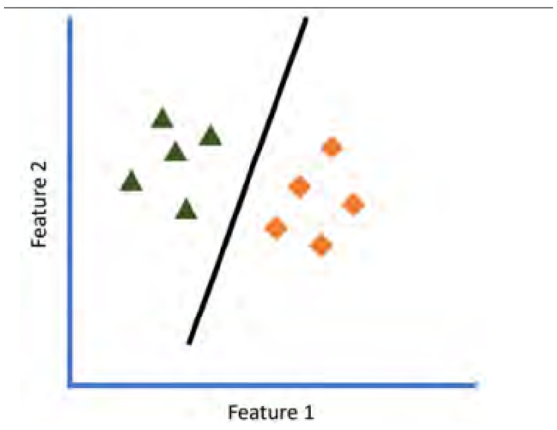
Support Vector Machine (SVM):

It is possible to use SVMs both for regression and classification, which are often referred to as support vector regressions (SVRs) and support vector classifiers (SVCs).

In classification tasks, SVMs allow us to mathematically find the best separation between groups of data.



A simple line is enough to separate groups in two dimensions, but if we are going to deal with higher dimensions (more than two), then we need to find the optimal “hyperplane” to separate the data, the hyperplane can be defined using the following equation:

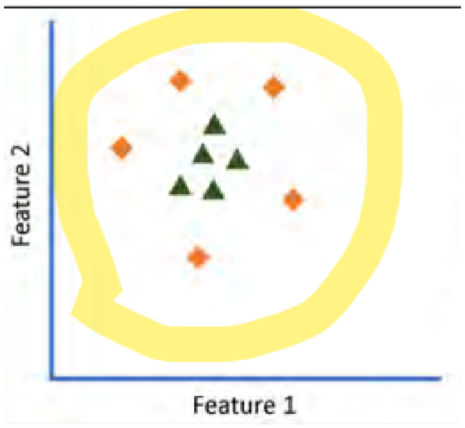
$$w.x+b = 0$$

Where w is a vector of coefficients, x is a vector of our features, and b is a constant.

The nearest points end up defining the hyperplane, and these nearest points form the support vectors, the basis of the name support vector machine.

All of this works by maximising the distance between the hyperplane and the nearest points of opposite classes. Following training, the points closest to the hyperplane are stored in memory as support vectors. Through optimization, this hyperplane is found iteratively

To separate two concentric circles of points, we need to transform the data to higher dimensions using the kernel trick.



There are four common kernels:

- Linear – the first example we saw
- Polynomial – can work for slightly more complex data
- Radial basis function (RBF) – works for very complex data
- Sigmoid – can work for complex data

Advantages of SVMs:

- They work well with a high number of dimensions (many features).
- They are memory-efficient, since they only use a subset of datapoints to classify new ones.
- Using kernels to transform data can make them more flexible for higherdimension and complex feature space.

Disadvantages of SVMs:

- Vanilla SVM implementations do not scale well with increasing features and datapoints (although there are implementations for big data with Spark and other software).
- Probability estimates for class predictions need to be found with crossvalidation, which is computationally expensive.

XGBoost:

First of all, we need to know what gradient boosting is before diving into the XGBoost methodology.

Machine learning techniques such as gradient boosting are used in regression and classification tasks, among others. Gradient-boosted trees usually outperform random forests when a decision tree is the weak learner; it gives a prediction model in the form of an ensemble of weak prediction models, which are typically decision trees. Gradient-boosted trees models are built the same way as other boosting methods, but allow optimization of an arbitrary differentiable loss function.

However, gradient boosting lacks the proper intelligibility and interpretability.

In contrast to plain gradient boosting, XGBoost uses Newton boosting to make several improvements. As opposed to finding the optimal multiplier to scale each weak learner by (which represents a step length in our gradient descent), XGBoost solves the direction and step length simultaneously, so we can say that XGBoost is faster than gradient boosting.

Despite those improvements, XGBoost has some issues, like:

- Not performing so well on sparse and unstructured data.
- Very sensitive to outliers.
- It’s hardly scalable.