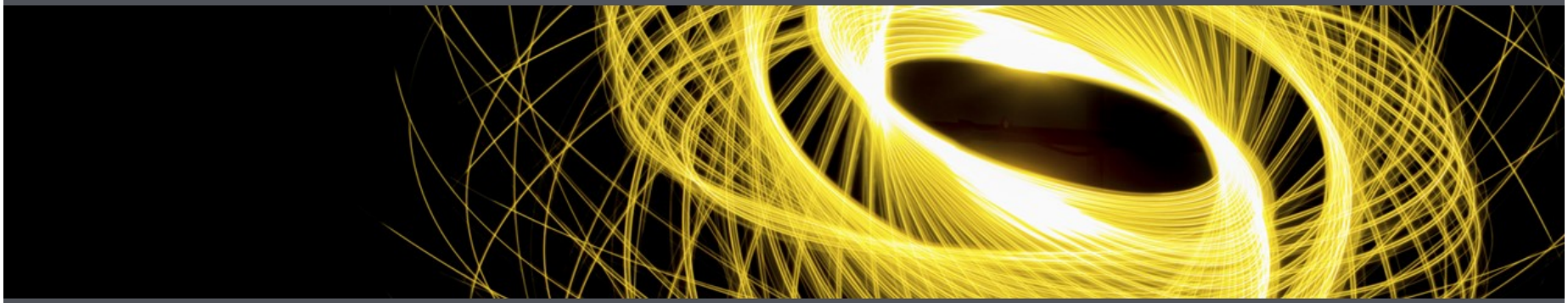
Department of Computer Science

University of Reading

CSMAD21 – Applied Data Science with Python



Classification

Supervised Learning

University of Reading

- Last week we covered regression, a method of supervised learning
- In supervised learning you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.

$$Y = f(x)$$

- The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data.

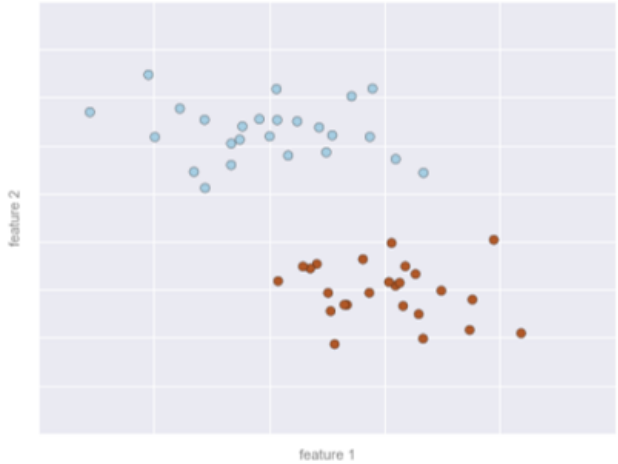
Classification

University of Reading

- In classification tasks the labels are in discrete categories as opposed to regression where labels are continuous.
- The pattern extracted from a set of labelled points are used to then classify unlabelled points.
- From a set of features and labels we would like to create a model that will let us decide whether a new point labelled ‘red’ or ‘blue’

Classifying data

- In 2 dimensional data each point is represented by (x,y) coordinates on the plane of 2 features



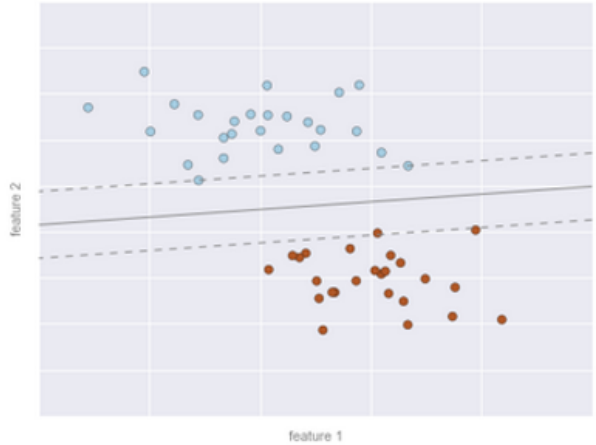
- Furthermore we have **class** labels for each point represented by the colours of the points

1. The Problem Statement

University of Reading

- From the features and labels we want to create a model whether any new data point would be ‘red’ or ‘blue’ class
- There are any number of possible models for this classification task however we will use the simplest one
- The assumption is that the two groups can be separated by drawing a straight line

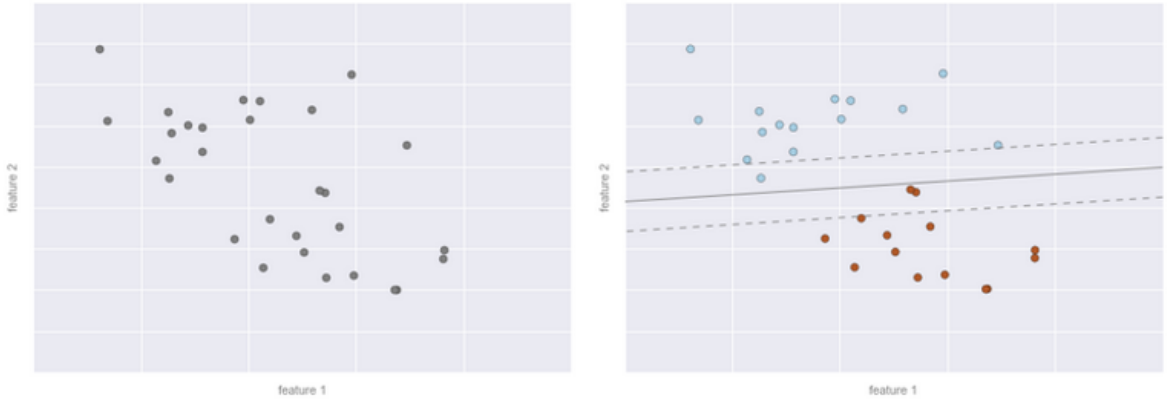
2. Training the model



- The **model** is a quantitative version of the statement "a straight line separates the classes"
- The model **parameters** are the numbers that describe the location and orientation of that line
- The optimal values for the parameters are learned from the data

