NYC DATA SCIENCE ACADEMY

Python Machine Learning Class 4: Classification 2

NYC Data Science Academy

Outline

- **Support Vector Machines**
  - Separating Hyperplanes
  - The Support Vector Classifier
  - Kernels
- **Tree-Based Methods**
  - Decision Trees
  - Bagging and Random Forests

Support Vector Machines

- **Support vector machines** (SVMs) are supervised learning methods used for classification analysis.
- Unlike linear discriminant analysis or logistic regression, **SVMs** approach the two-class classification problem in a direct way: constructing linear/nonlinear decision boundaries, by explicitly separating the data into two different classes as completely as possible.
- The linear decision boundaries in **SVMs** are **hyperplanes** in the feature space.
- The non-linear decision boundaries in the general **SVMs** are **hypersurfaces** in the feature space.

Outline

- Support Vector Machines
  - **Separating Hyperplanes**
    - The Support Vector Classifier
    - Kernels
- Tree-Based Methods
  - Decision Trees
  - Bagging and Random Forests

Hyperplanes

- A **hyperplane** of a *p-dimensional* Euclidean space *V* is an affine linear subspace of dimension *p*-1, which can be described by a single linear equation of the form (in Cartesian coordinates):

$$\beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p = 0$$

- It is more convenient to write the equation above in a matrix notation:

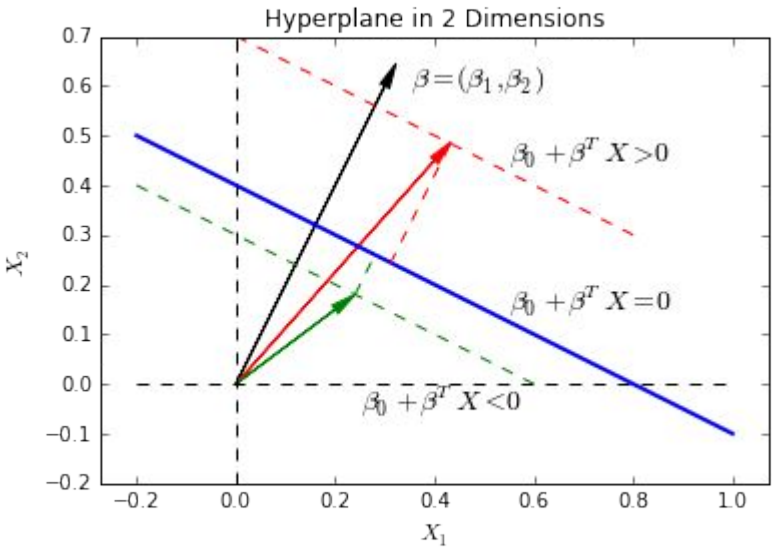
$$\beta_0 + X\beta^T$$

where $\beta = (\beta_1, \ldots \beta_p)$ and $X = (X_1, \ldots X_p)$ are *p-dimensional* vectors.

- In a 2-dimensional space, a hyperplane is nothing but a line and in a 3-dimensional space it is a plane.

Hyperplanes

- The coefficient vector β is the called **normal vector** - a vector orthogonal to all the movements within that hyperplane.
- In some cases we need to work with the normalized form: $\beta^* = \beta/|\beta|$ or equivalently, to require that $\sum_i^p \beta_i^2 = 1$.



Hyperplanes

- Here are some nice properties:
  - For any point x_0 lying in the hyperplane,

$$\beta^T x_0 = -\beta_0$$

  - The signed distance of any given point *x* to the hyperplane is given by:

$$f(x) = \frac{1}{|\beta|}(\beta^T x + \beta_0)$$

  - The signed distance function f can be used as the decision function of the classification (explained below).

