Berlin School of Economics and Law
Prof. Dr. Natalie Packham

Summer term 2021
LV-Nr. 425093

Computational Finance and FinTech

File I/O

Contents

8 Machine Learning in Finance **1**
8.1 Unsupervised learning . . . 1
8.2 k-means clustering . . . 2
8.3 Supervised learning . . . 3
8.4 Gaussian Naive Bayes (GNB) . . . 5
8.5 Logistic regression (LR) . . . 7
8.6 Decision trees . . . 8
8.7 Deep neural networks (DNN's) . . . 9
8.8 Support vector machines (SVM's) . . . 10

8 Machine Learning in Finance

Machine Learning

- Further reading: **Py4Fi, Chapter 13**, from page 444.
- The **Py4Fi** book is very brief and application oriented, a great resource to dive deeper is the book > James, Witten, Hastie, Tibshirani: An Introduction to Statistical Learning. Springer, 2013

Machine Learning

- Machine Learning is a vast field with diverse applications.
- This chapter gives an overview and some use cases, which can be used as a starting point.
- Machine Learning methods are split into:
  - unsupervised learning, and
  - supervised learning.
- Machine Learning problems are split into:
  - regression, and
  - classification.
- Python offers a number of libraries for Machine Learning:
  - `scikit-learn` http://scikit-learn.org
  - `TensorFlow` http://tensorflow.org

Machine Learning

- The usual initialisation:

[1]:
```
import numpy as np
import pandas as pd
import datetime as dt
from pylab import mpl, plt
plt.style.use('seaborn')
mpl.rcParams['font.family'] = 'serif'
np.random.seed(1000)
```

8.1 Unsupervised learning

Unsupervised learning

- The principal idea of unsupervised learning is to extract information from data without any guidance or feedback.
- A typical appication is **clustering** (a classification problem).
- One such algorithm is k-means clustering, which cluster data into k subsets, called clusters.

8.2 k-means clustering

k-means clustering

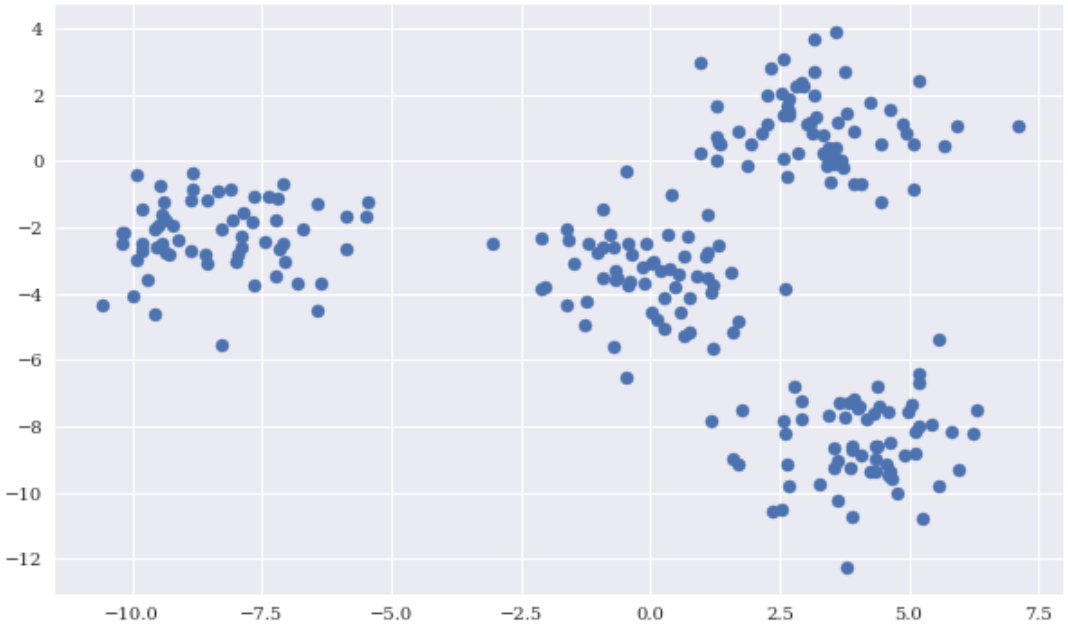
- `scikit-learn` allows the creation of sample data sets for different types of ML problems.
- Here we create a sample data set to illustrate k-means clustering.

