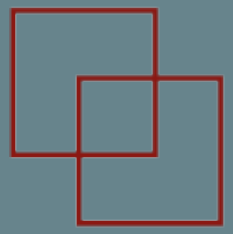
Tema 5. Fase de modelado

Dr. Manuel Castillo-Cara
Intelligent Ubiquitous Technologies – Smart Cities (IUT-SCi)
Web: www.smartcityperu.org

Índice

- Algoritmos de Machine Learning.
  - Algoritmos lineales.
  - Algoritmos no lineales.
- Rendimiento de los algoritmos.
- Algoritmos ensamblados.
  - Bagging.
  - Boosting.
  - Voting.
- Algoritmo Super Lerner

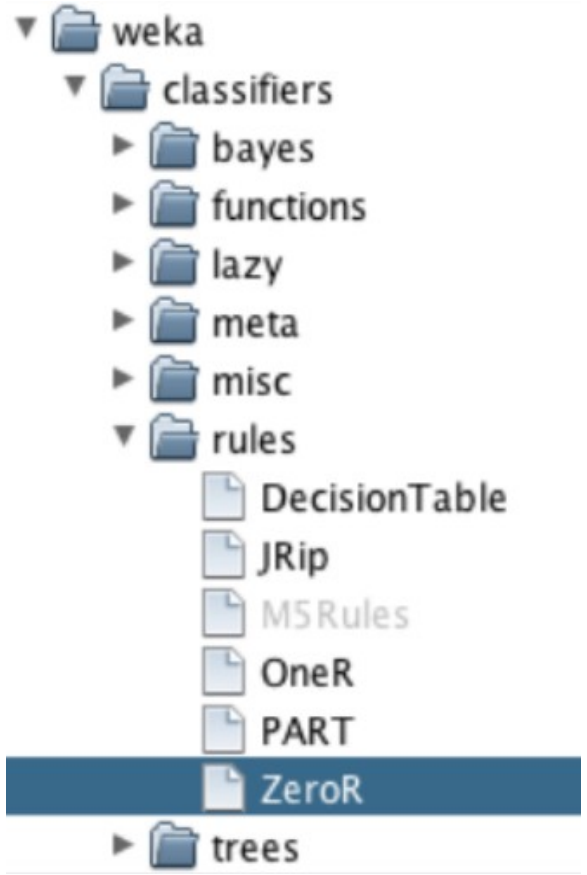


Smart City

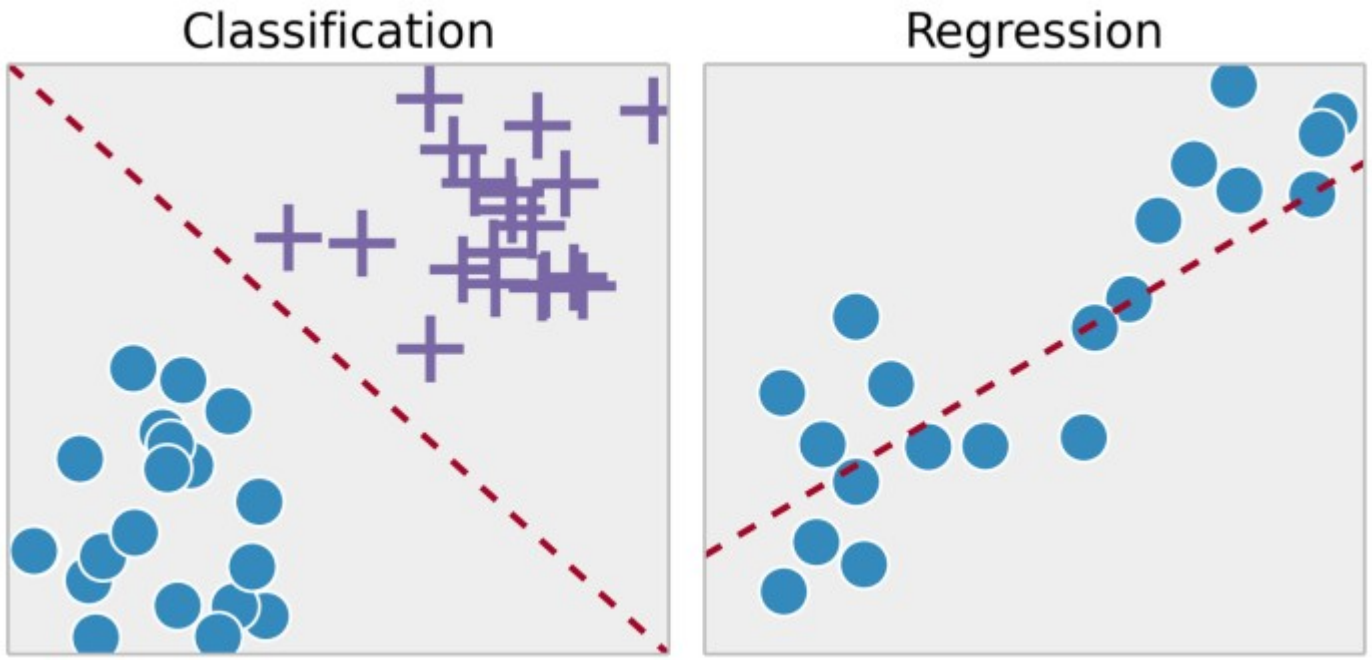
LAB CTIC UNI

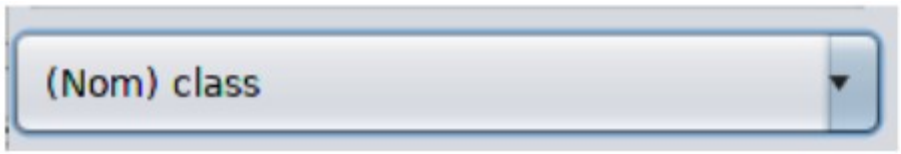
Algoritmos de Machine Learning

3.1. Algoritmos

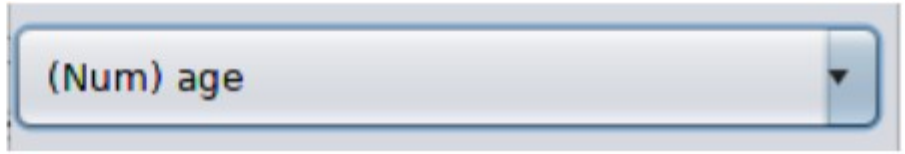


3.2. Clasificación Vs. Regresión





(a) Muestra de un atributo nominal.



(b) Muestra de un atributo numérico.

3.3. Taxonomía de los algoritmos

- **Lineales**: el valor objetivo se exprese como una combinación lineal de valores constantes o el producto entre un parámetro y una variable predictiva.
- **No lineales**: no se utilizan funciones como en los lineales.
- **Ensamblados**: combinan las predicciones de múltiples modelos para hacer predicciones más robusta.

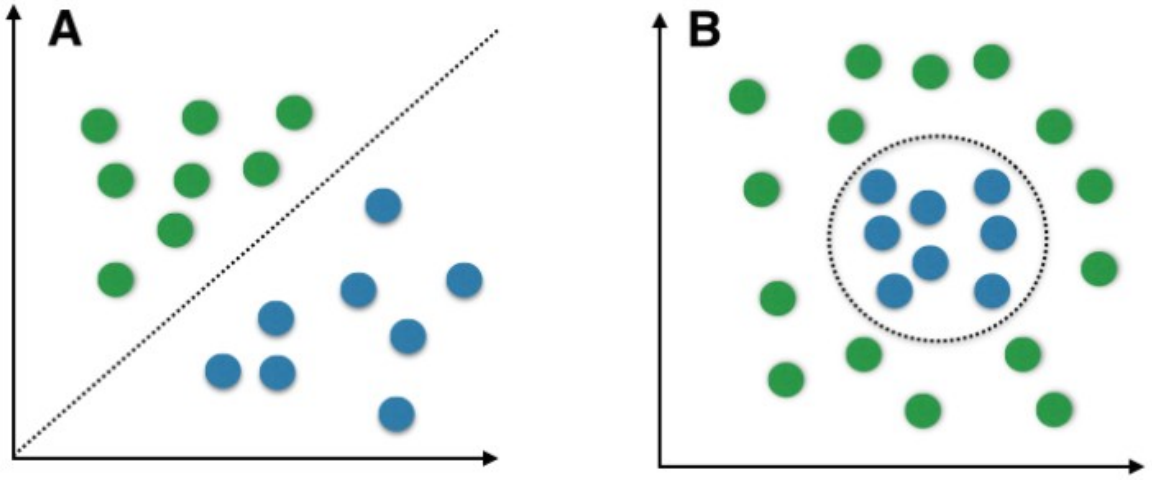
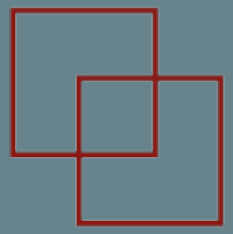


FIGURA 5.19: Algoritmos lineales Vs. No lineales.

3.4. Qué algoritmos revisar

- Probar una mezcla de representaciones de algoritmos:
  - Basados en instancias: Cuando se quiere clasificar un nuevo objeto, se extraen los objetos mas parecidos y se usa su clasificación para clasificar al nuevo objeto.
  - Árboles: Bifurcando y modelando los posibles caminos tomados y su probabilidad de ocurrencia para mejorar su precisión.
- Prueba una mezcla de algoritmos de aprendizaje (por ejemplo, diferentes algoritmos para aprender el mismo tipo de representación).
- Prueba una mezcla de tipos de modelos (por ejemplo, funciones lineales y no lineales o paramétricas y no paramétricas).



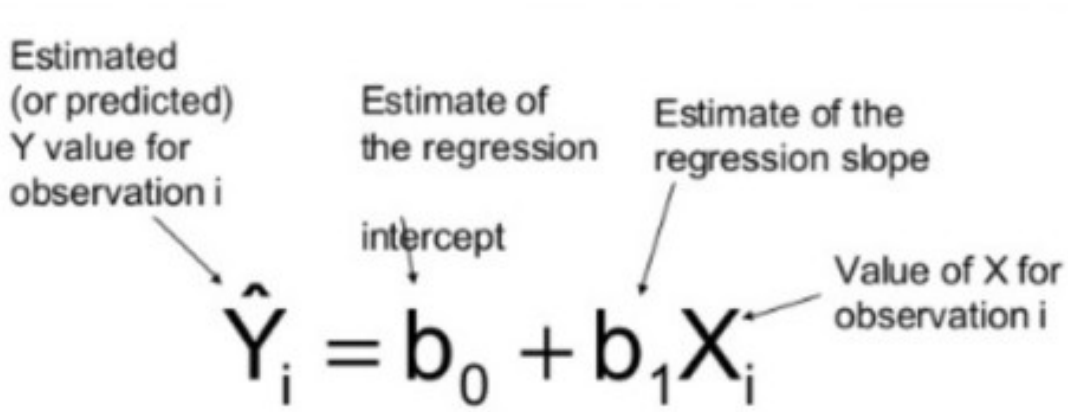
Smart City

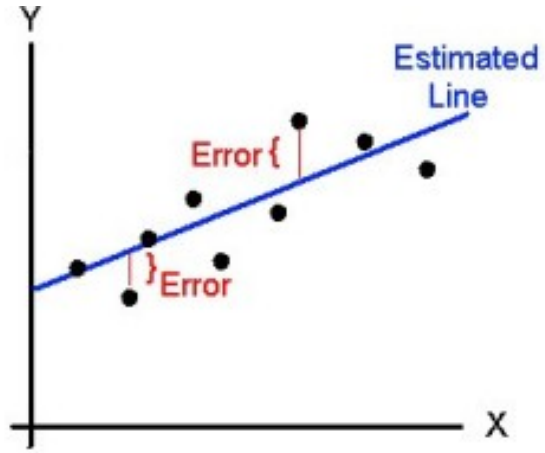
LAB CTIC UNI

Algoritmos lineales

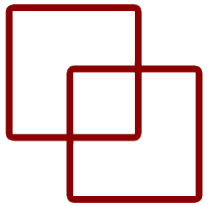
1. Linear Regression

- Para problemas de regresión.
- Es una aproximación para modelar la relación entre una variable escalar dependiente '*y*' y una o mas variables explicativas nombradas con '*X*'.
- En otras palabras, este modelo lo que realiza es "dibujar una recta" que nos indicará la tendencia de un conjunto de datos continuos.
- Se utiliza la clase *LinearRegression*.





1. Linear Regression - código



```
# Linear Regression
from sklearn.linear_model import LinearRegression

kfold = KFold(n_splits=10, random_state=7)
model = LinearRegression()
scoring = 'neg_mean_squared_error'
results = cross_val_score(model, X_reg, Y_reg, cv=kfold, scoring=scoring)
print(f"MSE: {results.mean()}")
```

```
MSE: -34.70525594452488
```

2. Ridge Regression

- Para problemas de regresión.
- Es una extensión de LiR donde la función de pérdida se modifica para minimizar la complejidad del modelo medido como el valor de la suma cuadrática de los valores del coeficiente (también llamada ***norma L2***).
- Se utiliza la clase *Ridge*.

```
# Ridge Regression
from sklearn.linear_model import Ridge

num_folds = 10
kfold = KFold(n_splits=10, random_state=7)
model = Ridge()
scoring = 'neg_mean_squared_error'
results = cross_val_score(model, X_reg, Y_reg, cv=kfold, scoring=scoring)
print(f"MSE: {results.mean()}")
```

```
MSE: -34.07824620925939
```

3. LASSO

- Para problemas de regresión.
- Es una modificación de LiR, como RiR, donde la función de pérdida se modifica para minimizar la complejidad del modelo medido como el valor absoluto de los valores del coeficiente (también llamada la ***norma L1***).
- Se utiliza la clase *Lasso*.

```
# LASSO Regression
from sklearn.linear_model import Lasso

kfold = KFold(n_splits=10, random_state=7)
model = Lasso()
scoring = 'neg_mean_squared_error'
results = cross_val_score(model, X_reg, Y_reg, cv=kfold, scoring=scoring)
print(f"MSE: {results.mean()}")
```

```
MSE: -34.46408458830232
```

4. ElasticNet

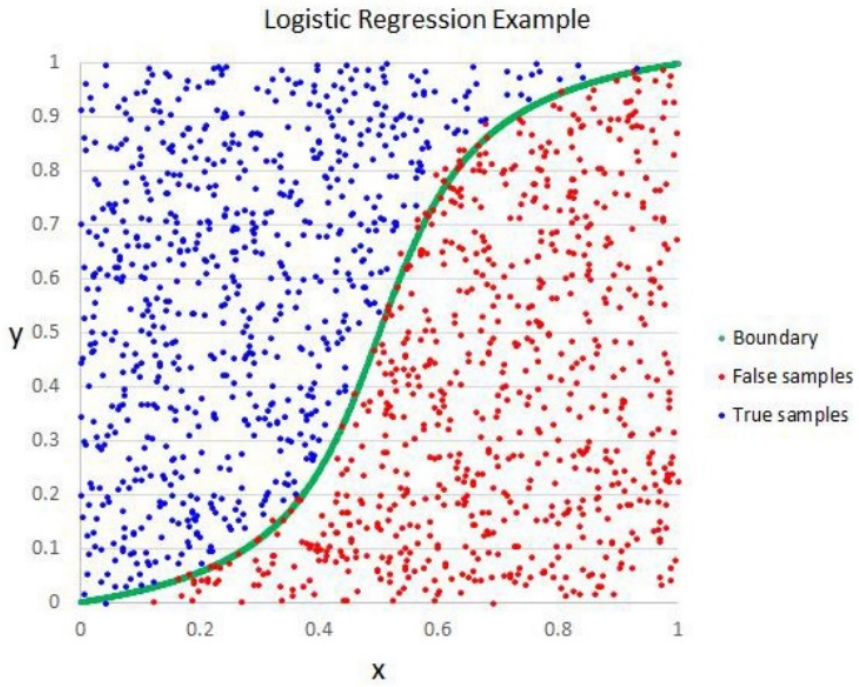
- Para problemas de regresión.
- Combina las propiedades de RIR y LASSO.
- Busca minimizar la complejidad del modelo de regresión (magnitud y número de coeficientes de regresión) penalizando el modelo utilizando tanto la ***norma L2*** (valores de coeficiente de suma cuadrática) como la ***norma L1*** (valores de coeficiente absoluto de suma).
- Se utiliza la clase *ElasticNet*.

```
# ElasticNet Regression
from sklearn.linear_model import ElasticNet

kfold = KFold(n_splits=10, random_state=7)
model = ElasticNet()
scoring = 'neg_mean_squared_error'
results = cross_val_score(model, X_reg, Y_reg, cv=kfold, scoring=scoring)
print(f"MSE: {results.mean()}")
```

```
MSE: -31.164573714249762
```

5. Logistic Regression

- Para problemas de clasificación binaria.
- Este modelo ayuda a determinar si la entrada pertenece a un sector específico.
- Utiliza la función sigmoide que tiene un rango de valores de salida entre 0 y 1.
- Se utiliza la clase *LogisticRegression*.

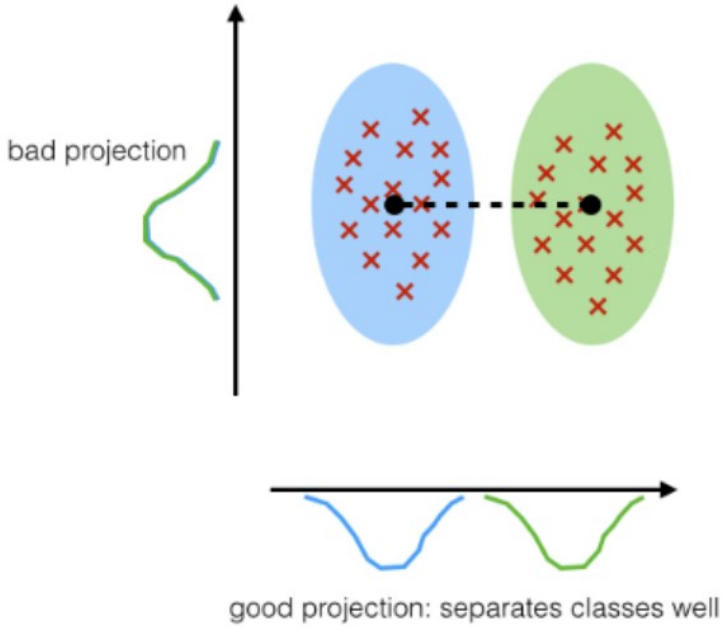



```
# Logistic Regression Classification
from sklearn.linear_model import LogisticRegression

num_folds = 10
kfold = KFold(n_splits=10, random_state=7)
model = LogisticRegression(solver = 'lbfgs', max_iter=1000)
results = cross_val_score(model, X_cla, Y_cla, cv=kfold)
print(f"Accuracy: {results.mean()*100.0:,.2f}% ({results.std()*100.0:,.2f}%)")
```

```
Accuracy: 77.60% (5.16%)
```

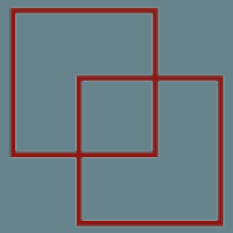
6. Linear Discriminant Analysis

- Para problemas de clasificación.
- También supone una distribución gaussiana para las variables de entrada numéricas.
- Se utiliza la clase *LinearDiscriminantAnalysis*




```
# LDA Classification
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

num_folds = 10
kfold = KFold(n_splits=10, random_state=7)
model = LinearDiscriminantAnalysis()
results = cross_val_score(model, X_cla, Y_cla, cv=kfold)
print(f"Accuracy: {results.mean()*100.0:,.2f}% ({results.std()*100.0:,.2f}%)")
```

```
Accuracy: 77.35% (5.16%)
```



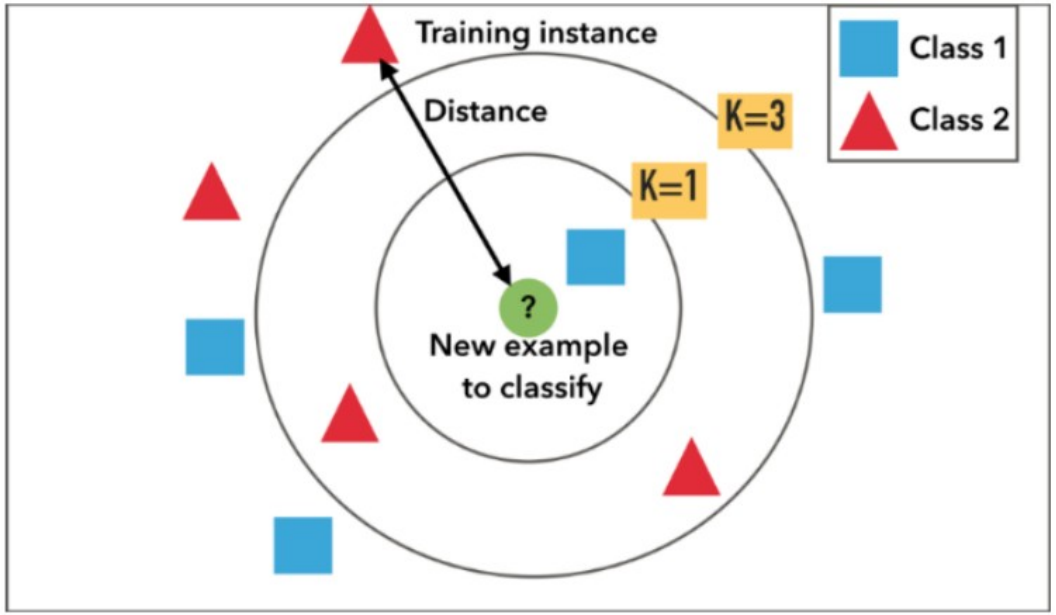
Smart City

LAB CTIC UNI

Algoritmos no lineales

1. *k*-Nearest Neighbors

- Clasifica la entrada basándose en una medida de similitud, que a menudo es la distancia en el espacio de los puntos de datos.
- Se hace una predicción eligiendo la clase más frecuente entre los *k* vecinos más cercanos.
- Clasificación: *KNeighborsClassifier*
- Regresión: *KNeighborsRegressor*



1. *k*-NN – código

CLASIFICACIÓN

```
# KNN Classification
from sklearn.neighbors import KNeighborsClassifier

num_folds = 10
kfold = KFold(n_splits=10, random_state=7)
model = KNeighborsClassifier()
results = cross_val_score(model, X_cla, Y_cla, cv=kfold)
print(f"Accuracy: {results.mean()*100.0:,.2f}% ({results.std()*100.0:,.2f}%)")
```

```
Accuracy: 72.66% (6.18%)
```

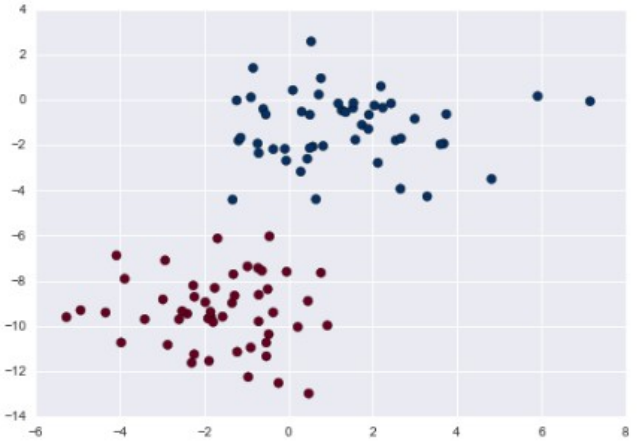
REGRESIÓN

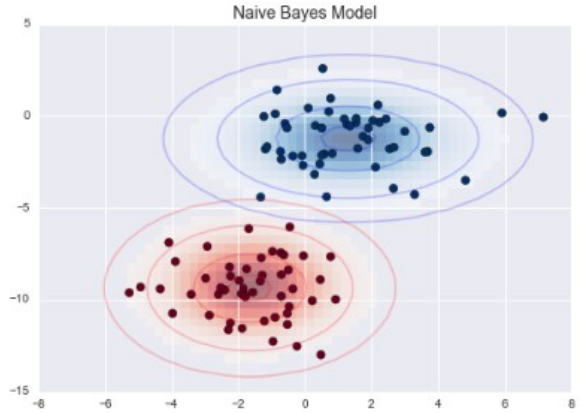
```
# k-NN Regressionore
from sklearn.neighbors import KNeighborsRegressor

kfold = KFold(n_splits=10, random_state=7)
model = KNeighborsRegressor()
scoring = 'neg_mean_squared_error'
results = cross_val_score(model, X_reg, Y_reg, cv=kfold, scoring=scoring)
print(f"MSE: {results.mean()}")
```

```
MSE: -107.28683898039215
```

2. Naive Bayes

- Realiza la clasificación asumiendo la independencia entre las características (ingenuo) y calcula las clases según la probabilidad bayesiana.
- Clasificación: *GaussianNB*



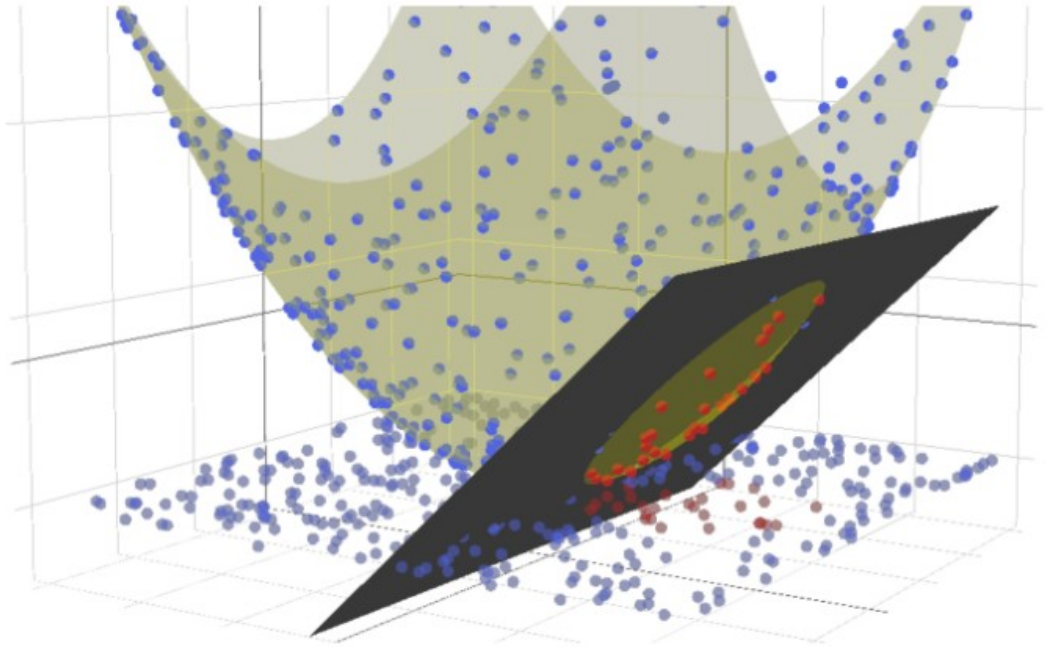



```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB

kfold = KFold(n_splits=10, random_state=7)
model = GaussianNB()
results = cross_val_score(model, X_cla, Y_cla, cv=kfold)
print(f"Accuracy: {results.mean()*100.0:,.2f}% ({results.std()*100.0:,.2f}%)")
```

```
Accuracy: 75.52% (4.28%)
```

3. Support Vector Machine

- Dados los datos en el espacio, SVM construye hiperplanos en un espacio de alta dimensión con una brecha máxima entre ellos.
- Con la ayuda de las funciones del kernel, puede realizar la clasificación de datos de alta dimensión.
- Clasificación: *SVC*
- Regresión: *SVR*



3. SVM – código

CLASIFICACIÓN

```
# SVM Classification
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC

kfold = KFold(n_splits=10, random_state=7)
model = SVC(gamma='scale')
results = cross_val_score(model, X_cla, Y_cla, cv=kfold)
print(f"Accuracy: {results.mean()*100.0:,.2f}% ({results.std()*100.0:,.2f}%)")
```

```
Accuracy: 76.04% (5.29%)
```

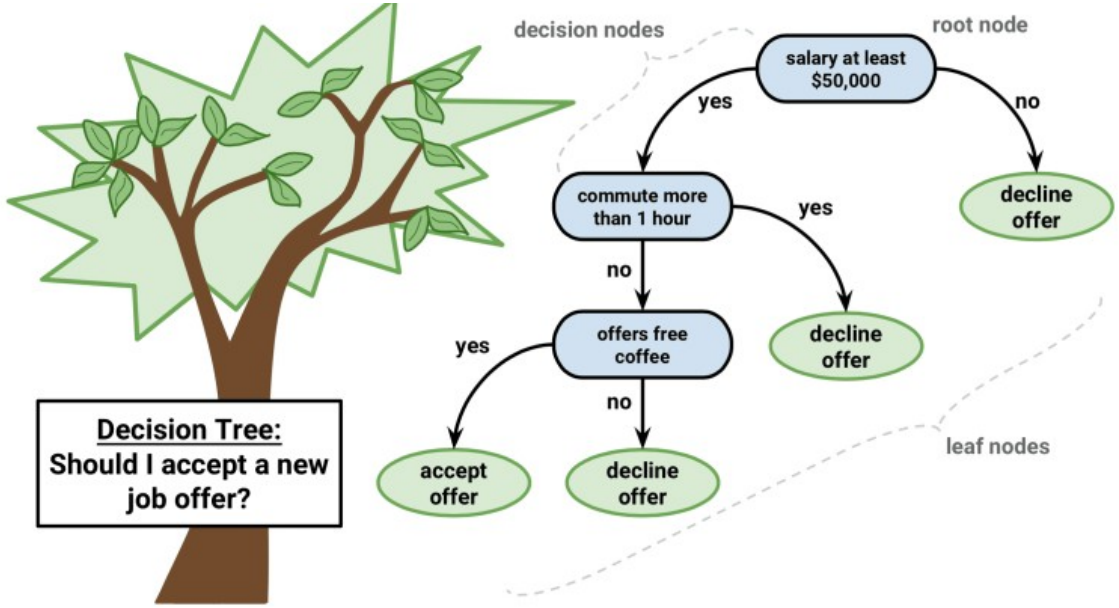
REGRESIÓN

```
# SVM Regression
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVR

num_folds = 10
kfold = KFold(n_splits=10, random_state=7)
model = SVR(gamma='auto')
scoring = 'neg_mean_squared_error'
results = cross_val_score(model, X_reg, Y_reg, cv=kfold, scoring=scoring)
print(f"MSE: {results.mean()}")
```

```
MSE: -91.04782433324428
```

4. Classification and Regression Trees

- Son modelos predictivos que coloca las observaciones realizadas a partir de los datos en las ramas; estos conducen a las hojas que están etiquetadas con la clasificación correcta.
- Utiliza un conjunto discreto de valores, y las hojas producen el resultado final.
- Tienen mejor comportamiento con atributos discretos (dummy, categóricos) → Es recomendable convertir si están los atributos en numéricos.
- Clasificación: *DecisionTreeClassifier*
- Regresión: *DecisionTreeRegressor*



4. CART – código

CLASIFICACIÓN

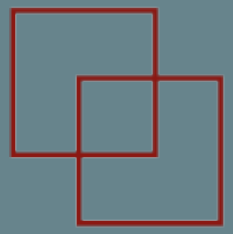
```
# CART Classification
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier

kfold = KFold(n_splits=10, random_state=7)
model = DecisionTreeClassifier()
results = cross_val_score(model, X_cla, Y_cla, cv=kfold)
print(f"Accuracy: {results.mean()*100.0:,.2f}% ({results.std()*100.0:,.2f}%)")
```

```
Accuracy: 70.83% (5.90%)
```

REGRESIÓN

```
# CART Regression
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeRegressor

kfold = KFold(n_splits=10, random_state=7)
model = DecisionTreeRegressor()
scoring = 'neg_mean_squared_error'
results = cross_val_score(model, X_reg, Y_reg, cv=kfold, scoring=scoring)
print(f"MSE: {results.mean()}")
```

```
MSE: -38.80579843137255
```



Smart City

LAB CTIC UNI

Rendimiento de los algoritmos

1. Evaluar el rendimiento

- Cómo comparar la habilidad del modelo usando la tabla resumen.
- Cómo revisar y comparar habilidades de modelos usando diferentes gráficos.
- Cómo comparar la habilidad del modelo usando gráficos entre pares de ellos.
- Cómo verificar si la diferencia en la habilidad del modelo es estadísticamente significativa.

2. Escoger el mejor modelo

- **Preparar el conjunto de datos**.
  - Proceso de carga de los paquetes y el conjunto de datos para entrenar a los modelos.
- **Resultados el modelo**.
  - Entrenar modelos estándar de machine learning en el conjunto de datos para su evaluación.
- **Comparar los modelos**.
  - Comparar los modelos entrenados usando 8 técnicas diferentes.

3. Preparar el conjunto de datos

LIBRERÍAS

```
#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

DATASET

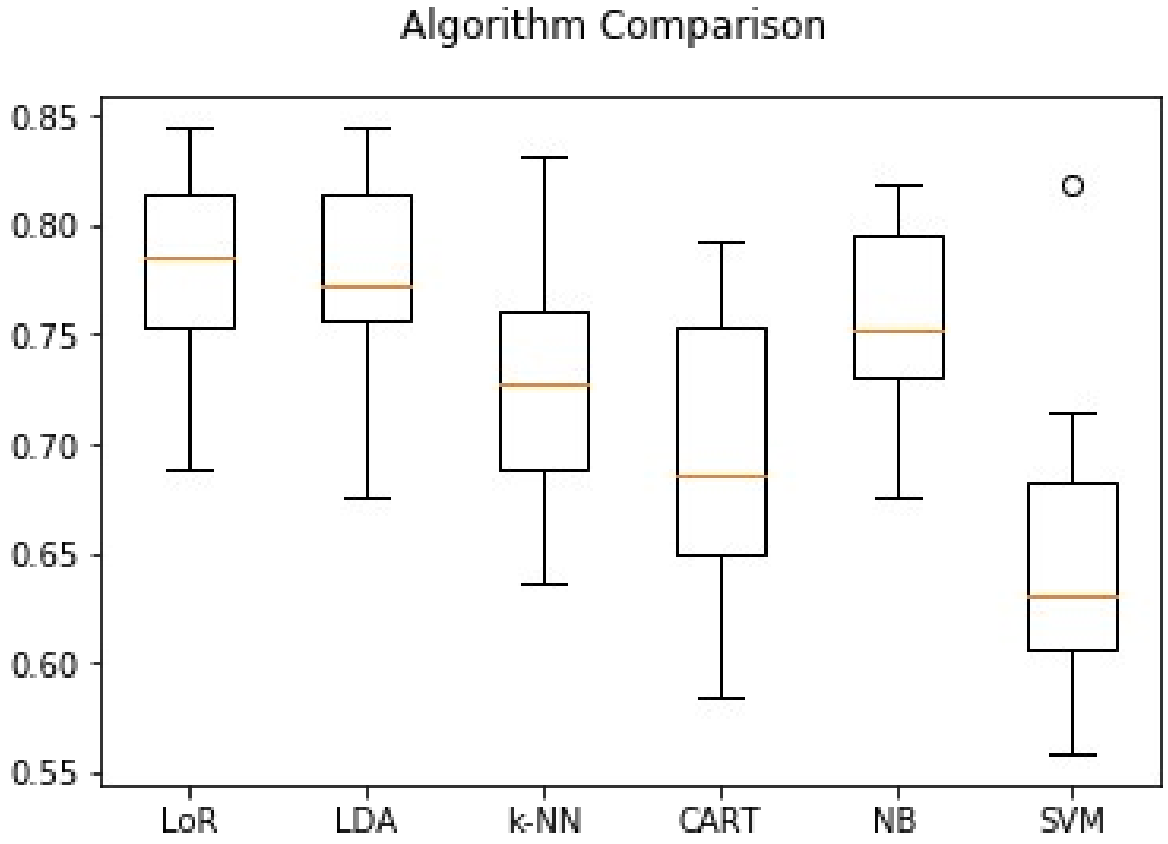
```
# Clasification problem
filename = 'data/pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pd.read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
```

4. Resultados de los modelos

```
# Compare Algorithms
# prepare models
models = []
models.append(('LoR', LogisticRegression(solver='lbfgs', max_iter=1000)))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('k-NN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))
# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
    kfold = KFold(n_splits=10, random_state=7)
    cv_results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    print(f"{name}: {cv_results.mean()*100.0:,.2f}% ({cv_results.std()*100.0:,.2f}%)")
```

```
LoR: 77.60% (5.16%)
LDA: 77.35% (5.16%)
k-NN: 72.66% (6.18%)
CART: 69.52% (6.47%)
NB: 75.52% (4.28%)
SVM: 65.10% (7.21%)
```

5. Visualizar resultados

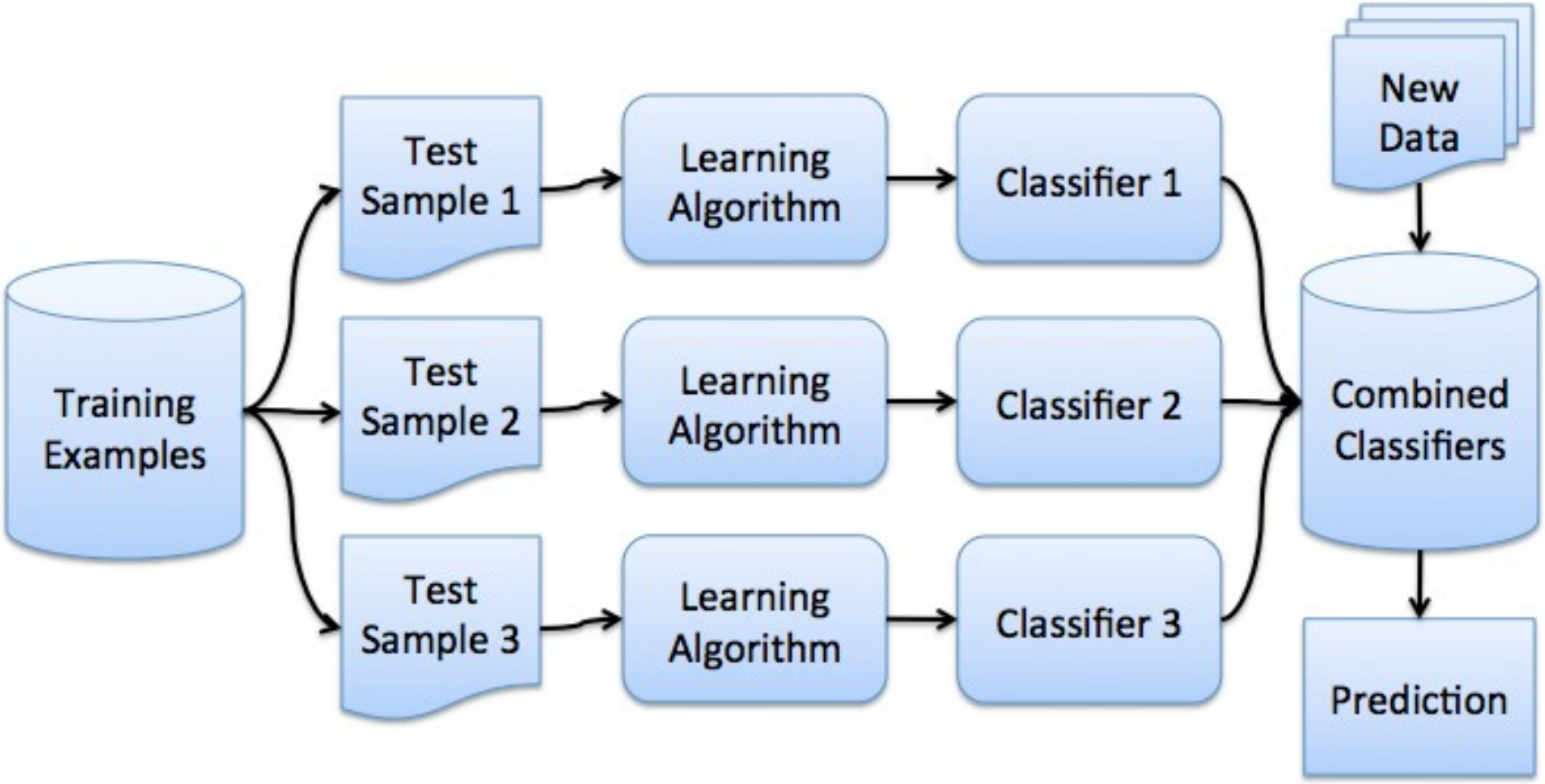


Smart City

LAB CTIC UNI

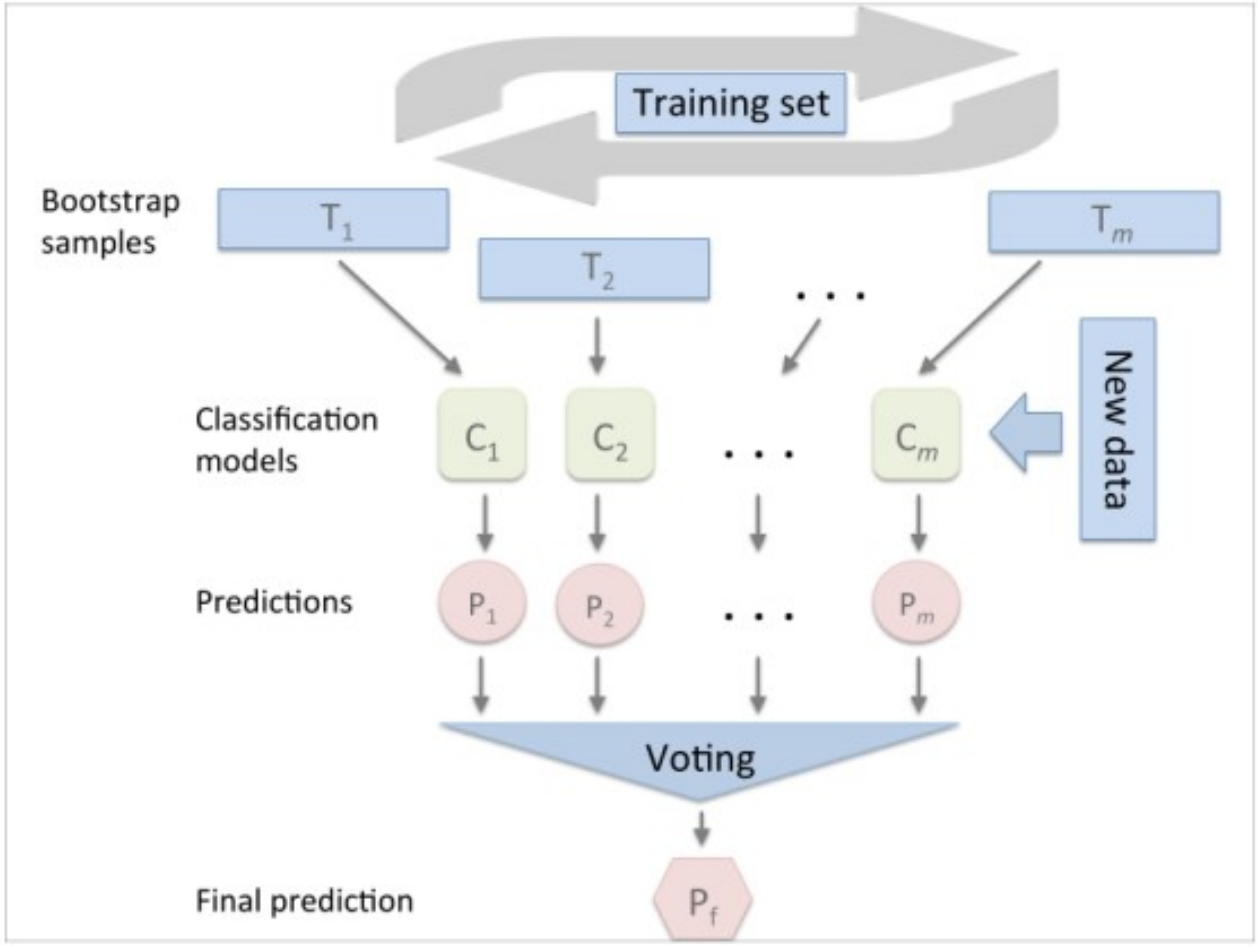
Algoritmos ensamblados

1. Algoritmos de conjunto



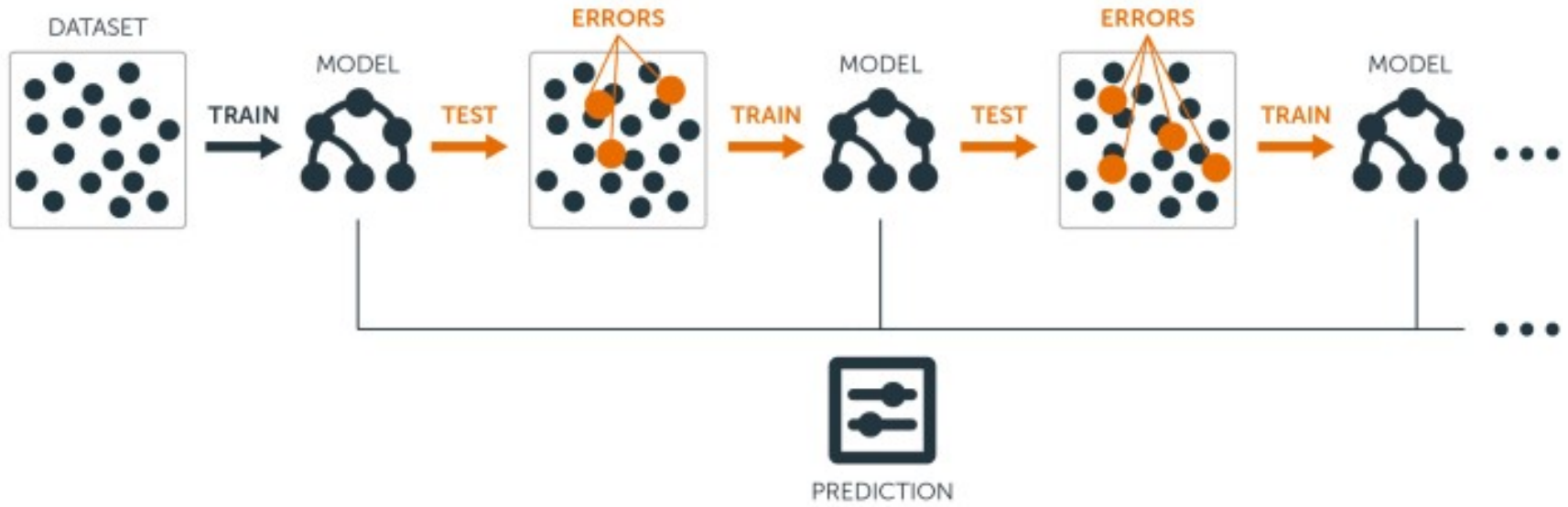
2. Bagging

- Construye múltiples modelos (típicamente modelos del mismo tipo) a partir de diferentes submuestras del conjunto de datos de entrenamiento. Algoritmos:
  - Bagged Decision Trees
  - Random Forest
  - Extra Trees



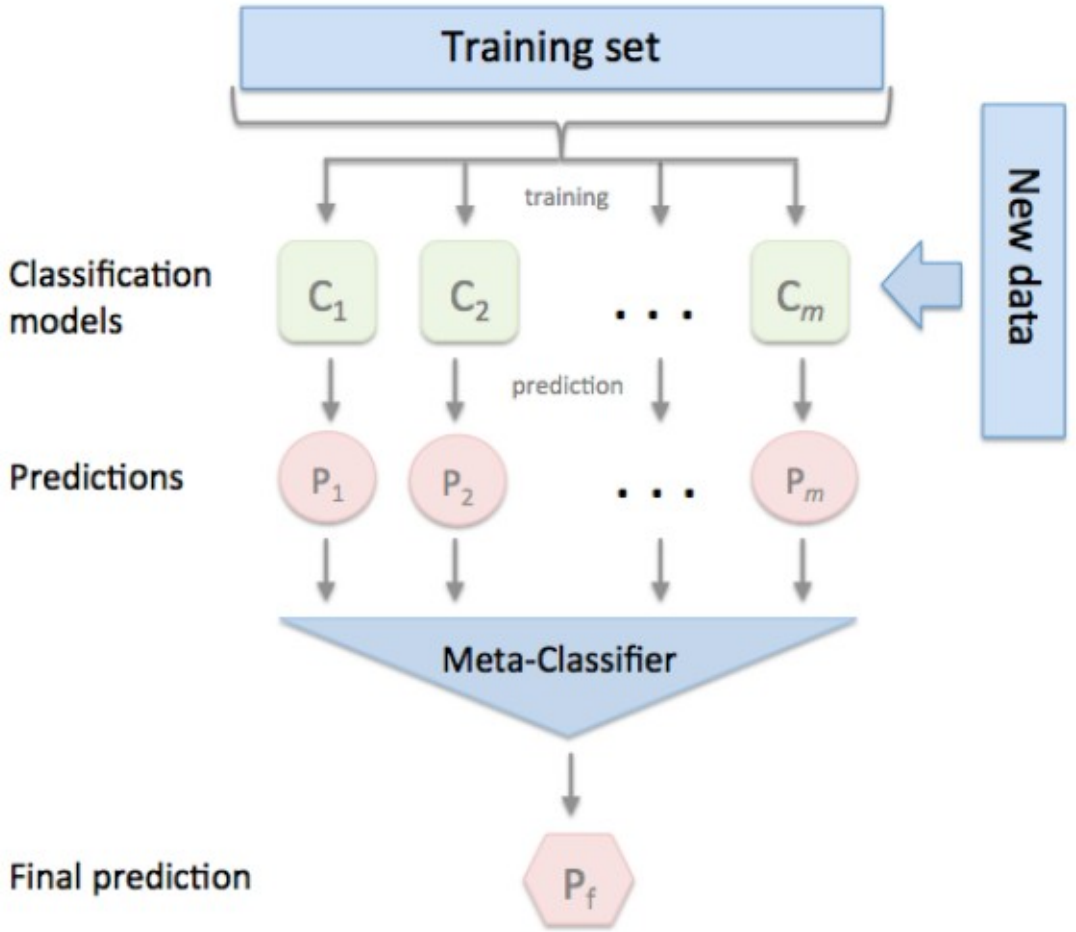
3. Boosting

- Construye múltiples modelos (típicamente modelos del mismo tipo), cada uno de los cuales aprende a corregir los errores de predicción de un modelo anterior en la cadena.
  - AdaBoost
  - Stochastic Gradient Boosting



4. Voting

- Construye múltiples modelos (típicamente modelos de diferentes tipos) y un modelo supervisado que aprende cómo combinar mejor las predicciones de los modelos primarios



Smart City

LAB CTIC UNI

Algoritmos Bagging

1. Bagged Decision Trees

- Bagging funciona mejor con algoritmos que tienen una alta varianza.
  - Un ejemplo popular son los árboles de decisión, a menudo construidos sin poda.
- En el ejemplo a continuación, se muestra un ejemplo del uso del *BaggingClassifier* con CART *(DecisionTreeClassifier)*.
  - Se crean un total de 100 árboles.

```
# Bagged Decision Trees for Classification
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

seed = 7
kfold = KFold(n_splits=10, random_state=seed)
cart = DecisionTreeClassifier()
num_trees = 100
model = BaggingClassifier(base_estimator=cart, n_estimators=num_trees, random_state=seed)
results = cross_val_score(model, X, Y, cv=kfold)
print(f"Accuracy: {results.mean()*100.0:,.2f}% ({results.std()*100.0:,.2f}%)")
```

```
Accuracy: 77.07% (7.39%)
```

2. Random Forest

- Las muestras del conjunto de datos de entrenamiento se toman con reemplazo, pero los árboles se construyen de una manera que reduce la correlación entre clasificadores individuales.
  - Específicamente, en lugar de elegir con avidez el mejor punto de división en la construcción de cada árbol, solo se considera un subconjunto aleatorio de características para cada división.
- Puede construir un modelo de Random Forest para la clasificación utilizando la clase *RandomForestClassifier*.
- El siguiente ejemplo se construye 100 árboles y puntos divididos elegidos de una selección aleatoria de 3 características.

```
# Random Forest Classification
from sklearn.ensemble import RandomForestClassifier

num_trees = 100
max_features = 3
kfold = KFold(n_splits=10, random_state=7)
model = RandomForestClassifier(n_estimators=num_trees, max_features=max_features)
results = cross_val_score(model, X, Y, cv=kfold)
print(f"Accuracy: {results.mean()*100.0:,.2f}% ({results.std()*100.0:,.2f}%)")
```

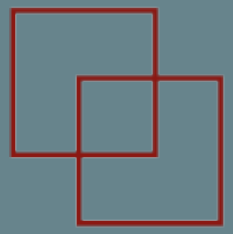
```
Accuracy: 77.34% (7.62%)
```

3. Extra Trees

- Extra Trees son otra modificación del ensacado donde se construyen árboles aleatorios a partir de muestras del conjunto de datos de entrenamiento.
- Puede construir un modelo Extra Trees para clasificación usando la clase *ExtraTreesClassifier*.
- El siguiente ejemplo proporciona una demostración de árboles adicionales con el número de árboles establecido en 100 y las divisiones elegidas entre 7 características aleatorias.

```
# Extra Trees Regression
from sklearn.ensemble import ExtraTreesClassifier

num_trees = 100
max_features = 7
kfold = KFold(n_splits=10, random_state=7)
model = ExtraTreesClassifier(n_estimators=num_trees, max_features=max_features)
results = cross_val_score(model, X, Y, cv=kfold)
print(f"Accuracy: {results.mean()*100.0:,.2f}% ({results.std()*100.0:,.2f}%)")
```

```
Accuracy: 75.90% (6.78%)
```



Smart City

LAB CTIC UNI

Algoritmos Boosting

1. AdaBoost

- AdaBoost fue quizás el primer algoritmo de conjunto Boosting exitoso.
- Generalmente funciona ponderando las instancias en el conjunto de datos según lo fácil o difícil que es clasificarlas, lo que permite que el algoritmo les preste más o menos menos atención en la construcción de modelos posteriores.
- Puede construir un modelo AdaBoost para clasificación utilizando la clase *AdaBoostClassifier*.
- El siguiente ejemplo demuestra la construcción de 30 árboles de decisión en secuencia.

```
# AdaBoost for Classification
from sklearn.ensemble import AdaBoostClassifier

num_trees = 30
seed=7
kfold = KFold(n_splits=10, random_state=seed)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
results = cross_val_score(model, X, Y, cv=kfold)
print(f"Accuracy: {results.mean()*100.0:,.2f}% ({results.std()*100.0:,.2f}%)")
```

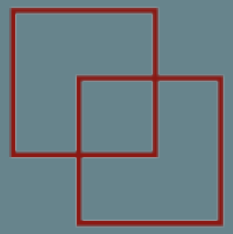
```
Accuracy: 76.05% (5.44%)
```

2. Stochastic Gradient Boosting

- Es una de las técnicas de conjunto más sofisticadas.
- También es una técnica que está demostrando ser quizás una de las mejores técnicas disponibles para mejorar el rendimiento a través de conjuntos.
- Puede construir un modelo para la clasificación utilizando la clase GradientBoostingClassifier.
- El siguiente ejemplo crea un GBM con 100 árboles.

```
# Gradient Boosting Machine for Classification
from sklearn.ensemble import GradientBoostingClassifier

seed = 7
num_trees = 100
kfold = KFold(n_splits=10, random_state=seed)
model = GradientBoostingClassifier(n_estimators=num_trees, random_state=seed)
results = cross_val_score(model, X, Y, cv=kfold)
print(f"Accuracy: {results.mean()*100.0:,.2f}% ({results.std()*100.0:,.2f}%)")
```

```
Accuracy: 76.82% (5.58%)
```



Smart City

LAB CTIC UNI

Algoritmo Voting

1. Voting

- Voting es una de las formas más simples de combinar las predicciones de múltiples algoritmos.
- Funciona creando primero dos o más modelos independientes a partir de su conjunto de datos de entrenamiento.
- Un clasificador Voting se puede usar para ajustar sus modelos y promediar las predicciones de los submodelos cuando se le pide que haga predicciones para nuevos datos.
- Puede crear un modelo Voting para la clasificación utiliza la clase *VotingClassifier*.
- El siguiente código proporciona un ejemplo de combinación de predicciones de LoR, CART y SVM juntas.

1. Voting - código

```
# Voting for Classification
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier

kfold = KFold(n_splits=10, random_state=7)
# create the sub models
estimators = []
model1 = LogisticRegression(solver= 'lbfgs', max_iter=1000)
estimators.append(('logistic', model1))
model2 = DecisionTreeClassifier()
estimators.append(('cart', model2))
model3 = SVC(gamma='auto')
estimators.append(('svm', model3))
# create the ensemble model
ensemble = VotingClassifier(estimators)
results = cross_val_score(ensemble, X, Y, cv=kfold)
print(f"Accuracy: {results.mean()*100.0:,.2f}% ({results.std()*100.0:,.2f}%)")
```

```
Accuracy: 74.21% (6.97%)
```

Smart City

LAB CTIC UNI

Algoritmo Super Lerner

1. Introducción

- Para problemas de Regresión y Clasificación.
- ¿Cuál es el mejor? ¿Cómo obtenemos la mejor configuración para suconjunto de datos específico?
- Es posible que tenga muchas decenas o cientos de modelos diferentes de su problema.
  - ¿Por qué no usar todos esos modelos en lugar del mejor modelo del grupo? → Esta es la idea de Super Lerner.
- Variante de Stacked Generalization en la que un Metamodelo se ajusta en las predicciones parciales de cada modelo.
- Propuesto en 2007 y, en 2010, con un trabajo más detallado.

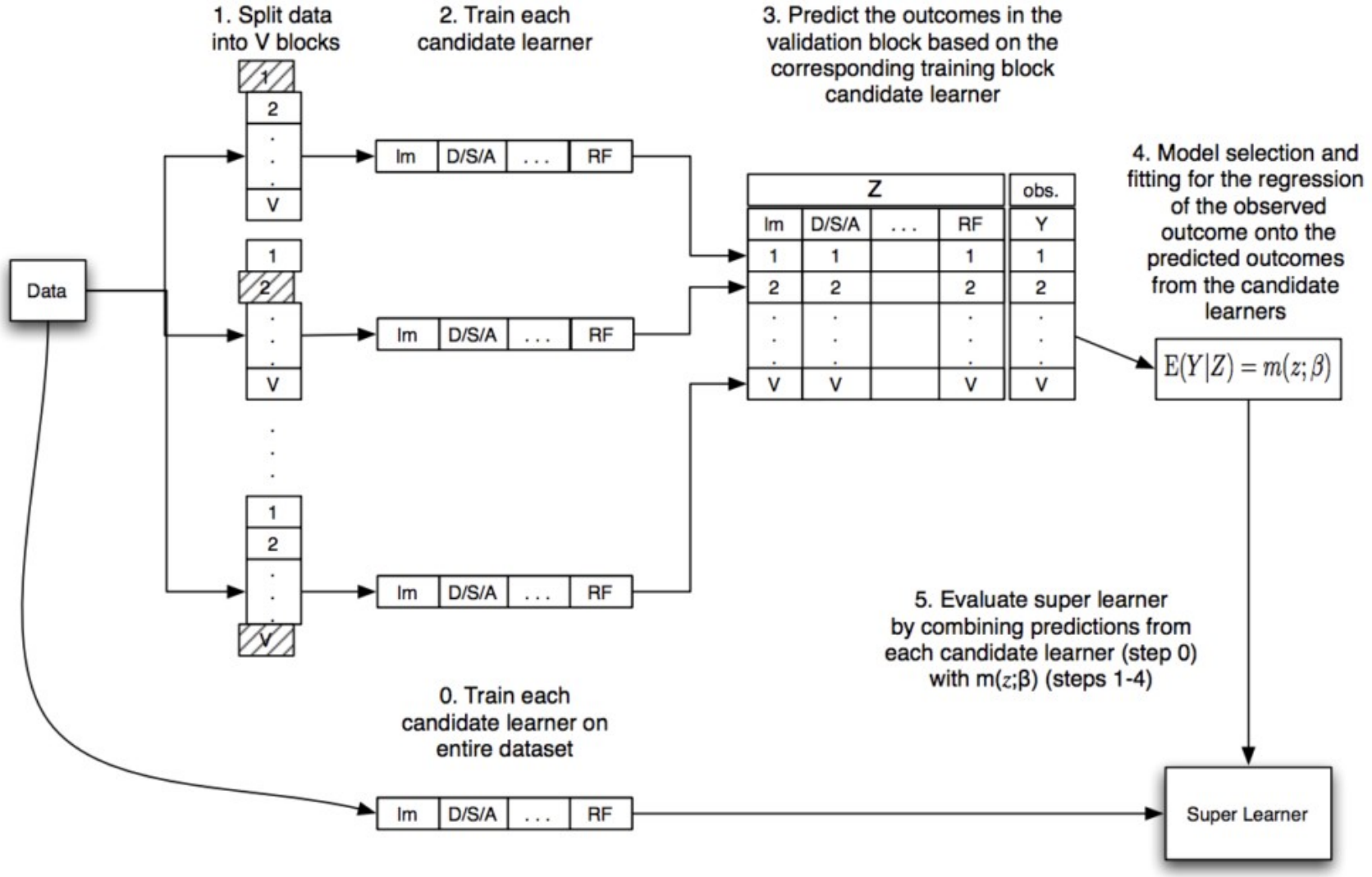
2. Procedimiento del modelo (I)

1) Seleccione una división de *k*-fold del conjunto de datos de entrenamiento.
2) Seleccione *M* modelos base o configuraciones de modelo.
3) Para cada modelo base:
   a) Evalúe usando la validación cruzada *k*-fold.
   b) Almacene todas las predicciones parciales de cada fold *(out-of-fold)*.
   c) Ajuste el modelo en el conjunto de datos de entrenamiento completo y almacene.
4) Ajuste un Metamodelo en las predicciones fuera del fold.
5) Evalúe el modelo en un conjunto de datos de reserva o use el modelo para hacer predicciones.

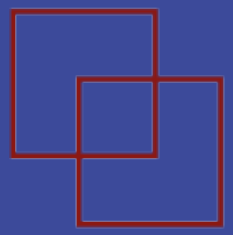
3. Esquema de trabajo (I)

1. Tome una muestra no etiquetada por las modelos durante el entrenamiento.
2. Para cada modelo base:
   a) Hace una predicción dada la muestra.
   b) Almacenar la predicción.
3. Concatene las predicciones del submodelo en un solo vector.
4. Proporciona el vector como entrada al metamodelo para hacer una predicción final.

3. Esquema de trabajo (II)



¡GRACIAS!



Smart City

LAB CTIC UNI

Dr. Manuel Castillo-Cara
Intelligent Ubiquitous Technologies – Smart Cities (IUT-SCi)
Web: www.smartcityperu.org