3. Model Prediction

- The model can be generalised new unlabelled data



- At first glance it would be easy to draw a discriminatory line.
- Machine learning approaches however can be applied to much larger datasets with many more dimensions

Real World Applications

University of Reading

- There are multiple real world applications of classification tasks:
    1. Image recognition
    2. Speech Recognition
    3. Medical Diagnosis
    4. Email Spam

- There are many implementations of common classification algorithms:
    1. **Logistic Regression**
    2. **Support Vector Machines**
    3. Naïve Bayes
    4. Decision Trees

University of Reading

1. Logistic Regression

- Logistic regression uses an equation as the representation, like linear regression

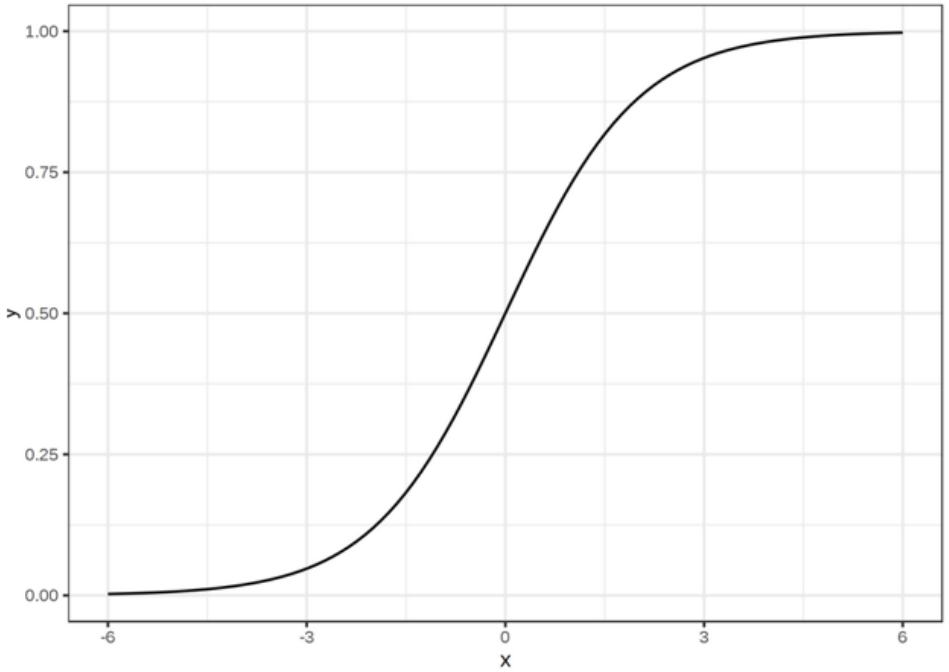
- Input values (x) are combined linearly using weights or coefficient values (referred to by the Greek capital letter Beta) to predict an output value (y)

- The difference from linear regression is that the output value is a binary value (0 or 1) rather than a continuous numeric value.

Theory for logistic regression

University of Reading

- Instead of fitting a straight line or hyperplane, the logistic regression model uses the logistic function to squeeze the output of a linear equation between 0 and 1.



- The logistic function is defined as:

$$logistic(\eta) = \frac{1}{1-exp^{(\eta)}}$$

University of Reading

From linear to logistic regression

- The step from linear regression to logistic regression is kind of straightforward. In a linear regression model, we have modelled the relationship between outcome and features with a linear equation

$$\hat{y} = B_0 + B_1 x_1 + B_2 x_2 + B_3 x_3 + \ \ldots + B_n x_n$$

- For classification we want probabilities between 1 and 0 therefore:

$$P = \frac{e^{(B_0 + B_1 x_1 + B_2 x_2 + B_3 x_3 + \ \ldots + B_n x_n)}}{1 - e^{(B_0 + B_1 x_1 + B_2 x_2 + B_3 x_3 + \ \ldots + B_n x_n)}}$$

Example of Logistic regression vs Linear Regression

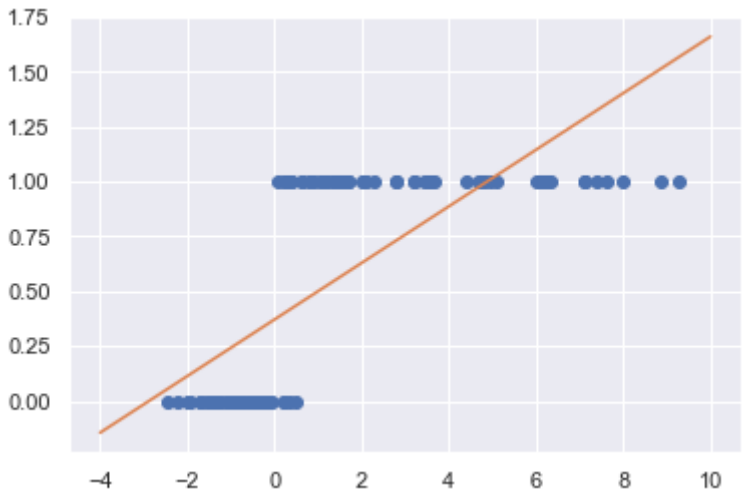
In [1]:

```
#Import Packages
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
import pandas as pd
from sklearn import linear_model
```

