INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Ceará

HUAWEI

Classification Models

Lecture 02

Douglas Chielle
douglas.chielle@ifce.edu.br

Content

1. Decision Trees
    - How it works
    - CART and ID3
    - Prunning
2. Support Vector Machines (SVM)
    - Linear SVM
    - Nonlinear SVM

Objectives

Upon completion of this lecture, you will be able to:

- Understand the main concepts of Decision Trees and Support Vector Machines;
- Understand how these models perform classification and regression;
- Build these models with scikit-learn and use them for classification or regression.

Decision Trees

Decision Trees

- *Decision Trees* are versatile Machine Learning algorithms that can perform both classification and regression tasks.
- They are probably one of the most intuitive and frequently used data science techniques.
- Decision Trees are also easy to set up and easy to interpret.

How it works

- A decision tree model takes a form of decision flowchart where an attribute is tested in each node.
- At end of the decision tree path is a ***leaf node*** where a prediction is made about the target variable based on conditions set forth by the decision path.
- The nodes split the dataset into subsets.
- These splits are based on the **“purity”** of the data.

Measures of impurity

- The most common measures of impurity used in Decision Trees are the ***Gini index*** and ***entropy***.
- These measures of impurity must satisfy the following conditions:
  - Maximum impurity is reached when all possible classes are equally represented.
  - When only one class is represented, the measure of impurity must be zero.

Evaluating Gini and Entropy

- Entropy:

$$H = -\sum_{k=1}^{K} p_k \log_2(p_k)$$

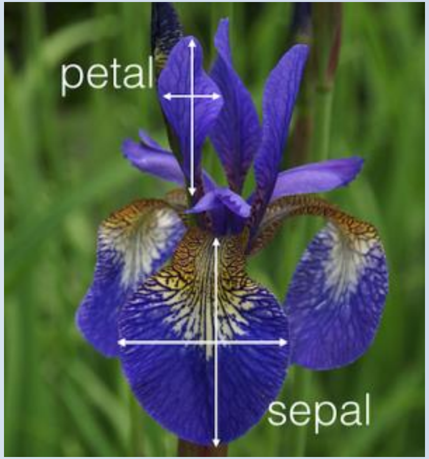
- Gini:

$$G = 1 - \sum_{k=1}^{K} p_k^2$$

  - $k = 1, \cdots, K$ represents the classes of the target variable; and
  - p_k represents the proportion of samples that belong to class k

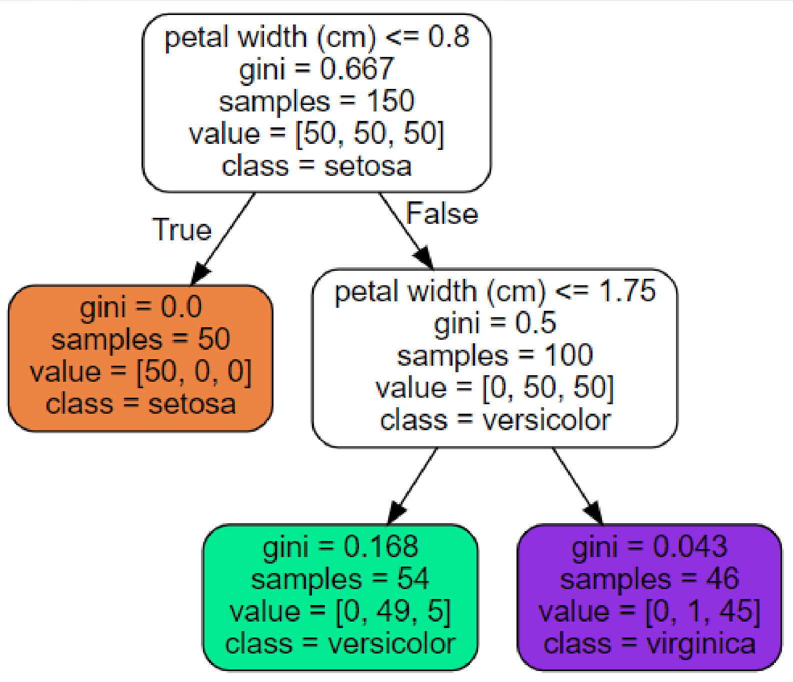
Example

- Let’s build a Decision Tree and see how it makes predictions.
- For this example, we will use the “iris dataset”.
- There are 150 samples in this dataset, 50 in each of the three classes.
- There are four numeric features:
  - sepal length in cm
  - sepal width in cm
  - petal length in cm
  - petal width in cm
- The classes are: *Iris-Setosa*, *Iris-Versicolour* and *Iris-Virginica*



Example

```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
import graphviz

np.random.seed(7)

iris = load_iris()
X = iris.data[:, 2:] # petal length and width
y = iris.target

tree_clf = DecisionTreeClassifier(max_depth=2)
tree_clf.fit(X, y)

dot = export_graphviz(tree_clf,
                feature_names=iris.feature_names[2:],
                class_names=iris.target_names,
                rounded=True, filled=True)

graph = graphviz.Source(dot)
graph
```



The CART Training Algorithm

- Scikit-Learn uses the ***Classification and Regression Tree (CART)*** algorithm to train Decision Trees.
- The algorithm works by first splitting the training set into two subsets using a single feature k and a threshold t_k.
- To select the feature and threshold, it searches for the pair (k, t_k) that produces the purest subsets (weighted by their size).
- Once the CART algorithm has successfully split the training set in two, it splits the subsets using the same logic, then the sub-subsets, and so on, recursively.

