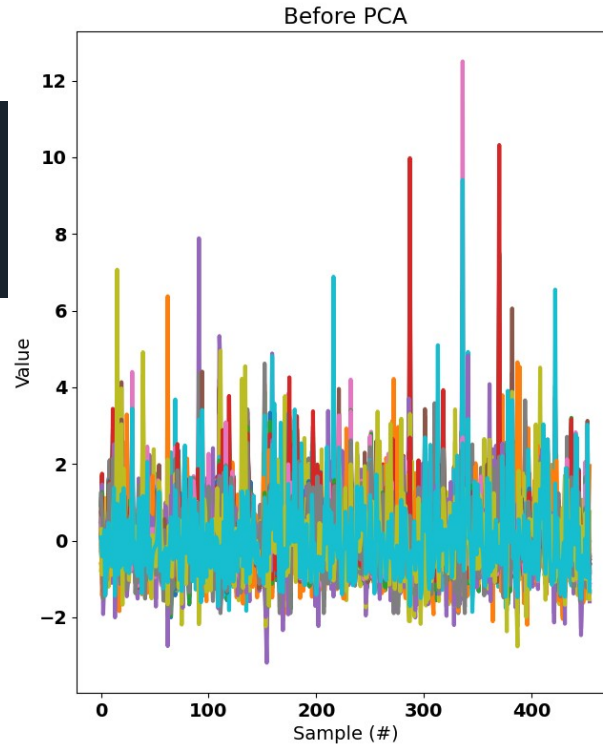
PCA

Each line is a feature

```
In [99]: print(X_train_std_pca.shape)
(455, 2)

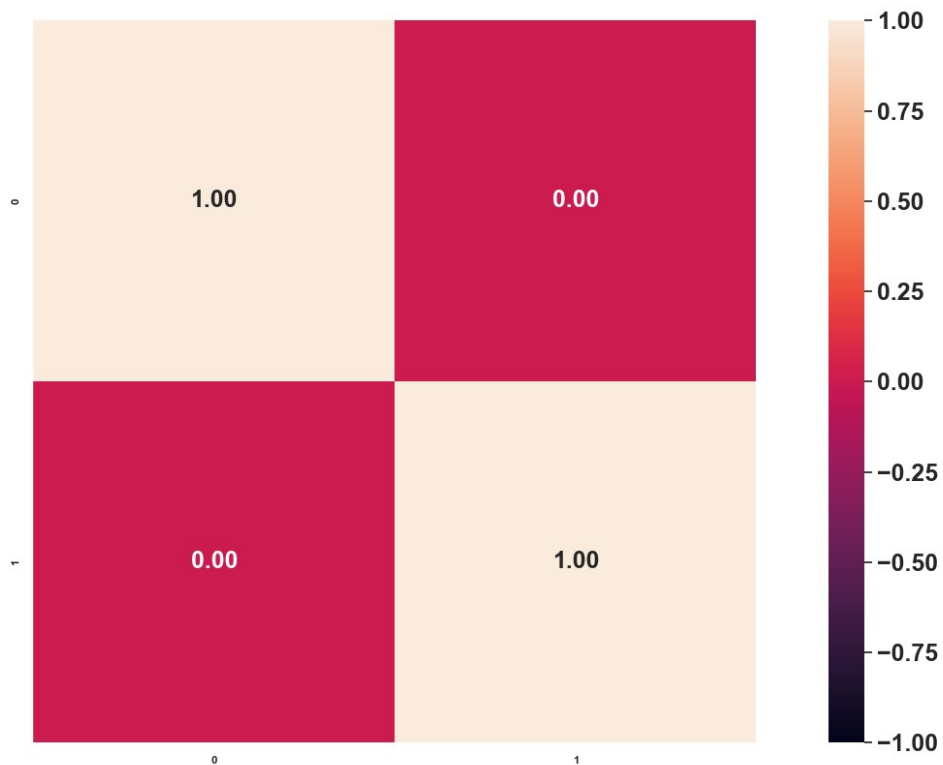
In [100]: print(np.std(X_train_std_pca, axis=0))
[3.68403185 2.31151566]

In [131]: print(np.mean(X_train_std_pca, axis=0))
[6.24652955e-17 3.90408097e-17]
```