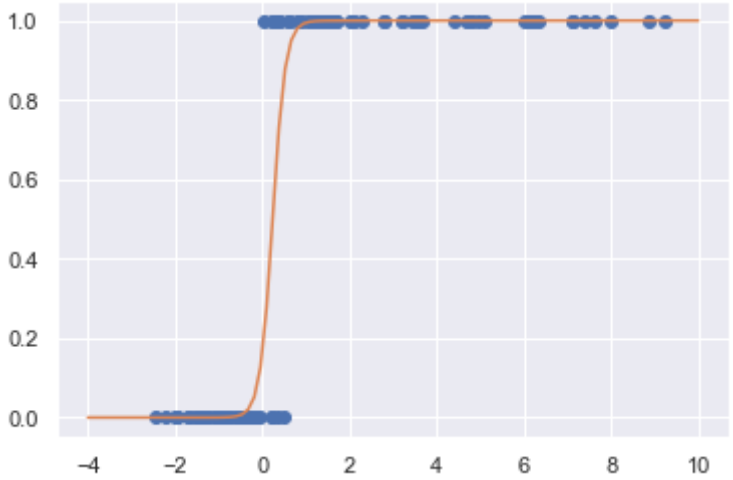
In [2]:
```
# Generate simulated data
np.random.seed(0)
X = np.random.normal(size=100)
y = (X > 0).astype(np.float)
X[X > 0] *= 4
X += .3 * np.random.normal(size=100)
X = X[:, np.newaxis]
plt.plot(X, y, 'o')

# Fit a linear regression to it using scikit learn
ols = linear_model.LinearRegression()
ols.fit(X, y)
X_test = np.linspace(-4, 10, 100)[:, np.newaxis]
plt.plot(X_test, ols.predict(X_test))
```

```
<ipython-input-2-e0946b30c143>:4: DeprecationWarning: `np.float` is a deprecated alias for the builtin `float`. To sile
nce this warning, use `float` by itself. Doing this will not modify any behavior and is safe. If you specifically wante
d the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecatio
ns (https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations)
  y = (X > 0).astype(np.float)
```

Out[2]:
```
[<matplotlib.lines.Line2D at 0x29076fa4c10>]
```



In [3]:
```
# Plot a logistic regression
plt.plot(X, y, 'o')
log_reg = linear_model.LogisticRegression(C=1e5)
log_reg.fit(X, y)
plt.plot(X_test, log_reg.predict_proba(X_test)[:, 1])
plt.show()
```

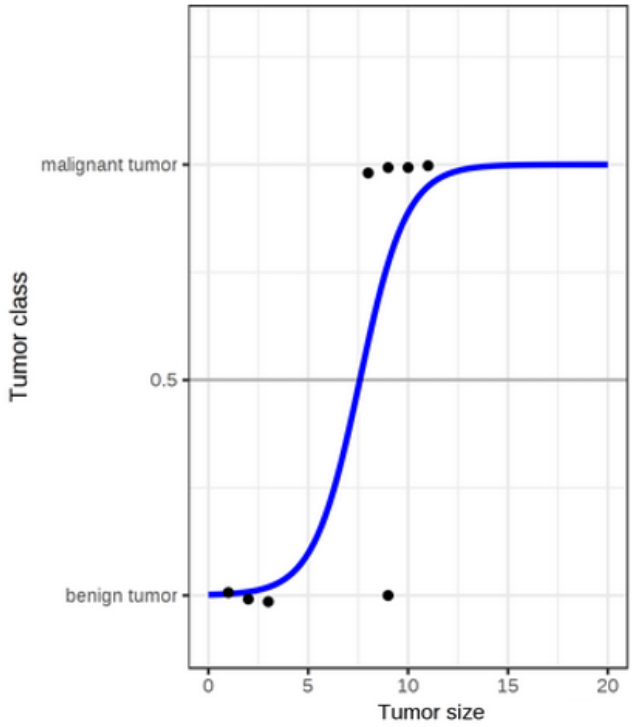


Interpretation of Logistic Regression

- The output of the model returns a decision boundary between two classes.