The ID3 Training Algorithm

- Another common training algorithm in Decision Trees is the *ID3* (*Iterative Dichotomiser 3*).
- It works similarly to CART. Here are the steps in ID3:
  1. Calculate the entropy of every feature of the training set;
  2. Split the training set into subsets using the feature for which the resulting entropy after splitting is minimized;
  3. Make a node containing that attribute;
  4. Recurse on subsets using the remaining attributes.

Overfitting

- Decision Trees make very few assumptions about the training data.
- If left unconstrained, the tree structure will adapt itself to the training data, fitting it very closely.
- To avoid overfitting, the tree’s growth may need to be restricted or reduced, using a process called *pruning*.
- The tree can be pruned before/during training (*pre-pruning*) or after it (*post-pruning*).

Prunning

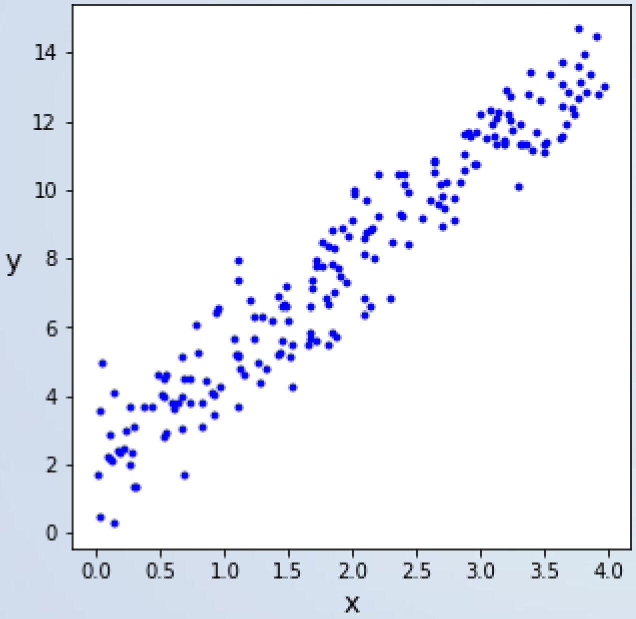
- Pre-pruning can be done by, for example:
  - setting a maximum depth for the tree;
  - setting the minimum number of samples a node must have before it can be split;
  - setting the maximum number of leaf nodes;
- Post-pruning methods do not restrict the tree and allow it to grow as deep as the data will allow. Then, after training, the unnecessary nodes are *pruned.*
- A node whose children are all leaf nodes is considered unnecessary if the purity improvement it provides is not statistically significant.

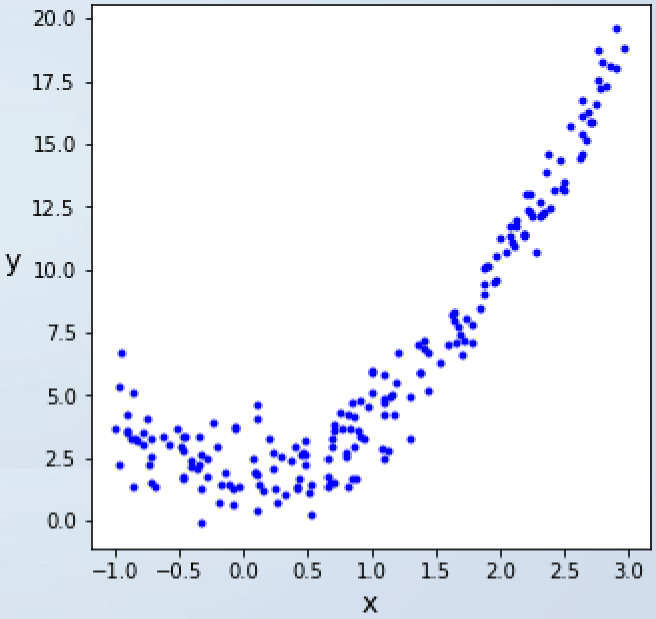
Regression

- Decisions Trees are also capable of performing regression.
- For regression problems, instead of predicting a class in each leaf node, it predicts a value.
- For training, the training algorithm will try to split the training set in order to minimize the Mean Squared Error (MSE), instead of impurity.
- We also must be careful when performing regression with Decision Trees, as it is prone to overfitting.
- Regularization can be done in the same way as in Classification.

Example

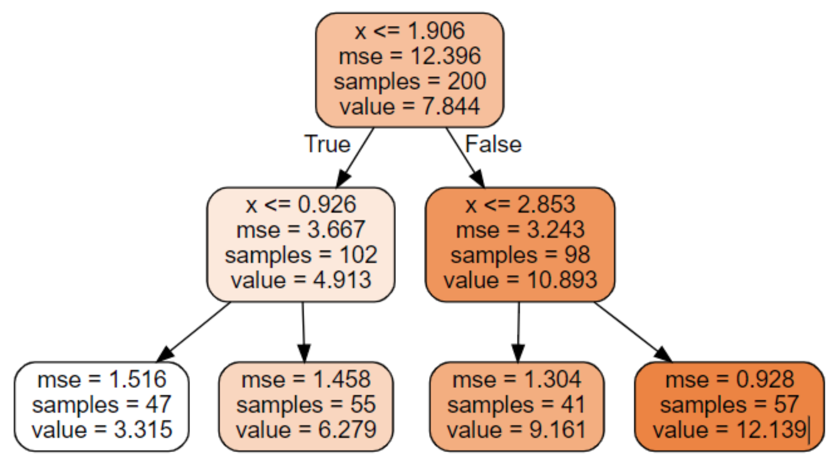
- For the regression examples, we will consider the following datasets:



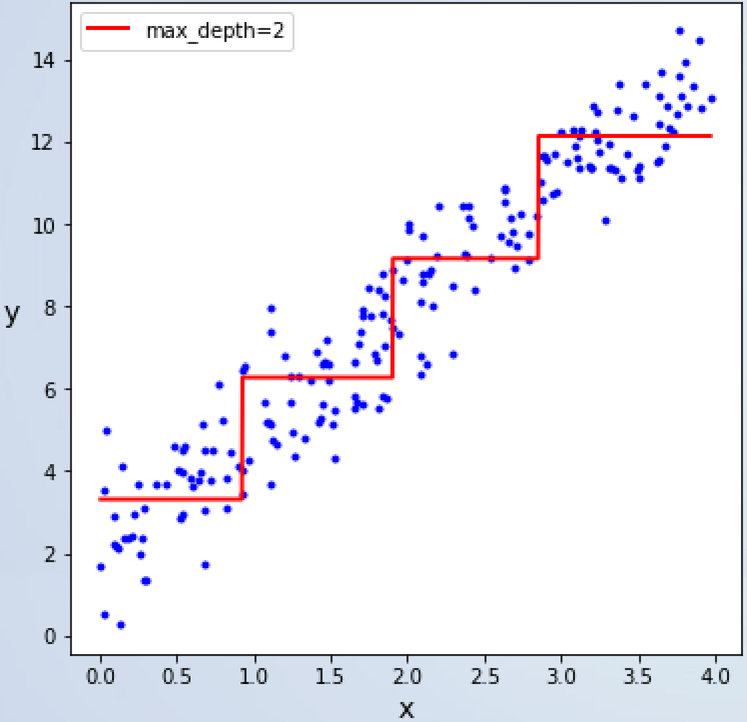


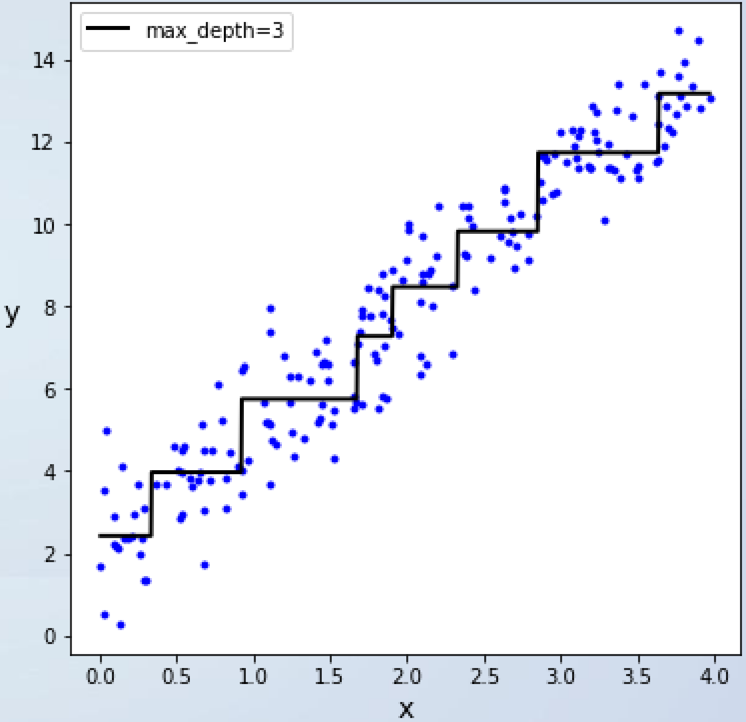
Example: linear data

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(max_depth=2)
tree_reg.fit(x, y)

dot = export_graphviz(tree_reg,
                      feature_names=["x"],
                      rounded=True, filled=True)