[2]:
```
from sklearn.datasets.samples_generator import make_blobs
```

```
/Users/natalie/anaconda3/lib/python3.7/site-
packages/sklearn/utils/deprecation.py:144: FutureWarning: The
sklearn.datasets.samples_generator module is  deprecated in version 0.22 and
will be removed in version 0.24. The corresponding classes / functions should
instead be imported from sklearn.datasets. Anything that cannot be imported from
sklearn.datasets is now part of the private API.
  warnings.warn(message, FutureWarning)
```

[3]:
```
X, y = make_blobs(n_samples=250, centers=4, random_state=500, cluster_std=1.25) #␣
↪create a sample data set with 250 samples and 4 clusters
```

[4]:
```
plt.figure(figsize=(10,6))
plt.scatter(X[:,0], X[:,1], s=50);
```



k-means clustering

- The following code demonstrates the use of k-means clustering.
- The algorithm determines k clusters and assigns each sample to a cluster.

[5]:
```
from sklearn.cluster import KMeans # import model class
```

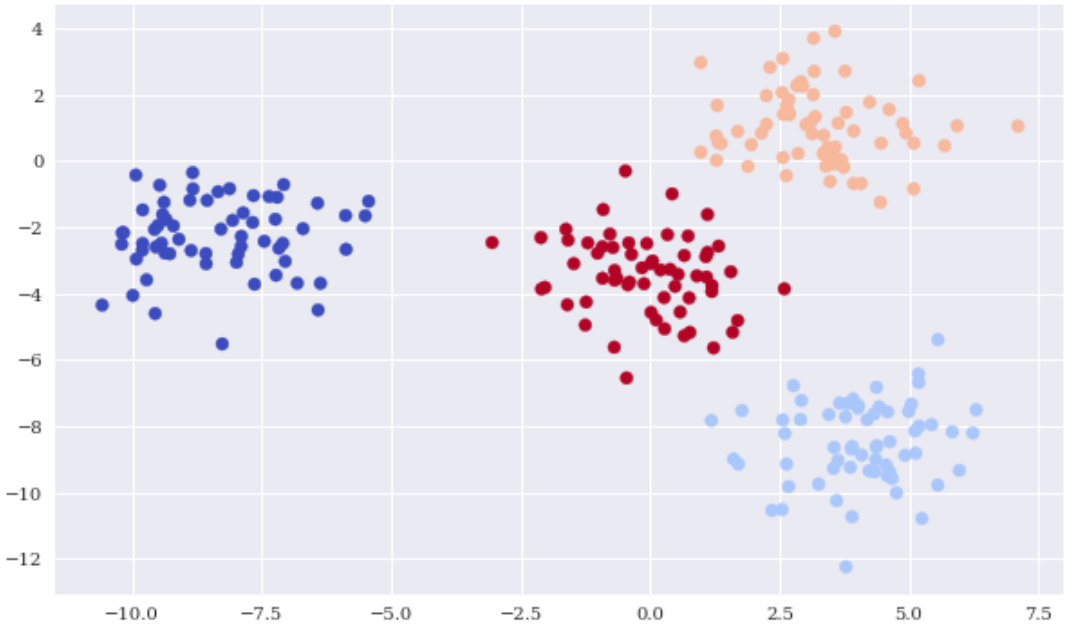
[6]:
```
model = KMeans(n_clusters=4, random_state=0) # instantiate the model
model.fit(X) # fit the model
```