- When we have a new instance:
  - If the returned probability is above 0.5 then will be assigned the value of 1

  - If the returned probability is below 0.5 then will be assigned the value of 0



University of Reading

Evaluation metrics for Classification

- The main tool for evaluating the performance of the model is the confusion matrix
- This an example confusion matrix for a binary classifier but can be expanded for multiclass problems:

	Predicted Class: No	**Predicted Class: Yes**
Actual Class: No	50	10
Actual Class : Yes	5	100

- What can we learn?
  - The number of classes, the number of predictions, which of those predictions are correct and those which are incorrect

Further Evaluation metrics

University of Reading

- This is used as the basis for all other basic measurements:

	Predicted Class 0	**Predicted Class 1**
Actual Class 0	True Positive	False negative (Type II error
Actual Class 1	False Positive (Type I error)	True Positive

- Which primary metric is used depends on the classification problem:

$$\text{Accuracy} = \frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Positve + False\ Negative}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$\text{F1 Score} = 2 * \frac{Precision * Recall}{Precision + Recall}$$

An example of logistic regression and evaluation using the Wisconsin Breast Cancer dataset

In [4]:
```
# Import packages and data
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression

cancer = datasets.load_breast_cancer()
```

In [5]:
```
#Split the dependent from the independent variables
X = cancer.data
y = cancer.target

print (X.shape)
print (y.shape)
```

```
(569, 30)
(569,)
```

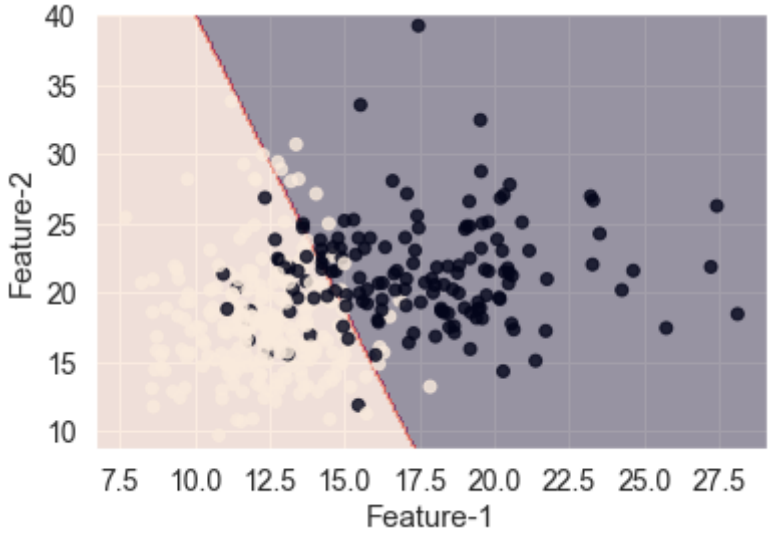
In [6]:
```
#Split into traing/test holdout dataset
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=42)
```

In [7]:
```
clf2 = linear_model.LogisticRegression(max_iter = 10000)
clf2.fit(X_train, y_train)
clf2.get_params()
```

Out[7]:
```
{'C': 1.0,
 'class_weight': None,
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1,
 'l1_ratio': None,
 'max_iter': 10000,
 'multi_class': 'auto',
 'n_jobs': None,
 'penalty': 'l2',
 'random_state': None,
 'solver': 'lbfgs',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

In [8]:
```
from decision_boundaries import plot_decision_boundaries
plot_decision_boundaries(X_train,y_train, LogisticRegression)
```

Out[8]:
```
<module 'matplotlib.pyplot' from 'C:\\Users\\Miguel\\anaconda3\\envs\\base2\\lib\\site-packages\\matplotlib\\pyplot.py'>
```



In [9]:
```
y_pred = clf2.predict(X_test)
```

In [10]:
```
#Plot the confusion matrix
confusion_matrix(y_test, y_pred)
```

Out[10]:
```
array([[ 61,   2],
       [  2, 106]], dtype=int64)
```

In [11]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.97      0.97        63
           1       0.98      0.98      0.98       108

    accuracy                           0.98       171
   macro avg       0.97      0.97      0.97       171
weighted avg       0.98      0.98      0.98       171
```

Now we can make a prediction on a new instance as we have evaluated the model

In [12]:
```
X_new = [[ 17.99,  10.38,  122.8,  1001, 0.1184, 0.2776, 0.3001, 0.1471, 0.2419, 0.07871, 1.095, 0.9053, 8.589, 153.4, 0
          0.4904, 0.05373, 0.01587, 0.03003, 0.006193, 25.38, 17.33, 184.6, 2019, 0.1622, 0.6656, 0.7119, 0.2654, 0.4601

print ('The probability of belonging to each class is:',clf2.predict_proba(X_new))
print('The most likely class is:',clf2.predict(X_new))
```

```
The probability of belonging to each class is: [[1.00000000e+00 1.16345447e-13]]
The most likely class is: [0]
```

This means that the model estimates that the sample in X_new has:

100% likelihood to belong to the ‘Malignant’ class (target = 0)
0.% likelihood to belong to the ‘Benign’ class (target = 1)

Be careful, though: what we've done here is not a very good model evaluation scheme. Cross validation is a methodology that removes the problem of sampling bias in model building introduced in earlier lectures. Cross Validation treats model evaluation a little bit more carefully.

