

# “APRENDIZAJE I”

Reconocimiento de Patrones

MAESTRÍA EN ELECTRÓNICA

Profesor: MSc. Felipe Meza

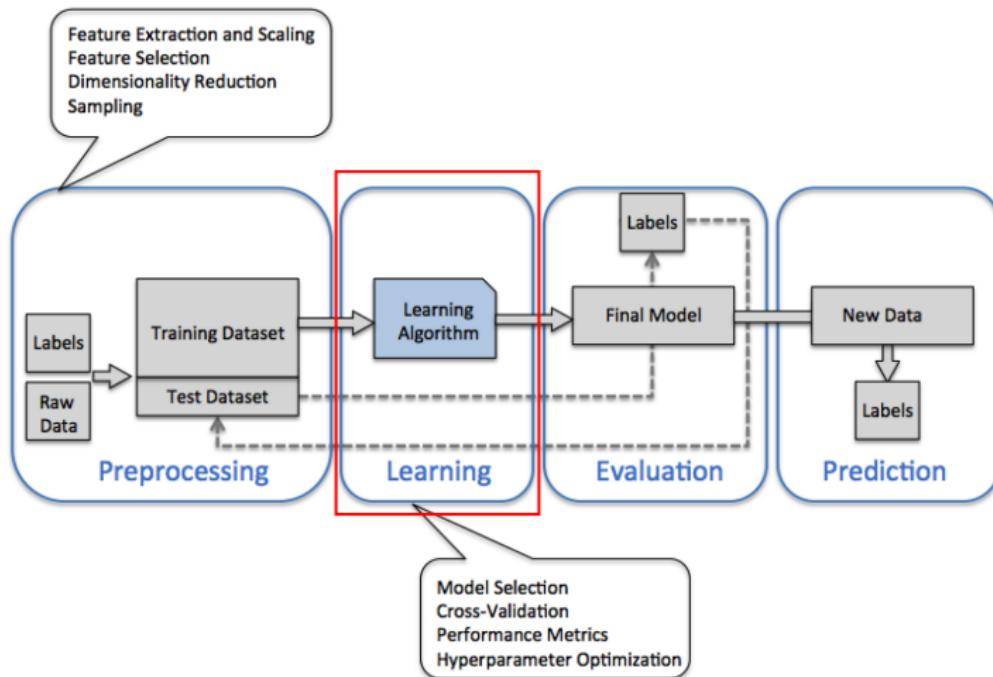


June 10, 2021

# Agenda

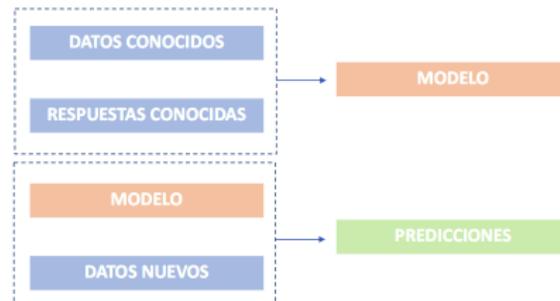
- ① Metodología de Diseño.
- ② Aprendizaje Supervisado.
- ③ Clasificación y Regresión.
- ④ Generalización, Overfitting, Underfitting.
- ⑤ Métodos de Aprendizaje Supervisado.

# Metodología de Diseño

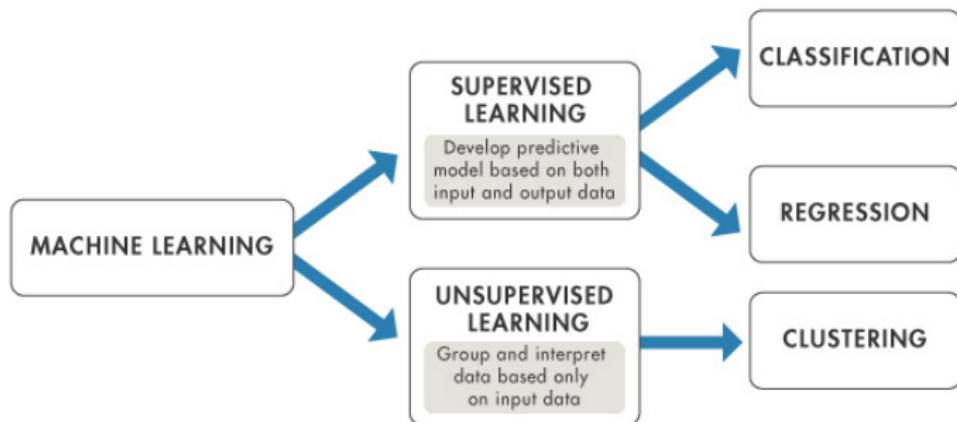


# Aprendizaje Supervisado

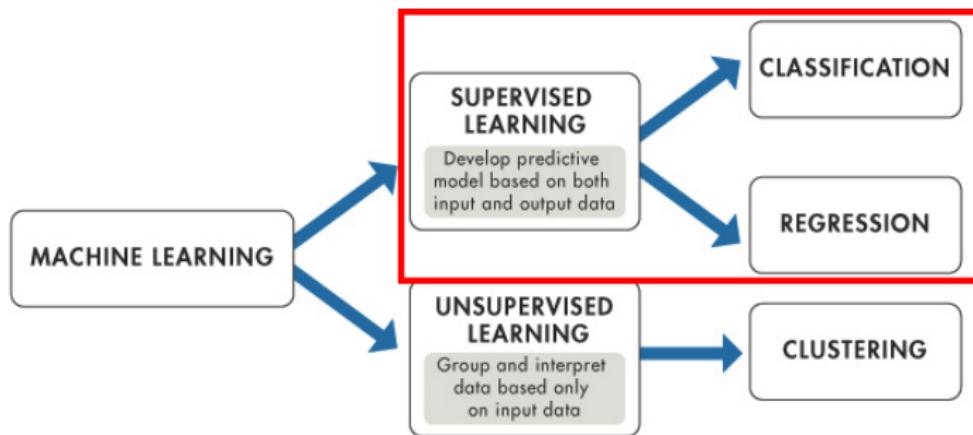
- Consiste en predecir una **salida** (clase) a partir de una **entrada** (atributos), y se cuenta con ejemplos “mapeados” de pares entrada/salidas (instancias).  $\{x_i, y_i\}_{i=1}^N$
- Se construye un modelo de aprendizaje automático a partir de los pares entrada/salidas (conjunto de **entrenamiento**). El objetivo consiste en utilizar dicho modelo para hacer predicciones precisas con datos nunca antes vistos (conjunto de **pruebas**).



# Aprendizaje Supervisado

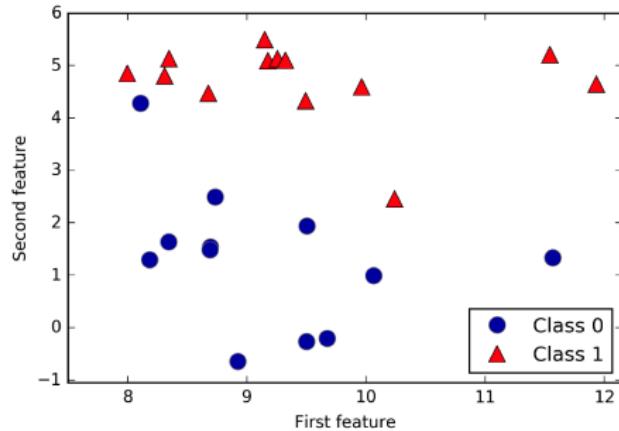


# Aprendizaje Supervisado



# Clasificación y Regresión

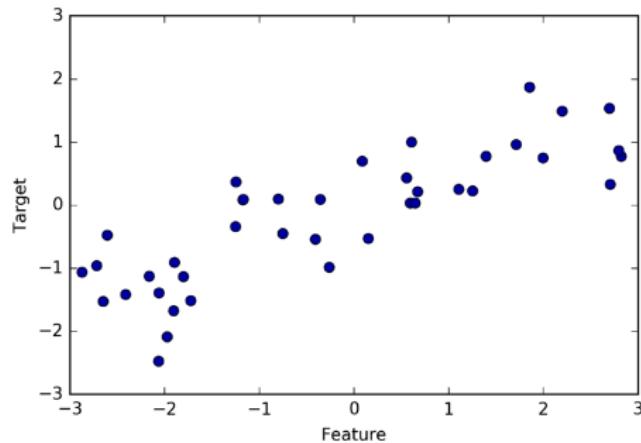
- Clasificación:
  - El objetivo es predecir una clase.
  - Hay *clasificación binaria* (yes/no, spam/no-spam), y también *clasificación multiclas* (e.g iris).



