



ARISA Learning Material

Educational Profile and EQF level: DATA SCIENTIST – EQF 6

PLO: 1, 2, 3, 4, 5

Learning Unit (LU): MACHINE LEARNING: SUPERVISED

Topic: 7. MODEL SELECTION



www.aiskills.eu

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

Copyright © 2024 by the Artificial Intelligence Skills Alliance

All learning materials (including Intellectual Property Rights) generated in the framework of the ARISA project are made freely available to the public under an open license [Creative Commons Attribution–NonCommercial](https://creativecommons.org/licenses/by-nc/4.0/) (CC BY-NC 4.0).

ARISA Learning Material 2024

This material is a draft version and is subject to change after review coordinated by the European Education and Culture Executive Agency (EACEA).

Authors: Universidad Internacional de La Rioja (UNIR)

Disclaimer: This learning material has been developed under the Erasmus+ project ARISA (Artificial Intelligence Skills Alliance) which aims to skill, upskill, and reskill individuals into high-demand software roles across the EU.



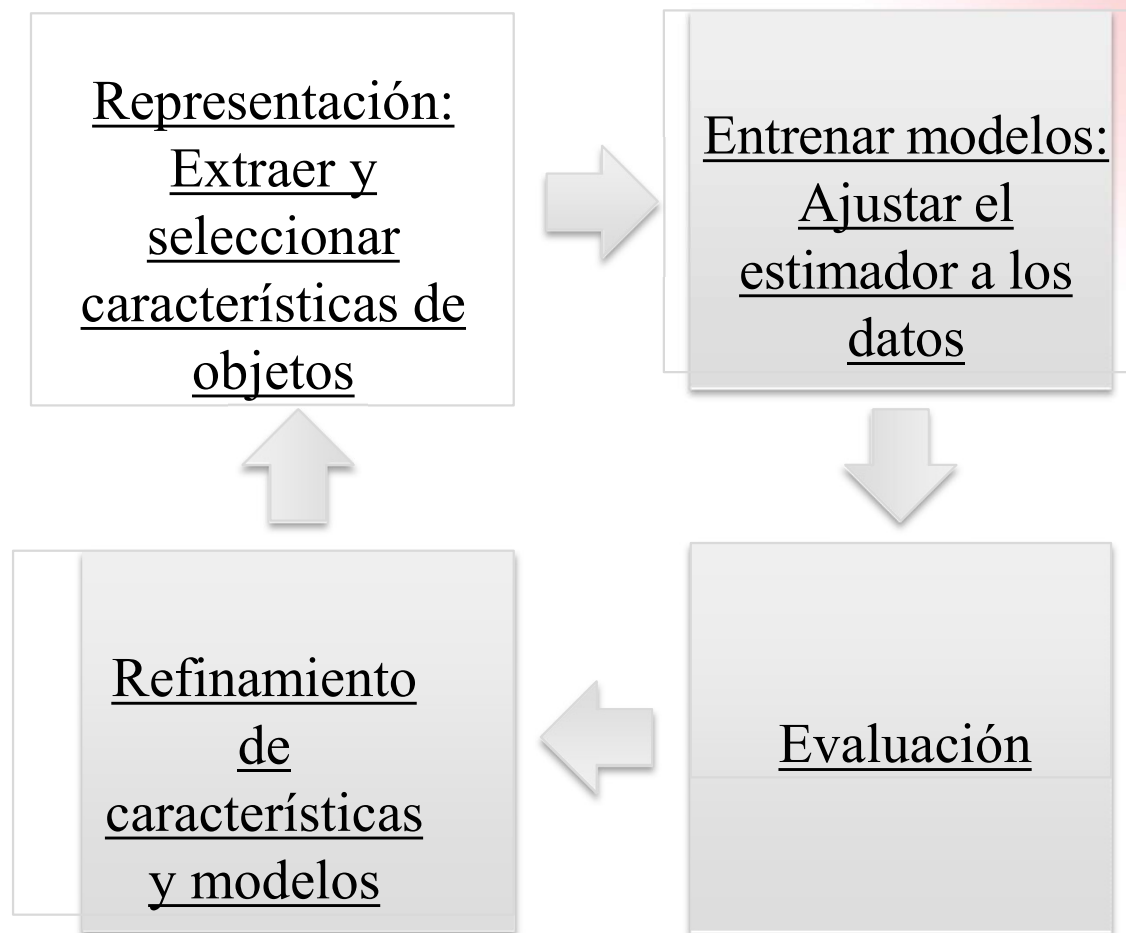
This project has been funded with support from the European Commission. The material reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- About ARISA

