Tema 05. Otras técnicas de aprendizaje automático supervisado

Autor: Ismael Sagredo Olivenza

5.1 Other Supervised Machine Learning Techniques

Although neural networks have become very popular and are the standard in Machine Learning today, it is usefull to learn about other techniques as they have some advantages over neural networks and for certain problems they may be good enough.

5.1.1 Advantages and disadvantages of neural networks

Advantages:

- Failover (input data noise)
- Scalable models
- Can solve non-linear problems

disadvantages:

- High computational cost
- Black box algorithm.

5.2 K-Nearest Neighbor o k vecinos más cercanos

Also known as **KNN**. The algorithm does not attempt to extract a feature from the data by doing an exhaustive training process on it. It simply stores the examples for future retrieval.

When a new case is presented, it is compared with the case base and the most similar case (or K most similar cases) is extracted to check which class it belonged to.

It is assumed that if the case is similar, the solution will be similar.

A similarity function needs to be defined between two examples. The quality of this similarity function is critical for this algorithm.

If the function is not able to measure the similarity between two cases correctly, the essence of the algorithm is lost.

The solution depend on K selected. I)f K = 1 will be the most similar instance. If K > 1 a creterium is needed, e.g. the majority class or the middle value if we are regression.

There are a multitude of **distance measures** that we can use:

- **Euclidian distance**: for each attribute of X and Y:

$$D - Euclidea(X, Y) = \sqrt[2]{\sum_{i=1}^{N} (x_i - y_i)^2}$$

- **Manhattan Distance**: for each attribute of X and Y:

$$D - Man(X, Y) = \sum_{i=1}^{N} |x_i - y_i|$$

- **Minkowski Distance**: generalisation of the Euclidean or Manhattan where the exponent and the root can be any number.

$$D - Minkowski(X,Y) = \sqrt[p]{\sum_{i=1}^{N} (x_i - y_i)^p}$$

- **Levenshtein distance (Distancia de edición)**: the number of changes that need to be made to convert one instance into another. Hamming generalisation.
  - casa → cala (substitution of 's' for 'l') = 1
  - casa → calle (s for L, a for e, l insertion) = 3

- **Mahalanobis Distance**: the similarity between two multidimensional random variables. It differs from the Euclidean distance in that it takes into account the correlation between the random variables.

Formally, the Mahalanobis distance between two random variables with the same probability distribution $\vec{x}$ and $\vec{y}$ is calculated as follows:

$$D - Mahalanobis(\vec{x}, \vec{y}) = \sqrt[2]{(x-\mu)^T * C^-1 * (x-\mu)}$$

Where μ is the mean of the data and C-1 is the inverse of the covariance matrix.

```python
def mahalanobis(x=None, data=None, cov=None):
    """Computa la covarianza entre cada fila de los datos
    x     : Vector o matriz de datos con p columnas
    data : ndarray el total de datos de la distribución
    cov   : matriz de covarianzas (p x p) de la
    distribución. Si no se invoca se establece como none None
    """
    x_minus_mu = x - np.mean(data) #Cálculo de la media
    if not cov: #Calculamos la covarianza i
        cov = np.cov(data.values.T)
    inv_covmat = sp.linalg.inv(cov) # inversión de la matriz de covarianza
    left_term = np.dot(x_minus_mu, inv_covmat) # producto escalar entre
    #x menos la media y la inversa de la matriz de covarianza
    mahal = np.dot(left_term, x_minus_mu.T)
    # producto escalar entre la traspuesta de x menos la media y el cálculo anterior
    return mahal.diagonal() # devolvemos la diagonal de la matriz
```

Note that we do not calculate the square root, so we are actually calculating the square of the mahalanobis distance.

Another approach is to weight the weight of each attribute in the distance calculation. This implies **introducing information from the domain** since we must know or intuit that certain variables provide more information when deciding the weights.

$$WeightedEuclideanDistance(X, Y, P) = \sqrt[2]{\sum_{i=1}^{N} p_i (x_i - y_i)^2}$$

Where P is the vector of weights associated with each input.

We can weight any measure funtion, but some distances make more sense than others.