# Clasificación y Regresión

- Regresión:

- El objetivo es predecir un número continuo, en un rango establecido.
- Se distingue de la clasificación en el sentido de que hay una continuidad a la salida (e.g sensores).



# Generalización, Overfitting, Underfitting

- Generalización:

- Consiste en la capacidad del modelo de aprendizaje de hacer predicciones correctas con el conjunto de pruebas a partir del conjunto de entrenamiento.
- Si lo hace correctamente se dice que el modelo “generaliza” del conjunto de entrenamiento al conjunto de pruebas.

# Generalización, Overfitting, Underfitting

Age	Number of cars owned	Owns house	Number of children	Marital status	Owns a dog	Bought a boat
66	1	yes	2	widowed	no	yes
52	2	yes	3	married	no	yes
22	0	no	0	married	yes	no
25	1	no	1	single	no	no
44	0	no	2	divorced	yes	no
39	1	yes	2	married	yes	no
26	1	no	2	single	no	no
40	3	yes	1	married	yes	no
53	2	yes	2	divorced	no	yes
64	2	yes	3	divorced	no	no
58	2	yes	2	married	yes	yes
33	1	no	1	single	no	no

- Podemos experimentar 3 situaciones:
  - **Regla 1:** Va comprar un bote el que tenga más de 45 años, menos de 3 hijos ó no esté divorciado. [muy complejo]
  - **Regla 2:** Va comprar un bote el que tenga una casa. [muy simple]
  - **Regla 3:** Alguna combinación intermedia.

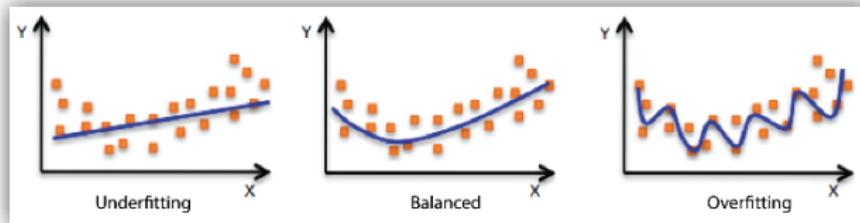
# Generalización, Overfitting, Underfitting

- Overfitting:

- Escoger un modelo de aprendizaje que sea muy complejo, funciona muy bien con el conjunto de entrenamiento pero no **generaliza** con nuevos datos.

- Underfitting:

- Escoger un modelo de aprendizaje que sea muy simple, funciona mal con el conjunto de entrenamiento y con nuevos datos no será capaz **generalizar**.



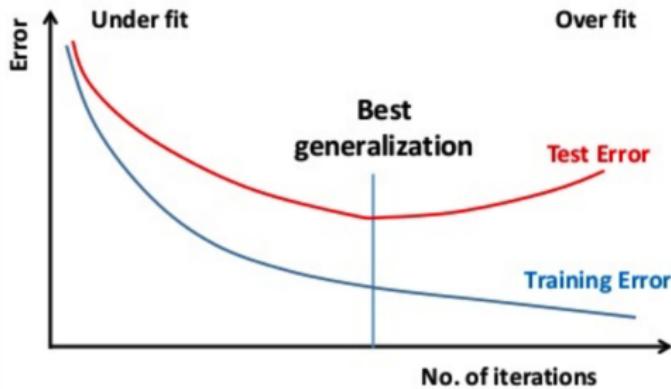
# Generalización, Overfitting, Underfitting

```
1  
2 Training Score: 0.99  
3 Testing Score: 0.55  
4  
5  
6 Training Score: 0.57  
7 Testing Score: 0.62  
8  
9  
10 Training Score: 0.82  
11 Testing Score: 0.86
```

# Generalización, Overfitting, Underfitting

```
1 # Overfit
2 Training Score: 0.99
3 Testing Score: 0.55
4
5 # Underfit
6 Training Score: 0.57
7 Testing Score: 0.62
8
9 # Fit
10 Training Score: 0.82
11 Testing Score: 0.86
```

# Generalización, Overfitting, Underfitting



# Métodos de Aprendizaje Supervisado

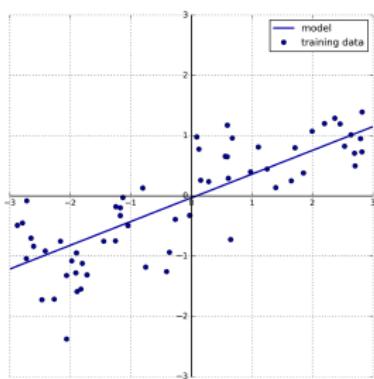
- Modelos Lineales:
  - Linear Regression (Regresión).
  - Logistic Regression (Clasificación).
  - SVM.
  - Modelos Lineales para clasificación multi-clase.
- kNN
  - kNN para clasificación.
  - kNN para regresión.
- Naive Bayes.
- Decision Trees.
- Random Forest.
- Kernel SVM.

# Modelos Lineales

- Son modelos de aprendizaje que hacen uso de la *función lineal* en los atributos de entrada:

$$y = w[0] \times x[0] + w[1] \times x[1] + \dots + w[p] \times x[p] + b$$

- Para el caso de único atributo:



$$y = w[0] \times x[0] + b$$

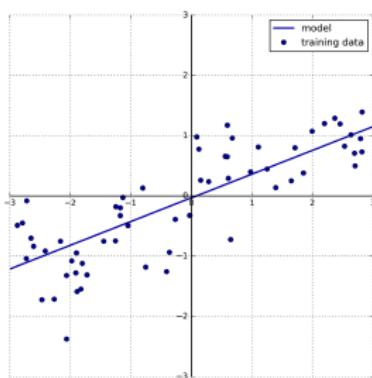
$$y = m \times x + b$$

# Modelos Lineales

- Son modelos de aprendizaje que hacen uso de la *función lineal* en los atributos de entrada:

$$y = w[0] \times x[0] + w[1] \times x[1] + \dots + w[p] \times x[p] + b$$

- Para el caso de único atributo:



MODELO

$$f_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$$

COSTO

$$\frac{1}{N} \sum_{i=1 \dots N} (f_{\mathbf{w},b}(\mathbf{x}_i) - y_i)^2$$

## REGRESIÓN

# Regresión Lineal Ordinaria (OLS)

- Consiste en encontrar los parámetros  $w$  y  $b$  que minimicen el MSE (mean square error) entre los valores reales y la predicción.
- En la mayoría de herramientas computacionales se le conoce como "Linear Regression".

```
1 from sklearn.linear_model import LinearRegression  
2  
3 lr = LinearRegression().fit(X_train, y_train)  
4  
5 lr.score(X_train, y_train)  
6 lr.score(X_test, y_test)
```

# Regresión Lineal RIDGE y LASSO

Se denomina **REGULARIZACIÓN** al efecto de influir en el modelo de aprendizaje con el fin de evitar “overfitting”, en el caso de LASSO regularización es  $L1$  y en el caso de RIDGE se llama regularización  $L2$ .