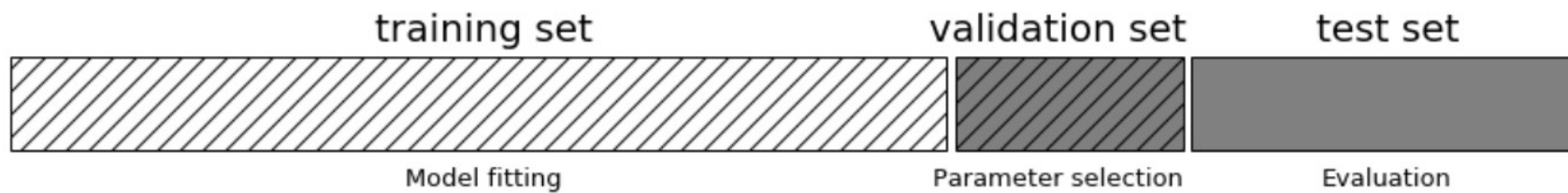
- The Artificial Intelligence Skills Alliance (ARISA) is a four-year transnational project funded under the EU's Erasmus+ programme. It delivers a strategic approach to sectoral cooperation on the development of Artificial Intelligence (AI) skills in Europe.
- ARISA fast-tracks the upskilling and reskilling of employees, job seekers, business leaders, and policymakers into AI-related professions to open Europe to new business opportunities.
- ARISA regroups leading ICT representative bodies, education and training providers, qualification regulatory bodies, and a broad selection of stakeholders and social partners across the industry.

[ARISA Partners & Associated Partners](#) | [LinkedIn](#) | [Twitter](#)