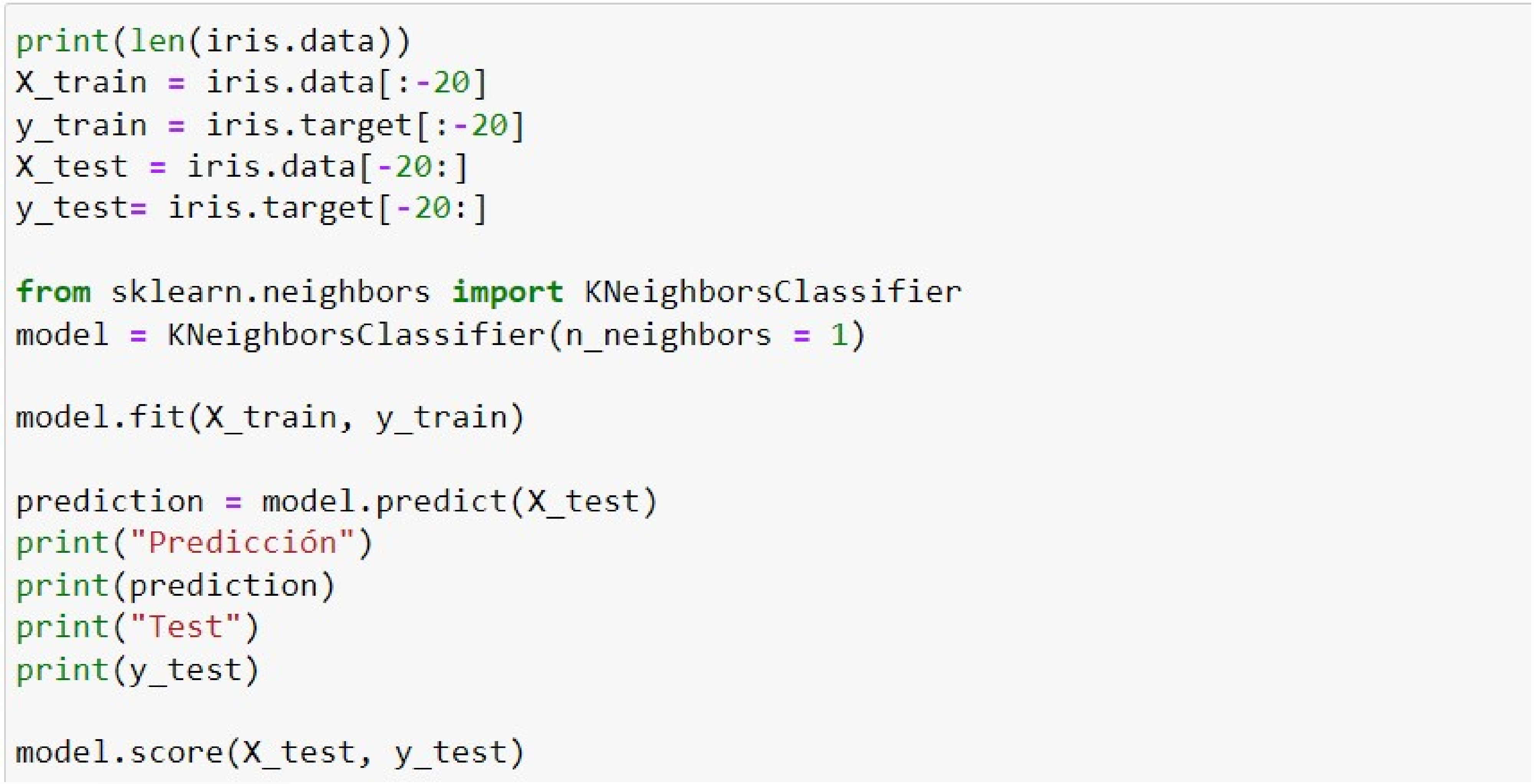
5.2.1 Advantages of KNN

- Non-parametric (unless we use weighted distances and k). Makes no explicit assumptions about the shape of the data.
- Simple algorithm both to explain and to interpret.
- High (relative) accuracy. Fairly high, though not superior to more sophisticated models. But despite its apparent simplicity, if the distance is chosen correctly it can give quite good results.
- The training process is immediate

5.2.2 Disadvantages of KNN

- It is very sensitive to irrelevant attributes. Making a good selection of relevant attributes is fundamental.
- It is sensitive to noise: if an example is a bad and it is the one selected as the most similar, we will give a wrong solution. This can be mitigated by making K large as it reduces noise.
- The execution is slow if there is a lot of training data, as it has to process all the data. There are methods to optimise it using spatial partitioning but it is still expensive.
- It is expensive in memory as it takes up a lot of memory if there are many cases =A limiting working memory and discard solutions.

Ejemplo de KNN en SKLearn



```python
print(len(iris.data))
X_train = iris.data[:-20]
y_train = iris.target[:-20]
X_test = iris.data[-20:]
y_test= iris.target[-20:]

from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors = 1)

model.fit(X_train, y_train)

prediction = model.predict(X_test)
print("Predicción")
print(prediction)
print("Test")
print(y_test)

model.score(X_test, y_test)
```

5.3 CBR

Storage of wisdom based on examples. We just need examples and how they were solved.

CBR only needs:

- A base of already solved cases
- A measure of similarity of cases
- A knowledge of how to adapt the cases if they are not the same.

It differs from pure KNN in several ways:

- It is not only based on tuples of examples attribute = value. The representation can be more complex (ontology, framework, etc).
- Cases are indexed for better retrieval
- Solutions are adapted if not directly applicable.
- The case base is updated and maintained.
- There is interaction with users (the user can add extra knowledge to help the system).

The University of Auckland

CASE 1

Problem (Symptoms)

- Problem: Front light doesn't work
- Car: VW Golf II, 1.6 L
- Year: 1993
- Battery voltage: 13,6 V
- State of lights: OK
- State of light switch: OK

Solution

- Diagnosis: Front light fuse defect
- Repair: Replace front light fuse

CASE 2

Problem (Symptoms)

- Problem: Front light doesn't work
- Car: Audi A6
- Year: 1995
- Battery voltage : 12,9 V
- State of lights: surface damaged
- State of light switch: OK

Solution

- Diagnosis: Bulb defect
- Repair: Replace front light

Each case describes one situation

Cases are independent of each other

Case are not rules

5.3.1 Similarity

The similarity of one case with respect to another is computed.

As in KNN, each attribute can have a different weight in the similarity.

The similarity in these cases will be a weighted sum of the similarities of each attribute.

Each attribute can have a similarity function according to its characteristics.

5.3.2 Adaptation of solutions

You get a very similar case, but the solution is not applicable (i.e. the solution does not use the same parameters) Example:

- We have a hit with the car and the brake light is broken.
- The most similar case is one where the headlight is broken.
- The solution of the recovered case is to replace the headlight.

We cannot apply the solution directly because the case is not exactly the same and the solution is not applicable. It has to be adapted => Replace the brake light.

5.3.4 Learning from the new case

- If the new diagnosis is correct, we can store the case as an example case.
- For this we need a system that gives us feedback on whether the new solution is correct.

Can the case base grow indefinitely?

No, if the case base increases, the retrieval time of a case increases. It is necessary to determine what is the maximum allowable case base size and the retrieval times required by the system and the memory size is also important.

5.3.5 Other measures of similarity: TF/IDF

TF/IDF: product of two measures, term frequency and inverse document frequency. $S(t,d) = tf(t,d) * idf(t,D)$

$$tf(t,d) = \frac{frecuencia(t,d)}{MAX(f(i,d)) \forall i \in d}$$

$$idf(t,D) = log(\frac{|D|}{1 + count(t \in D)})$$

$count(t \in D)$ indicates the number of documents where the term t appears. d current document, D collection of documents and t the term to search for.

Another way to calculate TF/IDF:

The weight of term t in document d is equal to the number of times term t appears in document d divided by the importance of term t, which is calculated as the number of times term t appears divided by the number of documents in the search corpus.

$$w(t,d) = \frac{|t \in d|}{imp(t)}$$

$$imp(t) = \frac{|t \in D|}{|D|}$$

Cosine similarity: given two vectors a and b of dimension N the cosine is calculated as:

$$cosSim(a, b) = \frac{\sum_i a_i, b_i}{\sqrt{\sum_i a_i^2}\sqrt{\sum_i b_i^2}}$$

soft cosine is the same but instead of calculating the product or sum of the normalised attribute values, we have a similarity matrix calculated for all the values of the vectors a and b.