LR

$$\min_{w,b} \frac{1}{N} \sum_{i=1}^N (f_{w,b}(x_i) - y_i)^2$$

L1

$$\min_{w,b} C|w| + \frac{1}{N} \sum_{i=1}^N (f_{w,b}(x_i) - y_i)^2, \text{ donde } |w| \stackrel{\text{def}}{=} \sum_{j=1}^D |w^{(j)}|$$

L2

$$\min_{w,b} C\|w\|^2 + \frac{1}{N} \sum_{i=1}^N (f_{w,b}(x_i) - y_i)^2, \text{ donde } \|w\|^2 \stackrel{\text{def}}{=} \sum_{j=1}^D (w^{(j)})^2$$

# Regresión Lineal RIDGE y LASSO

- LASSO (L1)

- Se pueden obtener valores de  $w$  **iguales** a cero.
- Reduce la **complejidad** y el **número de variables**.

- RIDGE (L2)

- Modelo que persigue obtener valores de  $w$  **cercanos** a cero, con el fin de minimizar el efecto de los atributos mientras se mantiene un buen desempeño.
- El parámetro  $\alpha$  permite balancear la simplicidad del modelo con su desempeño. Su valor óptimo depende de las características del conjunto de datos.
- Reduce la **complejidad** pero no el número de variables.

## CLASIFICACIÓN

# Modelos Lineales

- En el caso de la clasificación se usa la misma base matemática, con la diferencia de que el límite al valor por predecir se ubica en **cero**.
- Valores **menores** a cero corresponden a una clase y valores **mayores** a cero corresponden a la otra clase.
- La salida o clase por predecir corresponde a una linea denominada “**límite de decisión**” .
- Existen varios tipos de algoritmos de clasificación por modelos lineales pero en general difieren en dos aspectos:
  - Que tan bien la combinación de coeficientes ( $w$ ) ajuste a los datos de entrenamiento.
  - El tipo de regularización utilizada.

# Logistic Regression y SVM

- Logistic Regression

- No confundir con Linear Regression!!!!
- Logistic Regression se usa para la clasificación.

```
1 from sklearn.linear_model import LogisticRegression
```

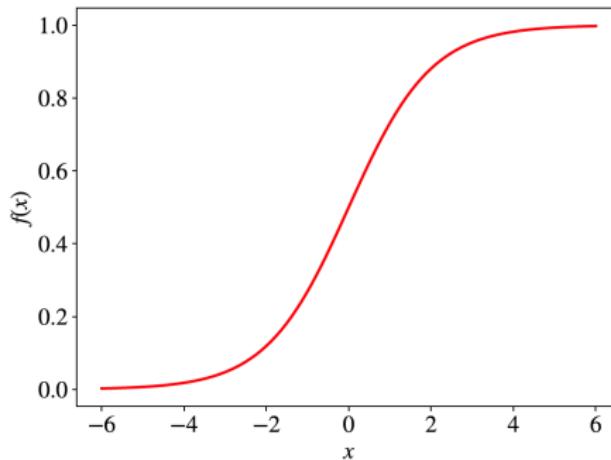
- SVM

- Máquina de vector de soporte o Clasificador de vector de soporte.

```
1 from sklearn.svm import LinearSVC
```

# Logistic Regression

- Se busca una función que logre operar como una probabilidad de la forma  $0 < p < 1$ , esa función es la logística estándar o sigmoid:



$$f(x) = \frac{1}{1+e^{-x}}$$

# Logistic Regression

- El modelo queda de la siguiente forma:

$$f_{w,b}(x) \stackrel{\text{def}}{=} \frac{1}{1+e^{-(wx+b)}}$$

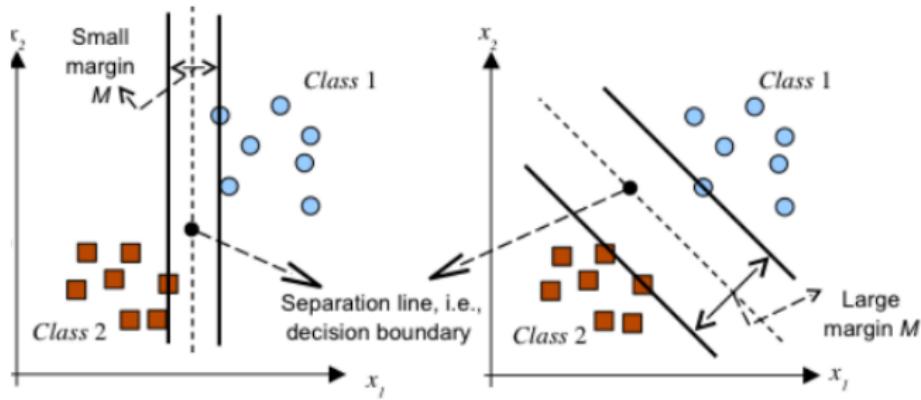
- El criterio consiste en maximizar la probabilidad de los datos de entrenamiento de acuerdo al modelo:

$$L_{w,b} \stackrel{\text{def}}{=} \prod_{i=1 \dots N} f_{w,b}(x_i)^{y_i} (1 - f_{w,b}(x_i))^{(1-y_i)}$$

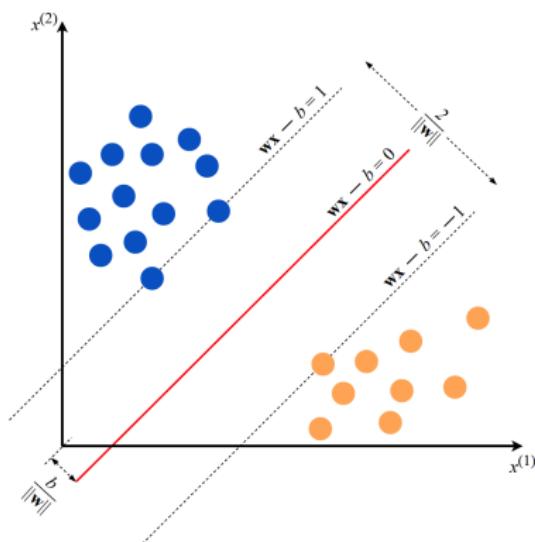
- Con el fin de facilitar el uso de la ecuación anterior, se recurre al uso de logaritmos:

$$\begin{aligned}\log L_{w,b} &\stackrel{\text{def}}{=} \ln(L_{w,b}(x)) = \\ &\sum_{i=1}^N y_i \ln f_{w,b}(x) + (1 - y_i) \ln(1 - f_{w,b}(x))\end{aligned}$$

# SVM (Máquinas de Vector de Soporte)



# SVM (Máquinas de Vector de Soporte)



Límite de decisión:

$$wx - b = 0, \text{ tal que}$$

$$w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + \dots + w^{(D)}x^{(D)}$$

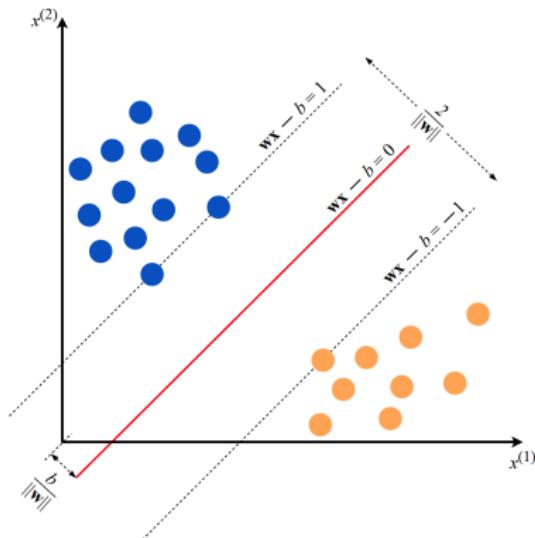
Predicción:

$$y = \text{sign}(wx - b)$$

Modelo:

$$f(x) = \text{sign}(w^*x - b^*)$$

# SVM (Máquinas de Vector de Soporte)



Se debe cumplir que:

$$\begin{aligned} \mathbf{w} \cdot \mathbf{x}_i - b &\geq 1 \text{ if } y_i = +1, \text{ y} \\ \mathbf{w} \cdot \mathbf{x}_i - b &\leq -1 \text{ if } y_i = -1 \end{aligned}$$