Separating Hyperplanes

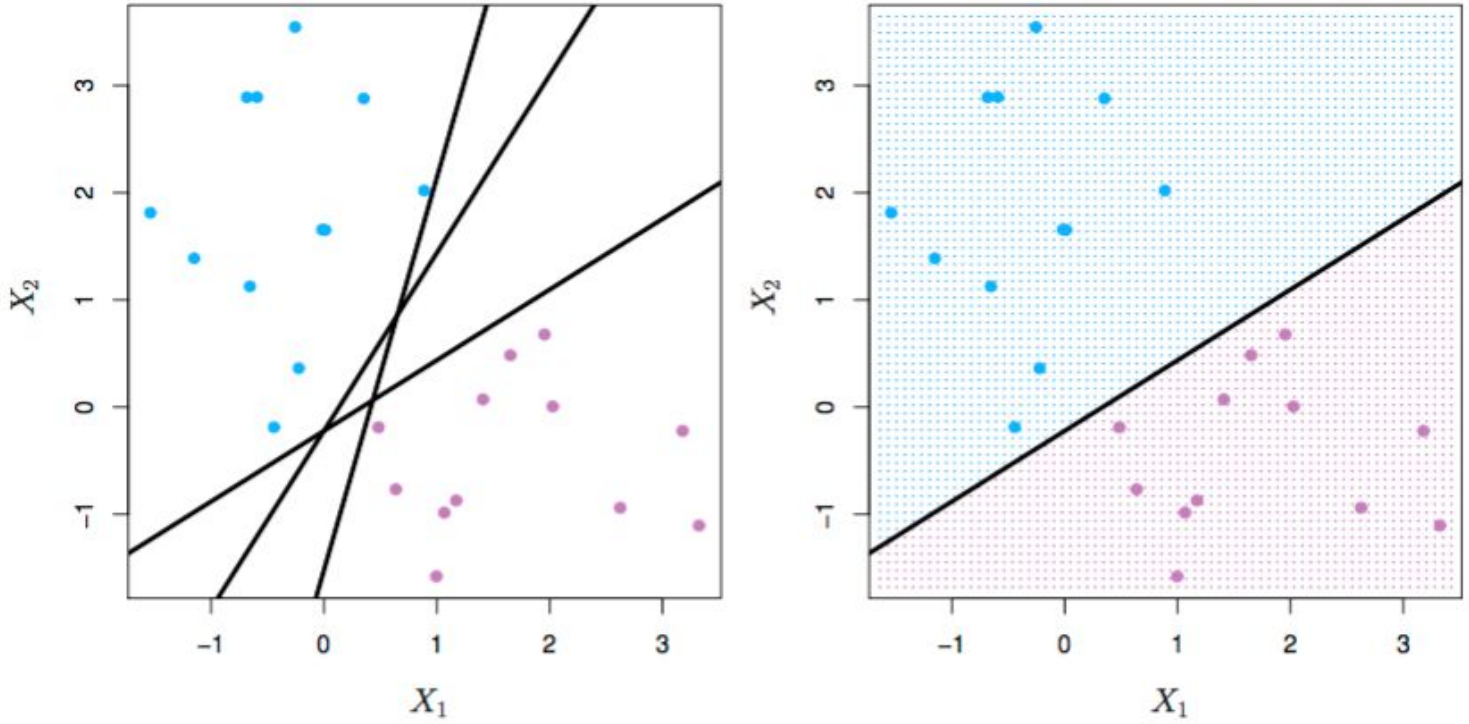
- If we assume that the data can be *well separated by a hyperplane* defined by $f(X) = \beta_0 + \beta^T X = 0$, then:
  - $f(X) > 0$, for points on one side of the hyperplane,
  - $f(X) < 0$, for points on the opposite side.
- If we code the two classes as:
  - $y = 1$, for $f(X) > 0$, and
  - $y = -1$, for $f(X) < 0$,

  then the distance multiplying the class becomes always positive:

$$y_i \cdot f(X_i) > 0$$

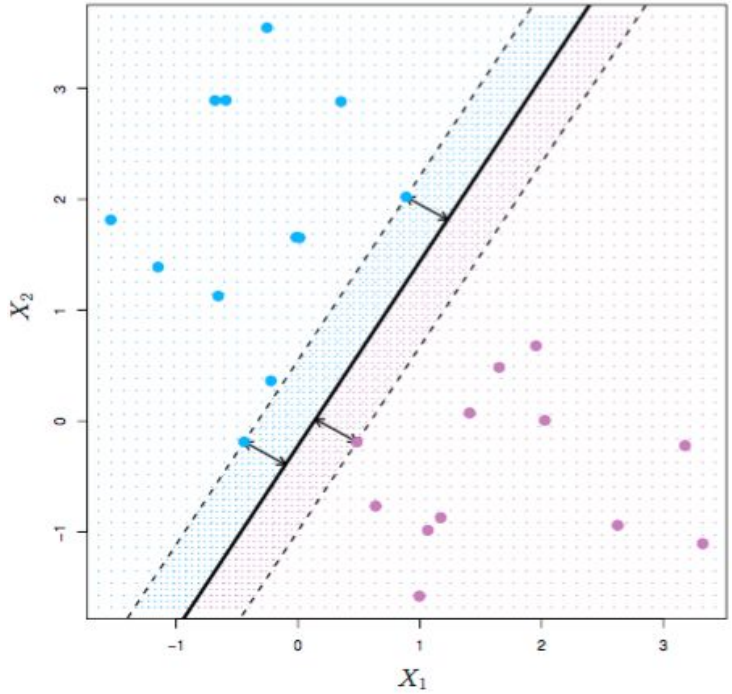
Separating Hyperplanes

- The figure on the left shows three of the infinitely many possible separating lines; the figure on the right shows that the values of function $f(X)$ on the two sides of the hyperplane are of opposite signs.



Optimal Separating Hyperplanes

- Goal: to maximize the **margin**, defined by the minimal distance from the data points to a hyperplane, between the two classes on the training data.