graph = graphviz.Source(dot)
graph
```

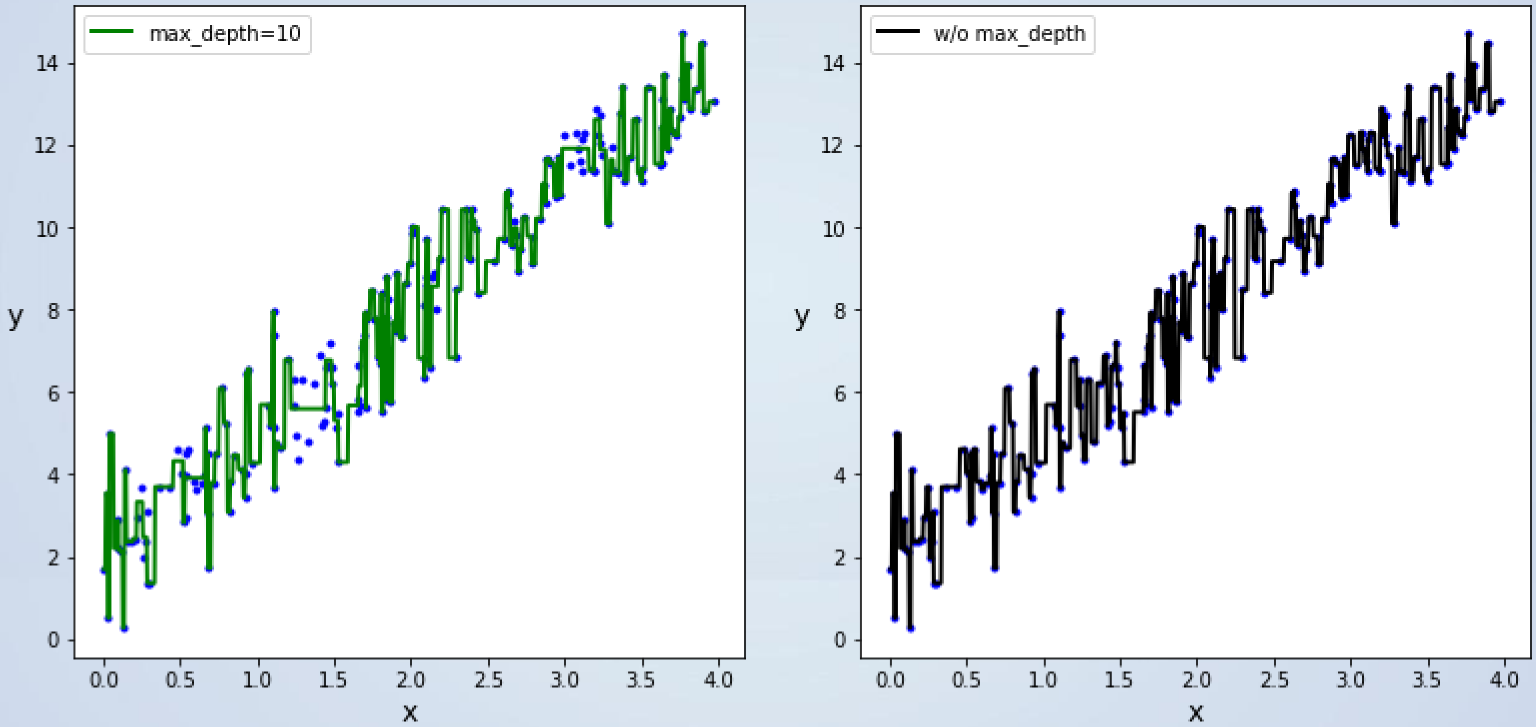


Example: linear data

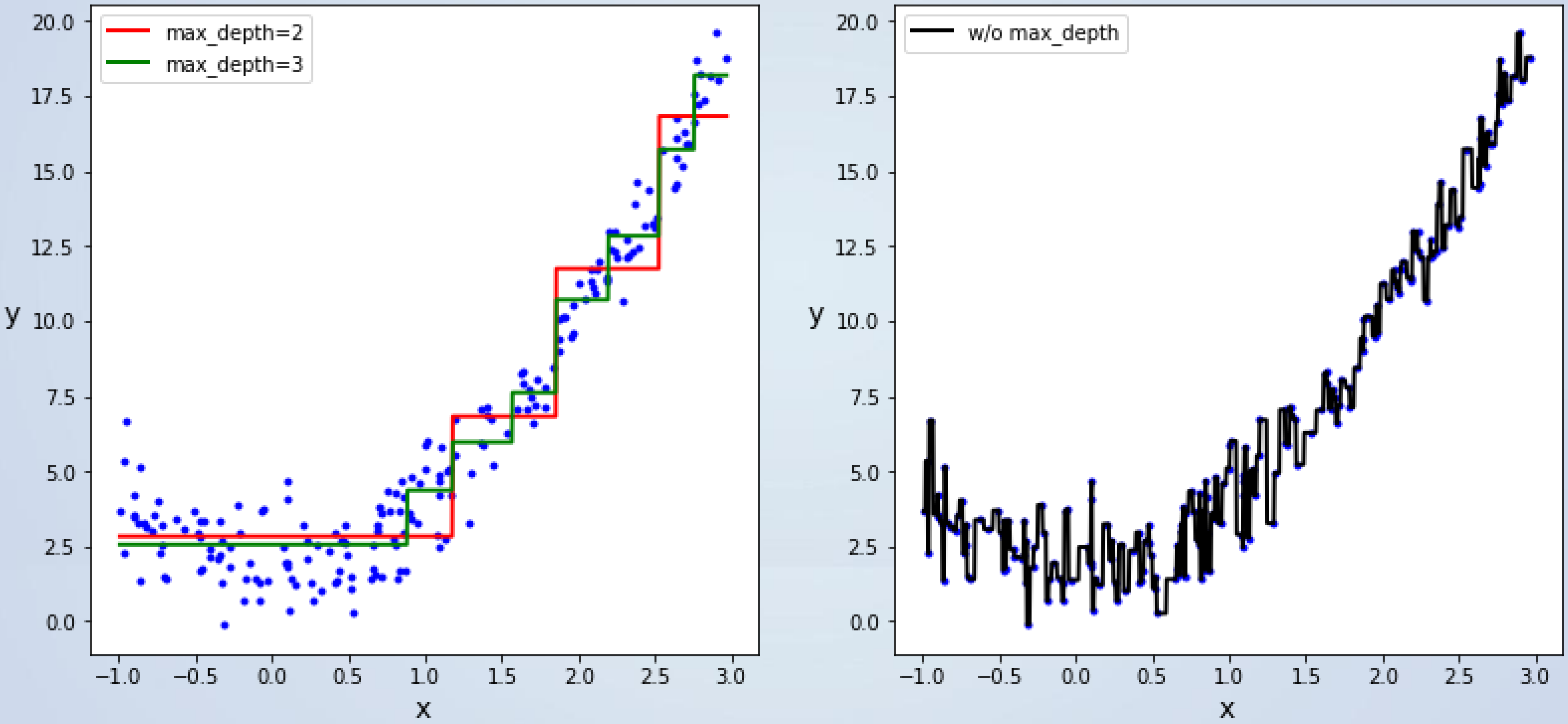




Example: linear data



Example: nonlinear data

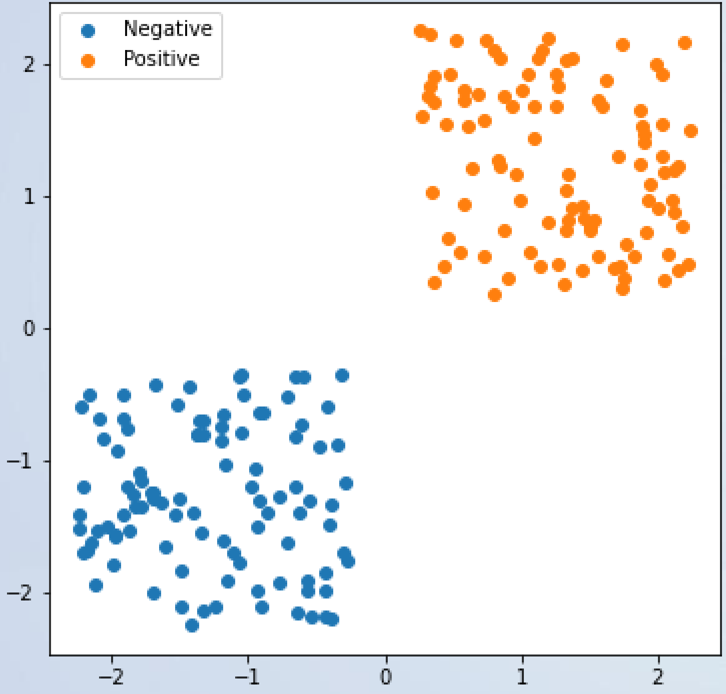


Support Vector Machines

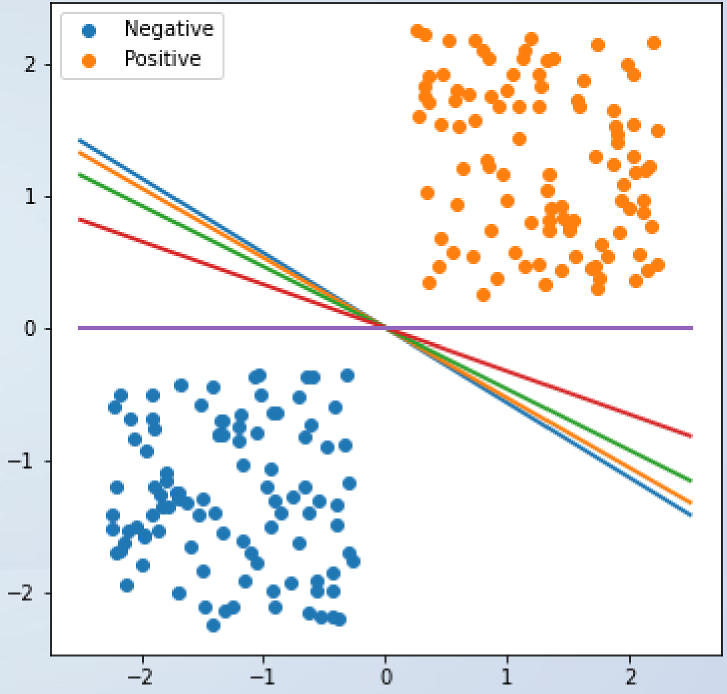
Support Vector Machines

- A ***Support Vector Machine (SVM)*** is a powerful and versatile Machine Learning model, capable of performing linear or nonlinear classification, regression, and even outlier detection.
- Briefly, for binary classification, a SVM performs an implicit nonlinear mapping of the input data to a very high-dimension feature space.
- In this space, a linear decision surface is constructed with special properties that ensure a high generalization ability.

Linear SVM



Negative
Positive
2
1
0
−1
−2
−2
−1
0
1
2

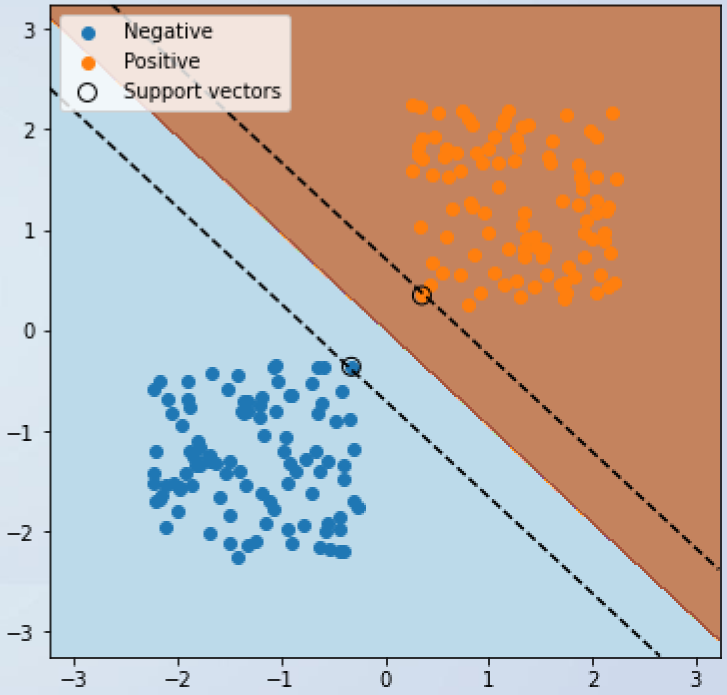


Negative
Positive
2
1
0
−1
−2
−2
−1
0
1
2

Hard Margin SVM

- For a linearly separable dataset, a SVM will look for the hyperplane ($\boldsymbol{w}^T\boldsymbol{x} + b = 0$) that separates the classes with maximum *margin*.
- Margin is the sum of the distance between the hyperplane and the closest elements of each class, and is given by:

$$\rho = \frac{2}{\|\boldsymbol{w}\|}$$



Hard Margin SVM

- Therefore, the optimal hyperplane is the one that minimizes $||\boldsymbol{w}||$.