La distancia se debe maximizar y la norma euclidiana minimizar:

$$\|\mathbf{w}\| = \sqrt{\sum_{j=1}^D (w^{(j)})^2}$$

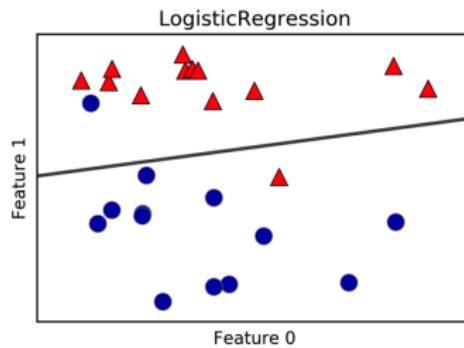
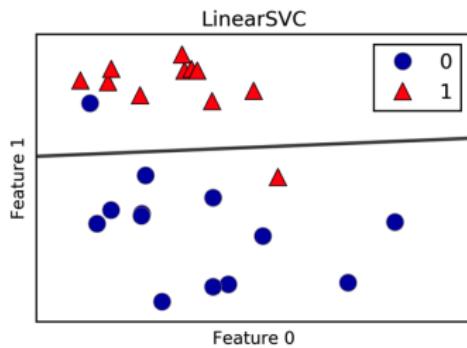
Criterio de optimización:

$$\min |\mathbf{w}| = \min \frac{1}{2} \|\mathbf{w}\|^2$$

# SVM (Máquinas de Vector de Soporte)

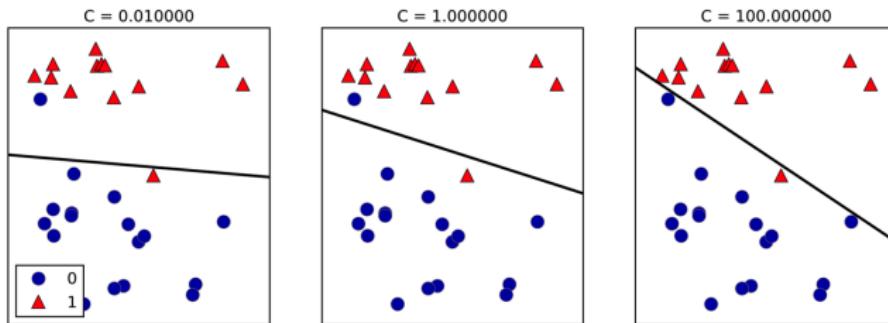
- Ventajas:
  - Buena precisión.
  - Opera bien con datos pequeños/limpios.
- Desventajas:
  - Cuando el conjunto de entrenamiento es muy *grande*, se puede volver lento.
  - No dà buenos resultados cuando los datos son “*ruidosos*”.

# Logistic Regression y SVM



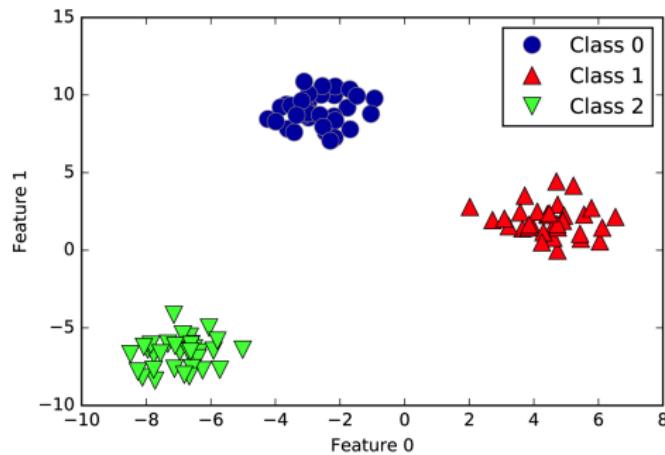
# Logistic Regression y SVM

- Se utiliza el hiperparámetro **C** para controlar la regularización.
- Un valor alto de **C** implica menos regularización, el modelo trata de ajustarse a los datos de training lo mayor posible.
- Un valor bajo de **C** implica encontrar coeficientes  $w$  cercanos a cero.



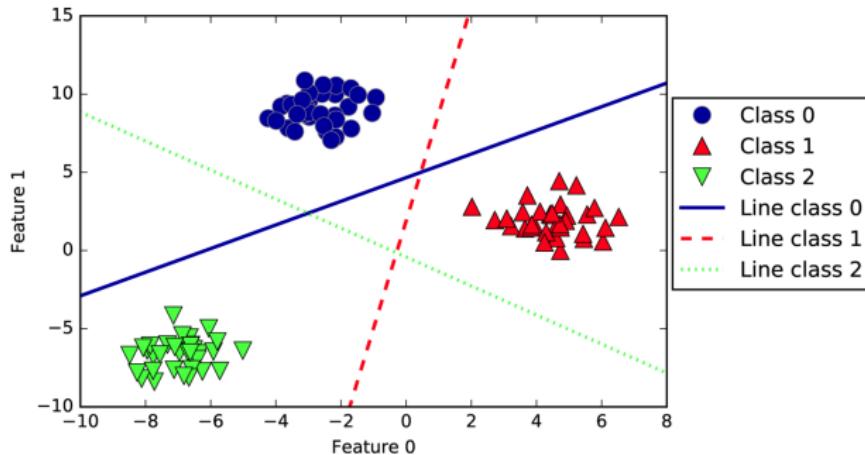
# Modelos Lineales Multi-Clase

- Consiste en usar modelos lineales usados comúnmente en clasificación binaria a clasificación con múltiples clases.



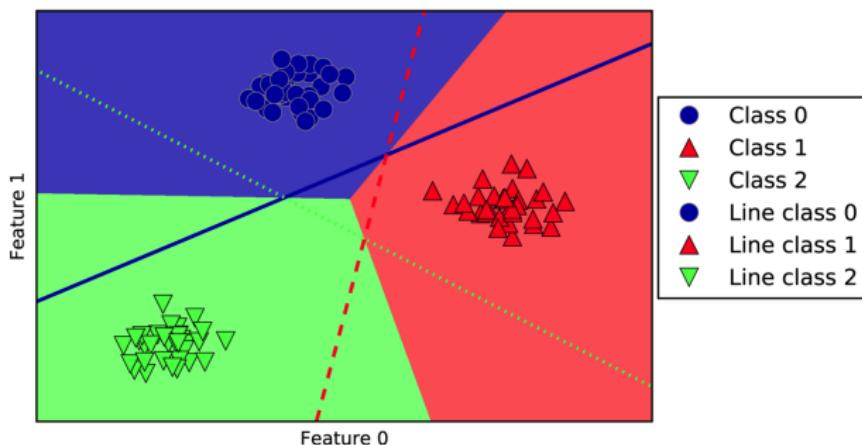
# Modelos Lineales Multi-Clase

- Se recurre al algoritmo *uno vrs. el resto*, el cual consiste en separar cada clase del resto como si fuera clasificación binaria.



# Modelos Lineales Multi-Clase

- Los datos de las zonas centrales (en este caso el triángulo central) se asocian con la clase de la línea más cercana, definiendo así una nueva área por clase.

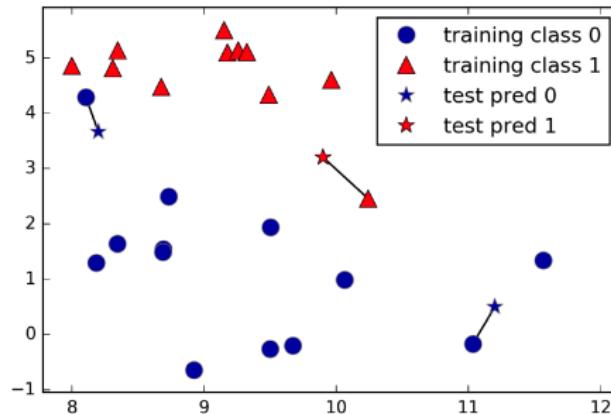


## Modelos Lineales - Recordar...

- Los hiperparámetros de regularización corresponden a  $\alpha$  para los modelos de regresión lineal y a  $\mathbf{C}$  para la regresión logística y SVM.
- Modelos simples implican un alto  $\alpha$  y un  $\mathbf{C}$  bajo.
- Otro factor importante es la regularización, L1 se escoge cuando se tiene claro que sólo algunos atributos son importantes.
- Los modelos lineales son rápidos de entrenar y de hacer predicciones.
- Escalan muy bien para conjuntos de datos grandes o dispersos.
- Son de fácil comprensión.
- Un detalle importante es que los coeficientes no son fácilmente interpretables.