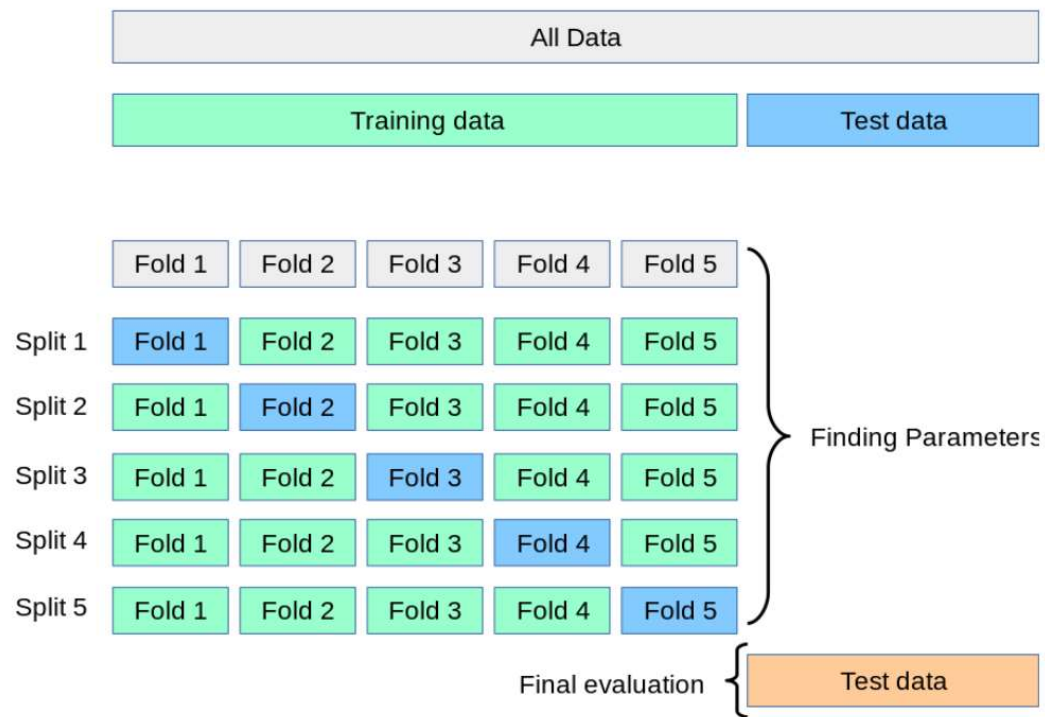
OPTIMIZACIÓN DE PARÁMETROS



Threefold split



Cross-validation + test set



```
X_trainval, X_test, y_trainval, y_test = train_test_split(X, y)
X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval)

val_scores = []
neighbors = np.arange(1, 15, 2)
for i in neighbors:
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    val_scores.append(knn.score(X_val, y_val))
print("best validation score: {:.3f}".format(np.max(val_scores)))
best_n_neighbors = neighbors[np.argmax(val_scores)]
print("best n_neighbors:", best_n_neighbors)

knn = KNeighborsClassifier(n_neighbors=best_n_neighbors)
knn.fit(X_trainval, y_trainval)
print("test-set score: {:.3f}".format(knn.score(X_test, y_test)))
```

```
best validation score: 0.991
best n_neighbors: 11
test-set score: 0.951
```

Grid-Search con validacion cruzada

```
from sklearn.model_selection import cross_val_score

X_train, X_test, y_train, y_test = train_test_split(X, y)
cross_val_scores = []

for i in neighbors:
    knn = KNeighborsClassifier(n_neighbors=i)
    scores = cross_val_score(knn, X_train, y_train, cv=10)
    cross_val_scores.append(np.mean(scores))

print("best cross-validation score: {:.3f}".format(np.max(cross_val_scores)))
best_n_neighbors = neighbors[np.argmax(cross_val_scores)]
print("best n_neighbors:", best_n_neighbors)

knn = KNeighborsClassifier(n_neighbors=best_n_neighbors)
knn.fit(X_train, y_train)
print("test-set score: {:.3f}".format(knn.score(X_test, y_test)))
```

```
best cross-validation score: 0.967
best n_neighbors: 9
test-set score: 0.965
```


GridSearchCV

```
from sklearn.model_selection import GridSearchCV

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y)

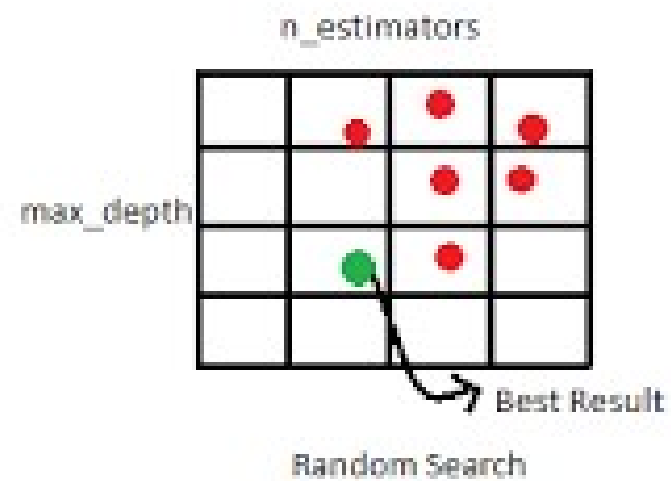
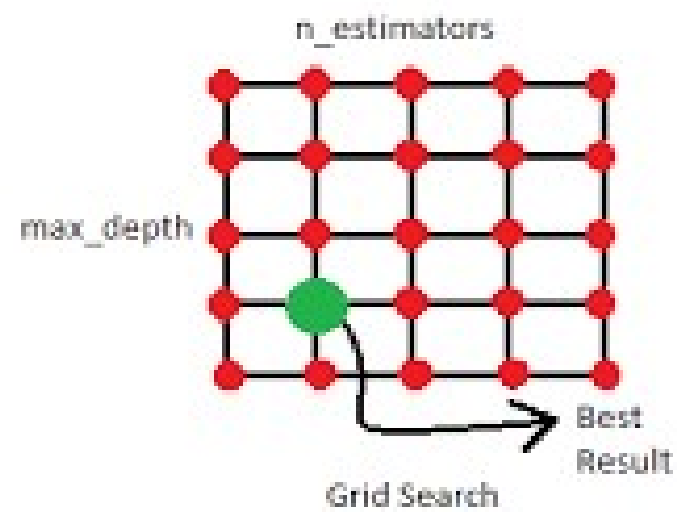
param_grid = {'n_neighbors': np.arange(1, 15, 2)}

grid = GridSearchCV(KNeighborsClassifier(), param_grid=param_grid, cv=10)
grid.fit(X_train, y_train)

print("best mean cross-validation score: {:.3f}".format(grid.best_score_))
print("best parameters:", grid.best_params_)

print("test-set score: {:.3f}".format(grid.score(X_test, y_test)))
```

```
best mean cross-validation score: 0.967
best parameters: {'n_neighbors': 9}
test-set score: 0.993
```



