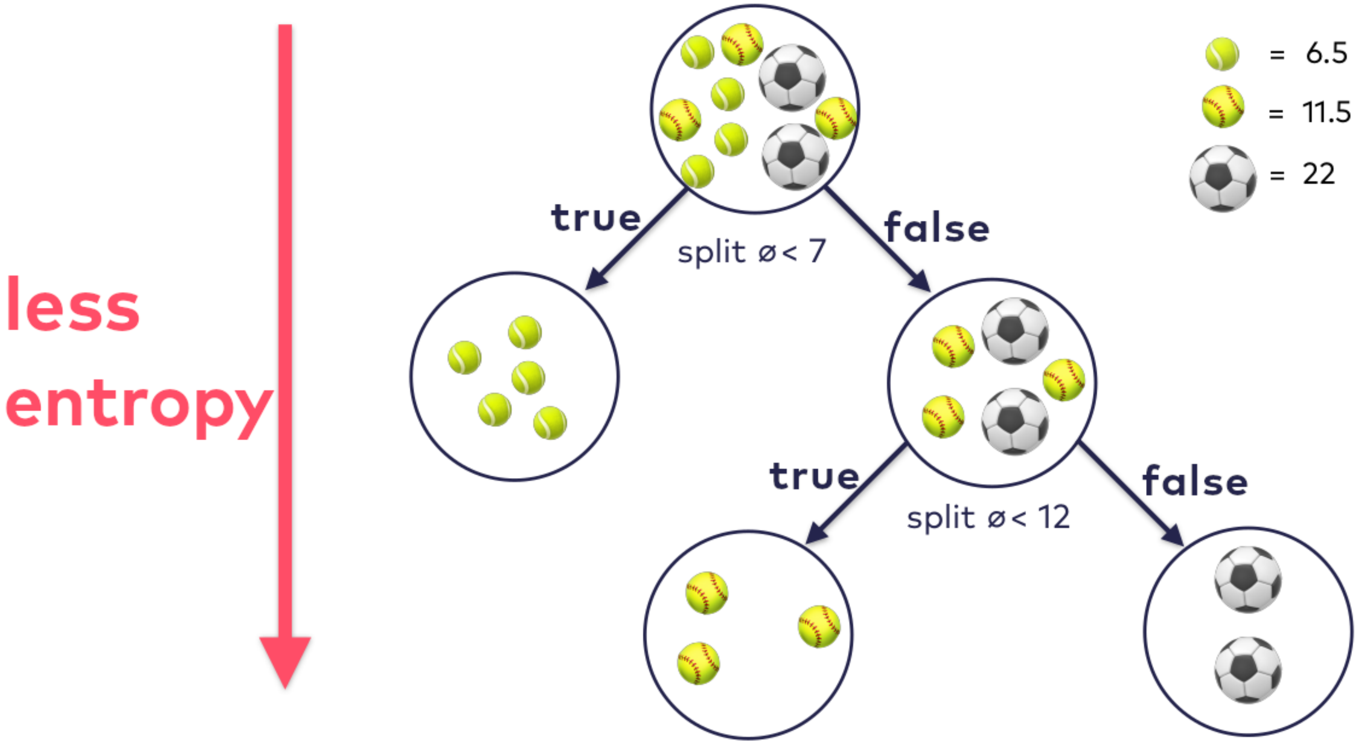
Decision Trees

How does it work



Advantages

- Easy to understand and to interpret. Trees can be visualised
- Uses a white box model: Results can be explained by boolean logic. In a black box model (e.g., in an artificial neural network), results may be more difficult to interpret
- Requires little data preparation: No data normalisation required, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Can handle both numerical and categorical data
- Can handle multi-output problems

Disadvantages

- Decision-tree learners can create over-complex trees that do not generalise the data well (overfitting)
- Can be unstable because small variations in the data might result in a different tree being generated
- The problem of learning an optimal decision tree is known to be NP-complete. Therefore practical decision-tree learning algorithms are based on heuristic algorithms which cannot guarantee to return the globally optimal decision tree.
- There are concepts that are hard to learn because decision trees can not express them easily, such as XOR, parity or multiplexer problems
- Decision Trees can be biased if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree

Decision Trees with scikit-learn

Splitting the dataset in test and training data

In [11]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.2,random_state=0)
```

Train the model

In [12]:

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(x_train, y_train)

dtc.predict(x_train)
```

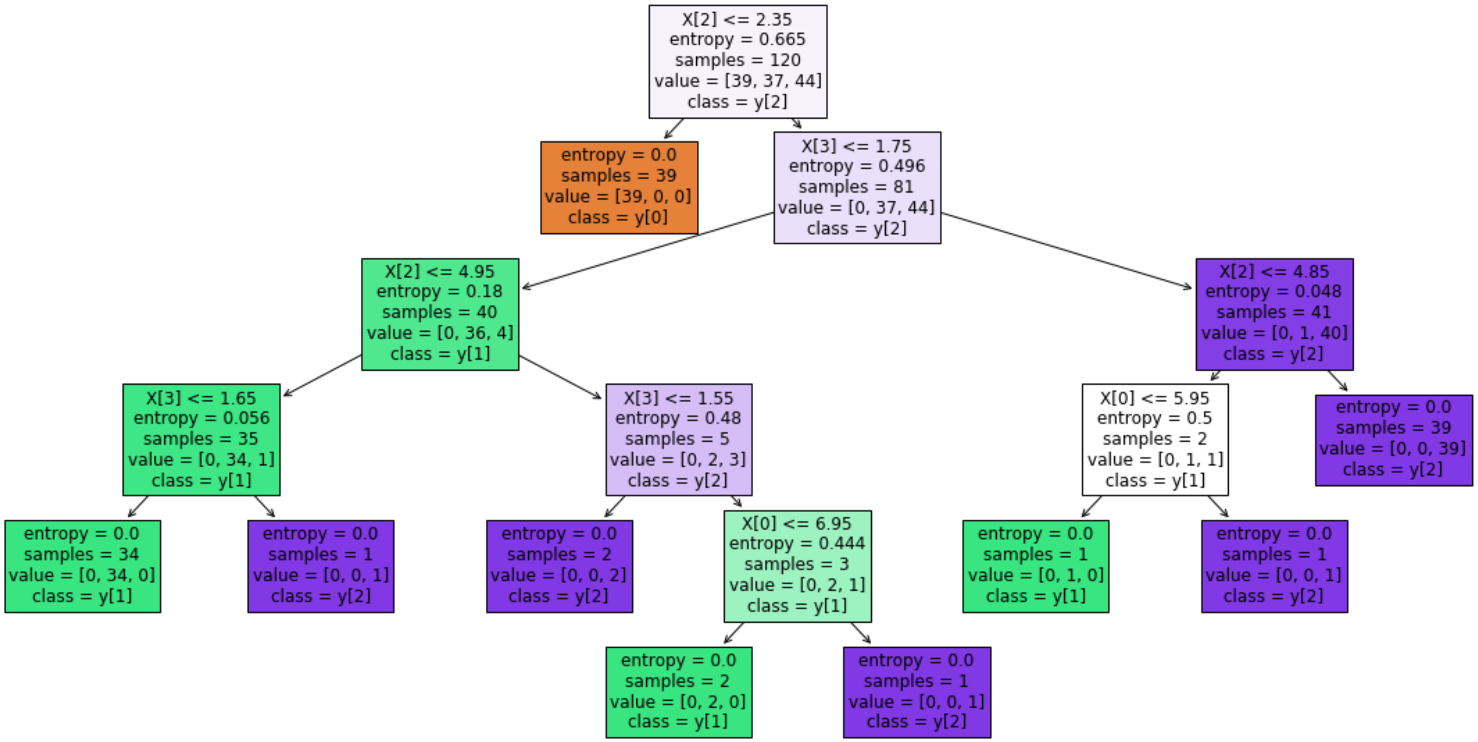
Out[12]:

```
array([2, 1, 0, 2, 2, 1, 0, 1, 1, 1, 2, 0, 2, 0, 0, 1, 2, 2, 2, 2, 1, 2,
       1, 1, 2, 2, 2, 2, 1, 2, 1, 0, 2, 1, 1, 1, 1, 2, 0, 0, 2, 1, 0, 0,
       1, 0, 2, 1, 0, 1, 2, 1, 0, 2, 2, 2, 2, 0, 0, 2, 2, 0, 2, 0, 2, 2,
       0, 0, 2, 0, 0, 0, 1, 2, 2, 0, 0, 0, 1, 1, 0, 0, 1, 0, 2, 1, 2, 1,
       0, 2, 0, 2, 0, 0, 2, 0, 2, 1, 1, 1, 2, 2, 1, 1, 0, 1, 2, 2, 0, 1,
       1, 1, 1, 0, 0, 0, 2, 1, 2, 0])
```