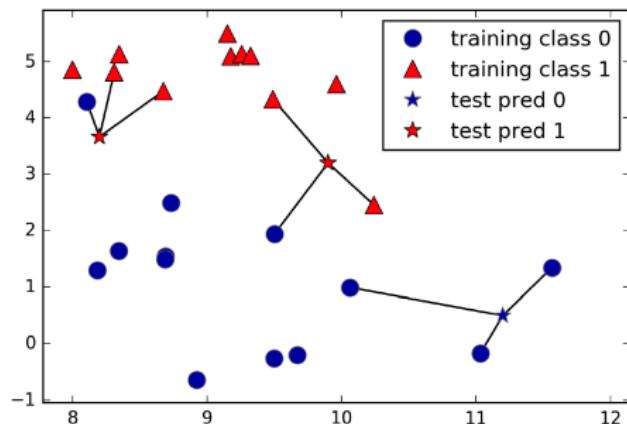
# kNN - k Nearest Neighbors (Clasificación)

- Es uno de los más **simples** algoritmos de aprendizaje, en cuanto a criterio de predicción.
- Para predecir un nuevo dato, busca el/los punto(s) más cercanos i.e vecino(s) más cercano(s).
- El caso más elemental consiste en hacer el calculo a partir de **1** vecino cercano:



# kNN - k Nearest Neighbors (Clasificación)

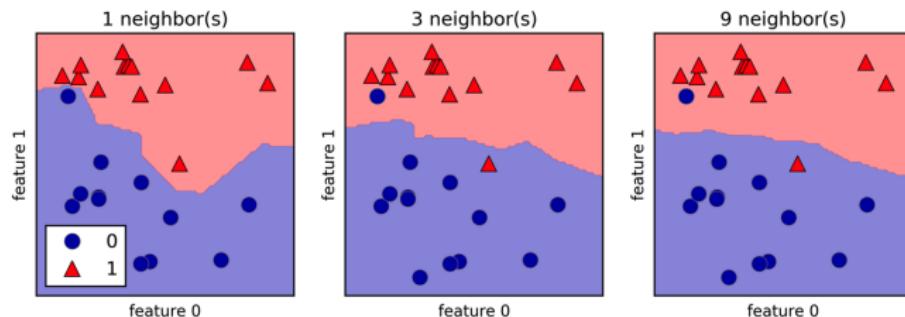
- En caso de considerar más de 1 vecino se usa la regla del “voto” i.e un conteo de la cantidad de vecinos cercanos pertenecientes a una clase, veamos el caso de  $k = 3$ .



- Este método puede ser aplicado a conjuntos de datos de múltiples clases.

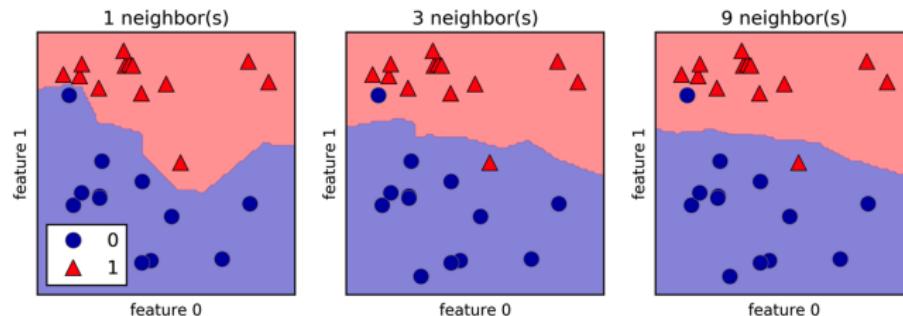
# kNN - k Nearest Neighbors (Clasificación)

- El **límite de decisión** se define como la línea que separa a una clase de otra.



# kNN - k Nearest Neighbors (Clasificación)

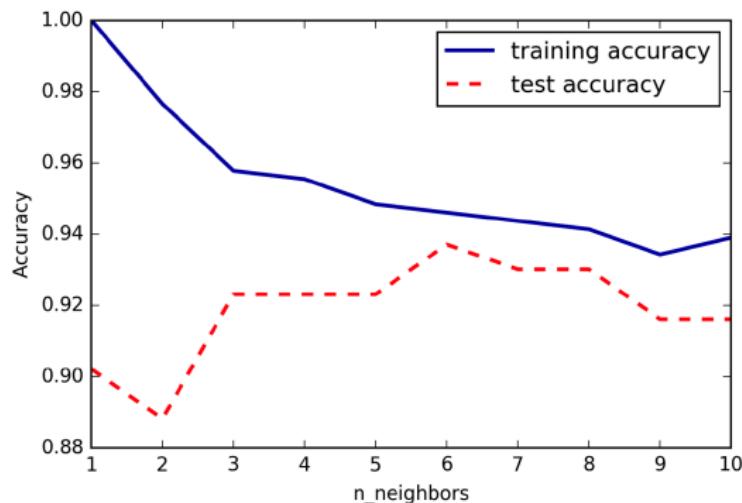
- El **límite de decisión** se define como la línea que separa a una clase de otra.



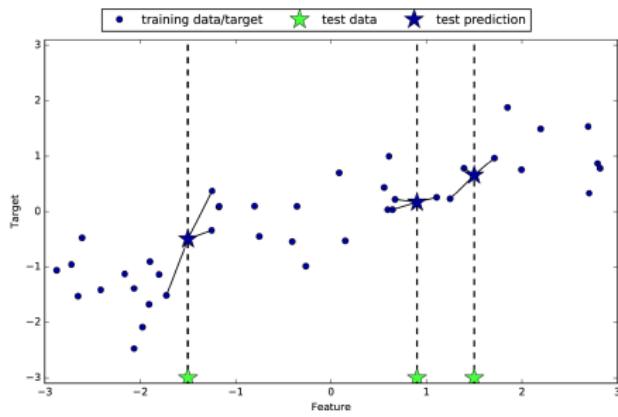
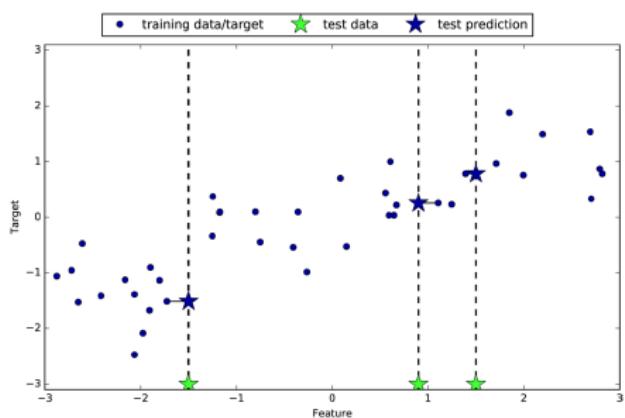
- En este caso observamos como un **k** bajo representa un modelo de alta complejidad, y un **k** alto un modelo de baja complejidad.
- Al ser **k** un valor del cual tenemos control, se le conoce como **hiper-parámetro**.