[6]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=4, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=0, tol=0.0001, verbose=0)
```

[7]:
```
# predict the cluster (number) for each sample in the raw data
y_kmeans=model.predict(X)
y_kmeans[:12] # some cluster (numbers)
```

[7]:
```
array([2, 2, 0, 3, 0, 2, 3, 3, 3, 0, 1, 1], dtype=int32)
```

[8]:
```
# plot the data with one colour per cluster
plt.figure(figsize=(10,6))
plt.scatter(X[:,0], X[:,1], c=y_kmeans, cmap='coolwarm');
```



k-means clustering

- Here is how it works (see Section 10.3 of James et al., 2013):
- Let $x_1, x_2, \ldots, x_n$ denote the sample of points.
- Let $C_1, \ldots, C_k$ denote sets containing the indices of the observations in each cluster.
- They must satisfy the following two properties:
- Each observation belongs to at least one cluster: $C_1 \cup C_2 \cup \cdots \cup C_k = \{1, \ldots, n\}$.
- Each observation belongs to no more than one cluster: $C_i \cap C_j$ for all $i, j \in \{1, \ldots, k\}$.
- The goal is to find k cluster that minimise *within-cluster-variation*.
- This is achieved by minimising least-square-distances (see James et al. for details).

8.3 Supervised learning

Supervised learning

- Most problems belong to the domain of supervised learning.
- Here, some guidance in the form of know results or observed data is available.
- Linear regression is one example of supervised learning.
- Here, we will continues to study classification problems:
  - Gaussian Naive Bayes
  - Logistic regression
  - Decision trees
  - Deep neural networks
  - Support vector machines

Supervised learning

- More formally, the setting is as follows:
- We have *training observations* $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$.
- (Note that x_k can be vectors, i.e., $x_k \in \mathbb{R}^d$, $d \geq 1$.)
- Given some learning method, we estimate $\hat{f}$, such that the *predictions* $\hat{f}(x_1), \hat{f}(x_2), \ldots, \hat{f}(x_n)$ are approximately $y_1, y_2, \ldots, y_n$.
- In practice, we are not so much interested in whether $\hat{f}(x_i) \approx y_i$.
- Instead, we want to know whether $\hat{f}(x_0)$ is approximately equal to y_0, where (x_0, y_0) is a previously unseen *test* observation not used in the training stage.