University of Reading

Rules of thumb for logistic regression

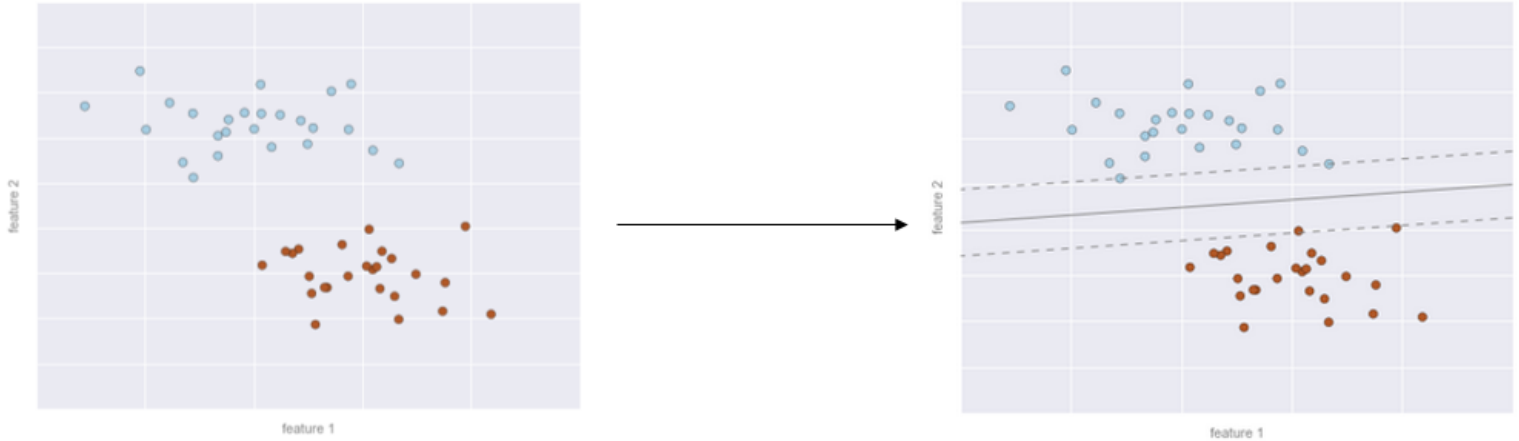
- **Remove Noise**: Logistic regression assumes no error in the output variable (y), consider removing outliers and possibly misclassified instances from your training data

- **Gaussian Distribution**: Logistic regression is a linear algorithm (with a non-linear transform on output). It does assume a linear relationship between the input variables with the output. Data transforms of your input variables that better expose this linear relationship can result in a more accurate model.

- **Remove Correlated Inputs**: Like linear regression, the model can overfit if you have multiple highly-correlated inputs. Consider calculating the pairwise correlations between all inputs and removing highly correlated inputs.

2. Support Vector Machines

- In the introduction we described the process to find a line that divides the classes in data from one another.



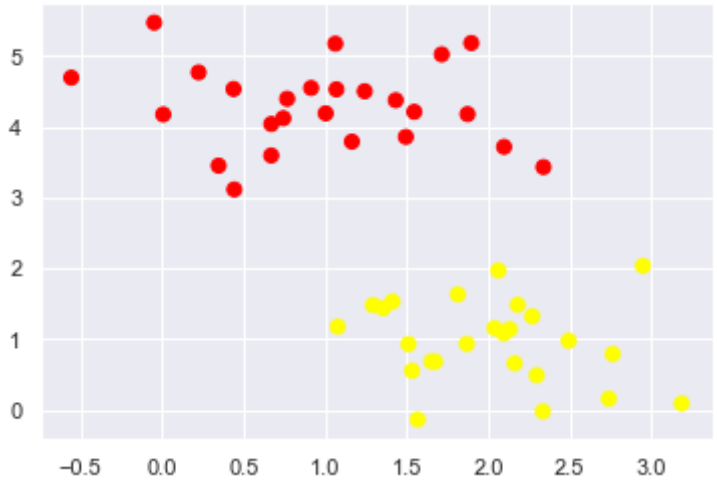
- A linear discriminative classifier would attempt to draw a straight line separating the two sets of data, and thereby create a model for classification.

Theory for Support Vector Machines

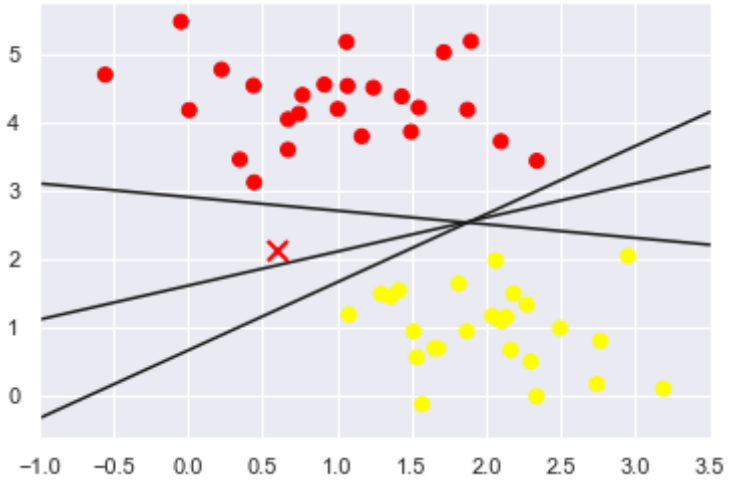
University of Reading

- However, there is more than a single dividing line that discriminates perfectly between the classes

In [13]:
```
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=50, centers=2,
                  random_state=0, cluster_std=0.60)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn');
```



In [14]:
```
xfit = np.linspace(-1, 3.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plt.plot([0.6], [2.1], 'x', color='red', markeredgewidth=2, markersize=10)

for m, b in [(1, 0.65), (0.5, 1.6), (-0.2, 2.9)]:
    plt.plot(xfit, m * xfit + b, '-k')

plt.xlim(-1, 3.5);
```