# kNN - k Nearest Neighbors (Clasificación)

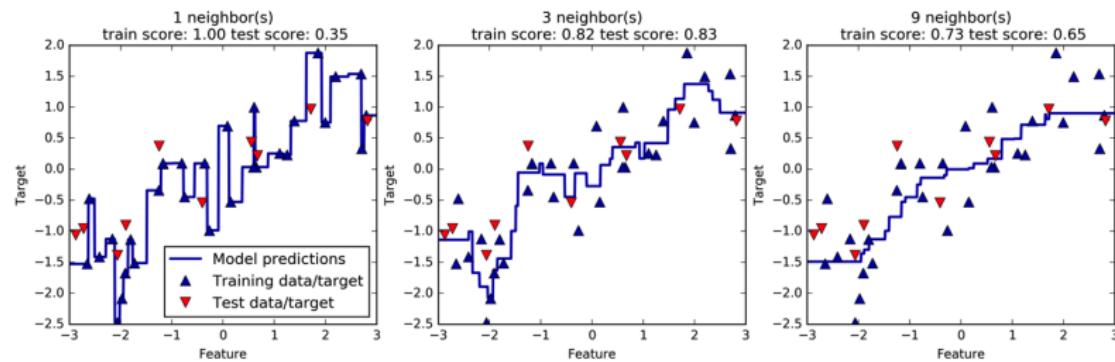
- Se trata de encontrar el punto en el cual: el conjunto de pruebas entregue la mejor precisión de la mano de una precisión del conjunto de entrenamiento “buena” (i.e no perfecta pero tampoco pésima).



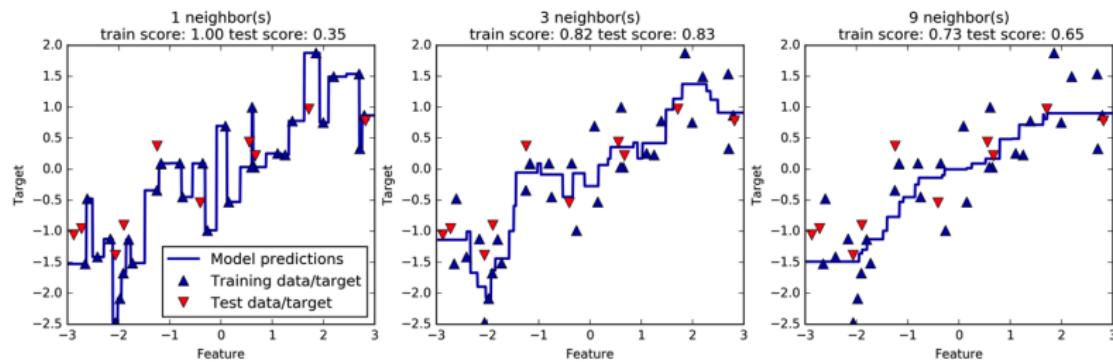
# kNN - k Nearest Neighbors (Regresión)



# kNN - k Nearest Neighbors (Regresión)



# kNN - k Nearest Neighbors (Regresión)



De nuevo, nótese como un **k** bajo representa un modelo de alta complejidad, y un **k** alto un modelo de baja complejidad.

# kNN - k Nearest Neighbors

- Hiper-Parámetros:
  - Requiere de ajuste del valor **k** y de la forma de medir la **distancia**, se asumirá la distancia euclidiana para efectos prácticos.
- Ventajas:
  - Muy fácil de entender.
  - Generalmente entrega buenos resultados sin excesivo ajuste.
- Desventajas:
  - Cuando el conjunto de entrenamiento es muy *grande*, se puede volver lento.
  - No dá buenos resultados cuando hay *muchos* "features" o muchos son ceros.

# naive Bayes

- El teorema de Bayes establece que:

$$P(A|B) = \frac{P(A) \times P(B|A)}{P(B)}$$

donde  $A$  es la hipótesis,  $P(A)$  la probabilidad a priori,  $P(B|A)$  la probabilidad condicional,  $P(B)$  la probabilidad marginal y  $P(A|B)$  la probabilidad a posteriori.

- Por ejemplo, se va fabricar una tarjeta electrónica y se requiere de un componente  $B$ , las partes se compran a 2 proveedores distintos.
- Un 30% se le compra al proveedor  $A_1$  y un 70% al proveedor  $A_2$ .
- Históricamente el 6% de las partes compradas al proveedor  $A_1$  son defectuosas, en el caso del proveedor  $A_2$  solo el 4% son defectuosas.

# naive Bayes

- Se tiene entonces que:
  - $P(A_1) = 0.3$
  - $P(A_2) = 0.7$
  - $P(B|A_1) = 0.06$
  - $P(B|A_2) = 0.04$
  - $P(B) = 6\% \text{ de } 30 = \mathbf{1.8\%} + 4\% \text{ de } 70 = \mathbf{2.8\%} == \mathbf{4.6\%}$

$$P(A_1|B) = \frac{P(A_1) \times P(B|A_1)}{P(B)} = \frac{0.3 \times 0.06}{0.046} = 0.39$$

$$P(A_2|B) = \frac{P(A_2) \times P(B|A_2)}{P(B)} = \frac{0.7 \times 0.04}{0.046} = 0.61$$

# naive Bayes

- Se dice “naive” por el hecho de que supone una idea simple: independencia de los atributos entre sí.
- Se puede encontrar en 3 tipos, dependiendo de las características de los datos:
  - BernoulliNB: Datos son binarios.
  - MultinomialNB: Datos representan un conteo.
  - GaussianNB: Distribución gaussiana de los datos.
- Es eficiente por que aprende a través de la recolección de estadísticas por clase de cada atributo.
  - MultinomialNB: Valor promedio de cada atributo para cada clase.
  - GaussianNB: Valor promedio de cada atributo y STD para cada clase.

# naive Bayes

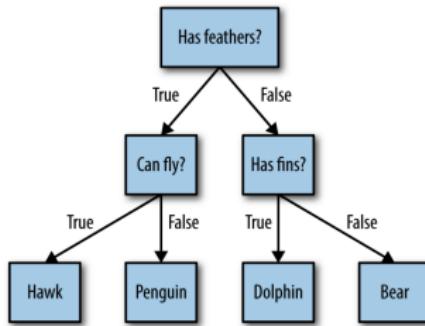
- Algoritmo suele ser muy rápido en el entrenamiento y predicción.
- Se suele usar mucho en grandes conjuntos de datos.
- MultinomialNB y GaussianNB, usan el parámetro  $\alpha$  para suavizado de la generalización, resultando en modelos menos complejos.

```
1 from sklearn.naive_bayes import MultinomialNB  
2 clf = MultinomialNB(alpha=1.0)  
3 clf.fit(X, Y)
```

- GaussianNB se usa con datos de grandes dimensiones, mientras MultinomialNB y BernoulliNB con datos dispersos.

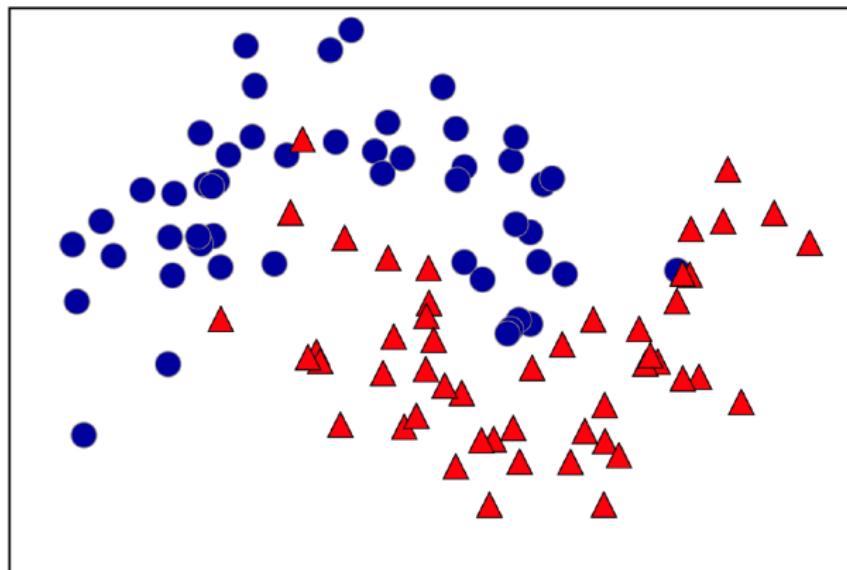
# Decision Trees

- Esquema de aprendizaje basado en preguntas tipo if/else, seguidas de una decisión.