Supervised learning - classification

- In a classification setting, the $y_1, y_2, \ldots, y_n$ can be qualitative data (corresponding to the classes).
- The accuracy of the estimate $\hat{f}(x_i) = \hat{y}_i$ is quantified for example by the training *error rate*, which is the proportion of incorrectly classified points: $\frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{\{y_i \neq \hat{y}_i\}}$
- Alternatively, one can specify the proportion of *correctly* classified observations in the training data set: $\frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{\{y_i = \hat{y}_i\}}$.

The data

- The following code produces a sample set with two features and a single binary label (0 or 1).

[9]:
```
from sklearn.datasets import make_classification
n_samples=100
X, y = make_classification(n_samples=n_samples, n_features=2, n_informative=2, \
                           n_redundant=0, n_repeated=0, random_state=250)
X[:5] # two real-valued features
```

[9]:
```
array([[ 1.68762365, -0.79757726],
       [-0.4312405 , -0.76063089],
       [-1.43934486, -1.23632519],
       [ 1.11799425, -1.86821958],
       [ 0.05020412,  0.65899067]])
```

[10]:
```
X.shape
```

[10]:
```
(100, 2)
```

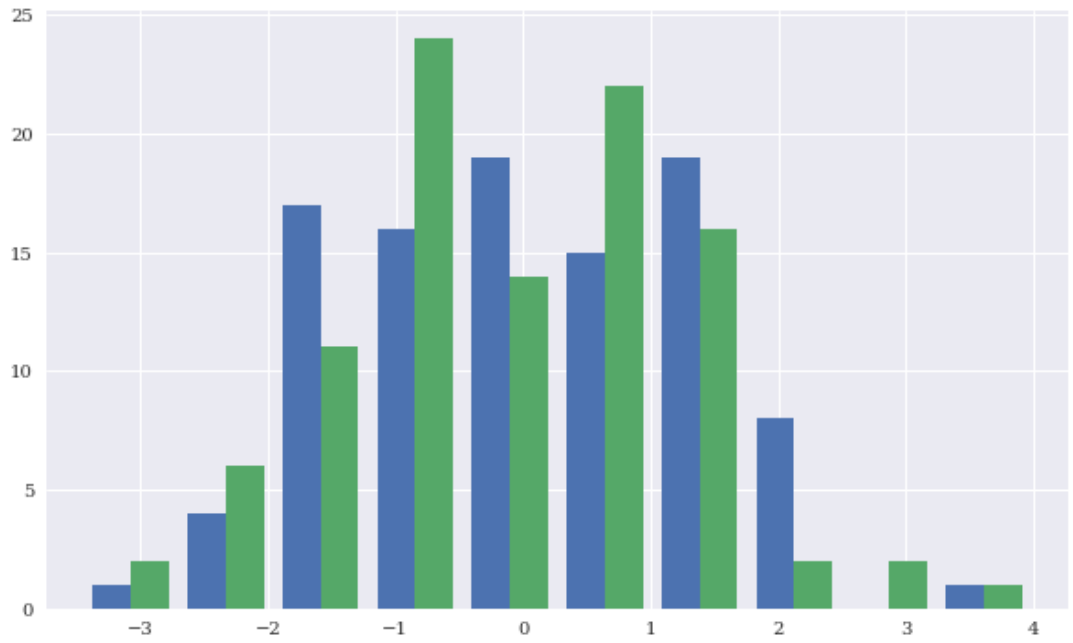
[11]:
```
y[:5] # single binary label
```

[11]:
```
array([1, 0, 0, 1, 1])
```

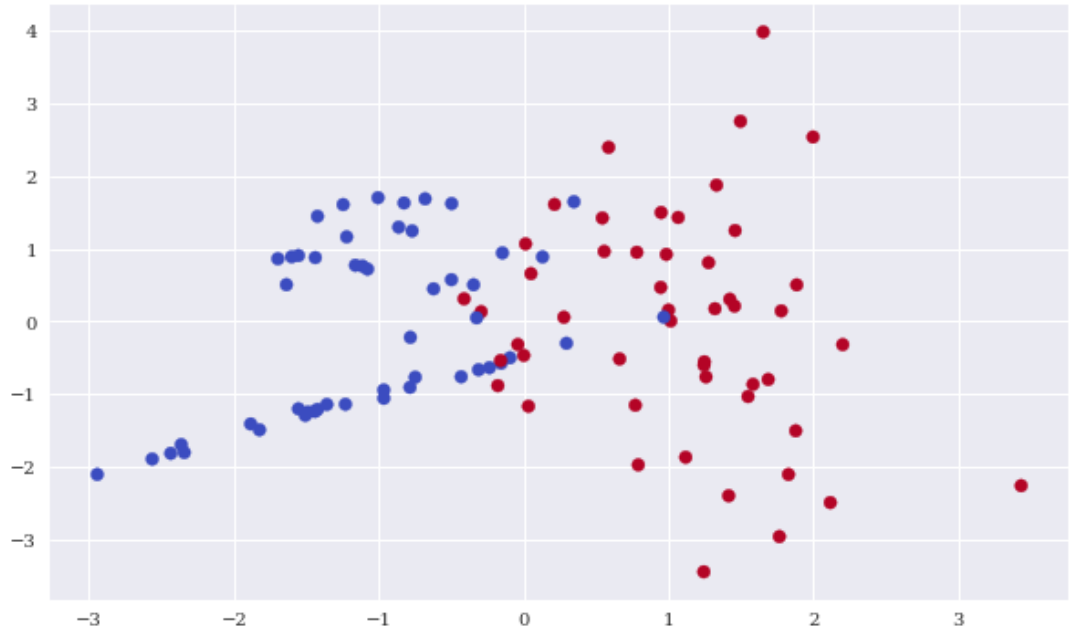
[12]:
```
y.shape
```

[12]: (100,)

[13]:
```
plt.figure(figsize=(10,6))
plt.hist(X);
```



[14]:
```
plt.figure(figsize=(10,6))
plt.scatter(x=X[:,0], y=X[:,1], c=y, cmap='coolwarm');
```



8.4 Gaussian Naive Bayes (GNB)

Bayes classifier

- The Bayes classifier estimates conditional probabilities $\mathbb{P}(Y = j|X = x_0)$ for all classes $j = 1, \ldots, k$.
- An observation is assigned its most likely class, i.e., it is assigned to the class for which the conditional probability is greatest.
- When the observation data is continuous, Gaussian naive Bayes is applied by calculating the conditional probability via a normal density, where the mean and variance are specific to each cluster.

[15]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
```