Normalised covariance matrix

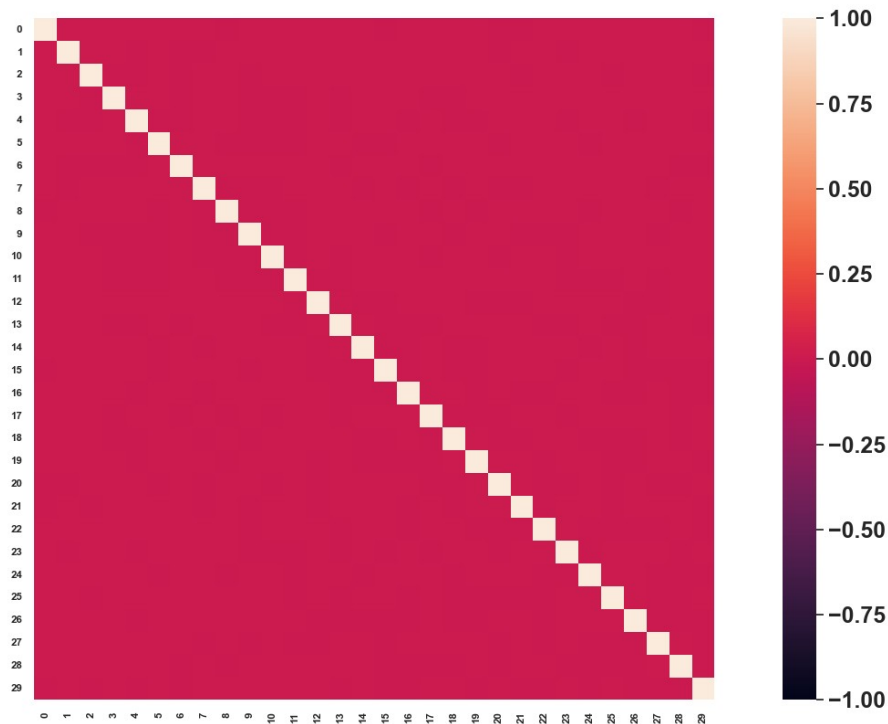
PCA: 2 FEATURES



Normalised covariance matrix

PCA: 30 FEATURES

```
In [311]: print(np.std(X_train_std_pca, axis=0))  
[3.68403185 2.31151566 1.70136353 1.39791498 1.27682331 1.11244795  
 0.8320431  0.69790209 0.63149539 0.59103773 0.53860885 0.50190151  
 0.4855613  0.40535005 0.30266507 0.28424771 0.24967564 0.2331132  
 0.21356572 0.17574297 0.17042989 0.16936146 0.15840141 0.13523772  
 0.12676504 0.0857295  0.07871836 0.03945616 0.0253066  
 0.01109384]
```



What does the pipeline do?

LOGISTIC REGRESSION

```
##### logistic regression
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(random_state=1, penalty='l2', C=1.0, tol=1e-4)
logreg.fit(X_train_std_pca, y_train)

print('Training accuracy: %.3f' % logreg.score(X_train_std_pca, y_train)) Training accuracy: 0.952
print('Test accuracy: %.3f' % logreg.score(X_test_std_pca, y_test)) Test accuracy: 0.947
```

What does the pipeline do?