Parámetros de búsqueda de cuadrícula (Grid Search Parameters)

```
sklearn.model_selection.GridSearchCV(estimator, param_grid, scoring=None,  
    n_jobs=None, iid='deprecated', refit=True, cv=None, verbose=0,  
    pre_dispatch='2*n_jobs', error_score=nan, return_train_score=False)
```

1. **estimator**: Pass the model instance for which you want to check the hyperparameters.
2. **params_grid**: the dictionary object that holds the hyperparameters you want to try
3. **scoring**: evaluation metric that you want to use, you can simply pass a valid string/ object of evaluation metric
4. **cv**: number of cross-validation you have to try for each selected set of hyperparameters
5. **verbose**: you can set it to 1 to get the detailed print out while you fit the data to GridSearchCV
6. **n_jobs**: number of processes you wish to run in parallel for this task if it -1 it will use all available processors.

GridSearchCV Results

```
import pandas as pd
results = pd.DataFrame(grid.cv_results_)
results.columns
```

```
Index(['mean_fit_time', 'mean_score_time', 'mean_test_score',
       'mean_train_score', 'param_n_neighbors', 'params', 'rank_test_score',
       'split0_test_score', 'split0_train_score', 'split1_test_score',
       'split1_train_score', 'split2_test_score', 'split2_train_score',
       'split3_test_score', 'split3_train_score', 'split4_test_score',
       'split4_train_score', 'split5_test_score', 'split5_train_score',
       'split6_test_score', 'split6_train_score', 'split7_test_score',
       'split7_train_score', 'split8_test_score', 'split8_train_score',
       'split9_test_score', 'split9_train_score', 'std_fit_time',
       'std_score_time', 'std_test_score', 'std_train_score'],
      dtype='object')
```

```
results.params
```