- The data points that are used to determine the margins are called *support vectors*.

Maximal Margin Classifier

- The optimal separating hyperplane leads to a constrained optimization problem on margin *M*:

$$\max_{\beta_0, |\beta|=1} M$$

$$\text{subject to } y_i(x_i^T\beta + \beta_0) \geq M, i = 1, ..., N$$

- The conditions ensure that the distances from all the points to the decision boundary specified by β and β_0 are at least *M*, and we seek the largest *M* by varying the parameters.
- We can get rid of the constraint $|\beta| = 1$ by replacing the inequalities with:

$$y_i(x_i^T\beta + \beta_0) \geq M|\beta|$$

Maximal Margin Classifier

- For any β and β_0 satisfying the inequalities, any positively scaled multiple satisfies them too.
- If we set $|\beta| = 1/M$, we can rephrase the original problem to a more elegant form by dropping the norm constraint on β:

$$\min_{\beta_0,\beta} \frac{1}{2}|\beta|^2$$

$$\text{subject to } y_i(x_i^T\beta + \beta_0) \geq 1$$

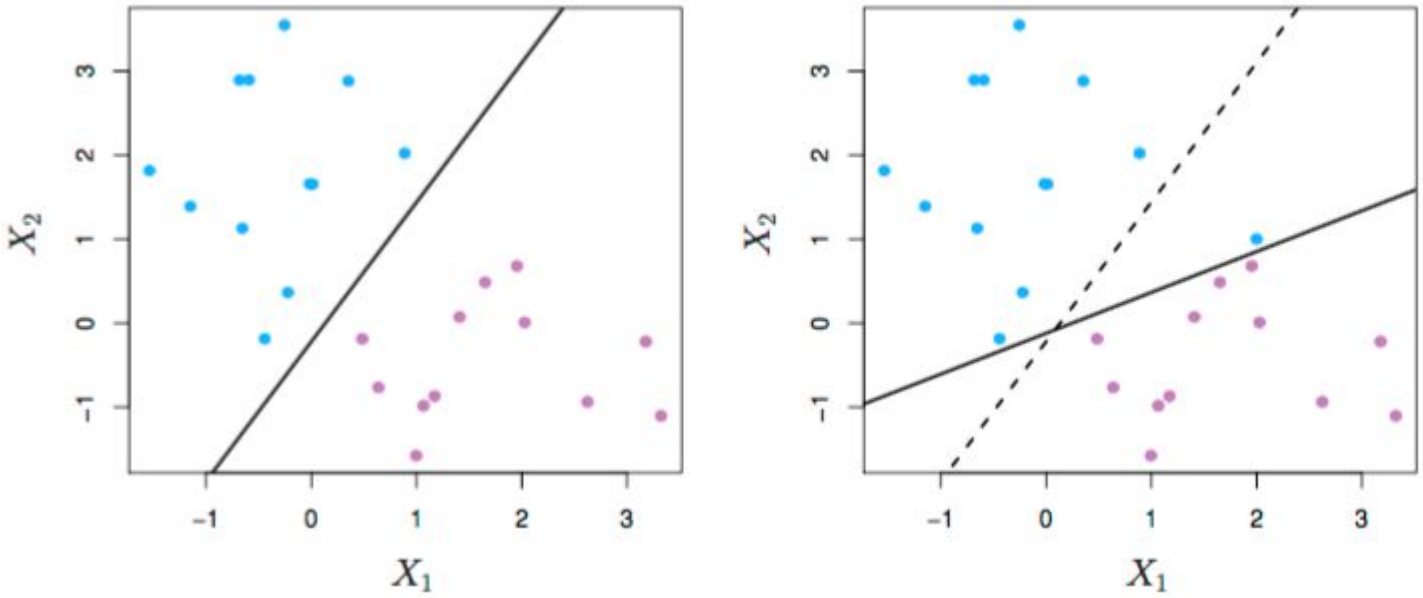
- This is a convex quadratic optimization problem, and can be solved more easily than a generic quadratic optimization problem.

Outline

- Support Vector Machines
  - Separating Hyperplanes
  - **The Support Vector Classifier**
  - Kernels
- Tree-Based Methods
  - Decision Trees
  - Bagging and Random Forests