ALL TOGETHER

```
#!/usr/bin/env python
# %% PIPELINE WAY

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

pipe_lr = Pipeline(
    [
        ('sc1', StandardScaler()),
        ('pca', PCA(n_components=2)),
        ('clf', LogisticRegression(random_state=1,
                                   penalty='l2',
                                   tol=1e-4,
                                   C=1.0))
    ]
)

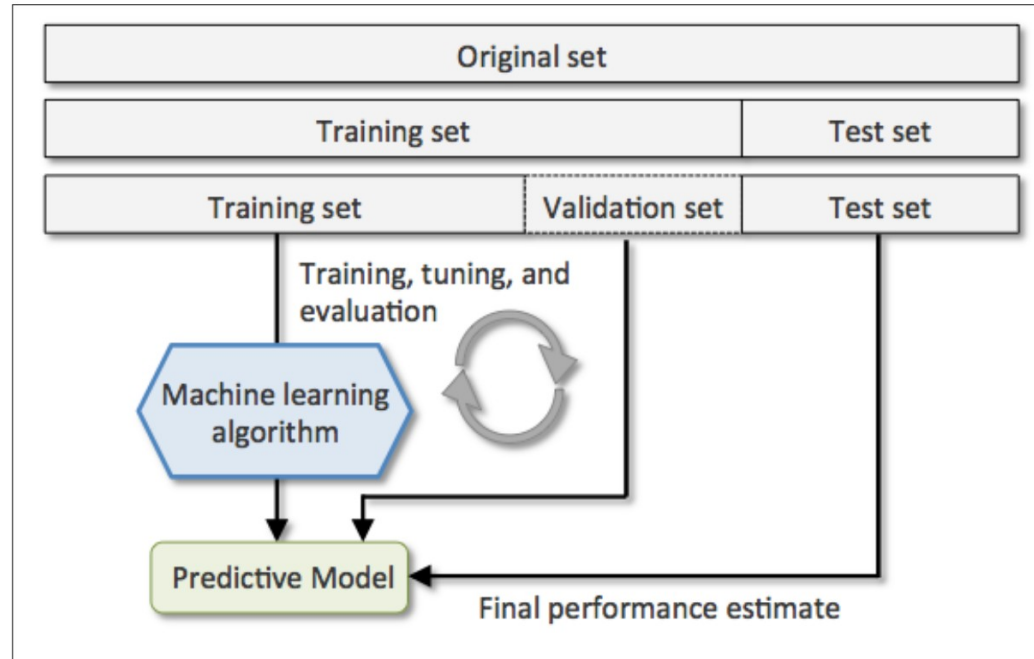
pipe_lr.fit(X_train, y_train)
print('Training accuracy: %.3f' % pipe_lr.score(X_train, y_train))
print('Test accuracy: %.3f' % pipe_lr.score(X_test, y_test))
```

```
Training accuracy: 0.952
Test accuracy: 0.947
```

DISCUSSION POINT

1. Which are the hyperparameters here?
2. Using the holdout method, *train_test_split*, what may be a disadvantage if we want to find the optimal set of hyperparameters?

Validation set



(p. 174: Raschka, 2015)

K-fold cross validation



(p. 176: Raschka, 2015)

Stratified K-fold validation

```
%% STRATIFIED K-FOLD - adapted from p. 177

import numpy as np
from sklearn.model_selection import StratifiedKFold
kfold = StratifiedKFold(n_splits=10)
scores = list() ## create an empty list

## just checking what kfold.split does
for index, (train_indices, test_indices) in enumerate(kfold.split(X, y)):
    print(f"Fold {index}:")
    print(f"Train indices: indices={train_indices[:10]}")
    print(f"Test indices: indices={test_indices[:10]}")

## actually applying it
for index, (train_indices, test_indices) in enumerate(kfold.split(X, y)):
    pipe_lr.fit(X[train_indices], y[train_indices])
    score = pipe_lr.score(X[test_indices], y[test_indices])
    scores.append(score)
    print(f'Fold: %s, Class dist.: %s, Acc: %.3f' % (index,
        np.bincount(y[train_indices]), score))

print(f'Cross-validation accuracy: %.3f +/- %.3f' % (
    np.mean(scores), np.std(scores)))
```

Stratified – making sure that there is an equal count of each category for each split – check the output of *np.bincount*

A slight improvement over the standard k-fold cross-validation approach is stratified k-fold cross-validation, which can yield better bias and variance estimates, especially in cases of unequal class proportions, as it has been shown in a study by R. Kohavi et al. (R. Kohavi et al. *A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection*. In *Ijcai*, volume 14, pages 1137–1145, 1995). In stratified cross-validation, the class proportions are preserved in each fold to ensure that each fold is representative of the class proportions in the training dataset, which we will illustrate by using the `StratifiedKFold` iterator in scikit-learn:

(p. 176: Raschka, 2015)