[16]:
```
model = GaussianNB()
model.fit(X,y)
```

[16]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

[17]:
```
model.predict_proba(X).round(4)[:5]
```

[17]:
```
array([[0.0041, 0.9959],
       [0.8534, 0.1466],
       [0.9947, 0.0053],
       [0.0182, 0.9818],
       [0.5156, 0.4844]])
```

[18]:
```
pred = model.predict(X)
```

[19]:
```
pred
```

[19]:
```
array([1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0,
       0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0])
```

[20]:
```
pred==y
```

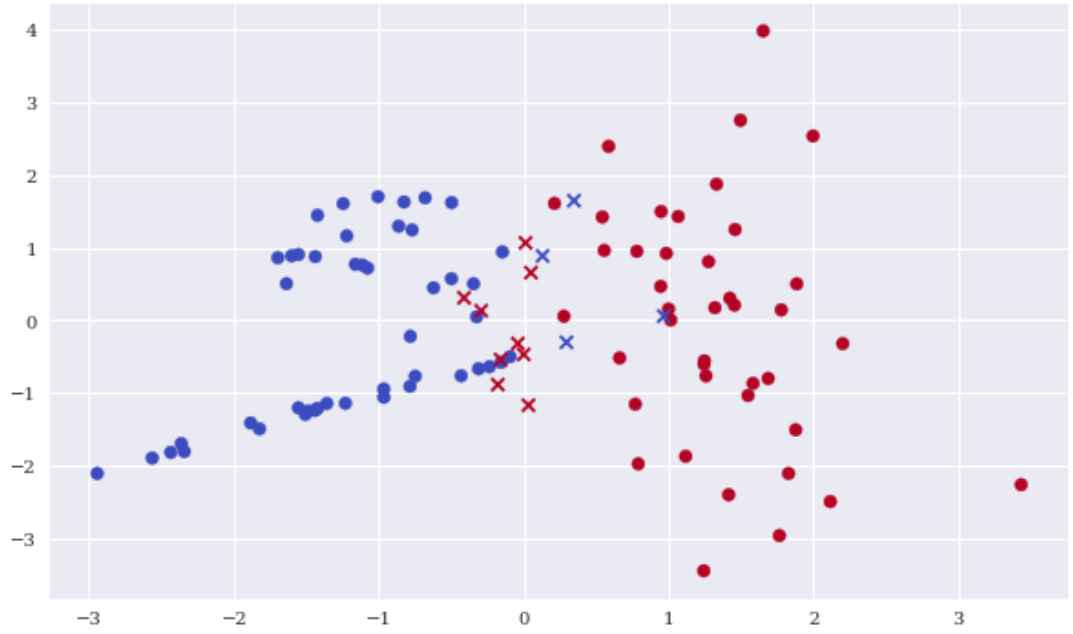
[20]:
```
array([ True,  True,  True,  True, False,  True,  True,  True,  True,
        True, False,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True, False, False, False,  True,  True,
        True,  True,  True,  True,  True,  True, False,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True, False,  True, False,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True, False,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True, False,  True, False,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True, False,  True, False,  True,  True,  True,  True,
        True])
```

[21]:
```
accuracy_score(y, pred)
```

[21]:
```
0.87
```

[22]:
```
Xc = X[y==pred]
Xf = X[y!=pred]
```

[23]:
```
plt.figure(figsize=(10,6))
plt.scatter(x=Xc[:,0], y=Xc[:,1], c=y[y==pred], marker='o', cmap='coolwarm')
plt.scatter(x=Xf[:,0], y=Xf[:,1], c=y[y!=pred], marker='x', cmap='coolwarm');
```



8.5 Logistic regression (LR)

- Logistic regression is a regression method where the dependent variable is a categorical variable (as opposed to a continuous variable).
- It models the probability of the categorical variable given the independent variables.