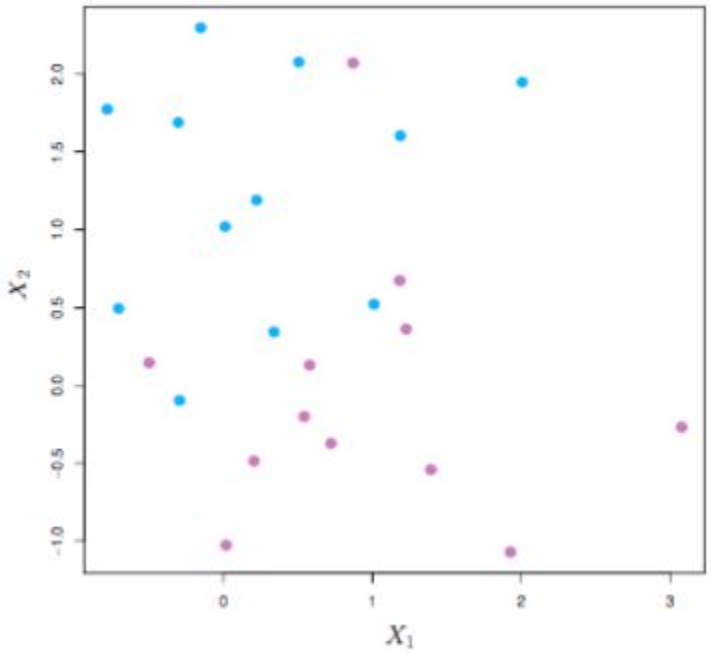
Noisy Data

- The technique for constructing an optimal separating hyperplane can be applied to cases of two perfectly separable classes.
- However, sometimes the data can be noisy, which can lead to a poor solution for the maximal margin classifier. (Note the one additional blue point on the right.)



Non-separable Data

- Even worse, quite often the data is not separable by a linear boundary.



- What shall we do?

The Support Vector Classifier

- To tolerate noise and classification errors, we maximize *M* but allow some points to be on the wrong side of the decision hyperplane.
- We introduce "slack" variables $\epsilon = (\epsilon_1, \ldots \epsilon_N)$ with constraints $\epsilon_i \geq 0$ and $\sum_i \epsilon_i \leq \text{Const}$. We modify the original optimization problem to be:

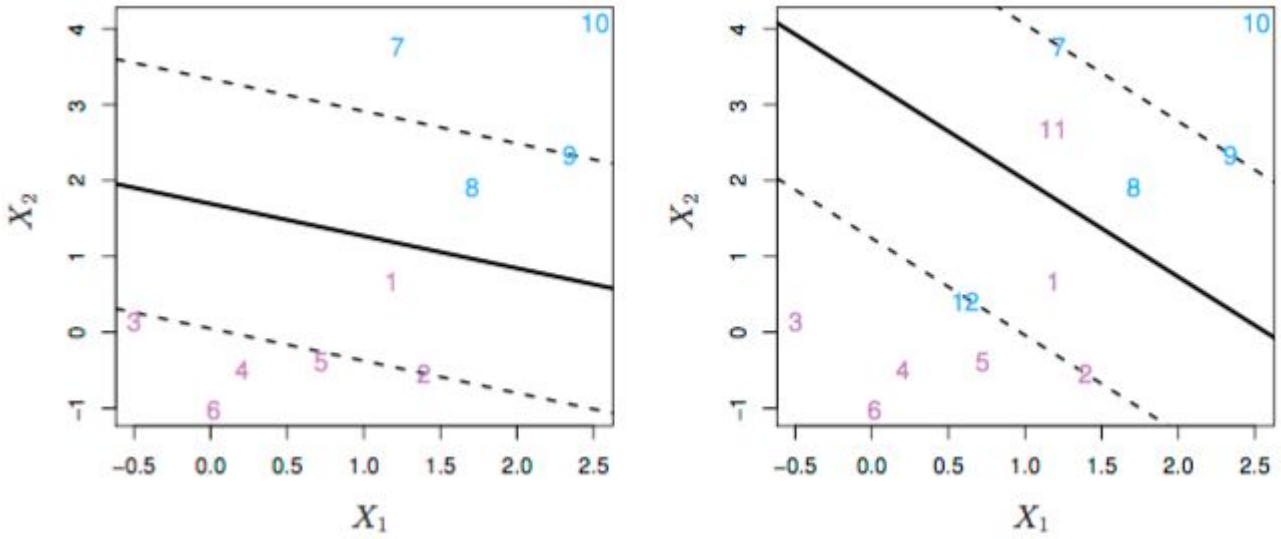
$$\max_{\beta_0, \epsilon, |\beta|=1} M$$

$$\text{subject to } \begin{cases} y_i(x_i^T \beta + \beta_0) \geq M(1 - \epsilon_i) \\ \epsilon_i \geq 0, \text{ and } \sum_i \epsilon_i \leq \text{Const} \end{cases}$$

  - ϵ_i are proportional to the degree by which the prediction is on the wrong side of their margin.
  - Misclassifications occurs when $\epsilon_i > 1$.

The Support Vector Classifier

- The data points that fall within the margin are penalized by the slack variables ϵ.