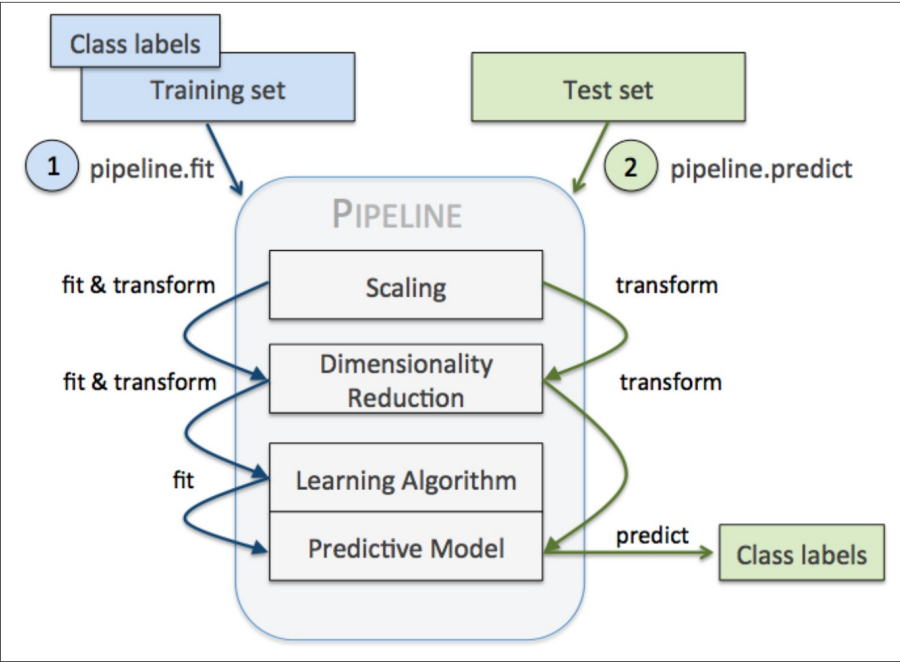
```
##### IMPLEMENTING IT IN PIPELINE

from sklearn.model_selection import cross_val_score
scores = cross_val_score(estimator=pipe_lr, X=X, y=y, cv=10, n_jobs=1)

print('Cross-validation accuracy scores: %s' % scores)
print('Cross-validation accuracy: %.3f +/- %.3f' % (
    np.mean(scores), np.std(scores)))
```

What does *n_jobs* do?

for cv_index in range(cv):



Grid Search

```
##%% GRID SEARCH CV - setup

from sklearn.model_selection import GridSearchCV
param_range_C = [1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3, 1e4]
param_range_PCA = np.arange(1, 15)
## Dictionary entry: name, e.g. 'clf' what it was called in the pipeline,
## two underscores and then the name of the parameter, e.g. C or n_components
param_grid = [{'clf__C': param_range_C, # you need two underscores
               'pca__n_components': param_range_PCA}]

grid_search = GridSearchCV(estimator=pipe_lr, param_grid=param_grid,
                           cv=10, n_jobs=-1) ## will use StratifiedKfold
```

```
##%% GRID SEARCH CV - fit

grid_search.fit(X, y)
print(grid_search.best_params_)
```

```
{'clf__C': 1.0, 'pca__n_components': 9}
```


Randomised grid search



Although grid search is a powerful approach for finding the optimal set of parameters, the evaluation of all possible parameter combinations is also computationally very expensive. An alternative approach to sampling different parameter combinations using scikit-learn is randomized search. Using the `RandomizedSearchCV` class in scikit-learn, we can draw random parameter combinations from sampling distributions with a specified budget. More details and examples for its usage can be found at http://scikit-learn.org/stable/modules/grid_search.html#randomized-parameter-optimization.

```
# specify parameters and distributions to sample from
param_dist = {
    "average": [True, False],
    "l1_ratio": stats.uniform(0, 1),
    "alpha": stats.loguniform(1e-2, 1e0),
```

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

Interim summary

- Standardisation
 - Makes linear and logistic regression fit more easily using gradient descent
- Feature extraction
 - PCA
 - Reduces the number of dimensions and thus the risk of overfitting
- Search of parameter space and cross-validation
 - Allows one to find the best hyperparameters