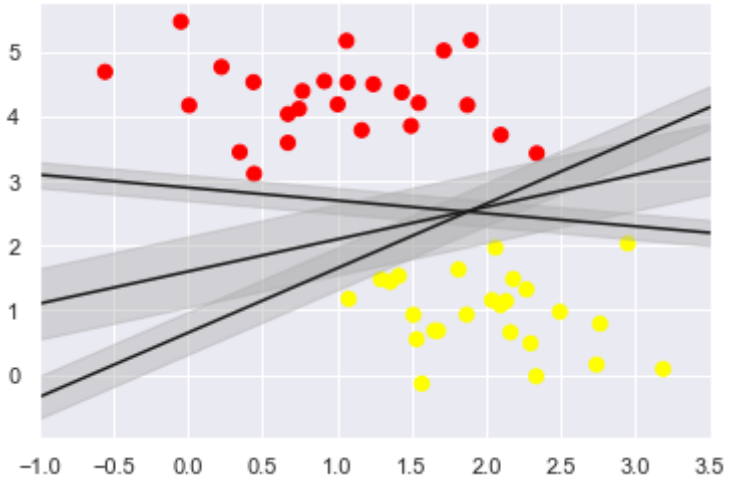
Depending on which model is chosen, a new data point (e.g., the one marked by the "X" in this plot) will be assigned to a different class. The simple intuition of "drawing a line between classes" is not enough

University of Reading

Theory for Support Vector Machines

- Support vector machines offer a way to choose the best model
- Rather than join a zero-width line between the classes a margin of some width can be drawn around each line

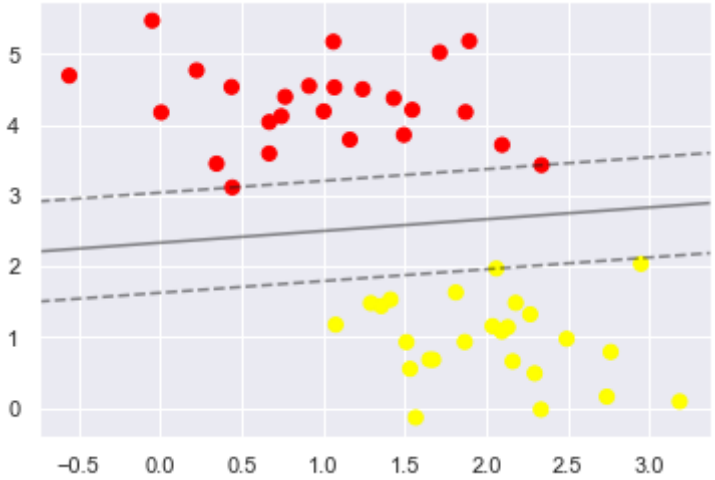
In [15]:
```
xfit = np.linspace(-1, 3.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')

for m, b, d in [(1, 0.65, 0.33), (0.5, 1.6, 0.55), (-0.2, 2.9, 0.2)]:
    yfit = m * xfit + b
    plt.plot(xfit, yfit, '-k')
    plt.fill_between(xfit, yfit - d, yfit + d, edgecolor='none',
                     color='#AAAAAA', alpha=0.4)

plt.xlim(-1, 3.5);
```



In [16]:

```
def plot_svc_decision_function(model, ax=None, plot_support=True):
    """Plot the decision function for a 2D SVC"""
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # create grid to evaluate model
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    # plot decision boundary and margins
    ax.contour(X, Y, P, colors='k',
               levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])

    # plot support vectors
    if plot_support:
        ax.scatter(model.support_vectors_[:, 0],
                   model.support_vectors_[:, 1],
                   s=300, linewidth=1, facecolors='none');
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
```

In [17]:
```
#from decision_boundaries import plot_svc_decision_function
from sklearn.svm import SVC # "Support vector classifier"
model = SVC(kernel='linear', C=1E10)
model.fit(X, y)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(model);
```



- In support vector machines, the line that maximizes the margin size is the one that is chosen as the optimal model.

- Support vector machines are an example of a *maximum margin* estimator.

Advantages of Support Vector Machines

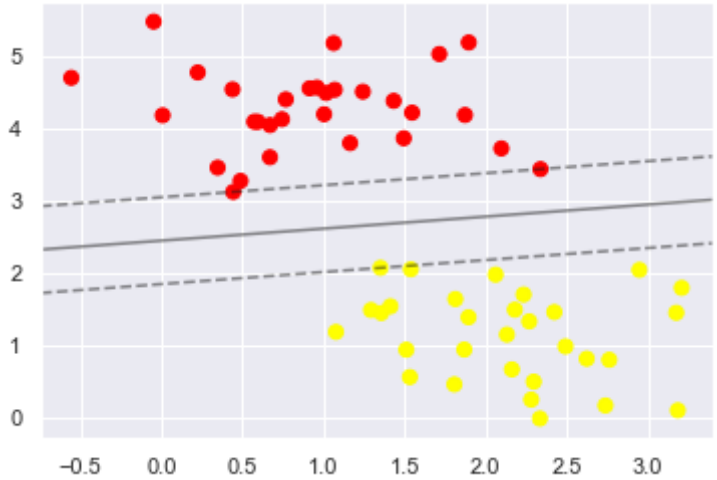
University of Reading

- A key to this classifier's success is that for the fit, only the position of the support vectors matter
- Any points further from the margin which are on the correct side do not modify the fit! Technically, this is because these points do not contribute to the loss function used to fit the model
- Therefore we plot the model learned from the first 60 points and first 120 points of this dataset, the model will still be the same