20 / 21

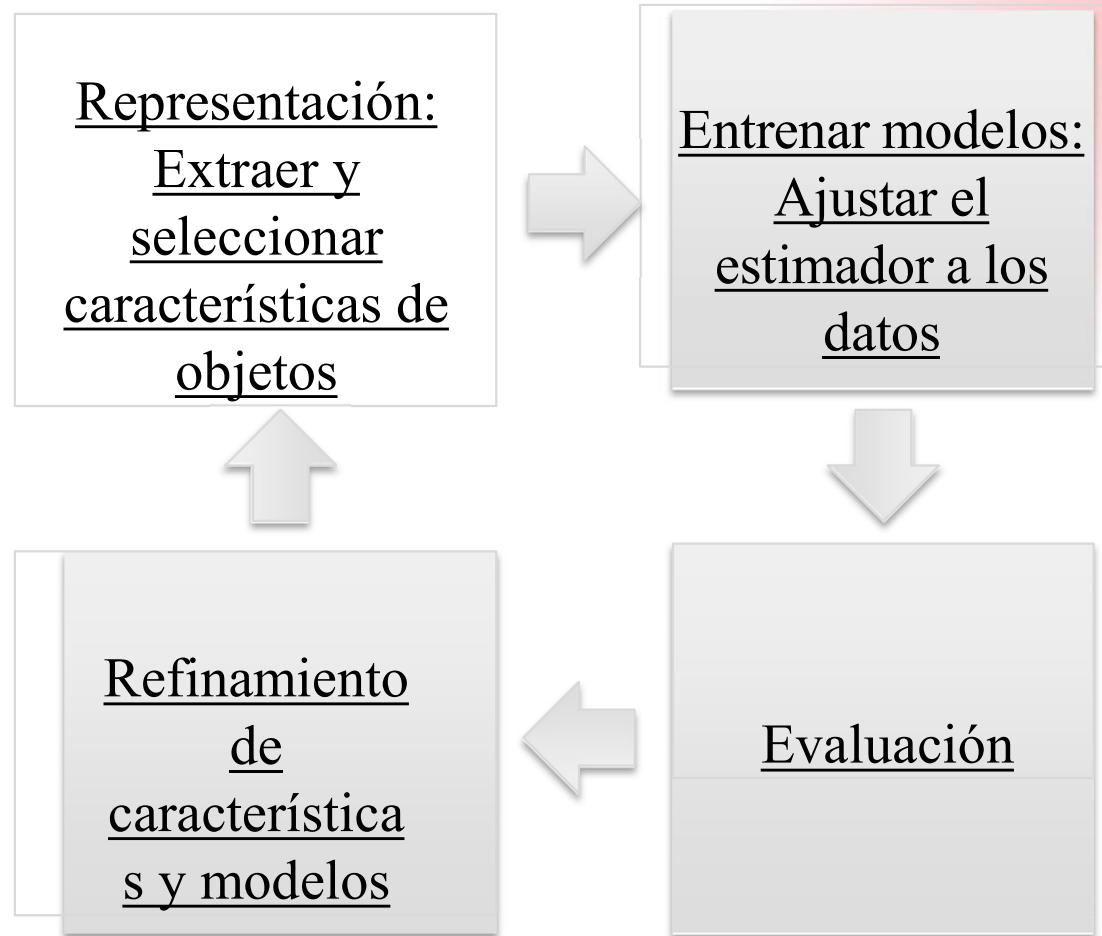
EVALUACIÓN DE MODELOS

(la métrica para GridSearch)

Límites de las métricas tradicionales ...

- LA PRECISIÓN SOLO DA UNA VISIÓN MUY PARCIAL DE LO QUE HACE UN CLASIFICADOR
- r^2 SOLO DA UNA VISIÓN MUY PARCIAL DE LO BUENA QUE ES LA APROXIMACIÓN DE UNA FUNCIÓN
- Es importante comprender la motivación detrás de estas métricas para comprender la información que proporcionan.
- Obtén información sobre cómo elegir la métrica correcta para seleccionar entre modelos o para ajustar parámetros.

Evaluación



Evaluación

- Diferentes aplicaciones tienen objetivos muy diferentes
- La precisión es ampliamente utilizada, pero muchas otras son posibles, por ejemplo:
 - Satisfacción del usuario (búsqueda web)
 - Cantidad de ingresos (comercio electrónico)
 - Aumento de las tasas de supervivencia de los pacientes (médico)

Evaluación

- Es muy importante elegir métodos de evaluación que coincidan con el objetivo de su solicitud.
-
- Calcule la métrica de evaluación seleccionada para varios modelos diferentes.
-
- A continuación, seleccione el modelo con el "mejor" valor de la métrica de evaluación.

Precisión con clases desequilibradas

- Supongamos que tienes dos clases:
 - Relevante (R): la clase positiva
 - No_Relevante (N): la clase negativa
- De 1000 artículos seleccionados al azar, en promedio
- Un artículo es relevante y tiene una etiqueta R
- El resto de los artículos (999 de ellos) no son relevantes y están etiquetados como N. Recordemos que:
 - Accuracy =
$$\frac{\text{\#correct predictions}}{\text{\#total instances}}$$

Precisión con clases desequilibradas

- Crea un clasificador para predecir elementos relevantes y ve que su precisión en un conjunto de pruebas es del 99,9 %.
- A modo de comparación, supongamos que tuviéramos un clasificador "ficticio" que no mirara las características en absoluto, y siempre predijera ciegamente la clase más frecuente (es decir, la clase N negativa).