Confusion matrix

```
## LOOK AT CONFUSION MATRIX

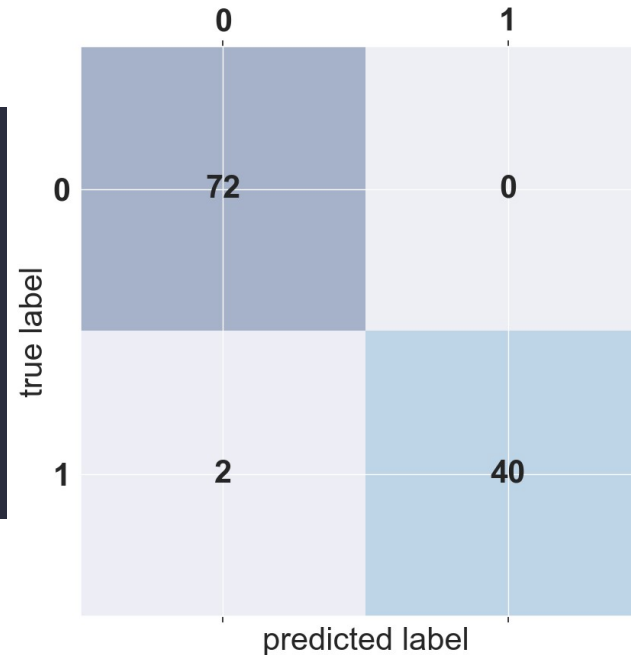
opt_pipe_lr = Pipeline(
    [
        ('sc1', StandardScaler()),
        ('pca', PCA(n_components=9)),
        ('clf', LogisticRegression(random_state=1,
                                   penalty='l2',
                                   tol=1e-4,
                                   C=1.0))
    ]
)

opt_pipe_lr.fit(X_train, y_train)
from sklearn.metrics import confusion_matrix
y_pred = opt_pipe_lr.predict(X_test)
confmat = confusion_matrix(y_true=y_test, y_pred=y_pred)
print(confmat)
```

Confusion matrix

```
#!/usr/bin/env python3
# %% plot more nicely pp. 190-191

fig, ax = plt.subplots()
ax.matshow(confmat, cmap=plt.cm.Blues, alpha=0.3)
for i in range(confmat.shape[0]):
    for j in range(confmat.shape[1]):
        ax.text(x=j, y=i, s=confmat[i, j], va='center', ha='center',
                fontdict=dict(fontsize=30))
plt.xlabel('predicted label', fontdict=dict(fontsize=30))
plt.ylabel('true label', fontdict=dict(fontsize=30))
plt.xticks(fontsize=30)
plt.yticks(fontsize=30)
plt.show()
```



Discussion point:
What do each of
the four squares
represent?

0: Benign
1: Malignant

Exercise in lecture

WINE DATASET



Wine Quality

Donated on 10/6/2009

Two datasets are included, related to red and white vinho verde wine samples, from the north of Portugal. The goal is to model wine quality based on physicochemical tests (see [Cortez et al., 2009], <http://www3.dsi.uminho.pt/pcortez/wine/>).