[24]:
```
from sklearn.linear_model import LogisticRegression
```

[25]:
```
model=LogisticRegression(C=1, solver='lbfgs')
model.fit(X,y)
```

[25]:
```
LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

[26]:
```
model.predict_proba(X).round(4)[:5]
```

[26]:
```
array([[0.011 , 0.989 ],
       [0.7266, 0.2734],
       [0.971 , 0.029 ],
       [0.04  , 0.96  ],
       [0.4843, 0.5157]])
```

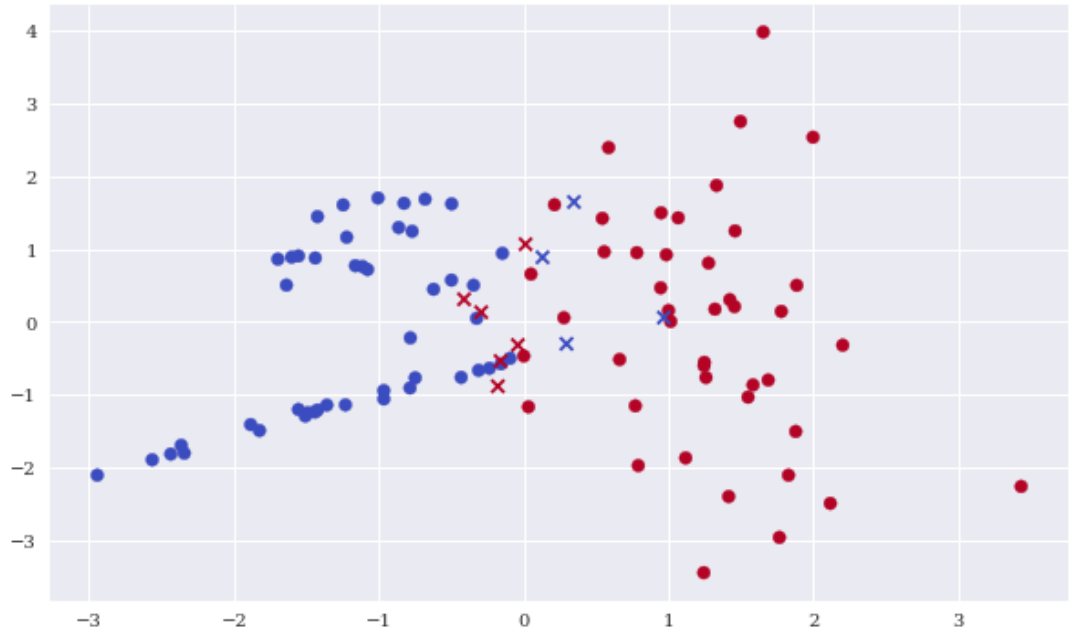
[27]:
```
pred = model.predict(X)
```

[28]:
```
accuracy_score(y,pred)
```

[28]:
```
0.9
```

[29]:
```
Xc = X[y==pred]
Xf = X[y!=pred]
```

[30]:
```
plt.figure(figsize=(10,6))
plt.scatter(x=Xc[:,0], y=Xc[:,1], c=y[y==pred], marker='o', cmap='coolwarm')
plt.scatter(x=Xf[:,0], y=Xf[:,1], c=y[y!=pred], marker='x', cmap='coolwarm');
```



8.6 Decision trees

- Decision tress classifiers can be thought of as a stepwise partitioning of the data space.

[31]:
```
from sklearn.tree import DecisionTreeClassifier
```

[32]:
```
model = DecisionTreeClassifier(max_depth=1)
```

[33]:
```
model.fit(X,y)
```

[33]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=1, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

[34]:
```
model.predict_proba(X).round(4)[:5]
```

[34]:
```
array([[0.08, 0.92],
       [0.92, 0.08],
       [0.92, 0.08],
       [0.08, 0.92],
       [0.08, 0.92]])
```

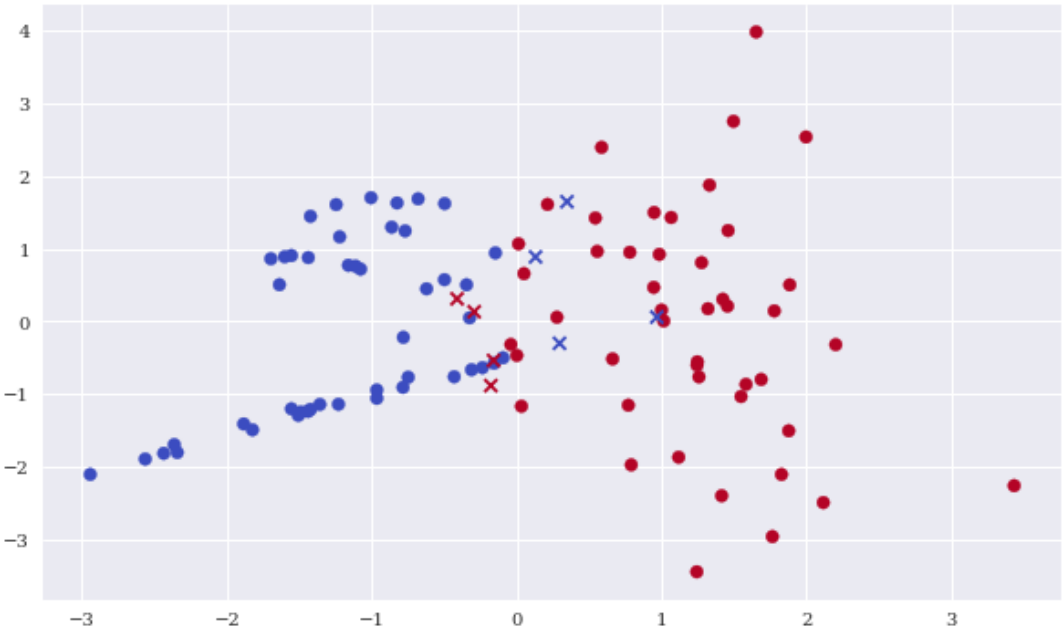
[35]:
```
pred = model.predict(X)
```

[36]:
```
accuracy_score(y, pred)
```

[36]:
```
0.92
```

[37]:
```
Xc = X[y==pred]
Xf = X[y!=pred]
```

[38]:
```
plt.figure(figsize=(10,6))
plt.scatter(x=Xc[:,0], y=Xc[:,1], c=y[y==pred], marker='o', cmap='coolwarm')
plt.scatter(x=Xf[:,0], y=Xf[:,1], c=y[y!=pred], marker='x', cmap='coolwarm');
```



- Increasing the maximum depth parameter for the decision tree allows to obtain a perfect result.
- But note that overfitting may occur.

[39]:
```
print('{:>8s} | {:8s}'.format('depth', 'accuracy'))
print(20 * '-')
for depth in range(1,7):
    model = DecisionTreeClassifier(max_depth=depth)
    model.fit(X,y)
    acc = accuracy_score(y, model.predict(X))
    print('{:8d} | {:8.2f}'.format(depth,acc))
```