The Support Vector Classifier

- Computationally, it's convenient to use the following equivalent form:

$$\min_{\beta_0,\beta} \left(\frac{1}{2}|\beta|^2 + C\sum_{i=1}^{N} \epsilon_i\right)$$

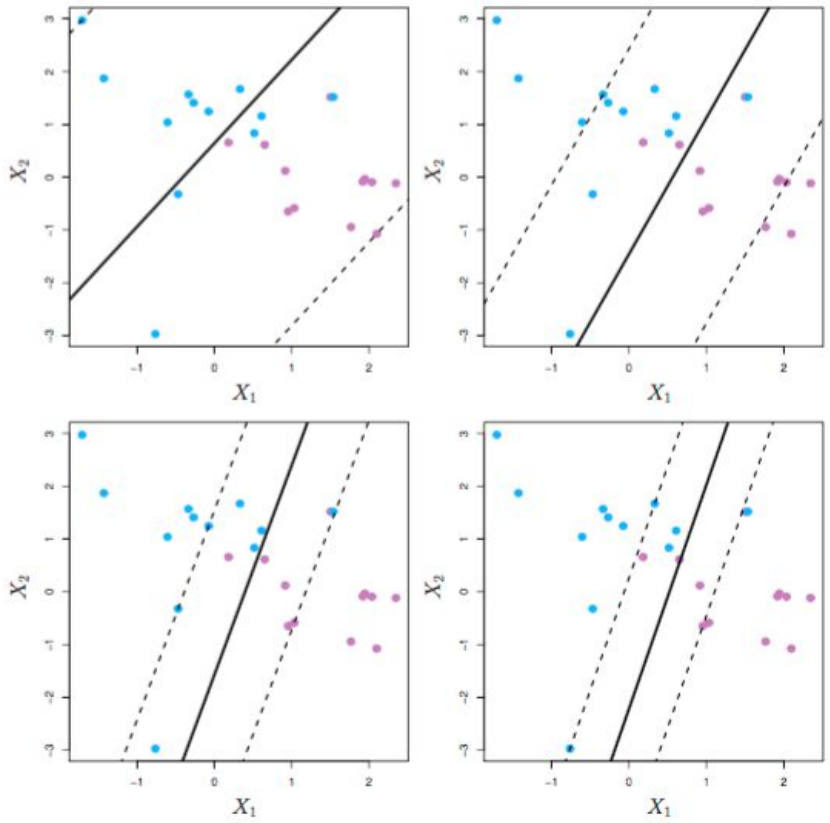
$$\text{subject to } \begin{cases} y_i(x_i^T\beta + \beta_0) \geq 1 - \epsilon_i \\ \epsilon_i \geq 0 \end{cases}$$

  where *C* is the penalty parameter of the total error term.

- The maximum margin classifier corresponds to $C = \infty$.

The Support Vector Classifier

- If *C* is close to 0 then we have a *wide*, *soft margin*.
- If *C* is large then we are close to the *hard-margin* formulation.

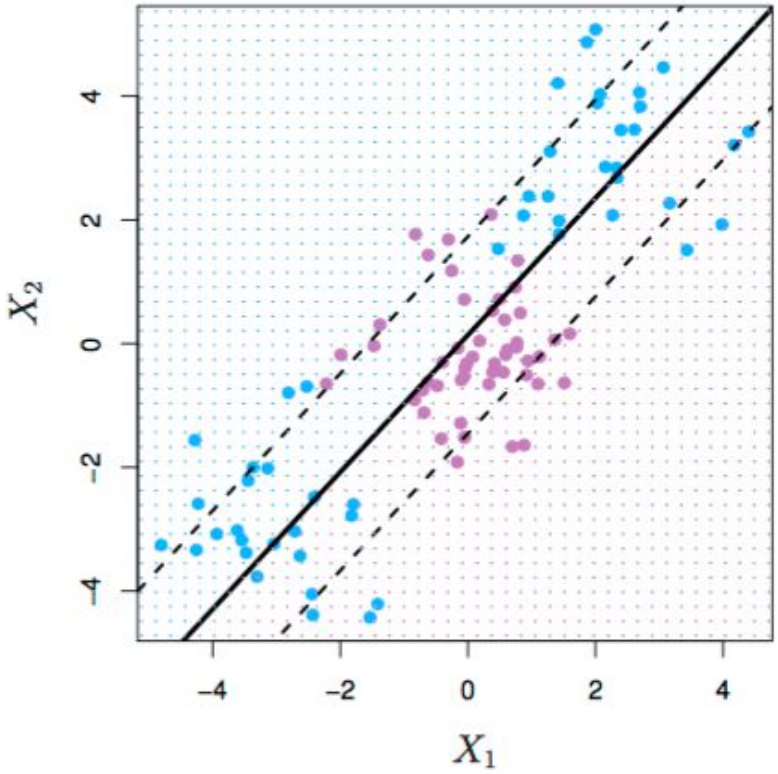


Outline

- Support Vector Machines
  - Separating Hyperplanes
  - The Support Vector Classifier
  - **Kernels**
- Tree-Based Methods
  - Decision Trees
  - Bagging and Random Forests

Beyond Linearity

- The support vector classifier described so far finds linear decision boundaries in the feature space.
- In reality, it's very unlikely that the true boundary is actually linear in *X*.