Dataset Characteristics

Multivariate

Subject Area

Business

Associated Tasks

Classification, Regression

Feature Type

Real

Instances

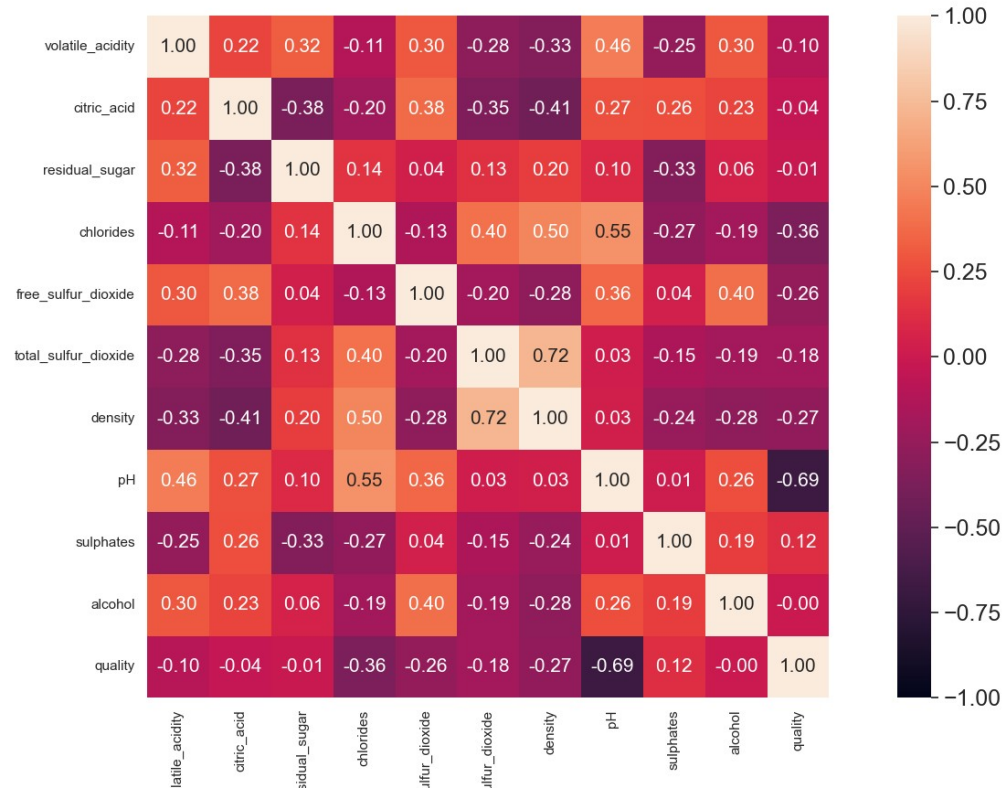
4898

Features

11

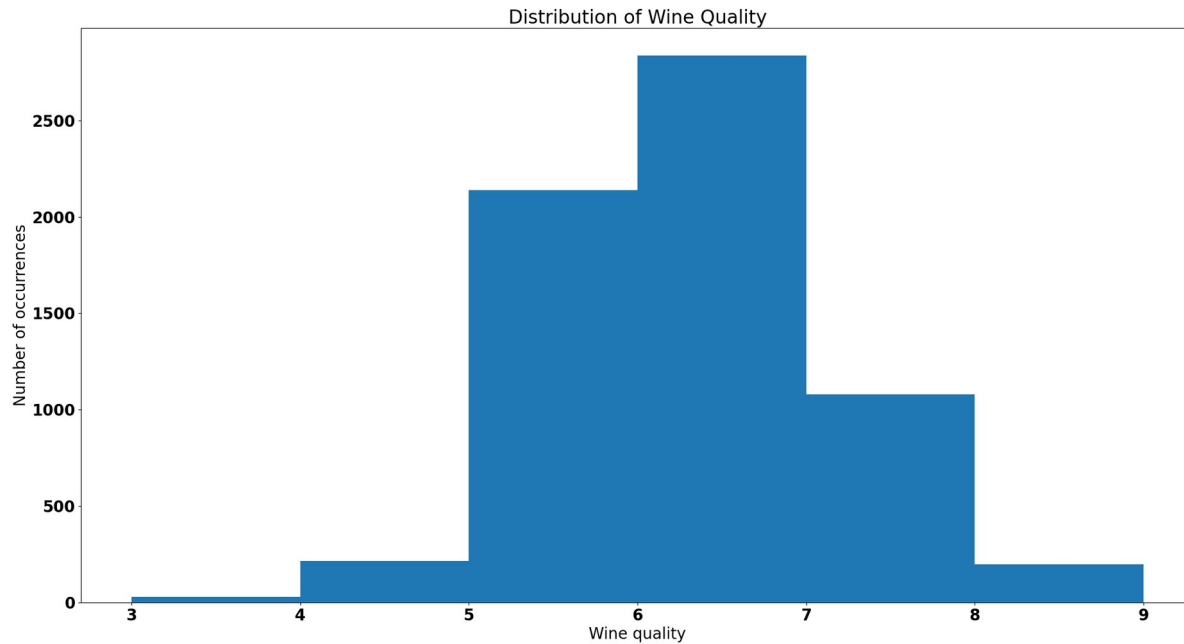
<https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic>