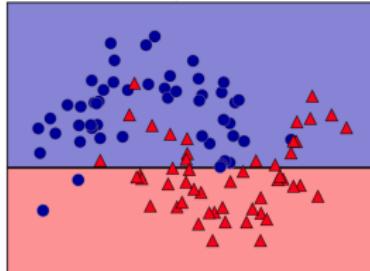
- A cada cuadro, sea una pregunta o nodo terminal, se le llama *hoja*.
- A cada pregunta se le conoce como *prueba*.
- Al primer nodo se le llama *raíz* y es el que parte los datos de la manera más significativa.

# Decision Trees



# Decision Trees

depth = 1



$X[1] \leq 0.0596$   
counts = [50, 50]

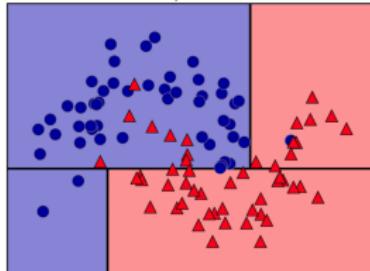
True

counts = [2, 32]

False

counts = [48, 18]

depth = 2



$X[1] \leq 0.0596$   
counts = [50, 50]

True

counts = [2, 32]

False

counts = [48, 18]

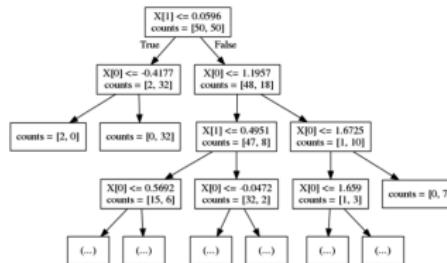
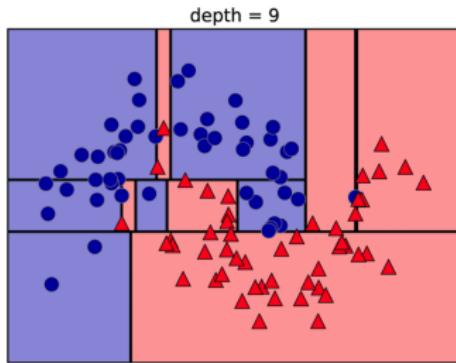
counts = [2, 0]

counts = [0, 32]

counts = [47, 8]

counts = [1, 10]

# Decision Trees



La hoja cuyos datos pertenecen solamente a una clase se llama *pura*.

Queremos llegar a alcanzar un DT con sólo hojas puras?

# Decision Trees

- Un árbol de decisión con sólo hojas puras implica que está 100% ajustado al conjunto de entrenamiento, en decir: **overfitting**.
- Existen dos técnicas para evitarlo:
  - *pre-pruning*: Detener la creación del árbol tempranamente (e.g limitar máximo de hojas o la profundidad del árbol).
  - *pruning (post-pruning)*: Remover o colapsar nodos que contienen poca información.
- Se puede tener una idea de la importancia de los atributos en el árbol, usando el comando:
  - 1 `tree.feature_importances_`
- O bien, mediante algún método de interpretación gráfica.

# Decision Trees

- Los árboles de decisión pueden ser usados para clasificación o regresión:
  - `DecisionTreeClassifier`
  - `DecisionTreeRegressor`
- Es muy recomendado el uso de técnicas para evitar overfitting:
  - `max_depth`
  - `max_leaf_nodes`
  - `min_samples_leaf`
- Son modelos fáciles de visualizar.
- No requieren pre-procesado en exceso por que se desempeña bien con datos en múltiples escalas.

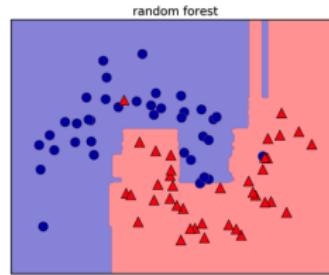
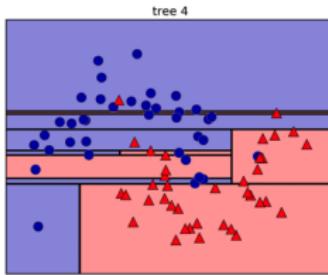
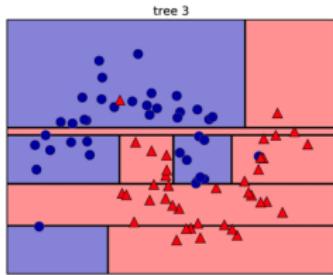
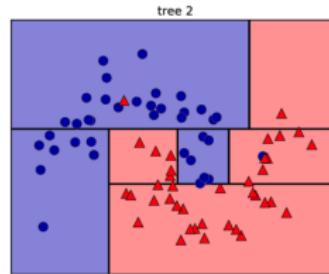
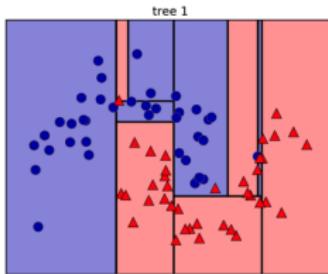
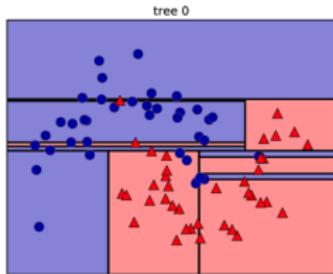
# Random Forest

- Algoritmo que pertenece a la familia de métodos en “conjunto” donde se recurre al uso de múltiples modelos de aprendizaje automático operando como uno sólo.
- Es una propuesta para evitar caer en “overfitting” usando árboles de decisión convencionales.
- Consiste en una colección de árboles de decisión, donde cada árbol es diferente.
- En conjunto se asume algún overfitting para cada árbol, sin embargo al promediar los resultados se reduce el overfitting.
- Su nombre proviene de la forma en la cual se construye cada árbol para mantenerlos diferentes, es una forma aleatoria.

# Random Forest

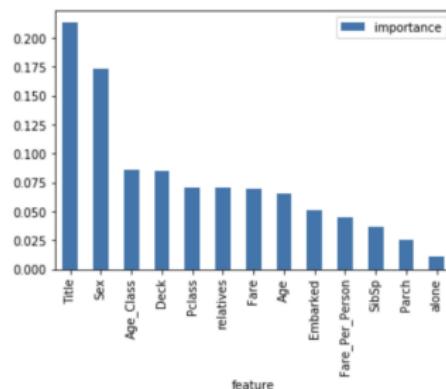
- Para construir el árbol, se debe indicar el número de árboles mediante el parámetro `n_estimators`.
- Es posible usar el método tanto para la clasificación como para la regresión:
  - `RandomForestRegressor`
  - `RandomForestClassifier`
- El parámetro `max_features` define que tan diferentes son los árboles que se crearán. Un valor alto implica similitud en los árboles, uno bajo significa amplia diferencia.
- Para la predicción usando Random Forest, primero se lleva a cabo la predicción para cada árbol y luego si es clasificación se hace un conteo del resultado de cada árbol (gana la mayoría) y si es regresión se estima el promedio.

# Random Forest



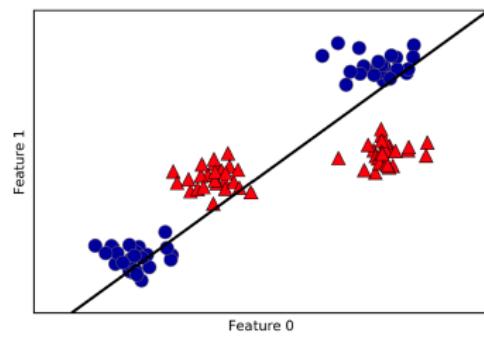
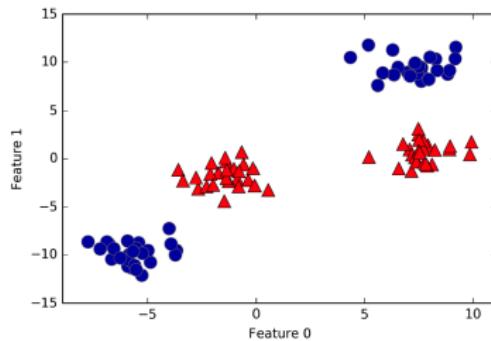
# Random Forest

feature	importance
Title	0.213
Sex	0.173
Age_Class	0.086
Deck	0.085
Pclass	0.071
relatives	0.070
Fare	0.069
Age	0.065
Embarked	0.051
Fare_Per_Person	0.045
SibSp	0.037
Parch	0.025
alone	0.011



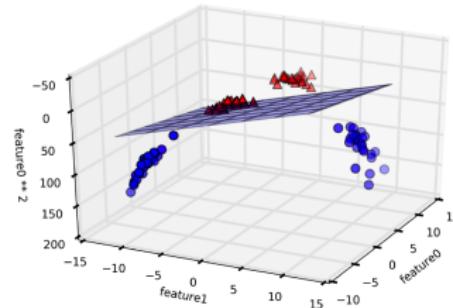
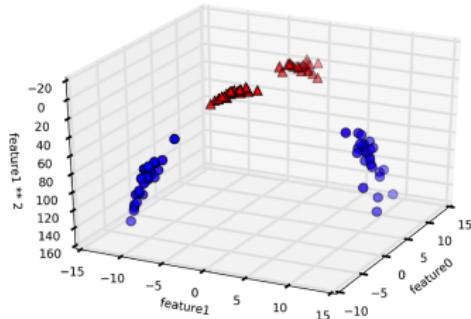
# Kernel SVM

- Extensión del modelo de clasificador para conjuntos de datos más complejos con más dimensiones.
- Los modelos lineales pueden enfrentar limitaciones en su flexibilidad, dadas las limitaciones de las líneas o hiper-planos.



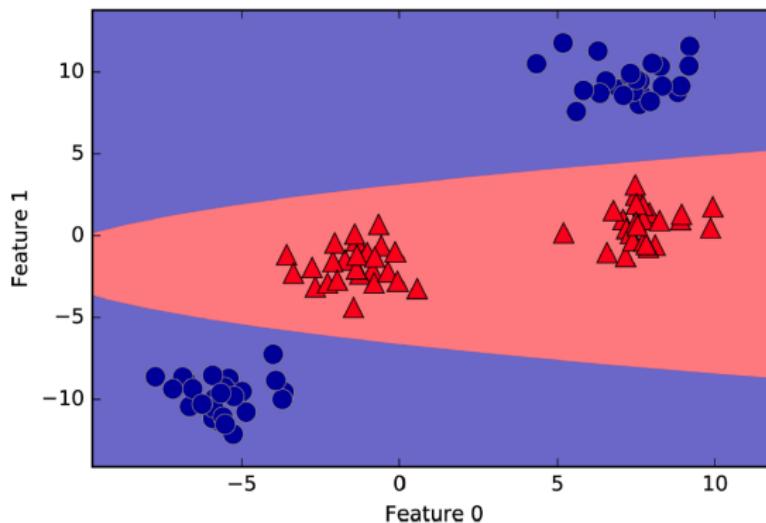
# Kernel SVM

- Una forma de mejorar la flexibilidad es agregando más atributos, bajo criterios técnicos.
- Por ejemplo: agregar en set de datos anterior un tercer atributo llamado ( $\text{feature1}^{**2}$ ).
- Un plano en 3D es ahora capaz de separar adecuadamente las clases.



# Kernel SVM

- Si llevamos el límite de decisión como una función de los 2 atributos originales, vemos como la linealidad se pierde y más bien se tiene una especie de elipse.



# Kernel SVM

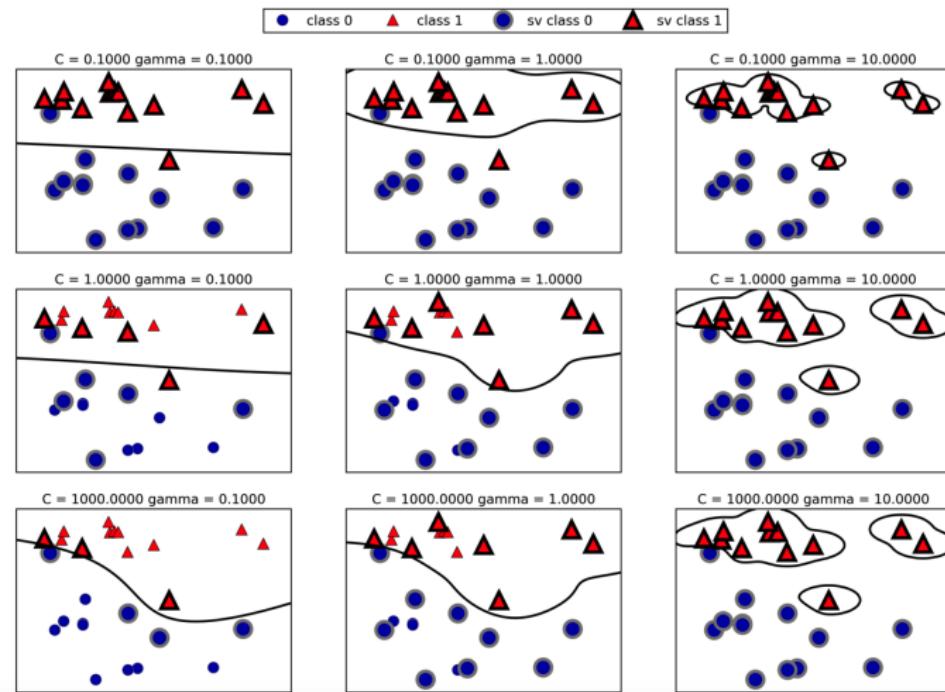
- Agregar atributos no lineales puede hacer que los modelos lineales sean más efectivos.
- Hay que tener cuidado con agregar más atributos, en especial en términos del costo computacional.
- La clave: El uso de un **kernel**.
- Encontramos dos tipos de kernels muy comunes:
  - *Polinomial*: Se hace un cálculo de todos los posibles polinomios hasta un cierto grado de los atributos originales.
  - *RBF*: También conocido como kernel gaussiano, considera todos los posibles polinomios de todos los grados pero a mayor grado del polinomio menor relevancia tienen el atributo.

# Kernel SVM

- Se dispone de varios hiper-parámetros de ajuste, donde los más comunes son:
  - *gamma*: Controla el ancho del kernel gaussiano, define la escala que determina la cercanía entre dos puntos.
  - *C*: Es el parámetro de regularización, limita el nivel de importancia de cada punto.

```
1 from sklearn.svm import SVC  
2 X, y = % dataset %  
3 svm = SVC(kernel='rbf', C=10, gamma=0.1).fit(X, y)
```

# Kernel SVM



# Kernel SVM

- Un  $\gamma$  bajo implica radio alto del kernel gaussiano, lo que implica que los puntos se consideran cercanos y se refleja como un límite de decisión menos complejo y más suave.
- Un  $\gamma$  alto implica radio pequeño del kernel gaussiano, lo que implica que los puntos se consideran lejanos y se refleja como un límite de decisión más complejo.
- Un  $C$  bajo implica que menos puntos tienen influencia en el límite de decisión.
- Un  $C$  alto implica que más puntos tienen influencia en el límite de decisión.

# Kernel SVM

- Se desempeñan muy bien en la mayoría de conjuntos de datos.
- Trabajan bien en conjuntos de pocas o muchas dimensiones.
- Con muchas instancias o ejemplos de datos puede consumir muchos recursos computacionales.
- El pre-procesamiento de los datos es muy común en kSVM, factor por el que random forest es muy usado.
- Son más complejos de explicar y de inspeccionar con detalle durante la operación.

# Questions?



Felipe Meza - fmeza@itcr.ac.cr