- To find it, we need to solve the following *quadratic programming problem (QPP)*:

$$\min_{(\boldsymbol{w},b)} \ \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$$
$$s.t. \quad y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1, \qquad i = 1, \cdots, m$$

- This version of SVM has two main problems:
  - It only works with linearly separable data;
  - It is sensible to outliers.

Soft Margin SVM

- To avoid these problems, the addition of slack variables (ξ_i) was proprosed:

$$\min_{(\boldsymbol{w},b)} \quad \frac{1}{2}\,\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{m}\xi_i$$

$$\begin{aligned} s.t. \quad & y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 - \xi_i, && i = 1,\cdots,m \\ & \xi_i \geq 0, && i = 1,\cdots,m \end{aligned}$$

- This formulation describes, for a sufficiently large C the problem constructing a separating hyperplane that minimizes the sum of deviations ξ (for training errors) and maximizes the margin for the correctly classified instances.

Dual problem

- Given a constrained optimization problem, known as the *primal problem*, it is possible to express a different but closely related problem, called its *dual problem*.
- For the SVM problem, the solutions of the primal and dual problems is the same.
- Therefore, we can choose to solve either of them.
- The Lagrange dual problem is:

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j$$

$$s.t. \sum_{i=1}^{m} \alpha_i y_i = 0$$

$$0 \le \alpha_i \le C, \qquad i = 1, \cdots, m$$

  - $\alpha_i \ge 0,\ i = 1, \cdots, m$ are the Lagrange multipliers and each is related to one constraint of the primal problem.

Dual problem

- When dealing with the dual problem, we derive

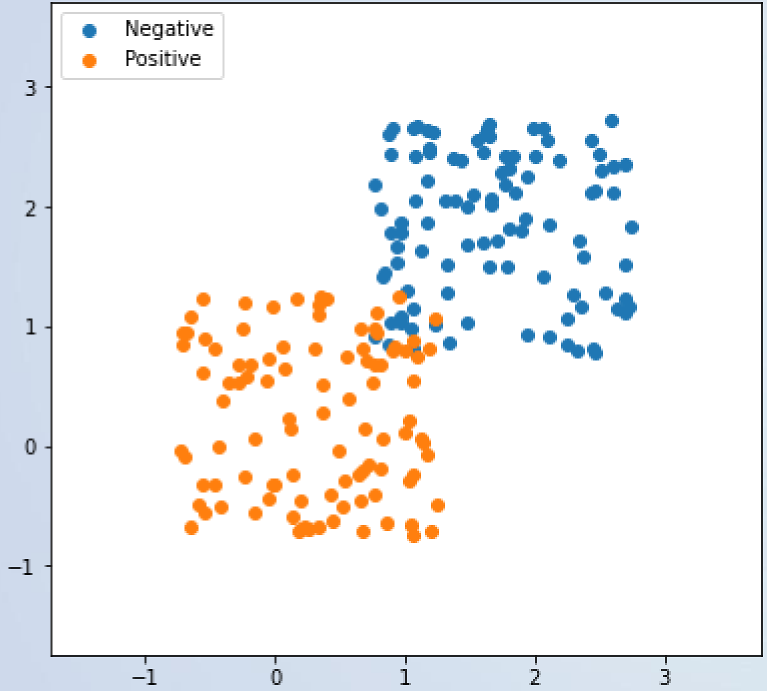
$$w_0 = \sum_{i=1}^{m} \alpha_i^0 y_i \boldsymbol{x}_i$$

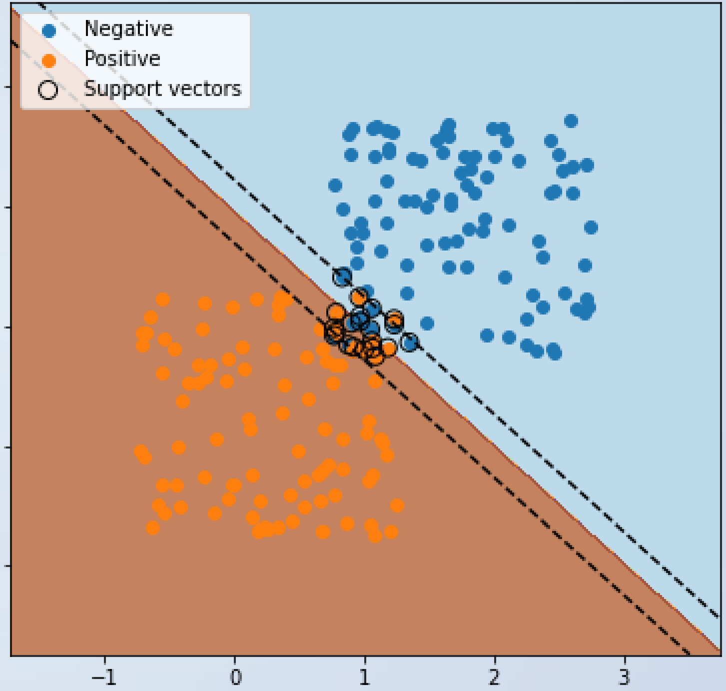
- According to the Karush-Kuhn-Tucker conditions, each Lagrange multiplier and its corresponding constraint are connected by the equation:

$$\alpha_i^0 [y_i (w_0^T x_i + b_0) - 1] = 0, \qquad i = 1, \cdots, m$$

- That means that $\alpha_i^0 \neq 0$ only when $y_i (w_0^T x_i + b_0) - 1 = 0$. The training instances (x_i, y_i) that satisfy this last equation are called ***support vectors***.

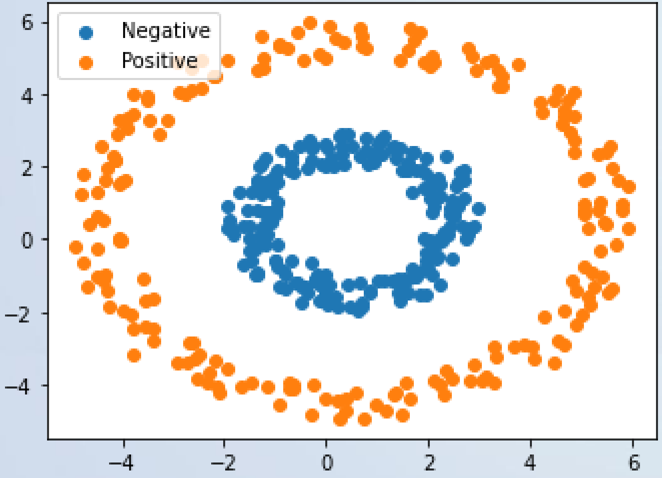
Soft Margin SVM

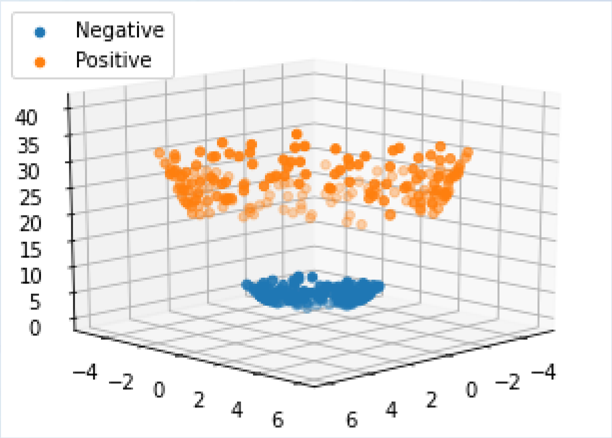




Nonlinear SVM

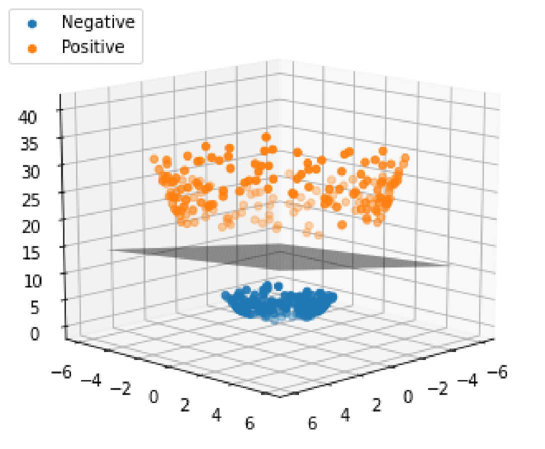
- Now, how to deal with data that is not linearly separable?
- One possibility would be to explictly map the data to a higher dimension and find the optimal separating hyperplane there.

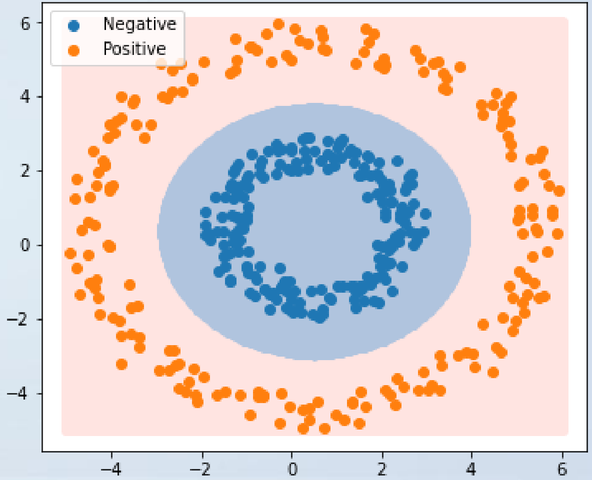




Nonlinear SVM

- Note that the decision surface is nonlinear when projected to the original data space.





- However, this approach has a few drawbacks. One of them is the computational cost, which can be absurdly high depending on them number of features and the choosen transformation.