Normalised covariance matrix



y-values

```
In [49]: np.unique(y)  
Out[49]: array([3, 4, 5, 6, 7, 8, 9])
```



Binary vs. multiclass classification

ONE-VS-ALL



One-vs.-All (OvA), or sometimes also called **One-vs.-Rest (OvR)**, is a technique, us to extend a binary classifier to multi-class problems. Using OvA, we can train one classifier per class, where the particular class is treated as the positive class and the samples from all other classes are considered as the negative class. If we were to classify a new data sample, we would use our n classifiers, where n is the number of class labels, and assign the class label with the highest confidence to the particular sample. In the case of the perceptron, we would use OvA to choose the class label that is associated with the largest absolute net input value.

(p. 28: Raschka, 2015)

Binary vs. multiclass classification

MULTINOMIAL CLASSIFICATION

$$p_k = \frac{e^{z_k}}{\sum_{j=1}^K e^{z_j}}$$

p_k : predicted probability for class k

z_k : net value for class k

K : number of classes

$$\mathbf{z} = \mathbf{w}^T \mathbf{x} = w_0 x_0 + w_1 x_1 + \dots + w_{m-1} x_{m-1} + w_m x_m$$

Example

$$z_1 = 2.0 ; z_2 = 4.0 ; z_3 = 0.1$$

$$p_1 = \frac{e^{2.0}}{e^{2.0} + e^{4.0} + e^{0.1}} \approx 0.12$$

$$p_2 = \frac{e^{4.0}}{e^{2.0} + e^{4.0} + e^{0.1}} \approx 0.87$$

$$p_3 = \frac{e^{0.1}}{e^{2.0} + e^{4.0} + e^{0.1}} \approx 0.02$$

Exercise in lecture

- Overall aim: get the best classification in terms of accuracy, on a test set using logistic regression
- There is a prize for the best classification found
- Work in pairs or triplets
- OPTIONAL: can you reach an even better score with SVM? (for inspiration, see p. 186)

Load the data

```
### TERMINAL COMMAND

## pip install ucimlrepo

### LOAD WINE QUALITY

from ucimlrepo import fetch_ucirepo

# fetch dataset
wine_quality = fetch_ucirepo(id=186)

# data (as pandas dataframes)
X = wine_quality.data.features
y = wine_quality.data.targets

# metadata
print(wine_quality.metadata)

# variable information
print(wine_quality.variables)
```

Split you must start with

```
from sklearn.model_selection import train_test_split

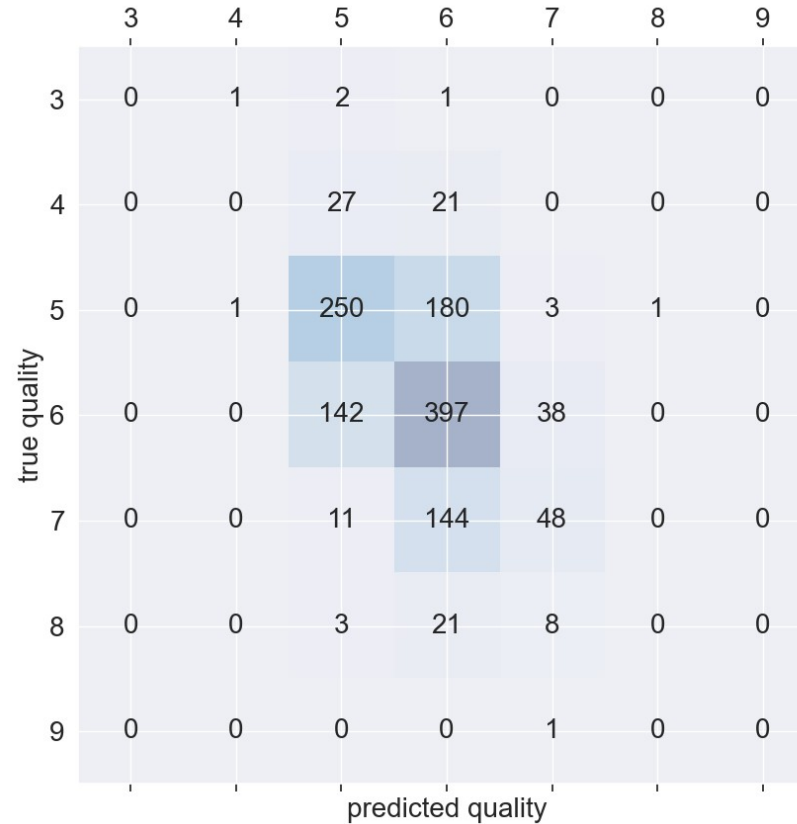
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.10, random_state=7)
```

- Thus the final test will be on `X_test` and `y_test` as defined here using `clf.score(X_test, y_test)`
- NB! This is different from how I did it – make sure

Suggested operations in no particular order

- test penalisers: L1, L2 or elastic net
- test different number of features in PCA space
- make sure to use cross-validation appropriately
- scale variables
- try different solvers (check the solver argument of LogisticRegression)
- try multinomial or OvR classification
- make sure to use the Pipeline function
- etc.

Also show your confusion matrix



Discuss

- Name some characteristics of this data that makes it hard to reach “high” classification rate
- Name some important steps in improving classification
- Did multinomial or OvR perform better?

Summary

- The space of hyperparameters can be quite large
 - Therefore we need consistent ways of exploring them
- Using a Pipeline is a good way to control things at a conceptual level within getting lost in the nitty-gritty
 - But do visit the nitty-gritty every now and then to make sure you understand what is going on

Learning goals and outline

Model evaluation and hyperparameter tuning

- 1) Understanding how cross-validation can improve validity
- 2) Understanding and being able to apply *Pipeline* from *scikit-learn*
- 3) Being able to run an analysis optimising the hyperparameters

The course plan

Week 1: Introduction

Instructor sessions: *Setting up R and Python and recollection of the general linear model*

Week 2: Multilevel linear regression

Instructor sessions: *Modelling subject level effects – and how do they differ from group level effects?*

Week 3: Link functions and fitting generalised linear multilevel models

Instructor sessions: *What to do when the response variable is not continuous?*

Week 4: Evaluating Generalised linear mixed models

Instructor sessions: *How do we assess how models compare to one another?*

Week 5: Explanation and Prediction

Instructor sessions: *Code review*

Week 6: Mid-way evaluation and Machine Learning Intro

Instructor sessions: *Getting Python Running*

Week 7: Linear and logistic regression revisited (machine learning)

Instructor sessions: *Categorizing responses based on informed guesses*

Week 8: Model evaluation and hyperparameter tuning

Instructor sessions: *Code review*

Week 9: Dimensionality Reduction, Principled Component Analysis (PCA)

Instructor sessions: *What to do with very rich data?*

Week 10: Outlook, unsupervised classification and neural networks

Instructor sessions: *Data with no labels and networks*

Week 11: Organising and preprocessing messy data

Instructor sessions: *Code review*

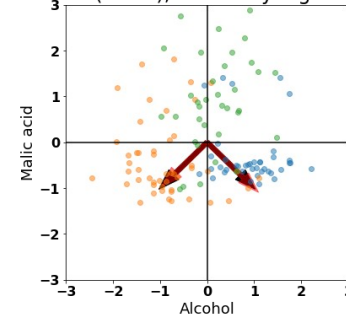
Week 12: Final evaluation and wrap-up of course

Instructor sessions: *Ask anything!*

Next time

- 1) Learning how we can extract the features that explain the most variance
- 2) Understanding how that can improve classification
- 3) Get acquainted with the concept of a eigenvector

Eigenvectors (black), scaled by eigenvalues (red)



Reading questions

- What is an eigenvalue?
 - <https://www.youtube.com/watch?v=PFDu9oVAE-g>
- How can multiplying by a matrix be seen as transformation of coordinate systems?
 - <https://www.youtube.com/watch?v=kYB8IZa5AuE>
- How would you find the optimal amount of principal components to keep?