Precisión con clases desequilibradas

- Suponiendo un conjunto de prueba de 1000 instancias, ¿cuál sería la precisión de este clasificador ficticio? Respuesta:
 - $\text{Accuracy}_{\text{DUMMY}} = 999 / 1000 = 99.9\%$

Los clasificadores ficticios(dummy) ignoran los datos de entrada

- Los clasificadores ficticios sirven como control
- Proporcionan una línea de base de métrica nula (por ejemplo, precisión nula).
- Algunas configuraciones de uso común para el parámetro de estrategia para
- DummyClassifier en scikit-learn:
 - ***most_frequent*** : predicts the most frequent label
 - ***stratified*** : random predictions
 - ***uniform*** : generates predictions uniformly at random.
 - ***constant*** : always predicts a constant label provided by the user.

Resultados de predicciones binarias

<u>False</u>	TN	FP
<u>True</u>	FN	TP
	<u>Predicted</u> negative	<u>Predicted</u> positive

Label 1 = positive class
(class of interest)

Label 0 = negative class
(everything else)

TP = true positive

FP = false positive (Type I error)

TN = true negative

FN = false negative (Type II error)

Confusion matrix for binary prediction task

True negative	TN = 356	FP = 51
True positive	FN = 38	TP = 5
	Predicted negative	Predicted positive

N = 450

- Cada instancia de prueba está exactamente en un cuadro (recuentos de enteros).
- Desglosa los resultados del clasificador por tipo de error.
- Por lo tanto, proporciona más información que la simple precisión.
- Le ayuda a elegir una métrica de evaluación que coincida con los objetivos del proyecto.
- Ni un solo número como la precisión. Pero hay muchas métricas posibles que se pueden derivar de la matriz de confusión.

Binary (two-class) confusion matrix

```
[13]: from sklearn.metrics import confusion_matrix

# Negative class (0) is most frequent
dummy_majority = DummyClassifier(strategy = 'most_frequent').fit(X_train, y_train)
y_majority_predicted = dummy_majority.predict(X_test)
confusion = confusion_matrix(y_test, y_majority_predicted)

print('Most frequent class (dummy classifier)\n', confusion)
```

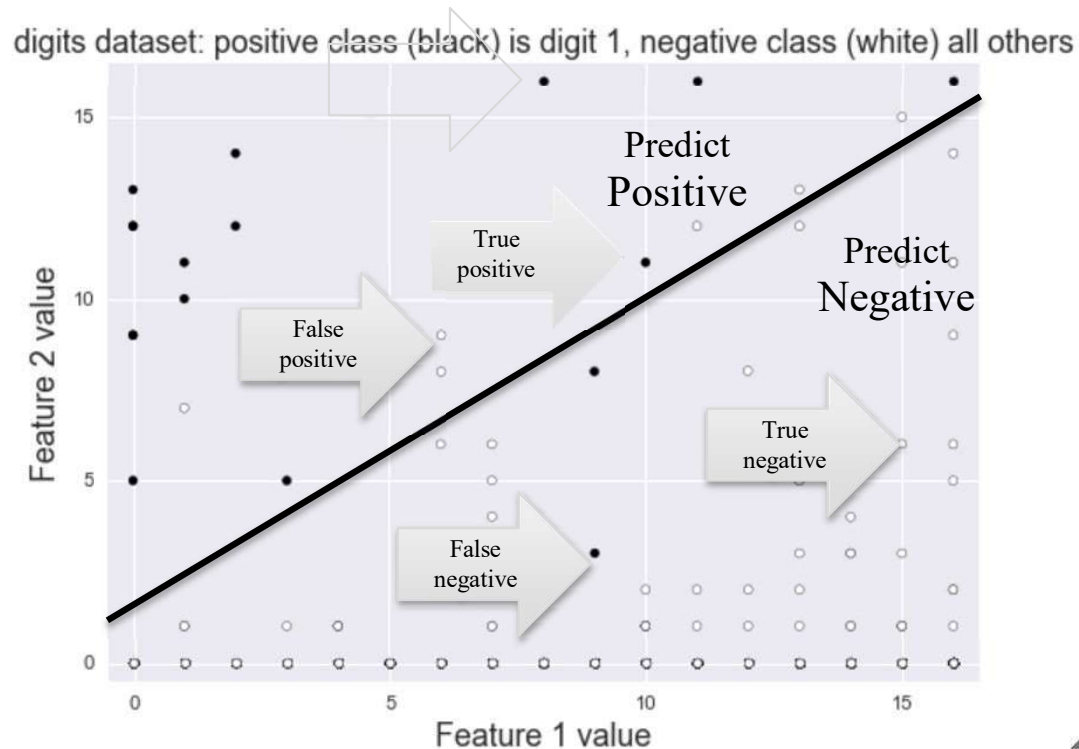
```
Most frequent class (dummy classifier)
[[407  0]
 [ 43  0]]
```

```
[14]: # produces random predictions w/ same class proportion as training set
dummy_classprop = DummyClassifier(strategy='stratified').fit(X_train, y_train)
y_classprop_predicted = dummy_classprop.predict(X_test)
confusion = confusion_matrix(y_test, y_classprop_predicted)

print('Random class-proportional prediction (dummy classifier)\n', confusion)
```

```
Random class-proportional prediction (dummy classifier)
[[358  49]
 [ 41   2]]
```


Visualización de diferentes tipos de error



$TN = 429$	$FP = 6$
$FN = 2$	$TP = 13$

Precisión:

True
negative

TN = 400

FP = 7

True
positive

FN = 17

TP = 26

Predicted
negative

Predicted
positive

$N = 450$

$$\begin{aligned}\text{Accuracy} &= \frac{TN+TP}{TN+TP+FN+FP} \\ &= \frac{400+26}{400+26+17+7} \\ &= 0.95\end{aligned}$$

Classification error (1 – Accuracy):

True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
	Predicted negative	Predicted positive	$N = 450$

$$\begin{aligned}\text{ClassificationError} &= \frac{FP + FN}{TN + TP + FN + FP} \\ &= \frac{7+17}{400+26+17+7} \\ &= 0.060\end{aligned}$$

Recall (sensibilidad/exhaustividad)

True
negative

TN = 400

FP = 7

True
positive

FN = 17

TP = 26

Predicted
negative

Predicted
positive

$N = 450$

$$\text{Recall} = \frac{TP}{TP+FN}$$

$$= \frac{26}{26+17}$$

$$= 0.60$$

Recall is also known as:

- True Positive Rate (TPR)
- Sensitivity
- Probability of detection

Precision

True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
	Predicted negative	Predicted positive	$N = 450$

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$= \frac{26}{26+7}$$

$$= 0.79$$

False positive rate (FPR)