Plotting the Decision Tree

In [13]:

```
from sklearn import tree
from matplotlib.pyplot import figure
plt.figure(figsize=(20,10))
tree.plot_tree(dtc.fit(x_train, y_train), fontsize=12, class_names=True,
filled=True);
```



Validation

In [14]:
```
print("Decision Tree Test Accuracy {:.2f}%".format(dtc.score(x_test, y_test)*100
))
```

```
Decision Tree Test Accuracy 100.00%
```

Support Vector Classifier (SVC)

How does it work?

- Find the maximum margin hyperplane or function to separate the dataset into two clusters (it's a so-called maximum-margin classificator)
- Multiple kernels to support non-linear problems
- Requires feature scaling

Advantages

- Works well for high number of features (high dimensional spaces)
- Memory efficient because only a subset of data points is requried to compute the margin (the support vectors, hence the name)

Disadvantes

- Bad runtime for large datasets, up to $O(n_{features} \times n_{samples}^3)$
- Binary classificator, no direct multiclass classification support (also applies to many other classifiers, workaround one-versus-all or one-versus-one)
- Does not provide probability estimates (workaround via Platt scaling with expensive cross-validation)

SVC with scikit-learn

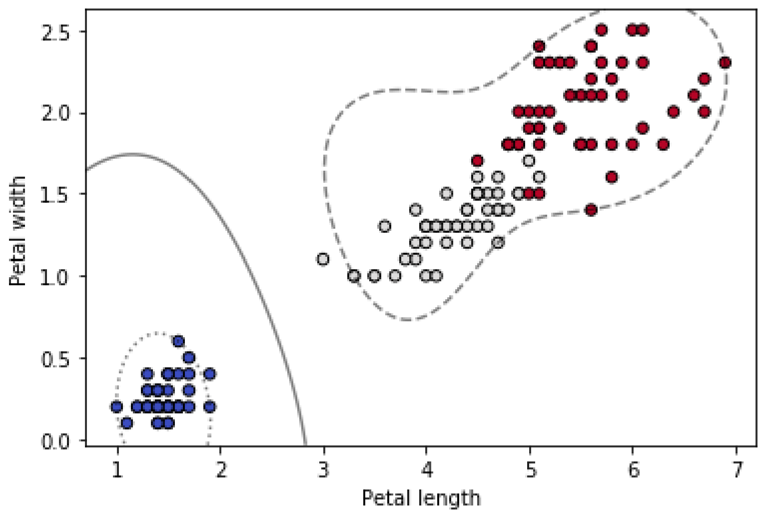
In [21]:

```
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

svc = Pipeline([
    ('scaler', StandardScaler()),
    ('svm', SVC(kernel='rbf'))
])
svc.fit(x_train, y_train);
```

Plotting single support vector for Setosa iris (RBF Kernel)

In [171]:

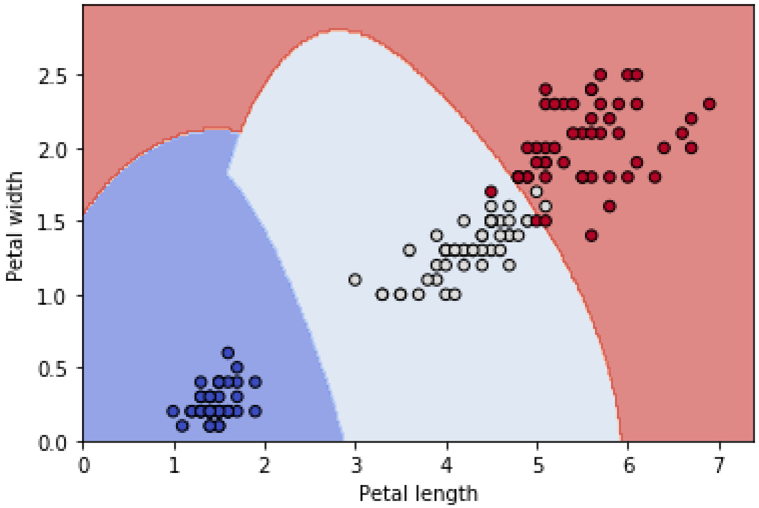
```
x = iris.data[:, (2,3)]
y = iris.target == 0 # 0=Setosa

clf = SVC(kernel="rbf"); clf.fit(x, y)

plt.scatter(x[:, 0], x[:, 1], c=iris.target, s=30, cmap=plt.cm.coolwarm, edgeco
lors='k') # plot the data

axes = plt.gca();xlim = axes.get_xlim();ylim = axes.get_ylim()
xx, yy = np.meshgrid(np.arange(xlim[0], xlim[1], .02), np.arange(ylim[0], ylim[
1], .02))
xy = np.vstack([xx.ravel(), yy.ravel()]).T

decision_boundaries = clf.decision_function(xy).reshape(xx.shape)
axes.contour(xx, yy, decision_boundaries, colors='k', levels=[-1, 0, 1], alpha=
0.5, linestyles=['--', '-', ':'])

axes.set_xlabel('Petal length'); axes.set_ylabel('Petal width'); plt.show()
```



Plot outcome of combined vectors for RBF Kernel

In [174]:

```
x = iris.data[:, (2,3)] # petal length and width
y = iris.target

clf = SVC(kernel='rbf'); clf.fit(x, y)

axes = plt.gca()
xx, yy = np.meshgrid(np.arange(0, x[:,0].max() + 0.5, .02), np.arange(0, x[:,1]
.max() + 0.5, .02))
axes.set_xlim(xx.min(), xx.max());axes.set_ylim(yy.min(), yy.max())

predicted_outcome = clf.predict(np.c_[xx.ravel(),
yy.ravel()]).reshape(xx.shape)
axes.contourf(xx, yy, predicted_outcome, cmap=plt.cm.coolwarm, alpha=0.6)

plt.scatter(x[:, 0], x[:, 1], c=iris.target, edgecolors='k', s=30, cmap=plt.cm.
coolwarm)

axes.set_xlabel('Petal length');axes.set_ylabel('Petal width'); plt.show()
```



Validation

In [22]:
```
print("Test Accuracy of the SVC model: {:.2f}%".format(svc.score(x_test,y_test)*
100))
```

```
Test Accuracy of the SVC model: 100.00%
```

Exercise - Preparing Data

In [ ]:

```
# Convert categorical variable into dummy/indicator variables, concerns cp, tha
l, slope
# See: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dum
mies.html

a = pd.get_dummies(heartdisease['chestpain type'], prefix = "cp")
b = pd.get_dummies(heartdisease['thal'], prefix = "thal")
c = pd.get_dummies(heartdisease['ST slope'], prefix = "slope")
```

In [ ]:

```
# add newly generated columns, drop the original columns
frames = [heartdisease, a, b, c]
df = pd.concat(frames, axis = 1).drop(columns = ['chestpain type', 'thal', 'ST
 slope'])
```

Exercise

- Use a decision tree classifier and a support vector classifier to create a model for predicting heart deseases
- Plot the decision tree
- Calculate error rate