Kernel trick

- Fortunately, by using *kernel functions*, SVMs can find the optimal separating hyperplane in a high dimension feature space **without** explicitly mapping the data into this space.
- A ***kernel*** $K(\boldsymbol{a}, \boldsymbol{b})$ is a function capable of computing the inner product $\langle \phi(\boldsymbol{a}), \phi(\boldsymbol{b}) \rangle$, based only on the input vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, without having to compute the transformation ϕ.

Commonly used kernels	
Linear	$K(\boldsymbol{a}, \boldsymbol{b}) = \boldsymbol{a}^T \boldsymbol{b}$
Polynomial	$K(\boldsymbol{a}, \boldsymbol{b}) = (\boldsymbol{a}^T \boldsymbol{b} + c)^d$
Gaussian RBF	$K(\boldsymbol{a}, \boldsymbol{b}) = \exp(-\gamma \|\boldsymbol{a} - \boldsymbol{b}\|^2)$
Sigmoid	$K(\boldsymbol{a}, \boldsymbol{b}) = \tanh(\gamma \boldsymbol{a}^T \boldsymbol{b} + c)$

Regression

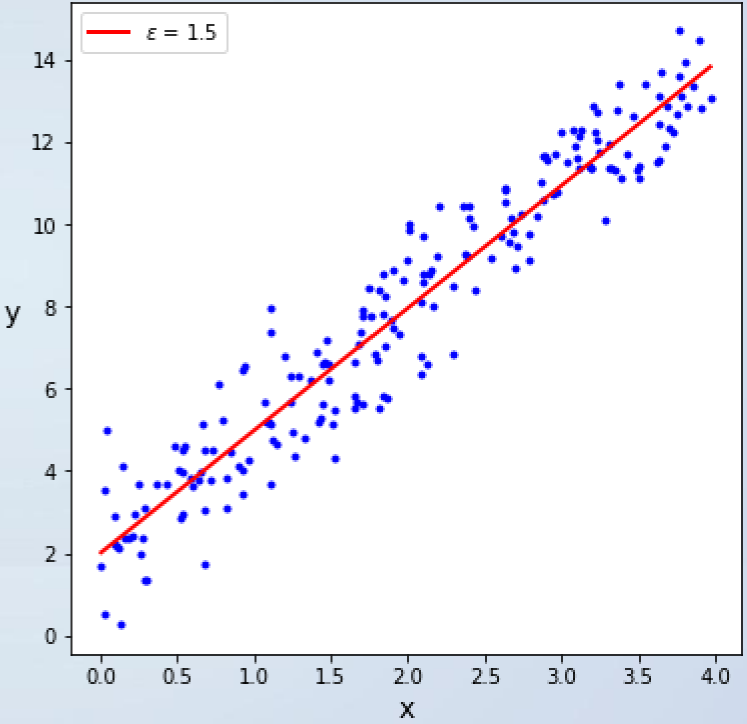
- In ε-SV regression, our goal is to find a function $f(x)$ that has at most ε deviation from the targets y_i for all the training data, and at the same time is as flat as possible.

$$
\begin{aligned}
\min_{(\boldsymbol{w},b)} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{m}(\xi_i + \xi_i^*) \\
s.t. \quad & y_i - \boldsymbol{w}^T\boldsymbol{x}_i - b \leq \epsilon + \xi_i, \qquad i = 1, \cdots, m \\