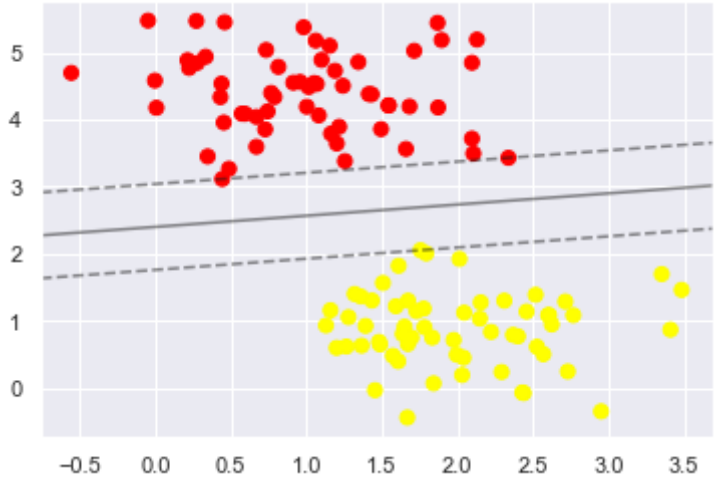
```
In [18]: model.support_vectors_
```

```
Out[18]: array([[0.44359863, 3.11530945],
                [2.33812285, 3.43116792],
                [2.06156753, 1.96918596]])
```

In [19]:
```
X, y = make_blobs(n_samples=60, centers=2, random_state=0, cluster_std=0.60)

model = SVC(kernel='linear', C=1E10)
model.fit(X, y)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(model);
```



In [20]:
```
X, y = make_blobs(n_samples=120, centers=2, random_state=0, cluster_std=0.60)

model = SVC(kernel='linear', C=1E10)
model.fit(X, y)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(model);
```



Example of support vector machine classification using the Wiconsin Breast Cancer dataset

In [21]:
```
from sklearn.svm import SVC # "Support vector classifier"

X = cancer.data
y = cancer.target

print (X.shape)
print (y.shape)
```

```
(569, 30)
(569,)
```

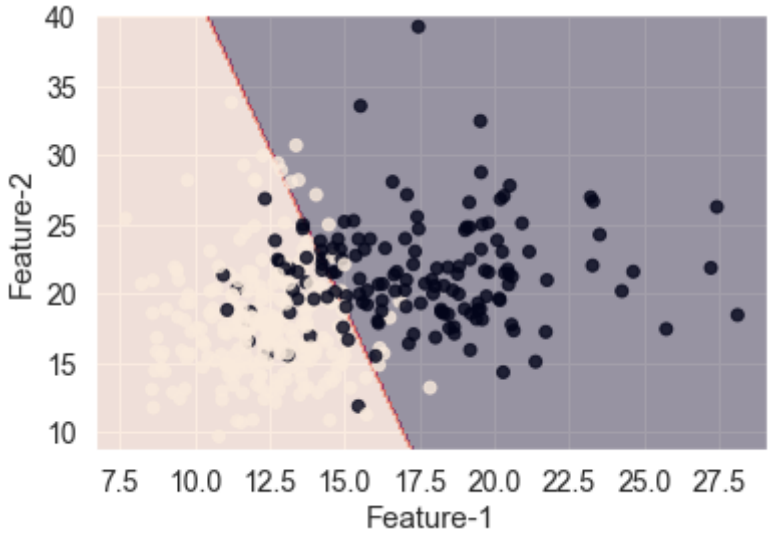
In [22]:
```
#Split into traing/test holdout dataset
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=42)
```

In [23]:
```
model = SVC(kernel='linear')
model.fit(X_train, y_train)
```

Out[23]:
```
SVC(kernel='linear')
```

In [24]:
```
plot_decision_boundaries(X_train,y_train, SVC, kernel='linear')
```

Out[24]:
```
<module 'matplotlib.pyplot' from 'C:\\Users\\Miguel\\anaconda3\\envs\\base2\\lib\\site-packages\\matplotlib\\pyplot.p
y'>
```



In [25]:
```
y_pred = model.predict(X_test)
```

In [26]:
```
#Plot the confusion matrix
confusion_matrix(y_test, y_pred)
```

Out[26]:
```
array([[ 59,   4],
       [  2, 106]], dtype=int64)
```

In [27]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.94      0.95        63
           1       0.96      0.98      0.97       108

    accuracy                           0.96       171
   macro avg       0.97      0.96      0.96       171
weighted avg       0.96      0.96      0.96       171
```

In [28]:
```
X_new = [[ 17.99,  10.38,  122.8,  1001, 0.1184, 0.2776, 0.3001, 0.1471, 0.2419, 0.07871, 1.095, 0.9053, 8.589, 153.4, 0
          0.4904, 0.05373, 0.01587, 0.03003, 0.006193, 25.38, 17.33, 184.6, 2019, 0.1622, 0.6656, 0.7119, 0.2654, 0.4601