- **Pearson correlation**: Pearson's correlation coefficient is a measure of linear dependence between two quantitative random variables. Unlike covariance, Pearson's correlation is independent of the scale of measurement of the variables.

$$\rho(a,b) = \frac{Cov(a,v)}{\sqrt{Var(a)Var(b)}}$$

5.4 Decision Tree

They are very useful for visualising the various options available for solving a problem or modelling behaviour. They can be converted into rules.

Backward induction methods can be applied to discover the reasoning behind the logic of the system.

A **decision tree** is composed of two types of nodes, a **conditional node** and a **class**. The conditional nodes are the internal nodes of the tree and the class, the leaf or terminal nodes.

Each condition (conditional node) is a question to the system about a variable and has two possibilities, either true or false. Depending on that it will follow one path or another. The end nodes are leaf nodes and show us the class.

Source (https://es.wikipedia.org/wiki/Árbol_de_decisión#/media/Archivo:Arbol_decision.jpg)

Some of the most famous implementations of these algorithms are **ID3, J48 or C4.5 (classification) and M5 (regression)**.

This algorithm uses the **entropy** (measure of uncertainty or clutter) to help decide which attribute should be evaluated next in the tree. That is, the attribute selected is the one that leaves the information more ordered or, better classified.

$$Entropy(s) = \sum_{n=1}^{c} -p_i * Log_2 p_i$$

Where c are the possible classification values, S is the set of all examples, and p_i is the proportion of examples of S that are in class i.

Information gain:

$$Gain(S, A) = Entropy(S) - \sum_{v \in V(A)}^{|A|} \frac{|sv|}{|s|} Entropy(S_v)$$

Where V(A) is the set of all possible values for the attribute A and Sv is a subset of S for which the attribute A has the value v

Since it selects the most promising attribute among all, we can conclude that it performs a **voracious search** among the best attributes. Then it applies **Divide and Conquer (divide y vencerás)** recursively with the problem.

5.4.1 Algorithm

```
ID3(E,A,N):N
E: ejemplos, A: atributos, N: nodoRaíz
------------------------------------
Si vacio(A) o Todos lso ejemplos de E pertenecen a una clase
  N.Clase = ClaseMayoritaria(E)
si no
  a = MejorAtributo(A) => aplicando entropía
  A = A - a => quitamos a
  Para cada valor v del atributo a de todas las instancias
    newNode = CrearNodo(a,v)
    N.AddHijos(newNodo)
    E = E - aquellos cuyo valor de de a sea v
    ID3(E,A,newNode)
return N
```

Example

Sitio de acceso A1	Cantidad gastada A2	Vivienda A3	Última compra A4	Clase
1	0	2	Book	Good
1	0	1	Disc	Bad
1	2	0	Book	Good
0	2	1	Book	Good
1	1	1	Book	Bad
2	2	1	Book	Bad

We choose the best attribute:
$A1 = \frac{1}{6}.I_{10} + \frac{4}{6}.I_{11} + \frac{1}{6}.I_{12}$ Where I_{1i} are the possible values of attribute 1 = {0,1,2}
We now calculate I_{10}, I_{11} e I_{12} using Entropy:
I_{10} we have for class 0 (Good) and value 0 a total of 1 examples. For class 1 (Bad) and value 0 we have 0 examples. The total of elements with value 0 is 1, from this we can deduce the following:

$I_{10} = -\frac{1}{1}log_2\frac{1}{1} - \frac{0}{1}log_2\frac{0}{1} = -1 * 0 + 0 = 0$ and for the rest...
$I_{11} = -\frac{2}{4}log_2\frac{2}{4} - \frac{2}{4}log_2\frac{2}{4} = 1$ (there are 2 good and 2 bad)
$I_{12} = 0$ because there are 0 good and 1 bad

Applying the equation we have:

$A1 = \frac{1}{6}.0 + \frac{4}{6}.1 + \frac{1}{6}.0 = 0,66$

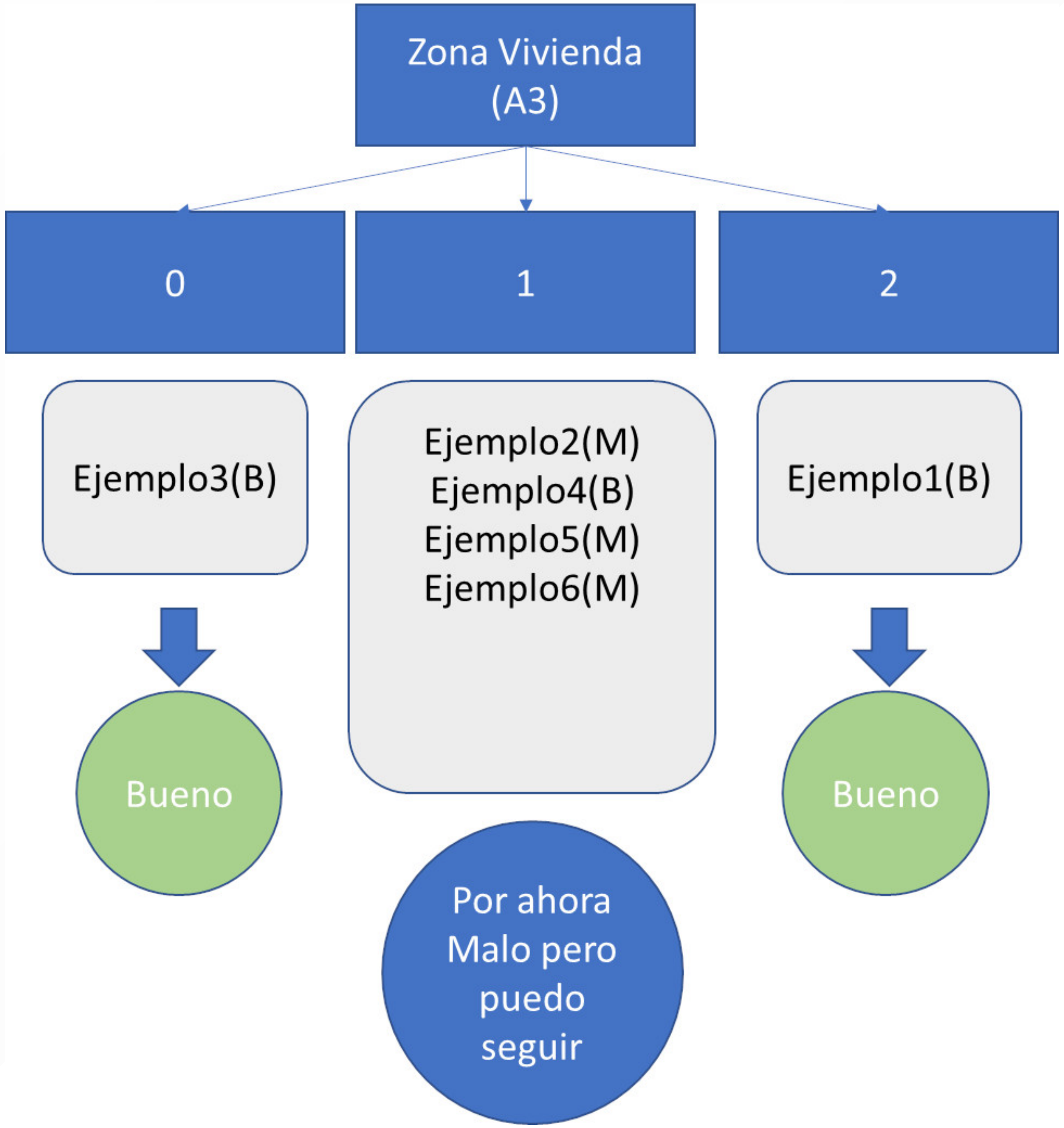
$A2 = 0,79$

$A3 = 0,54$

$A4 = 0,81$

The selected attribute would be the one with the lowest entropy (highest order) and therefore would be A3.

Leaves the tree with some terminals. But I could keep expanding and repeating the operation for the rest of the attributes (A1,A2,A4).

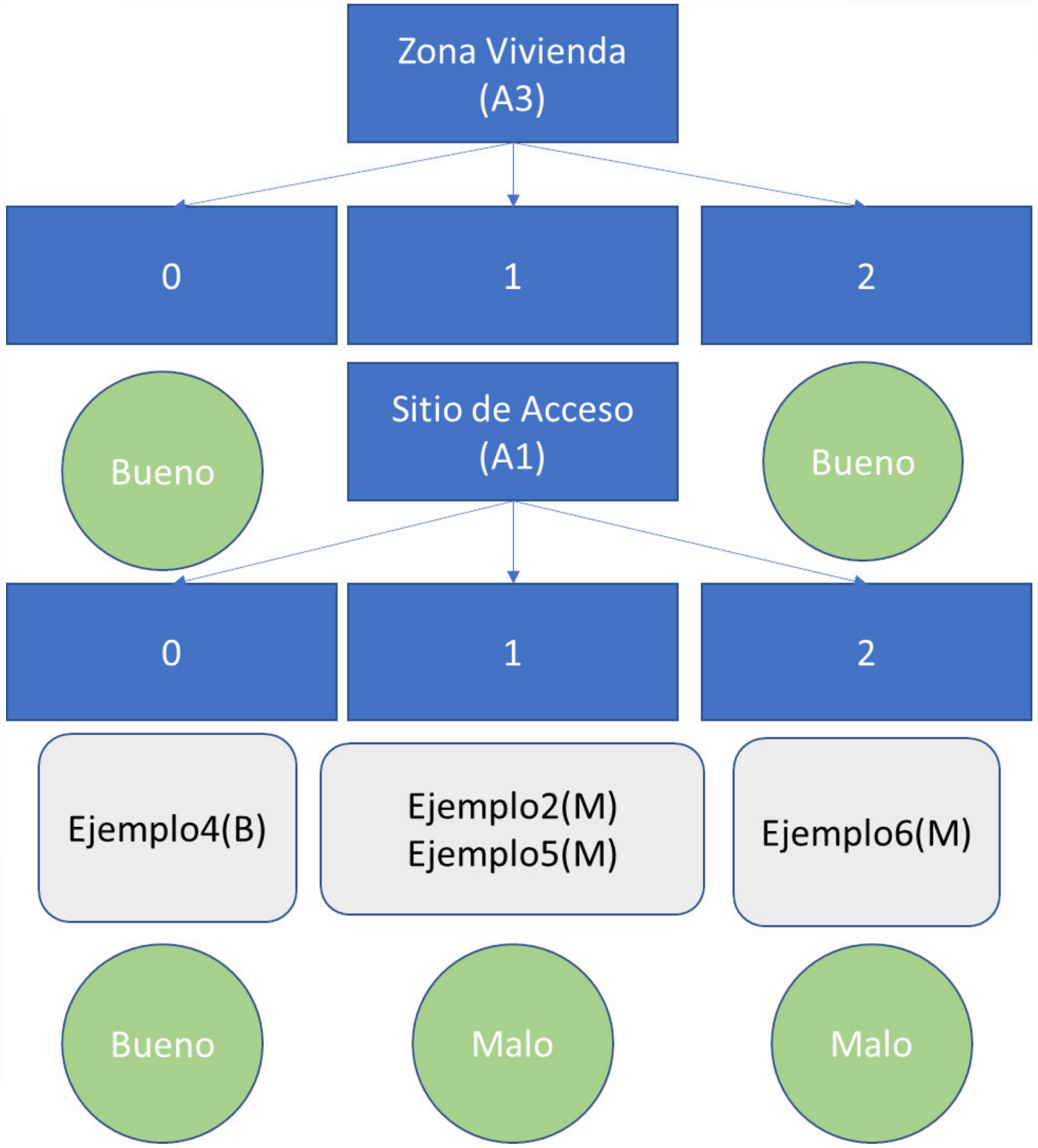

Zona Vivienda (A3)
0
1
2
Ejemplo3(B)
Ejemplo2(M)
Ejemplo4(B)
Ejemplo5(M)
Ejemplo6(M)
Ejemplo1(B)
Bueno
Por ahora Malo pero puedo seguir
Bueno

El siguiente atributo sería A1 sitio de acceso

$A1 = 0$
$A2 = 0,5$
$A4 = 0,23$

That would leave us with a totally ordered tree.

Note what I said before, the DT makes a voracious search, never rethinks if having put another node initially the configuration would have been better.


Zona Vivienda (A3)
0
1
2
Bueno
Sitio de Acceso (A1)
Bueno
0
1
2
Ejemplo4(B)
Ejemplo2(M)
Ejemplo5(M)
Ejemplo6(M)
Bueno
Malo
Malo

5.4.2 Graphic representation of the algorithm

Decision trees divide the solution space using hyperplanes by setting one of the variables to a boundary value.