& \boldsymbol{w}^T\boldsymbol{x}_i + b - y_i \leq \epsilon + \xi_i^*, \qquad i = 1, \cdots, m \\
& \xi_i, \xi_i^* \geq 0, \qquad i = 1, \cdots, m
\end{aligned}
$$

- It is also possible to use kernel functions with this model. All we need to do is consider the dual problem of the one above.

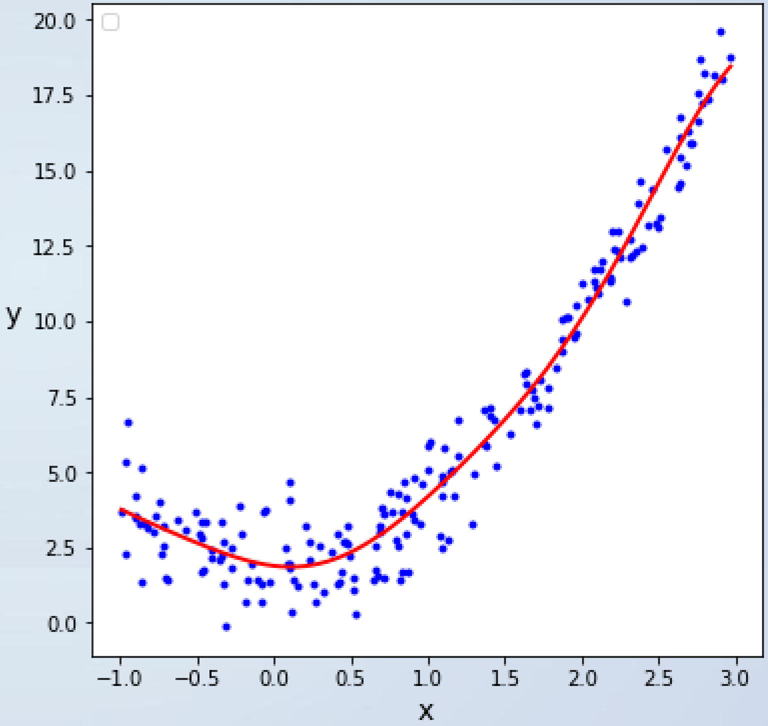
Example: linear data

```
from sklearn.svm import LinearSVR

svm_reg = LinearSVR(epsilon=1.5)
svm_reg.fit(x, y.ravel())
```



Example: linear data

```
from sklearn.svm import SVR

svm_reg2 = SVR(kernel='rbf', C=10)
svm_reg2.fit(x2, y2)
```



Decision Trees and SVMs with scikit-learn

Decision Trees and SVMs with scikit-learn

- Decision Tree:
  - Classification: sklearn.tree.DecisionTreeClassifier
  - Regression: sklearn.tree.DecisionTreeRegressor
  - Use sklearn.tree.export_graphviz and graphviz to visualize the model.

- SVM:
  - Classification: sklearn.svm.SVC
  - Regression: sklearn.svm.SVR

References

1. Cortes, Corinna, and Vladimir Vapnik. *Support-vector networks*. Machine learning 20.3 (1995): 273-297.
2. Géron, Aurélien. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.
3. Kotu, Vijay, and Bala Deshpande. *Data science: concepts and practice*. Morgan Kaufmann, 2018.
4. Smola, Alex J., and Bernhard Schölkopf. *A tutorial on support vector regression*. Statistics and computing 14.3 (2004): 199-222.


That's all Folks!