```
   depth | accuracy
--------------------
       1 |     0.92
       2 |     0.92
       3 |     0.94
       4 |     0.97
       5 |     0.99
       6 |     1.00
```

8.7 Deep neural networks (DNN's)

Deep neural networks

- DNN's are very powerful and versatile algorithms for estimation and for classification.
- They are particularly useful for learning non-linear relationships, but this also makes them computationally demanding.
- *Deep* refers to so-called hidden layers in the network, which makes them powerful, but can also make it hard to understand how a DNN operates ("black box").
- `TensorFlow` is a popular open-source platform for DNN's.

[40]:
```
from sklearn.neural_network import MLPClassifier
```

[41]:
```
model = MLPClassifier(solver = 'lbfgs', alpha=1e-5, hidden_layer_sizes=2 * [75], ␣
↪random_state=10)
%time model.fit(X,y)
```

```
CPU times: user 948 ms, sys: 10.4 ms, total: 958 ms
Wall time: 277 ms

/Users/natalie/anaconda3/lib/python3.7/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:470:
ConvergenceWarning: 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
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

[41]:
```
MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=[75, 75], learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=200,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=10, shuffle=True, solver='lbfgs',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
```

[42]:
```
pred = model.predict(X)
pred
```

[42]:
```
array([1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1,
       0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0])
```

[43]:
```
accuracy_score (y, pred)
```

[43]:
```
1.0
```

8.8 Support vector machines (SVM's)

- Aside from introducing SVM's we investigate how to split a data set into separate training and testing data sets.
- A SVM is a classifier that linearly splits a hyperplane.