print('The predicted class is:',model.predict(X_new))
```

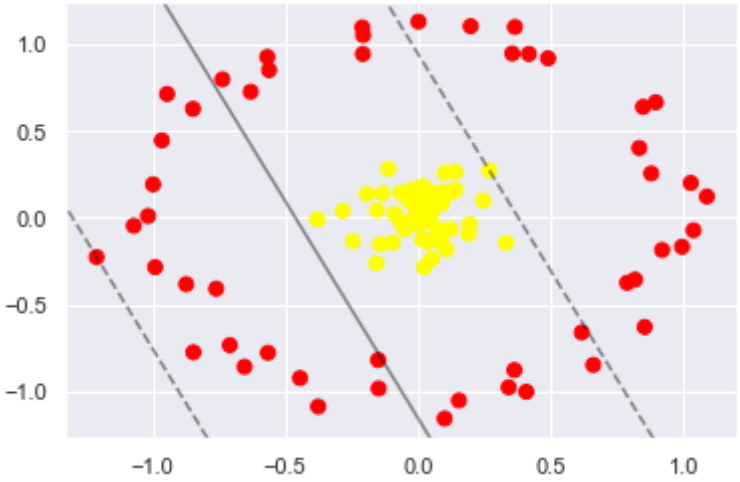
```
The predicted class is: [0]
```

University of Reading

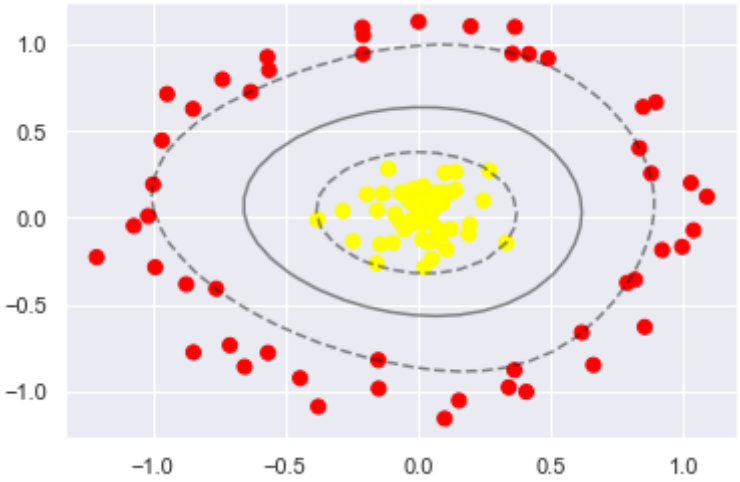
Beyond Linear Boundaries

- What makes support vector machines so powerful is when they are combined with kernels
- This is where data is projected into higher dimensional space defined by polynomials and Gaussian processes
- This is used to fit non-linear relationships with a linear classifier

In [29]:
```
from sklearn.datasets import make_circles
X, y = make_circles(100, factor=.1, noise=.1)

clf = SVC(kernel='linear').fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(clf, plot_support=False);
```

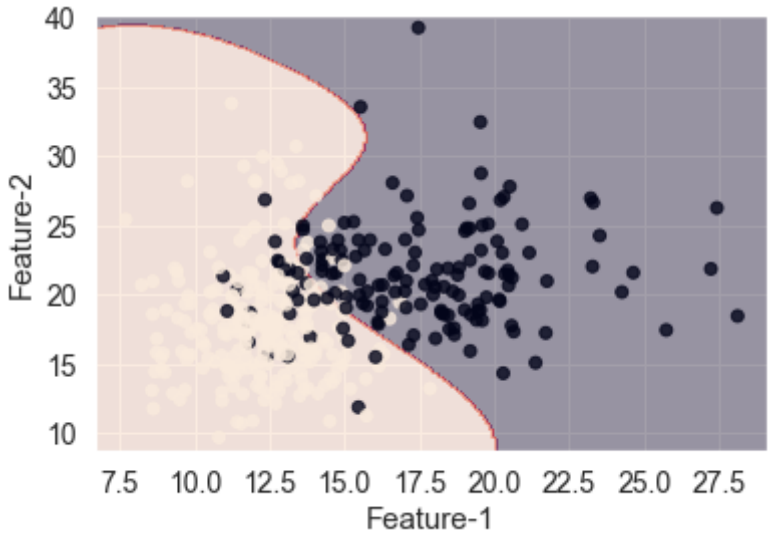


In [30]:
```
clf = SVC(kernel='rbf', C=1E6)
clf.fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(clf)
plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1],
            s=300, lw=1, facecolors='none');
```



In [31]:
```
model = SVC(kernel='rbf')
model.fit(X_train, y_train)
plot_decision_boundaries(X_train,y_train, SVC, kernel='rbf')
```

Out[31]:
```
<module 'matplotlib.pyplot' from 'C:\\Users\\Miguel\\anaconda3\\envs\\base2\\lib\\site-packages\\matplotlib\\pyplot.py'>
```



Rules of thumb for SVMs

University of Reading

- **Model is resistant high dimensionality:** Only points near the margin affect the model, the algorithm works well with high dimensionality data

- **Numerical Inputs**: SVM assumes that your inputs are numeric. If you have categorical inputs you may need to covert them to binary dummy variables (one variable for each category).

- **Poor scaling for Large datasets:** The scaling with the number of samples N is $\theta[N^3]$ at worst or $\theta[N^2]$ for efficient implementations

University of Reading

Data Science Methodology

In order to implement machine learning algorithms on a dataset there are a number of steps to follow:

1. Choose the class of model
2. Choose the model hyperparameters
3. Arrange the data into independent features and the target vector
4. Fit the model to the data
5. Predict labels for unknown data

University of Reading

Summary

- In this lecture we have covered:
  - Supervised learning using Classification Algorithms
  - The application of classification methods in the real world
  - Introduced 2 discriminative classification algorithms: Logistic regression and support vector machines
  - The confusion matrix and the evaluation metrics for classification models
  - How to implement logistic regression and SVMs using scikit learn
  - Rules of thumb for using these algorithms on real data
  - The steps involved data science methodology

In [ ]: