Día 5 - 19/11/21 (Viernes)

Clusterización y

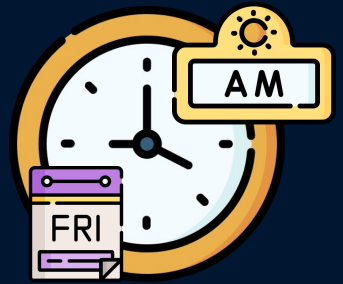
Métodos de ensamble

Lic. Ronaldo Armando Canizales Turcios

UNIVERSIDAD DE GRANADA

Universidad Centroamericana José Simeón Cañas

Agenda Día 5



Bloque A

- Aprendizaje no supervisado: clusterización.

- Agrupación no supervisada de perfiles económicos: k-medias.

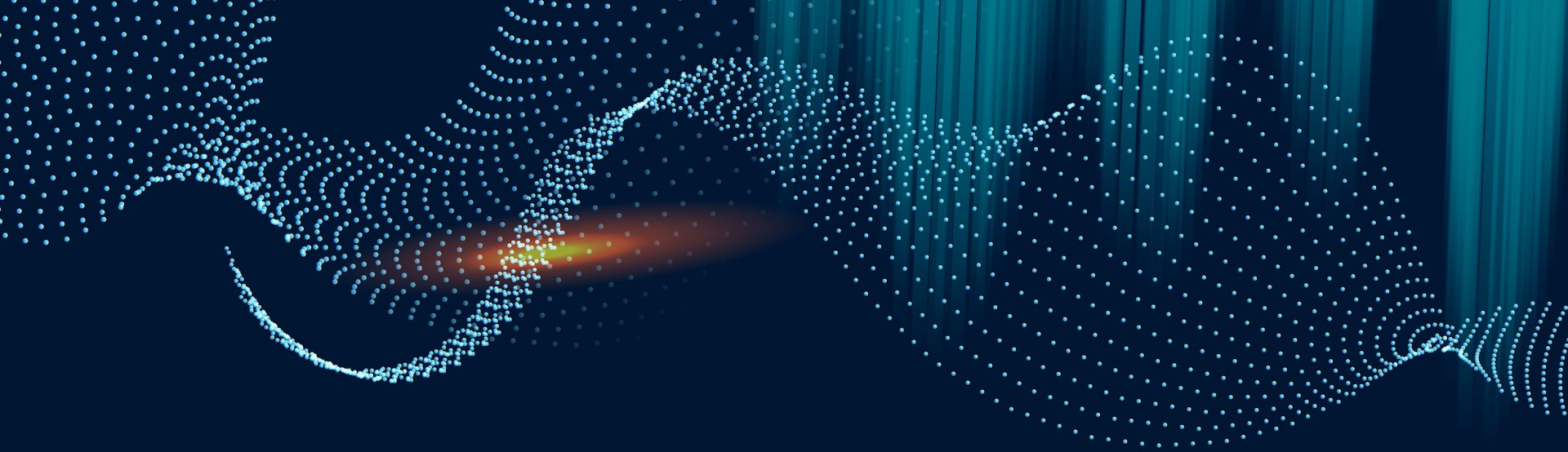
- Clusterización utilizando datos generados en tiempo real.

Bloque B

- Aplicando clusterización mediante k-medias. **[Práctica]**

- Métodos de ensamble: modelos y conceptos.

- Cierre del curso: concurso (rifa de libros) y siguientes pasos.



01 Clusterización

Intro. al aprendizaje no supervisado (continuación)

Clusterización

Pertenece al **aprendizaje no supervisado**, es una técnica que nos permite **descubrir estructuras ocultas** en los datos.

Ambos algoritmos (reducción de la dimensionalidad y clusterización) permiten **resumir** nuestros datos.

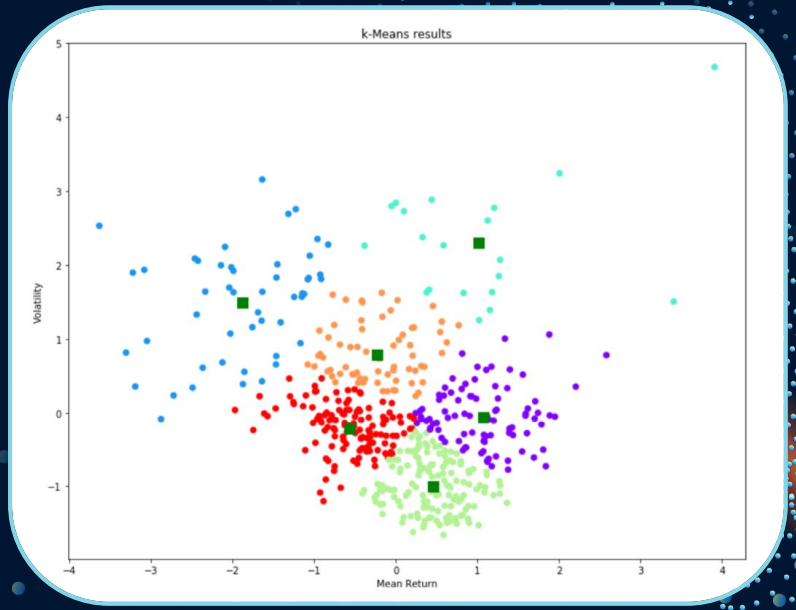
- PCA **comprime** nuestros datos mediante la representación de ellos en nuevas (y menor cantidad) de características, mientras que simultáneamente captura la mayor cantidad de información relevante.
- De forma similar, la clusterización, es una forma de reducir el volumen de datos y **encontrar patrones**. Lo logra mediante la categorización de la data, no mediante la creación de nuevas variables.

Clusterización

El objetivo de la clusterización es **encontrar una forma natural de agrupar nuestros datos**, de tal manera que los datos que pertenecen a un subgrupo (llamado clúster) son **más similares entre sí** que con los datos de los demás subgrupos.

Es un algoritmo no supervisado, ya que **no se sabe de antemano** cuántos subgrupos (clústers) se formarán, ni cuántos datos pertenecerán a cada uno de ellos.

También permite la **categorización automática** de nuevos datos mediante la regla que ha aprendido.



Algoritmo de k-medias

El objetivo es **encontrar k centroides** y asignar uno de ellos a cada registro, de tal forma que se **minimice** la varianza intra-clúster (llamada **inercia**).

Usualmente se utiliza la distancia Euclidiana (distancia entre dos puntos), pero es posible ocupar otras métricas.

K-medias se encarga de encontrar un mínimo para una “k” dada, se procede de la siguiente manera:

1. **Se elige una cantidad de clústers.**
2. **Algunos puntos se eligen aleatoriamente como los centroides de los clústers.**
3. **Cada dato se asigna al clúster de cuyo centro esté más cercano.**
4. **El centroide del clúster se actualiza con la media de los puntos asignados a él.**
5. **Los pasos 3 al 4 se repiten hasta que todos los centroides se mantengan sin cambios.**

Hiperparámetros de k-medias

Cantidad de clústers

- Clústers (y centroides) a generar.

Cantidad máxima de iteraciones

- Para **una ejecución** del algoritmo.

```
from sklearn.cluster import KMeans
#Fit with k-means
k_means = KMeans(n_clusters=nclust)
k_means.fit(X)
```

“Número inicial”

- Cantidad de veces que el algoritmo se ejecutará, utilizando **diferentes semillas** para la generación de los **centroides**. El resultado final será el que mejor rendimiento haya tenido (en términos de la **inercia**).
- En la librería sklearn, el algoritmo se ejecuta **al menos 10 veces, con diferentes valores iniciales**.

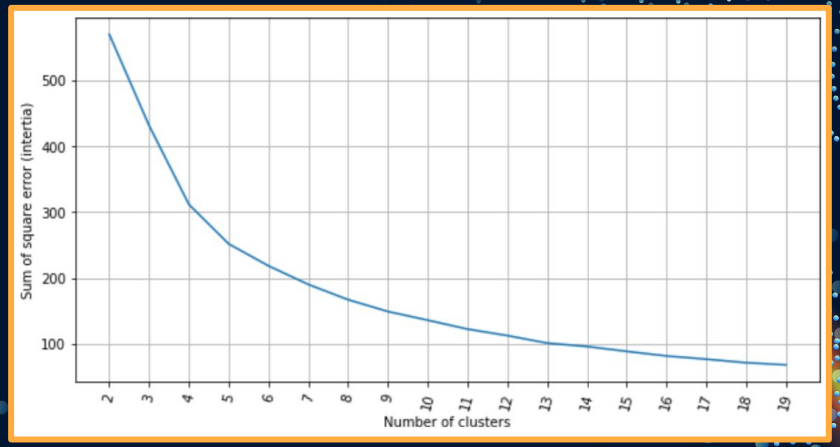
Características de k-medias

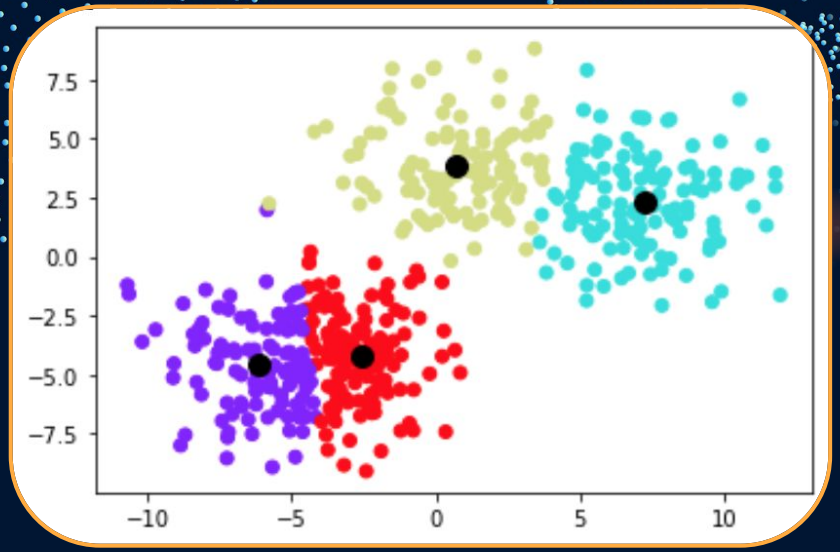
Ventajas

- Su principal ventaja radica en su simplicidad, su amplio rango de aplicabilidad, rápida velocidad de convergencia y su escalabilidad para conjuntos grandes de datos.

Desventajas

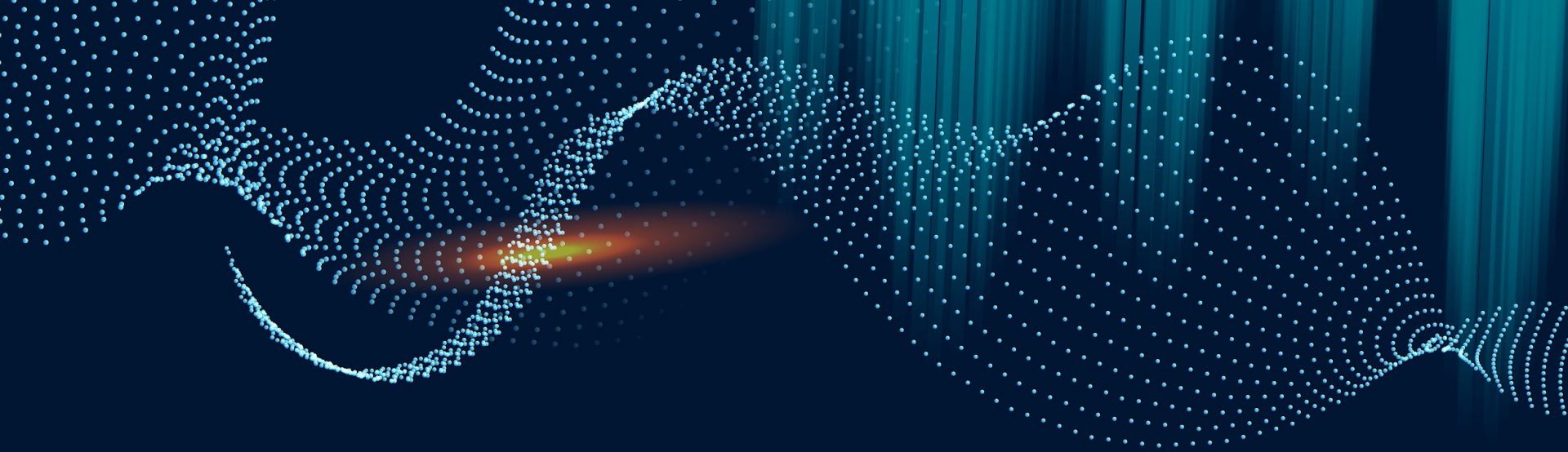
- Falta de garantía para encontrar el mínimo global (sin tener que ejecutar el algoritmo una considerable cantidad de veces).
- Puede ser sensible a valores atípicos.





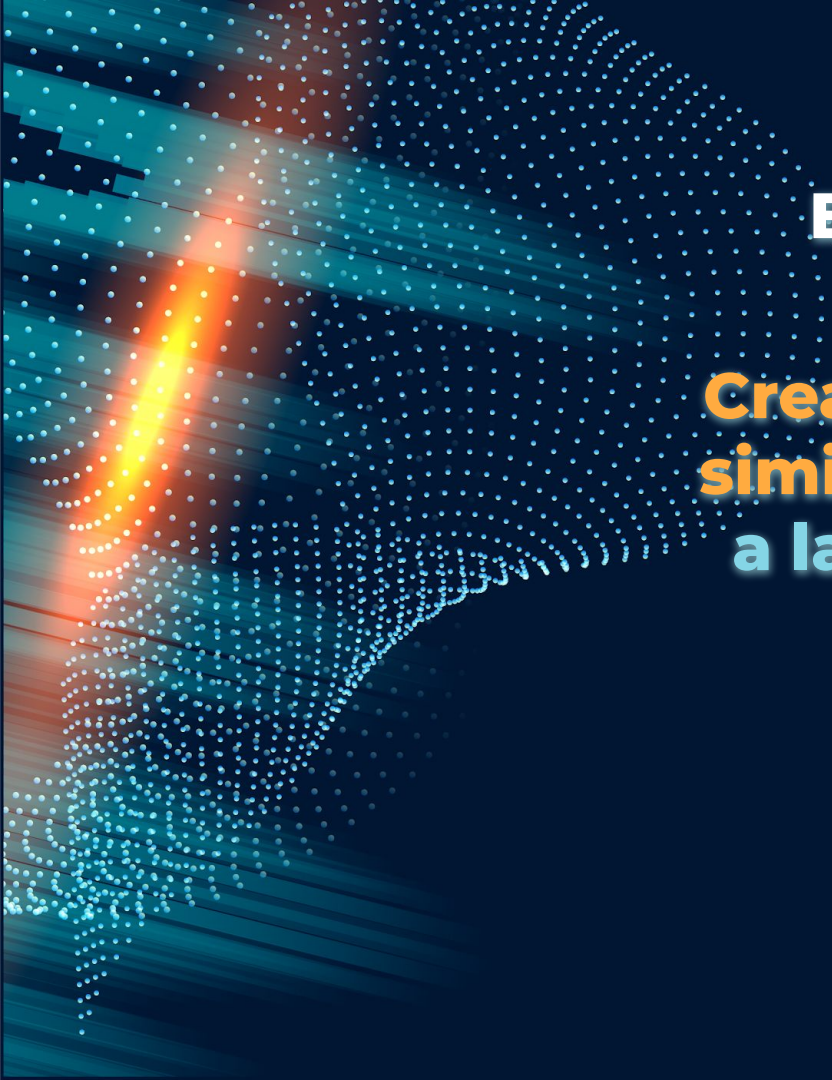
02 Clusterización

Utilizando datos generados en tiempo real



03 Agrupación no supervisada

Aplicación de k-medias: perfiles económicos



El objetivo del ejercicio es clusterizar individuos

Crear subconjuntos de personas similares entre ellas y diferentes a las personas de otros grupos

Con cada clúster, se podrán crear de forma personalizadas:

- Estrategias de negocios.
- Campañas de publicidad.
- Descuentos, planes, etc.

Existen 12 variables. Ojo: no hay una “variable objetivo”, pues se trata de un algoritmo no supervisado.

	ID	AGE	EDUC	MARRIED	KIDS	LIFECL	OCCAT	RISK	HHOUSES	WSAVED	SPENDMOR	NWCAT	INCCL
0	1	3	2	1	0	2	1	3	1	1	5	3	4
1	2	4	4	1	2	5	2	3	0	2	5	5	5
2	3	3	1	1	2	3	2	2	1	2	4	4	4
3	4	3	1	1	2	3	2	2	1	2	4	3	4
4	5	4	3	1	1	5	1	2	1	3	3	5	5

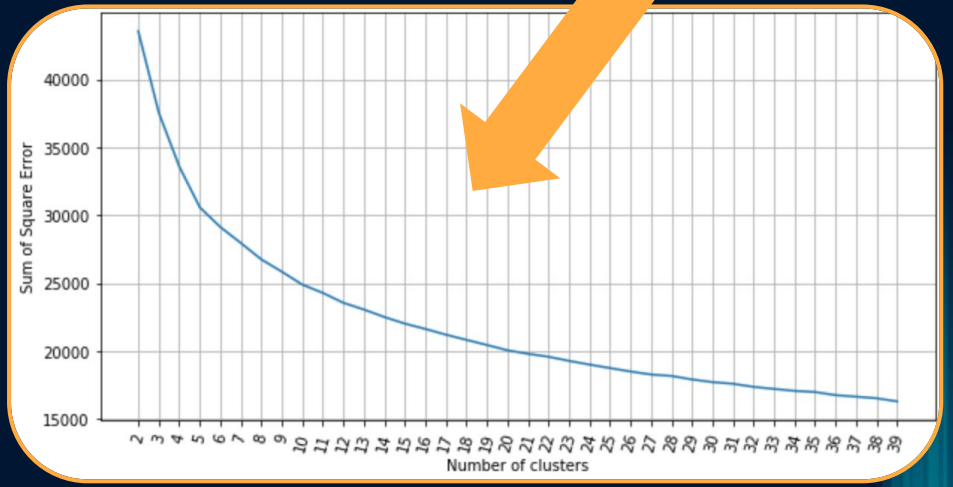
```
dataset = pd.read_excel('ProcessedData.xlsx')
```

```
X=dataset.copy("deep")
X=X.drop(['ID'], axis=1)
X.head()
```

	AGE	EDUC	MARRIED	KIDS	LIFECL	OCCAT	RISK	HHOUSES	WSAVED	SPENDMOR	NWCAT	INCCL
0	3	2	1	0	2	1	3	1	1	5	3	4
1	4	4	1	2	5	2	3	0	2	5	5	5
2	3	1	1	2	3	2	2	1	2	4	4	4
3	3	1	1	2	3	2	2	1	2	4	3	4
4	4	3	1	1	5	1	2	1	3	3	5	5

La decisión dependerá de los datos... ¡o del negocio!

```
distorsions = []
max_loop=40
for k in range(2, max_loop):
    k_means = KMeans(n_clusters=k)
    k_means.fit(X)
    distorsions.append(k_means.inertia_)
fig = plt.figure(figsize=(10, 5))
plt.plot(range(2, max_loop), distorsions)
plt.xlabel("Number of clusters")
plt.ylabel("Sum of Square Error")
plt.grid(True)
```

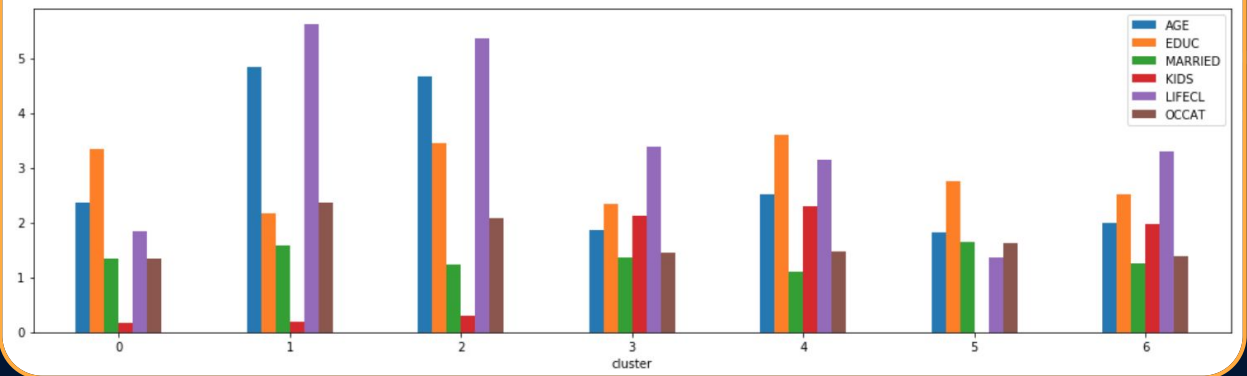


Visualizando cada centroide (clúster)

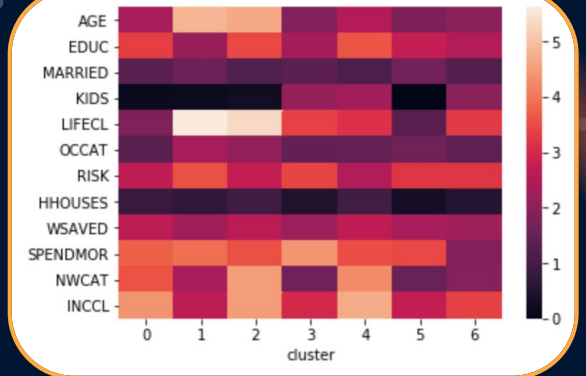
```
cluster_output= pd.concat([pd.DataFrame(X),
                           pd.DataFrame(k_means.labels_, columns = ['cluster'])],axis = 1)
output=cluster_output.groupby('cluster').mean()
output
```

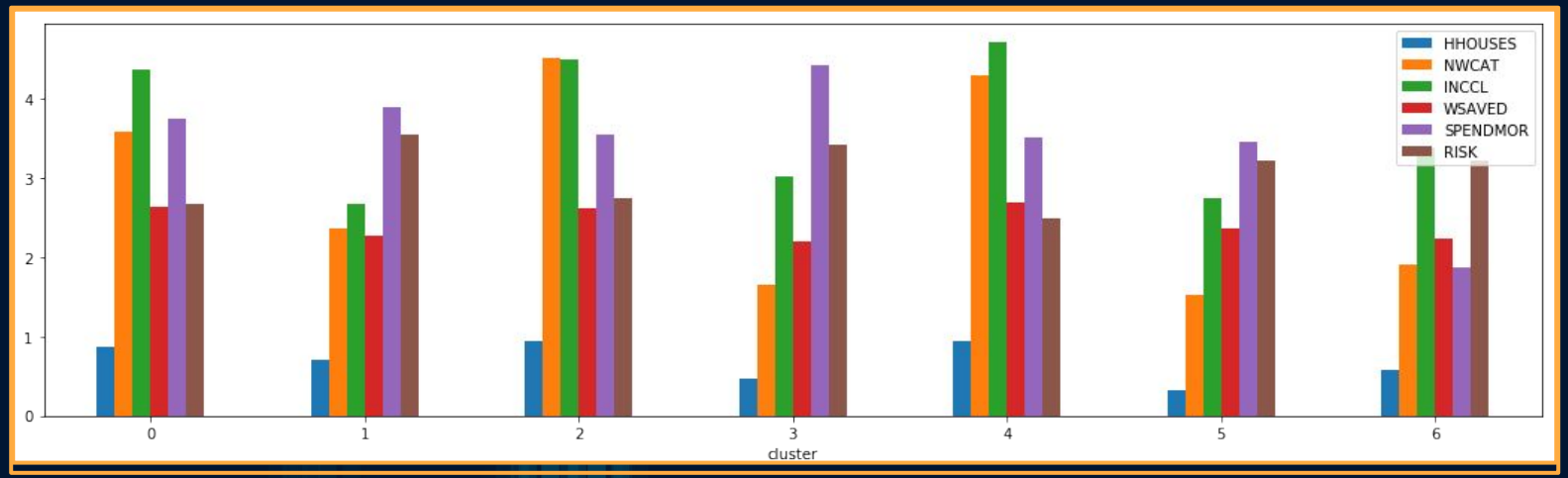
cluster	AGE	EDUC	MARRIED	KIDS	LIFECL	OCCAT	RISK	HHOUSES	WSAVED	SPENDMOR	NWCAT	INCCL
0	2.355	3.349	1.336	0.169	1.833	1.334	2.683	0.865	2.647	3.745	3.589	4.375
1	4.839	2.158	1.579	0.189	5.621	2.371	3.552	0.709	2.270	3.905	2.362	2.674
2	4.666	3.458	1.226	0.299	5.351	2.077	2.745	0.942	2.624	3.551	4.515	4.506
3	1.874	2.350	1.354	2.124	3.373	1.452	3.416	0.475	2.205	4.426	1.657	3.024
4	2.520	3.610	1.102	2.307	3.147	1.480	2.502	0.952	2.686	3.519	4.294	4.717
5	1.811	2.746	1.649	0.002	1.354	1.627	3.226	0.323	2.367	3.462	1.527	2.744
6	1.983	2.525	1.252	1.971	3.288	1.390	3.225	0.571	2.245	1.874	1.913	3.380

```
output[['AGE','EDUC','MARRIED','KIDS','LIFECL','OCCAT']].plot.bar(rot=0, figsize=(18,5));
```



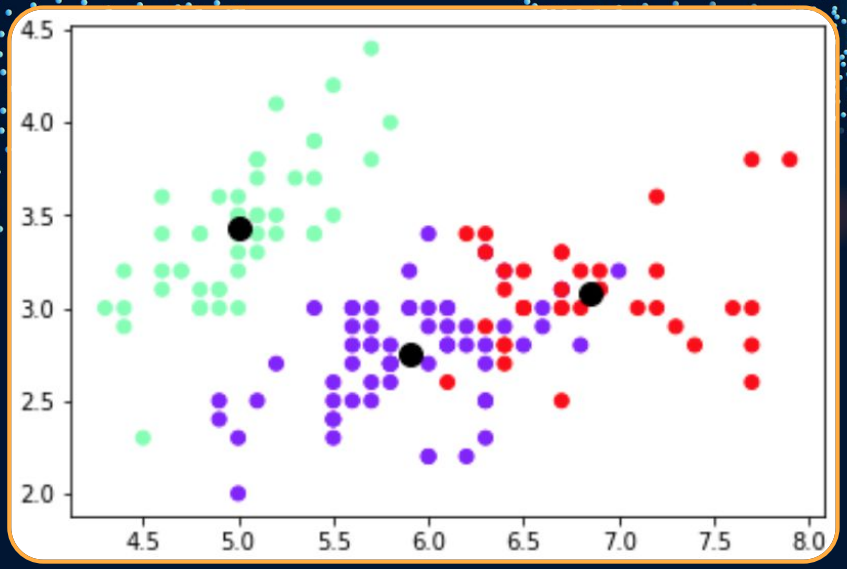

```
sns.heatmap(output.T)
```





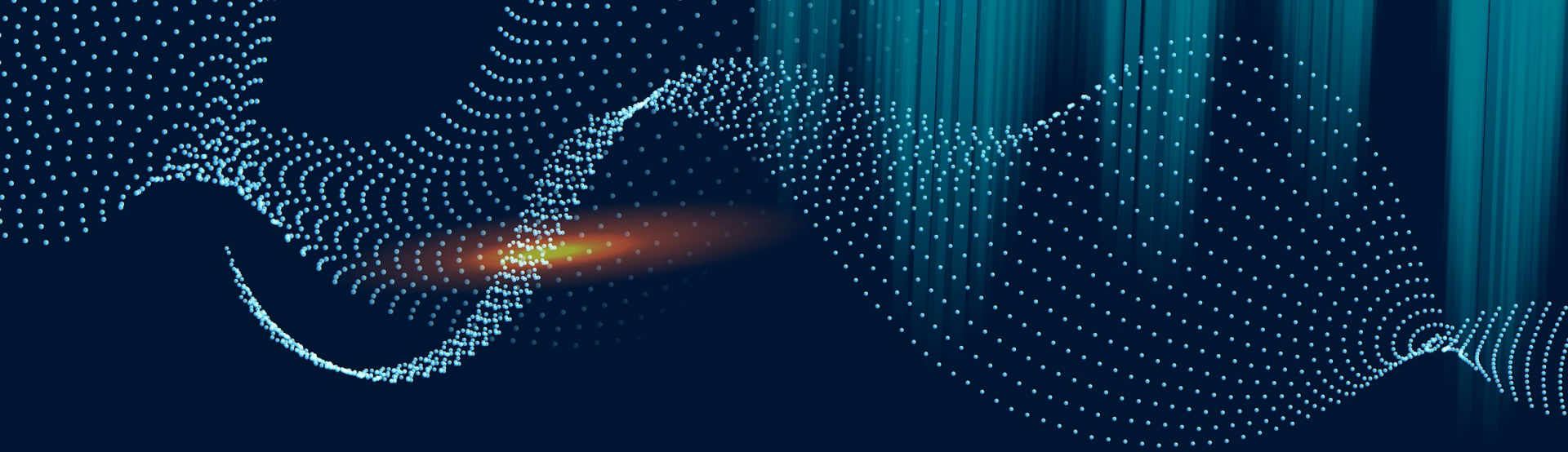
¿Con cuál te identificas?

Cluster	Features	Risk Capacity
Cluster 0	Low Age, High Networth and Income, Less risky life category, willingness to spend more	High
Cluster 1	High Age, low net worth and Income, highly risky life category, Willing ness to take risk, low education	High
Cluster 2	High Age, high net worth and Income, highly risky life category, Willing ness to to take risk, own house	Medium
Cluster 3	Low age, very low income and net worth, high willingness to take risk, many kids	Low
Cluster 4	Medium age, very high income and net worth, high willingness to take risk, many kids, own house	High
Cluster 5	Low age, very low income and net worth, high willingness to take risk, no kids	Medium
Cluster 6	Low age, medium income and net worth, high willingness to take risk, many kids, own house	Low



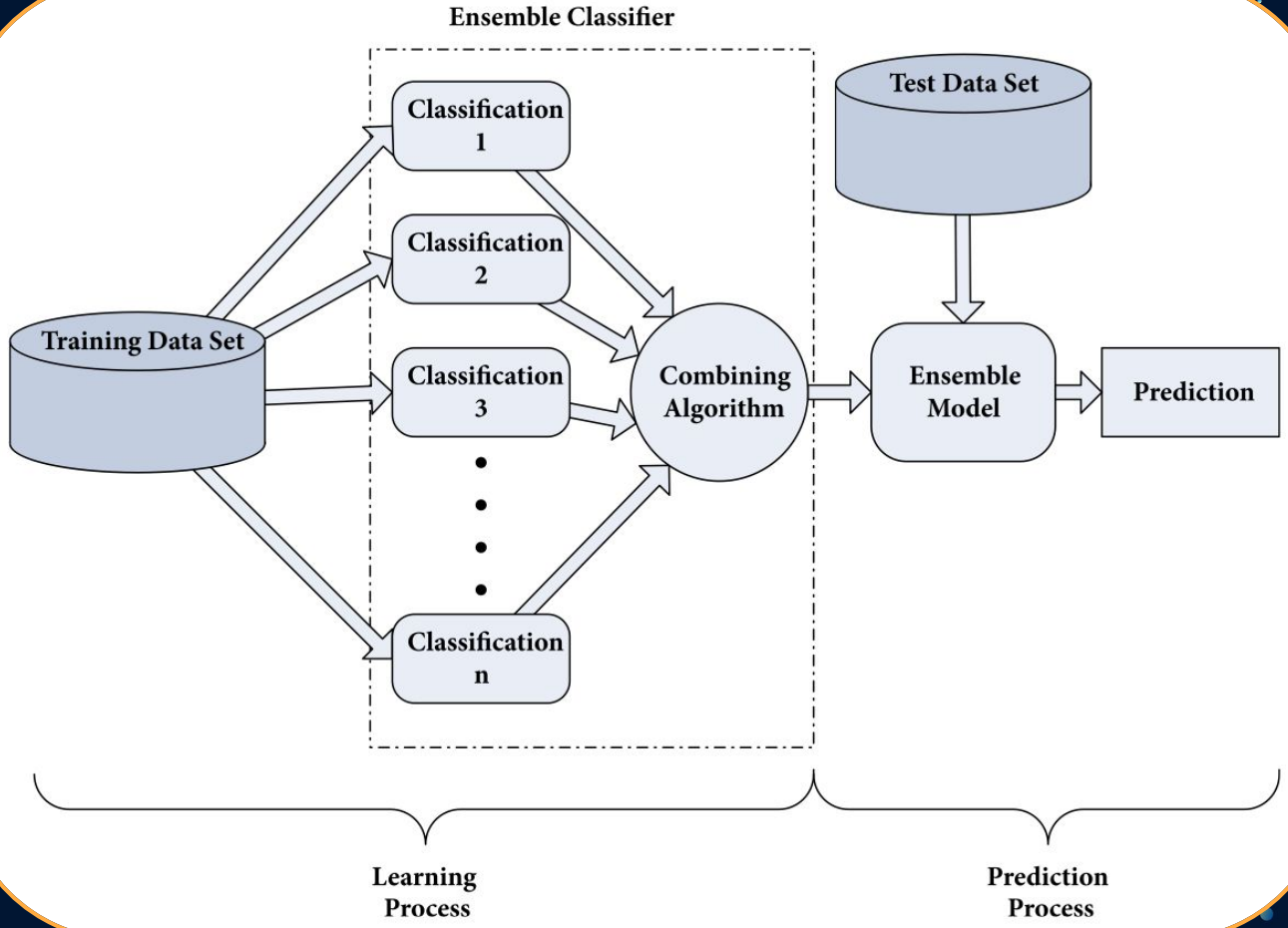
04 Aplicando k-medias

¡Manos a la obra: clusterización! **[Práctica]**



05 Métodos de ensamble

Modelos y conceptos

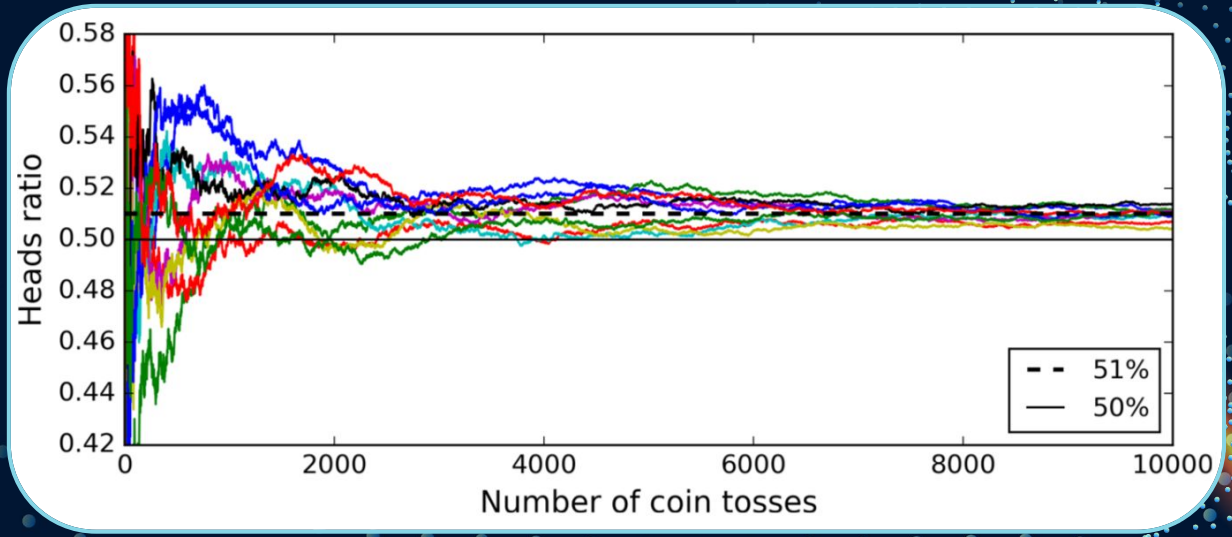


Idea intuitiva

Supongamos que tenemos una **moneda ligeramente sesgada** que tiene un 51% de posibilidades de salir cara y un 49% de probabilidad de salir cruz. Si la lanza 1.000 veces, generalmente obtendrá más o menos 510 caras y 490 cruz, y por tanto una mayoría de caras.

Es posible calcular que la probabilidad de obtener una **mayoría de caras** después de 1.000 lanzamientos es cercana al 75%. Cuanto más se lance la moneda, mayor será la probabilidad (por ejemplo con 10.000 lanzamientos, la probabilidad supera el 97%).

Esto se debe a la **ley de los grandes números**: a medida que se lanza la moneda, la proporción de caras se acerca cada vez más a la probabilidad de salir cara (51%).



Métodos de ensamble

Del mismo modo, suponga que se construye un conjunto que contiene **1.000 clasificadores que, individualmente, sólo aciertan el 51\% de las veces** (apenas mejor que la adivinación aleatoria). Si predice la clase mayoritariamente votada, puede esperar una **precisión de hasta el 75%**.

Sin embargo, esto sólo es cierto si todos los clasificadores son **perfectamente independientes**, cometiendo **errores no correlacionados**, lo que claramente no es el caso ya que están entrenados con los mismos datos. Es probable que cometan los mismos tipos de errores, por lo que habrá muchos **votos mayoritarios para la clase equivocada**, reduciendo la precisión del conjunto.

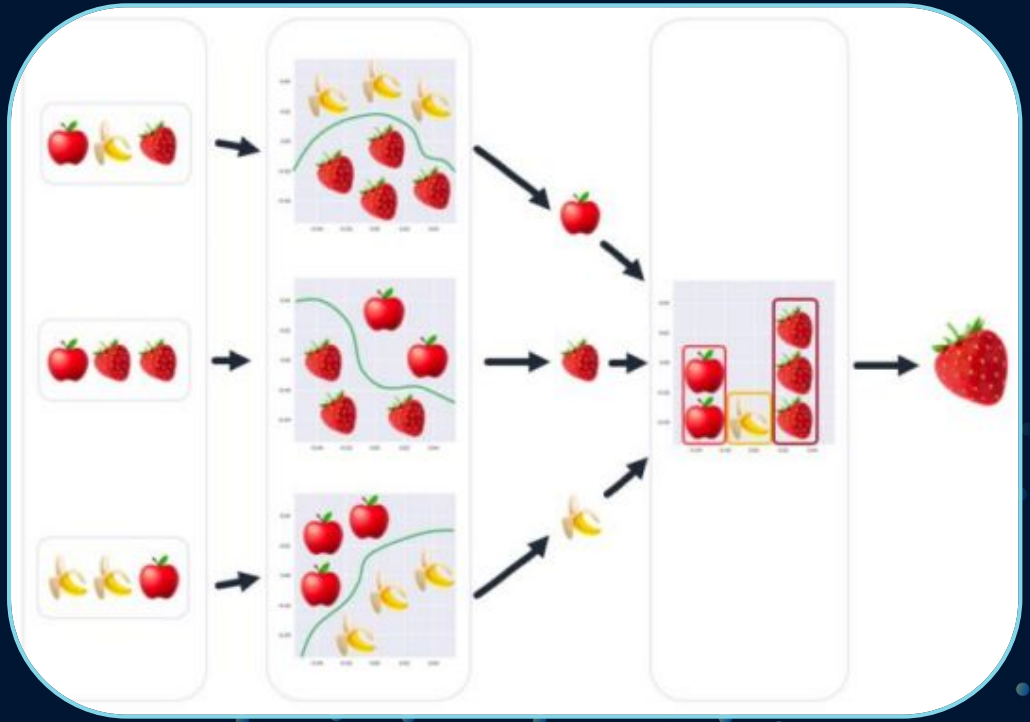
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC()

voting_clf = VotingClassifier(
        estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
        voting='hard'
    )
voting_clf.fit(X_train, y_train)
```

```
>>> from sklearn.metrics import accuracy_score
>>> for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
>>>     clf.fit(X_train, y_train)
>>>     y_pred = clf.predict(X_test)
>>>     print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
```

¡Muy bien!

```
LogisticRegression 0.864
RandomForestClassifier 0.872
SVC 0.888
VotingClassifier 0.896
```

Tipos de métodos de ensamble más utilizados



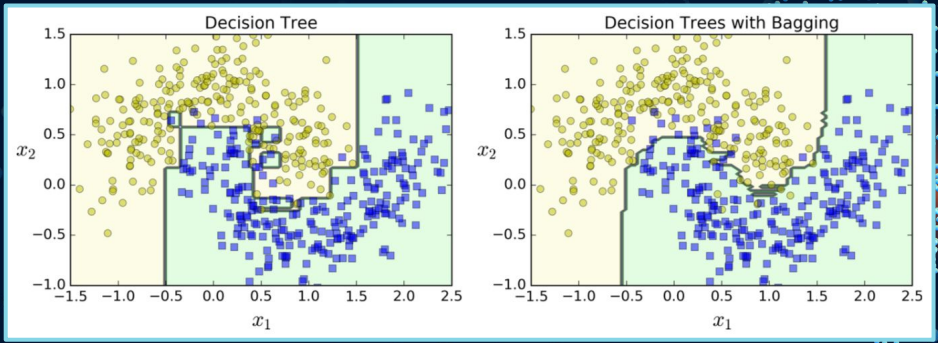
Utiliza **el mismo algoritmo** de entrenamiento para cada predictor, pero **entrenarlos en diferentes subconjuntos** aleatorios del conjunto de entrenamiento.
Cuando el muestreo se realiza con reemplazo, este método se denomina bagging (**abreviatura de bootstrap aggregating**).

Bagging

Una vez entrenados todos los predictores, el conjunto puede hacer una predicción para una nueva instancia **simplemente agregando las predicciones de todos los predictores**.

La función de agregación suele ser el modo estadístico (es decir, la predicción más frecuente, **como un clasificador de voto duro**) para la clasificación, o la media para la regresión. Cada predictor individual tiene un sesgo mayor que si se entrenara con el conjunto de entrenamiento original, pero la agregación reduce tanto el sesgo como la varianza. **En general, el resultado neto tiene un sesgo similar pero una varianza menor que un predictor individual entrenado en el conjunto de entrenamiento original.**

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
        DecisionTreeClassifier(), n_estimators=500,
        max_samples=100, bootstrap=True, n_jobs=-1
    )
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

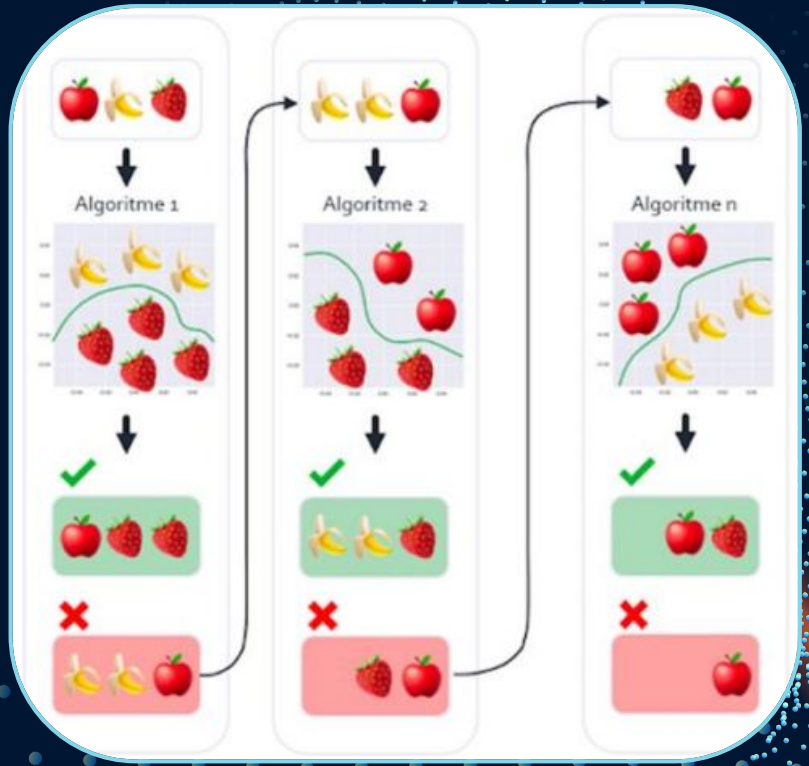


Boosting

El refuerzo (originalmente llamado refuerzo de hipótesis) se refiere a cualquier método de ensamble que pueda combinar varios aprendices débiles en un aprendiz fuerte.

La idea general de la mayoría de los métodos de refuerzo es entrenar predictores de forma secuencial, cada uno de los cuales intenta corregir a su predecesor.

Existen muchos métodos de boosting, pero los más populares son AdaBoost (abreviatura de Adaptive Boosting) y Gradient Boosting.

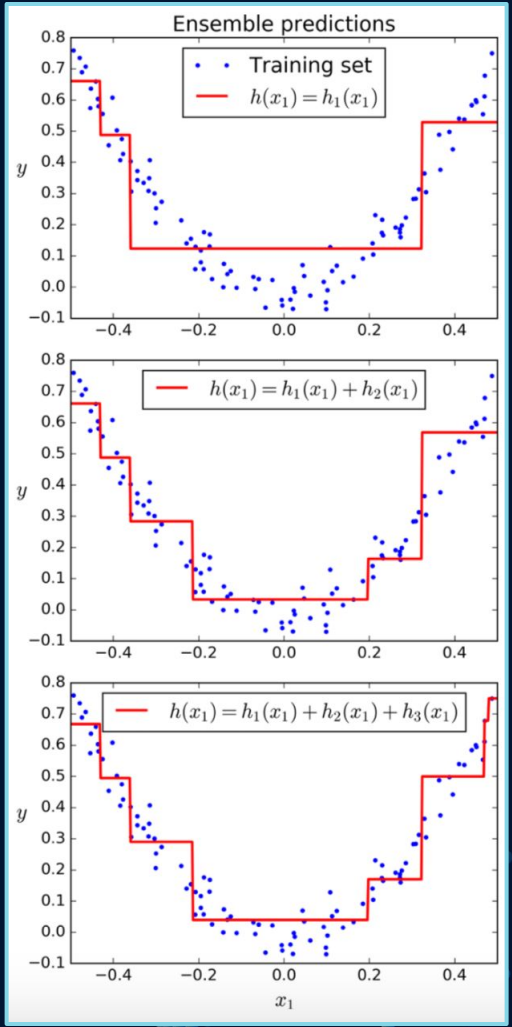


Una forma de que un nuevo predictor corrija a su predecesor es **prestar un poco más de atención a las instancias de entrenamiento que el predecesor infravaloró.** Esto hace que los nuevos predictores se centren cada vez más en los casos difíciles. Esta es la técnica utilizada por Ada-Boost.

```
from sklearn.ensemble import AdaBoostClassifier

ada_clf = AdaBoostClassifier(
        DecisionTreeClassifier(max_depth=1), n_estimators=200,
        algorithm="SAMME.R", learning_rate=0.5
    )
ada_clf.fit(X_train, y_train)
```

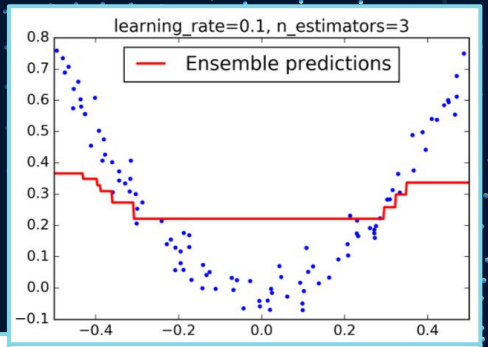
Por ejemplo, para construir un clasificador AdaBoost, se entrena un primer clasificador base (como un árbol de decisión) y se utiliza para hacer predicciones sobre el conjunto de entrenamiento. A continuación, **se aumenta el peso relativo de las instancias de entrenamiento mal clasificadas**.

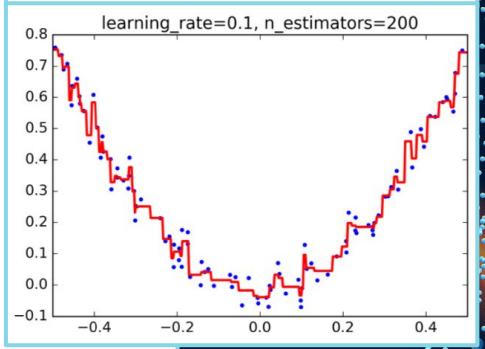
Se entrena un segundo clasificador utilizando los pesos actualizados y, de nuevo, realiza predicciones sobre el conjunto de entrenamiento, se actualizan los pesos, y así sucesivamente.



Otro algoritmo de Boosting muy popular es el **Gradient Boosting**. Al igual que AdaBoost, Gradient Boosting funciona añadiendo secuencialmente predictores a un conjunto, cada uno de los cuales corrige a su predecesor.

Sin embargo, en lugar de ajustar los pesos de las instancias en cada iteración, como hace AdaBoost, este método intenta ajustar el nuevo predictor a los errores residuales cometidos por el predictor anterior.



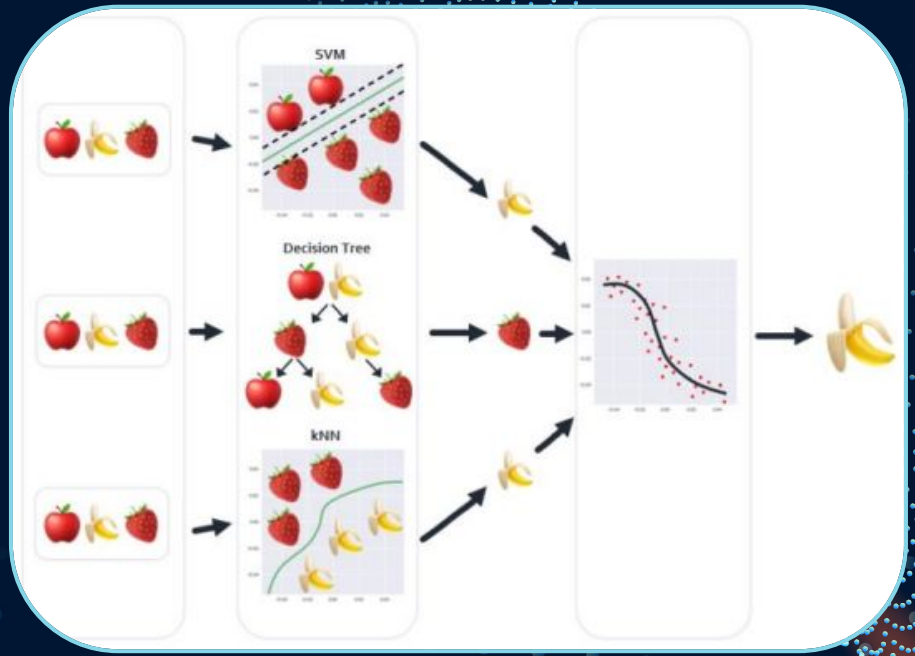



```python
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X_train, X_val, y_train, y_val = train_test_split(X, y)

gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=120)
gbrt.fit(X_train, y_train)

errors = [mean_squared_error(y_val, y_pred)
          for y_pred in gbrt.staged_predict(X_val)]
bst_n_estimators = np.argmin(errors)

gbrt_best = GradientBoostingRegressor(max_depth=2,n_estimators=bst_n_estimators)
gbrt_best.fit(X_train, y_train)
```

Stacking

Stacking (abreviatura de stacked generalization). Se basa en una idea sencilla: en lugar de utilizar funciones triviales (como el voto duro) para agregar las predicciones de todos los predictores en un conjunto, **¿por qué no entrenamos un modelo para realizar esta agregación?**

La figura muestra un conjunto de este tipo realizando una tarea de regresión sobre una nueva instancia.



06 Cierre del curso

Rifa de libros: mini copa del conocimiento

¡Muchas gracias!

Ha sido un honor y un gusto visitar Granada

Espero volver a España algún día, Dios mediante

UNIVERSIDAD DE GRANADA

Universidad Centroamericana José Simeón Cañas