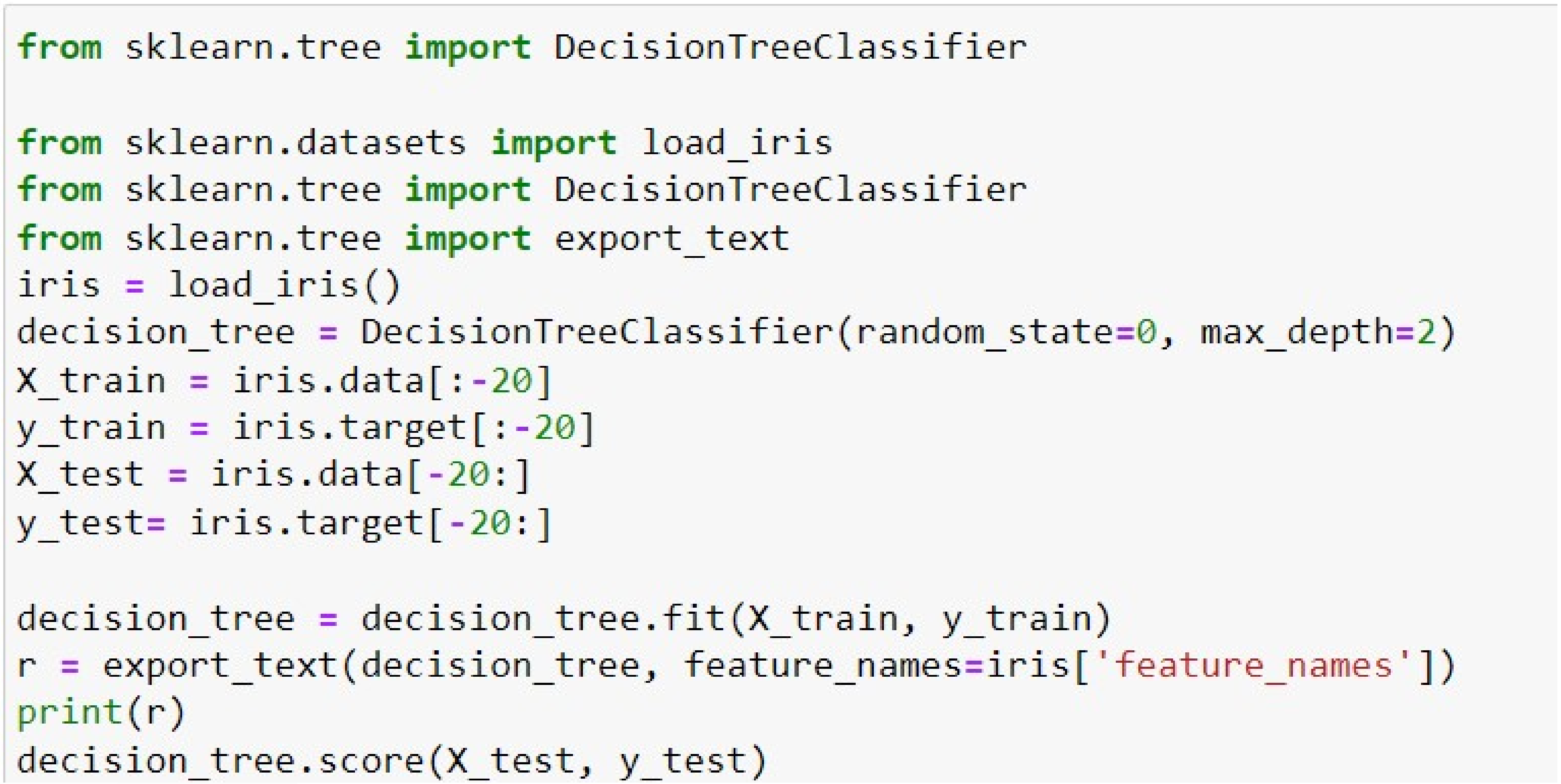
5.4.3 Adventages

- The training is very fast
- It is easy for a human to interpret the results, it is a white box algorithm.
- For some problems it achieves good accuracy.
- It can be easily converted into rules.
- It does not require a very demanding data preparation.
- It can work with qualitative and quantitative variables.

5.4.4 Disadventages

- It is highly dependent on input noise
- Decision trees are prone to overfiting (if the model is not lneal).
- There is no guarantee that the generated tree is optimal
- Assume as bias a linear model, if it is not, they may misclassify
- It is recommended to balance the data set before training.

Ejemplo de DT en SKLearn

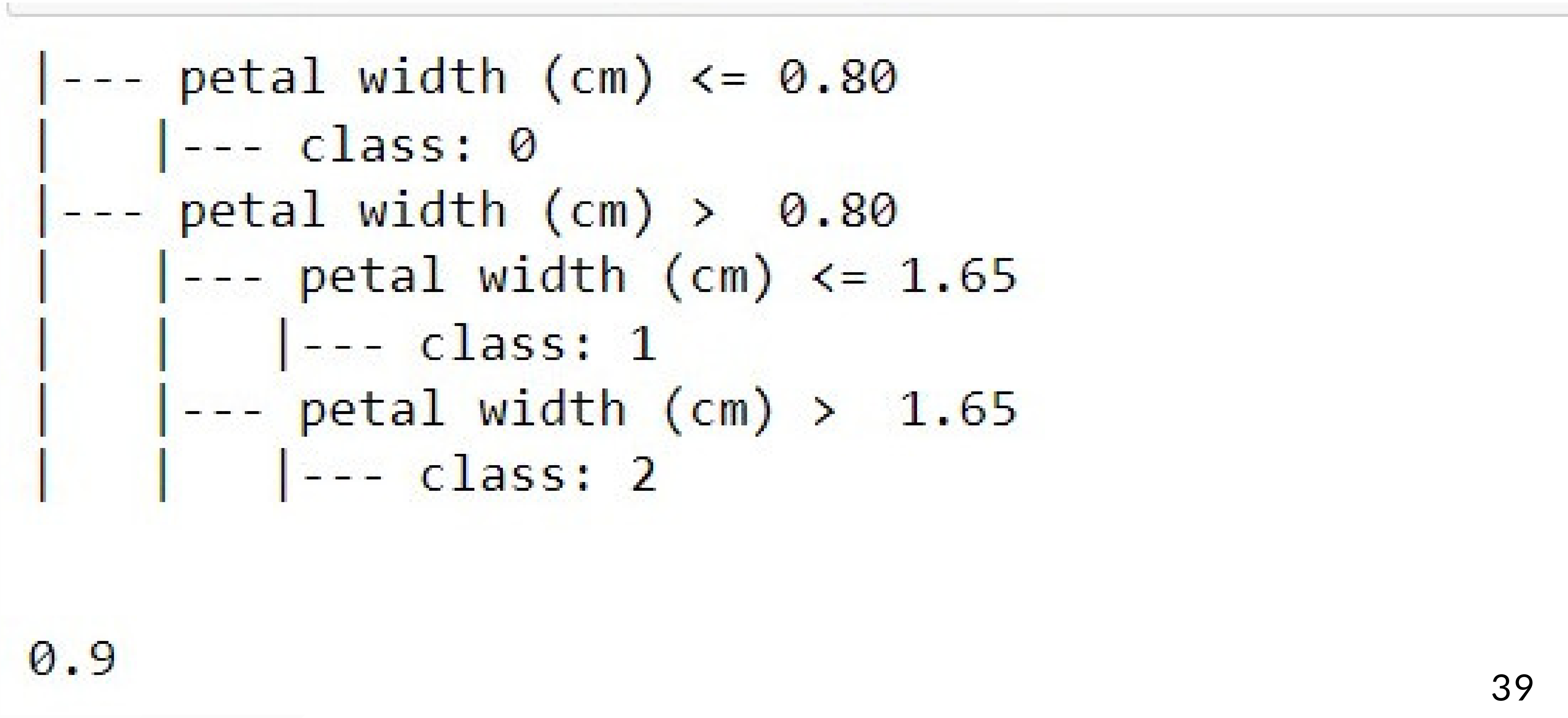


```python
from sklearn.tree import DecisionTreeClassifier

from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_text
iris = load_iris()
decision_tree = DecisionTreeClassifier(random_state=0, max_depth=2)
X_train = iris.data[:-20]
y_train = iris.target[:-20]
X_test = iris.data[-20:]
y_test= iris.target[-20:]

decision_tree = decision_tree.fit(X_train, y_train)
r = export_text(decision_tree, feature_names=iris['feature_names'])
print(r)
decision_tree.score(X_test, y_test)
```

Ejemplo de DT en SKLearn



```
|--- petal width (cm) <= 0.80
|   |--- class: 0
|--- petal width (cm) >  0.80
|   |--- petal width (cm) <= 1.65
|   |   |--- class: 1
|   |--- petal width (cm) >  1.65
|   |   |--- class: 2


0.9
```

5.4.5 Random Forest

This method runs different decision trees and a voting process is performed to choose the final prediction. RFs are ensemble methods for this reason. The number of trees it generates is a system parameter:

- We divide our data set into several randomly composed subsets of samples.
- We train a model on each subset.
- For each node, randomly choose m variables on which to base the decision. m must be < than the total number of variables.
- Combine all model results (by voting).

These methods perform better in general than conventional decision trees, although they make the generated model more difficult to understand.

**Advantages:

- Generally generates very good results.
- Easy to calculate
- Give estimates of which variables are important to rank.

Disadvantages:

- Overfit if there is a lot of noise.
- More difficult to interpret than DT

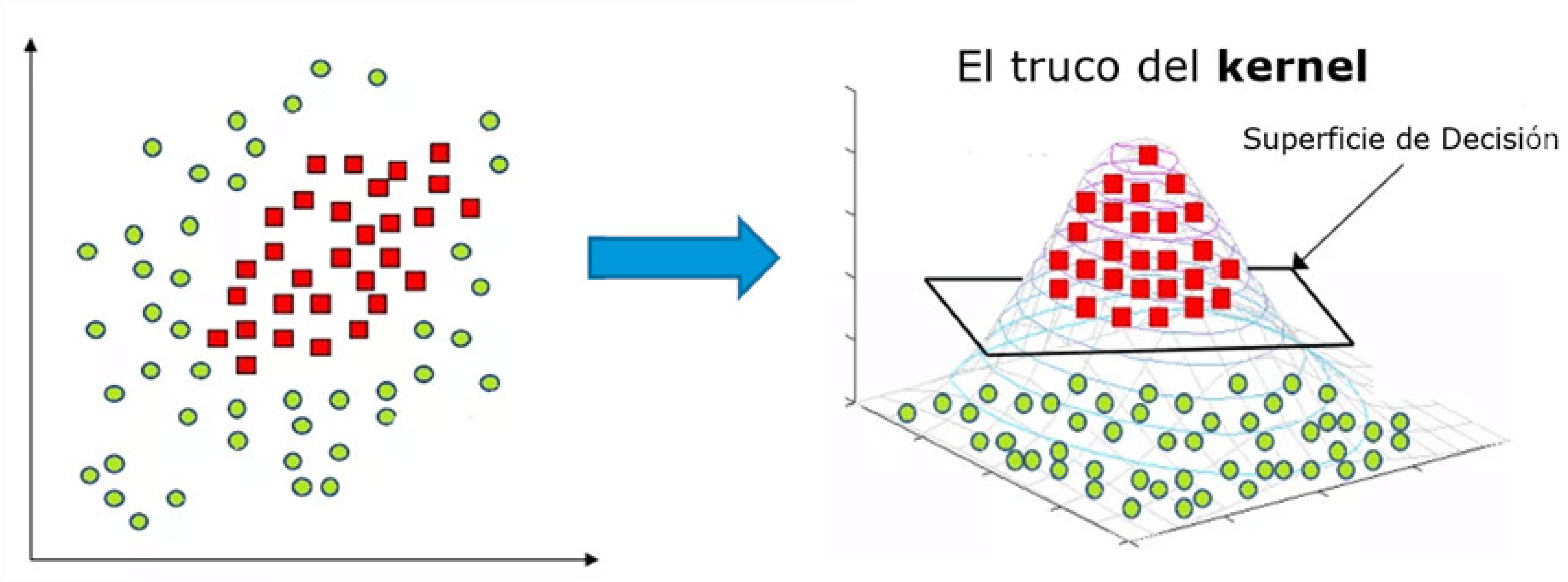
5.5 Support Vector Machine

They are a set of supervised learning algorithms developed by **Vladimir Vapnik** and his team at AT&T Labs.