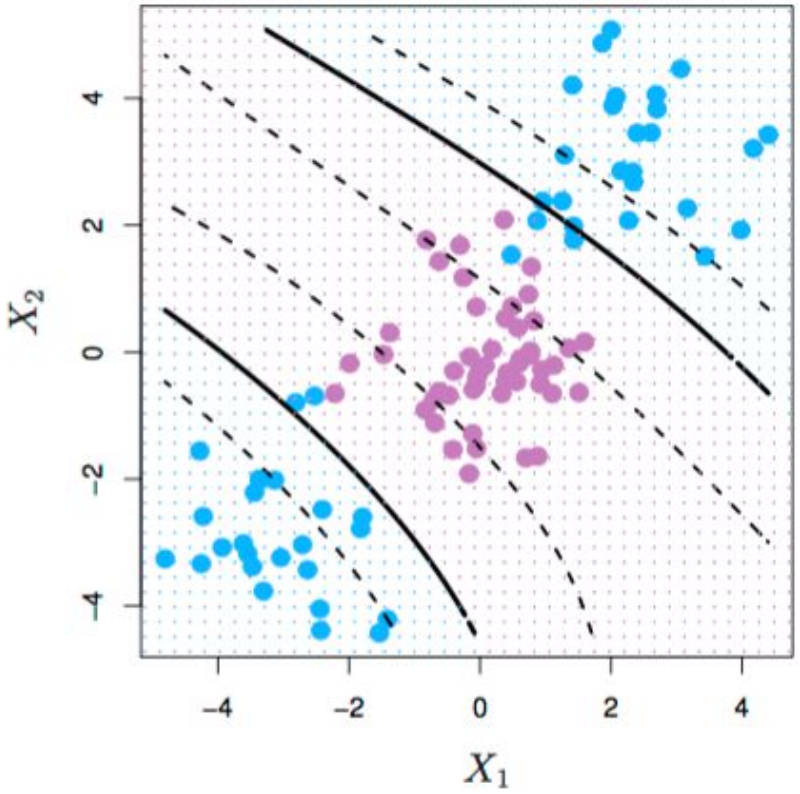
Basis Expansions

- If searching for linear boundary fails, we need to move beyond linearity.
- The core idea can be summarized as:
  - Expand the features *X* using basis expansions such as polynomials,
  - use linear models in the enlarged space of derived input features,
  - then converts to nonlinear boundaries in the original space.
- Example:
  - We enlarge the feature space (X_1, X_2) to $(X_1, X_2, X_1^2, X_2^2, X_1X_2)$
  - The decision boundary is then determined by:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2, + \beta_3 X_1^2 + \beta_4 X_1 X_2 + \beta_5 X_2^2 = 0$$

Basis Expansions

- Back to the previous example, a basis expansion of cubic polynomials will create a nonlinear boundary in the original feature space.



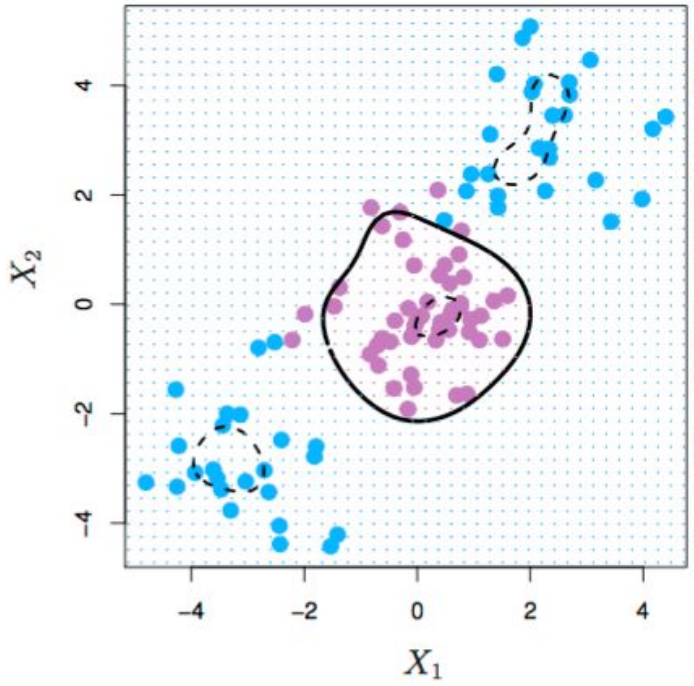
Kernels

- Some popular choices for SVM kernel functions are:
  - *d*th-Degree polynomial: $K(x, x') = (1 + \langle x, x' \rangle)^d$
  - Radial basis: $K(x, x') = \exp(-\gamma |x - x'|^2)$
- By using the kernel function, the classification function can be written as:

$$f(x) = \beta_0 + \sum_{i=1}^{N} \alpha_i y_i K(x, x_i)$$

Kernels

- The figure below shows the decision boundary with Radial Kernel.



Hands-on Session

- Please go to the **"SVM in Scikit Learn"** in the lecture code.

Outline

- Support Vector Machines
  - Separating Hyperplanes
  - The Support Vector Classifier
  - Kernels
- Tree-Based Methods
  - Decision Trees
  - Bagging and Random Forests

Tree-Based Methods

- Tree based methods partition the feature space into a set of rectangular regions and then fit a simple model (usually the mean in that region) in each subregion.
- Methods we'll talk about include
  - Decision Trees
  - Bagging and Random Forests

Outline

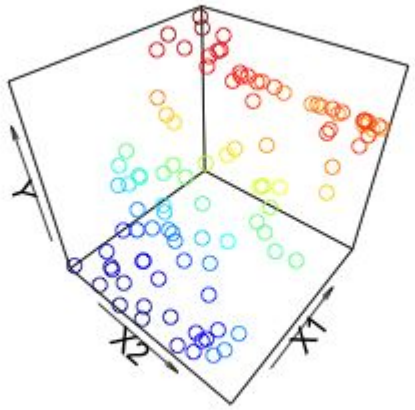
- Support Vector Machines
  - Separating Hyperplanes
  - The Support Vector Classifier
  - Kernels
- Tree-Based Methods
  - **Decision Trees**
  - Bagging and Random Forests

Decision Trees

- Decision trees partition the feature space into a set of simple rectangular regions.
- **Decision trees** can be used for both regression and classification problems.
- We'll first consider an example of regression problem and then focus on classification.

Decision Trees - Regression

- Let's consider a regression problem with predictors X_1 and X_2 and continuous response Y.



- To fit a decision tree, we divide the feature space into non-overlapping regions and in each region R_j we model the response by its mean $\hat{y}_{Rj}$.
- The goal is to find rectangular regions that minimize the RSS:

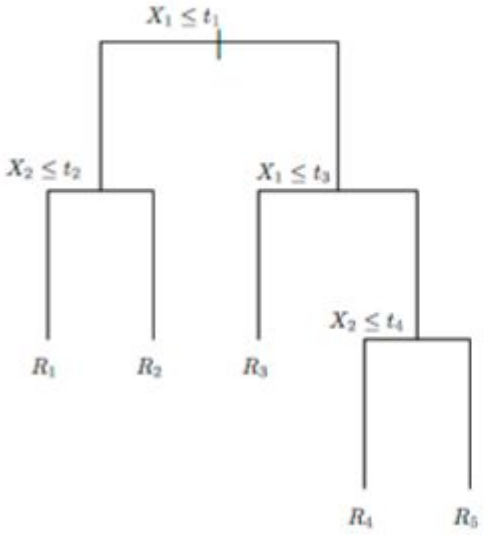
$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

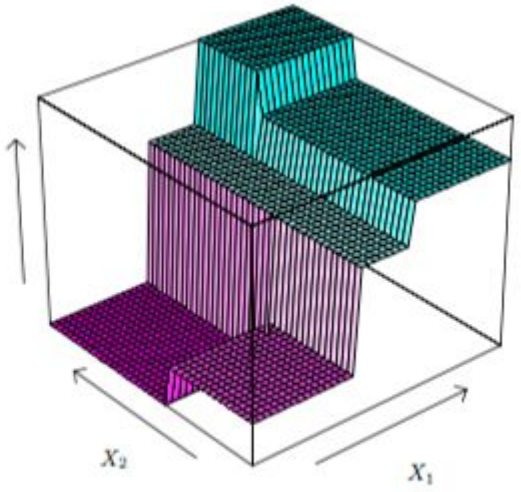
Decision Trees - Regression

- To simplify the process, we restrict the partitioning to recursive binary splittings.
- We first select the predictor X_j and the cutpoint that splits the dataset to the greatest reduction in *RSS*.
- Then we repeat the process within each resulting region (not the entire predictor space) to reduce the overall *RSS*.
- We continue the process recursively until we reach a stopping criterion.
- Finally the decision tree partitions the feature space into a set of regions, each of which is modeled by the mean responses of the data points that fall in that region.

Decision Trees - Regression

- A five-region splitting is achieved.
  1. split the overall dataset at $X_1 = t_1$;
  2. split the region $X_1 < t_1$ at $X_2 = t_2$ and the region $X_1 > t_1$ at $X_1 = t_3$;
  3. split the region $X_1 > t_3$ at $X_2 = t_4$.