[44]:
```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
```

[45]:
```
train_x, test_x, train_y, test_y = train_test_split(X, y, test_size=0.33,␣
↪random_state=0)
```

[46]:
```
model = SVC(C=1, kernel='linear')
model.fit(train_x, train_y) # fit the model using the training data
```

[46]:
```
SVC(C=1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

[47]:
```
pred_train = model.predict(train_x) # predict the training data label values
```

[48]:
```
accuracy_score(train_y, pred_train) # "in-sample" prediction rate
```

[48]:
```
0.9402985074626866
```

[49]:
```
pred_test = model.predict(test_x)
```

[50]:
```
test_y == pred_test
```

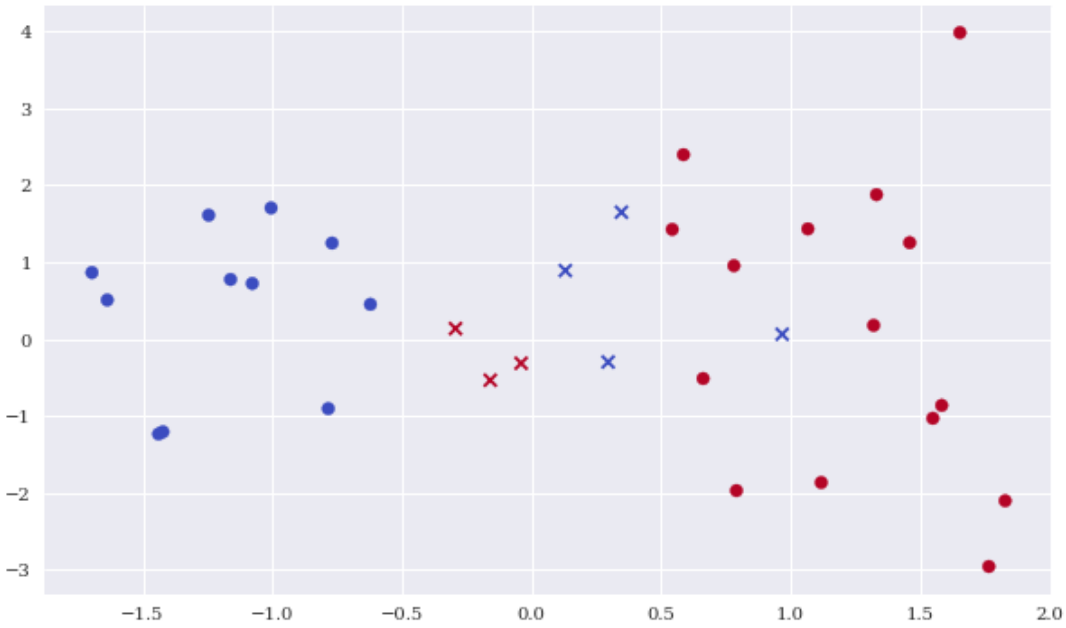
[50]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True, False, False, False,  True,  True,  True, False, False,
       False,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True, False,  True])
```

[51]:
```
accuracy_score(test_y, pred_test) # "out-of-sample" prediction rate
```

[51]:
```
0.7878787878787878
```

[52]:
```
test_c = test_x[test_y == pred_test]
test_f = test_x[test_y != pred_test]
```

[53]:
```
plt.figure(figsize=(10,6))
plt.scatter(x=test_c[:,0], y=test_c[:,1], c=test_y[test_y == pred_test],␣
↪marker='o', cmap='coolwarm')
plt.scatter(x=test_f[:,0], y=test_f[:,1], c=test_y[test_y != pred_test],␣
↪marker='x', cmap='coolwarm');
```


4
3
2
1
0
−1
−2
−3
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0