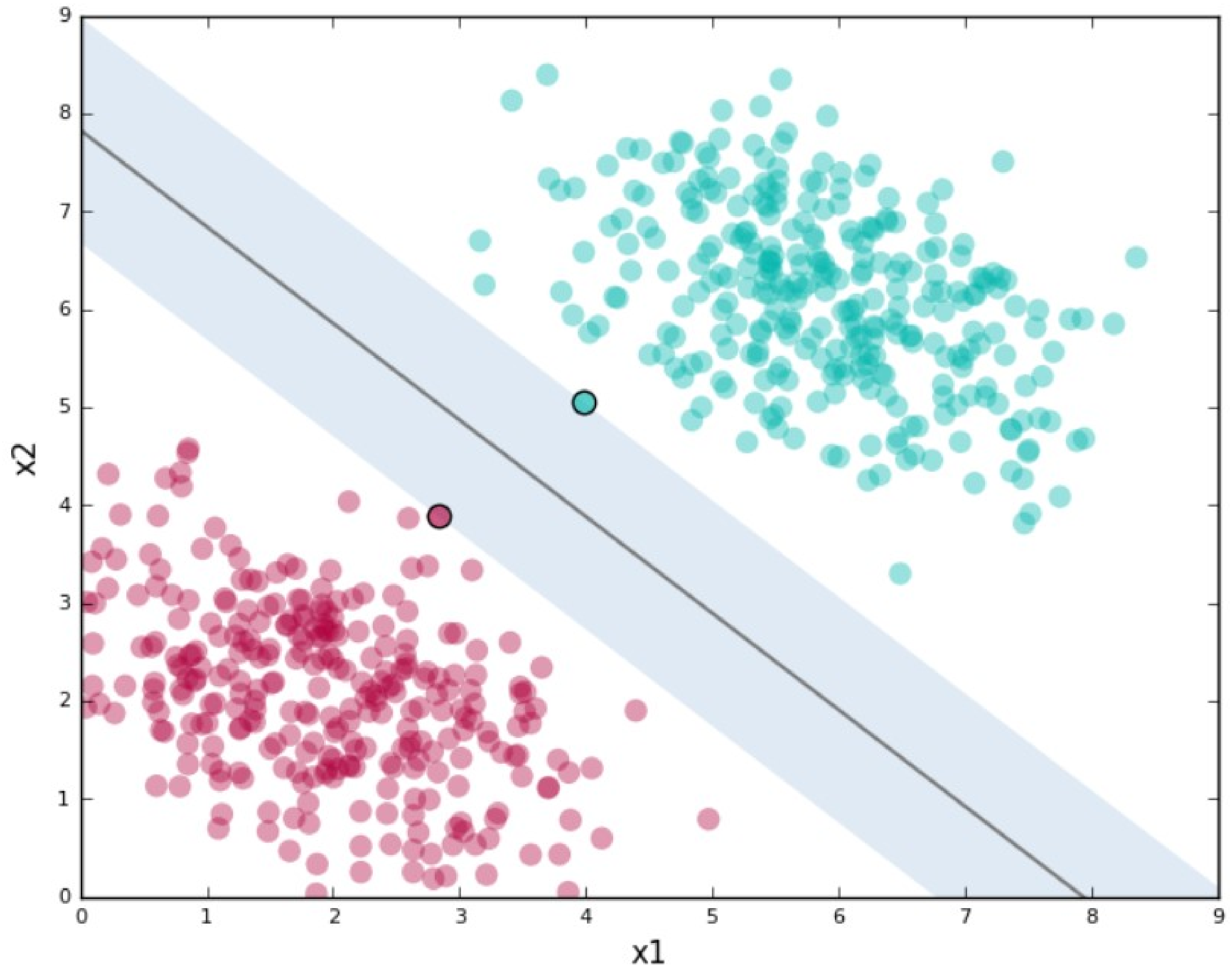
These methods are properly related to classification and regression problems.

Support vectors are the points that define the maximum margin of separation of the hyperplane separating the classes. They are called vectors because these points have as many elements as there are dimensions in our input space.

There are times when there is no way to find a hyperplane that allows two classes to be separated. In these cases, we say that the classes are not linearly separable. To solve this problem, we can use a **kernel**. The trick of the kernel is to invent a new dimension in which we can find a hyperplane to separate the classes.



The closest points that identify the lines of the margin are known as *support vectors*



Support vector machine

Logistic regression:

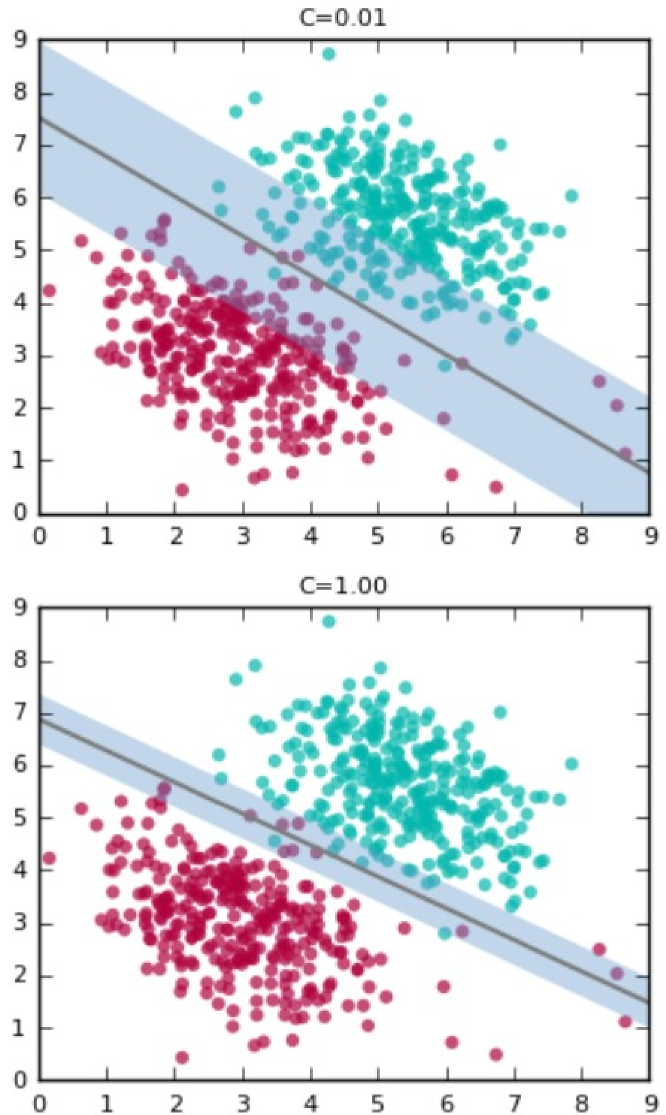
$$\min_\theta \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\left(-\log h_\theta(x^{(i)})\right) + (1-y^{(i)})\left((-\log(1-h_\theta(x^{(i)}))\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

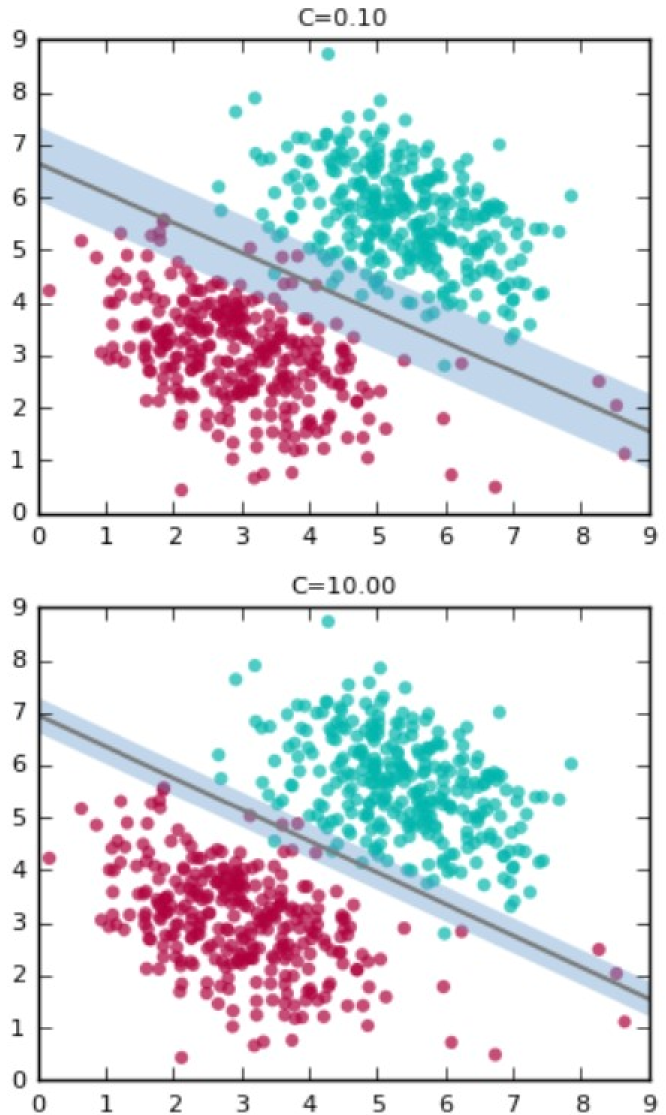
Support vector machine:

$$\min_\theta \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) cost_0(\theta^T x^{(i)})\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

$$\min_\theta C\sum_{i=1}^{m}\left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

C controls overfitting





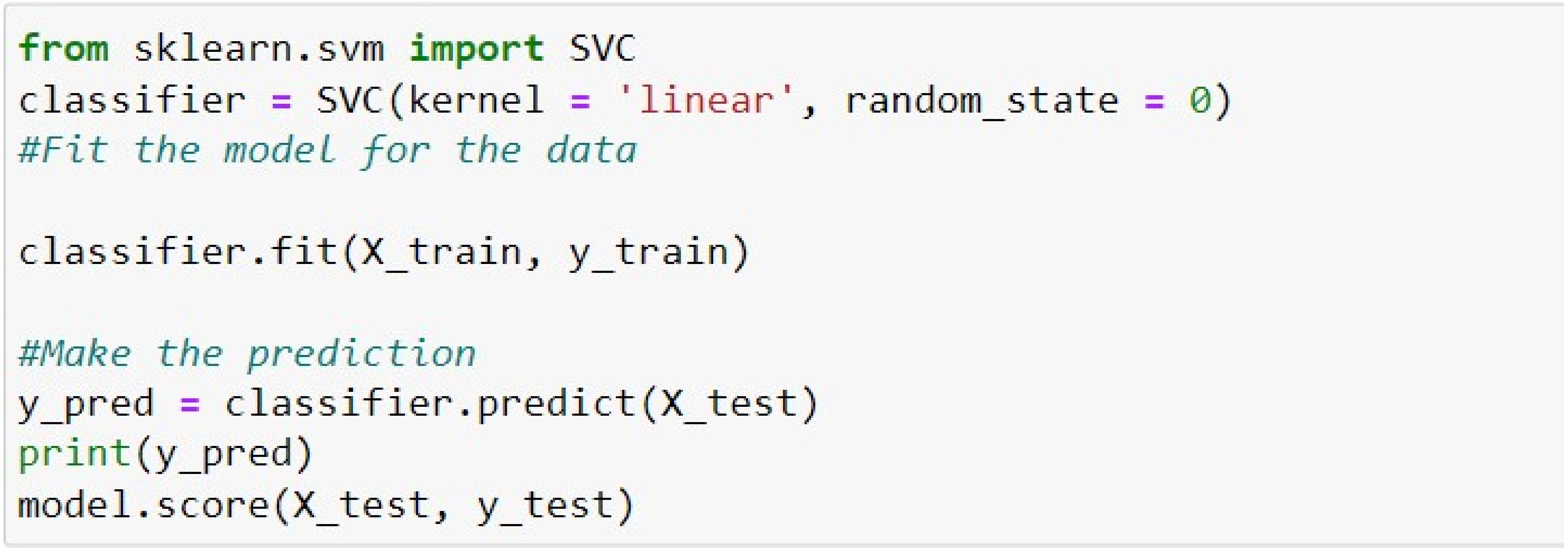
5.5.1 Advantages

- Effective in large spaces.
- Still effective in cases where the number of dimensions is larger than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

5.5.2 Disadvantages

- If the number of features is much larger than the number of samples avoid over-fitting when choosing Kernel functions and the regularisation term is crucial.
- SVMs do not directly provide probability estimates, these are calculated using cross-validation.

SVM in SKLearn



```
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
#Fit the model for the data

classifier.fit(X_train, y_train)

#Make the prediction
y_pred = classifier.predict(X_test)
print(y_pred)
model.score(X_test, y_test)
```