Decision Trees - Classification

- Similar to regression, we use the same procedure to fit a classification tree.
- For regression we used the *RSS* to determine the splitting, what should we use for classification? Two commonly used metrics are:
  - **Gini impurity**: the probability a randomly chosen item from the set would be incorrectly labeled if it were randomly labeled according to the distribution of labels in the subset.

$$I_G(f) = \sum_{i=1}^{m} f_i(1 - f_i)$$

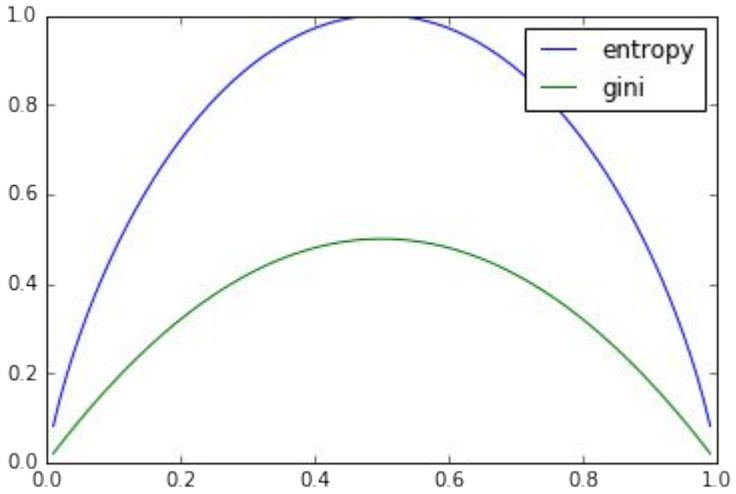
  - **Information Entropy** also works in a similar way:

$$I_E(f) = -\sum_{i=1}^{m} f_i \log_2(f_i)$$

where f_i is the fraction of items labeled with i in the set and $\sum f_i = 1$.

Decision Trees - Gini impurity and Information Gain

- For a binary classification problem, the gini impurity and entropy can be visualized as:



where the horizontal axis refers to the proportion of one of the classes.

Decision Trees - Gini impurity and Information Gain

- Gini impurity is a measure of node purity.
- When using gini impurity as the metric for splitting, our goal is to have two subregions as pure as possible by reducing the weighted sum of gini impurities.
- Similarly, when using information entropy we want the information gain as great as possible after the splitting.
- In practice, the behaviors of the two metrics are very similar.
  - It is often more useful to focus on different pruning cut-offs

Decision Trees

- If we keep sub-dividing the regions until there's only one class in each region, we may end up with *overfitting*: high accuracy on the training dataset, but poor prediction performance.
- A decision tree with fewer splits may lead to lower model variance and better interpretation.
- To prevent overfitting, we can set the threshold upon:
  - the maximal depth of the tree,
  - the minimum number of observations in a tree node to split, or
  - the minimum number of observations in each region (node).

Hands-on Session

- Please go to the **"Decision Tree in Scikit Learn"** in the lecture code.

Outline

- Support Vector Machines
  - Separating Hyperplanes
  - The Support Vector Classifier
  - Kernels
- Tree-Based Methods
  - Decision Trees
  - **Bagging and Random Forests**

Decision Trees Pros and Cons

- Pros:
  - Interpretability: easier to interpret than most other regression methods.
  - Easy to handle qualitative/categorical predictors.
  - Can be displayed graphically.
- Cons:
  - Instability: a small change in the data may result in very different splits.
  - Predictive accuracy usually not as good as other approaches.
- By aggregating many decision trees, the predictive performance can be improved substantially.

Ensemble Learning with Decision Trees

- Ensemble learning is the idea of combining **weak learners** like decision trees to build a more robust model, a **strong learner**

Bagging on Decision Trees

- By averaging a collection of bootstrap (repeated) samples from the training data set, we can dramatically reduce the model variance of trees, leading to improved prediction. This is called **bagging**.
  - For a given B, generate *B* bootstrapped training datasets.
  - For each test observation, we record the class predicted by each of the B decision trees, and choose the majority vote.
  - The overall prediction is the most commonly occurring class among the B predictions.

Random Forests

- The idea for bagging of decision trees is to average many noisy trees and hence reduce the model variance.
- However, since each tree generated in bagging is identically distributed, the expectation value of the averages is the same as the expectation of any one of them. This means the bias will not be improved.
- Random forests is a modification of bagging that builds a large collection of de-correlated trees, and then averages them.

Random Forest Algorithm

The random forest algorithm can be summarized in four simple steps:

1. Draw a random bootstrap sample of size *n* (randomly choose *n* samples from the training set with replacement)
2. Grow each decision tree from the bootstrap sample. At each node:
    a. Randomly select *d* features without replacement.
    b. Split the node using the feature that provides the best split according to the objective function, for instance, by maximizing the information gain.
3. Repeat steps 1 to 2 *k* times.
4. Aggregate the prediction by each tree to assign the class label by **majority vote**.

Hands-on Session

- Please go to the **"Random Forest in Scikit Learn"** in the lecture code.