True
negative

TN = 400

FP = 7

True
positive

FN = 17

TP = 26

Predicted
negative

Predicted
positive

$N = 450$

$$FPR = \frac{FP}{TN+FP}$$

$$= \frac{7}{400+7}$$

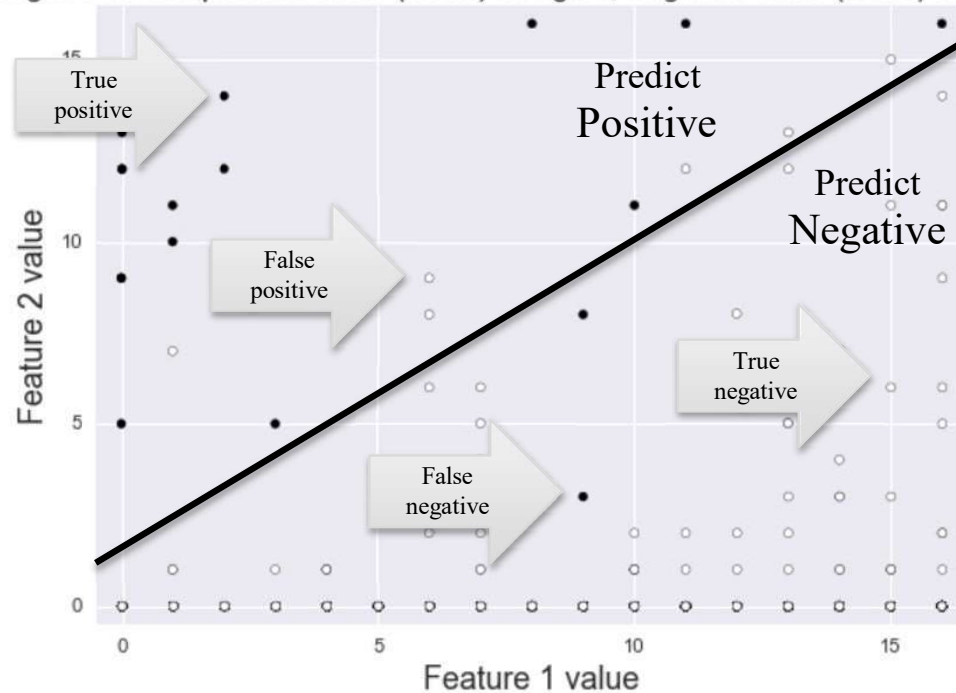
$$= 0.02$$

False Positive Rate is also known as:

- Specificity

The Precision-Recall Tradeoff (sensibilidad vs. Precision)

digits dataset: positive class (black) is digit 1, negative class (white) all others



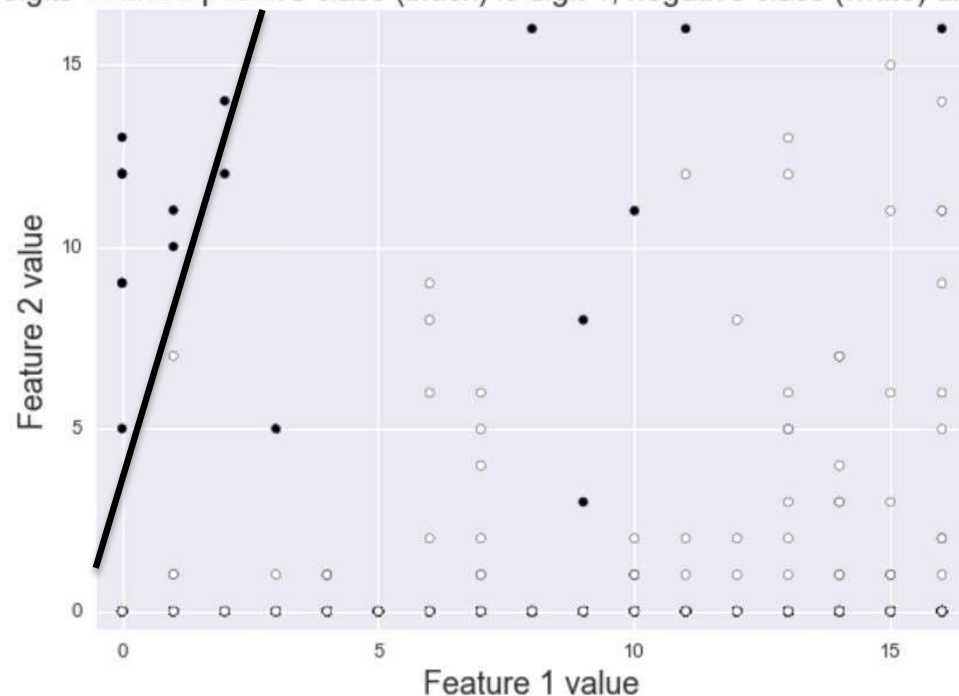
TN = 429	FP = 6
FN = 2	TP = 13

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{13}{19} = 0.68$$

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{13}{15} = 0.87$$

High Precision, Lower Recall

digits dataset: positive class (black) is digit 1, negative class (white) all others



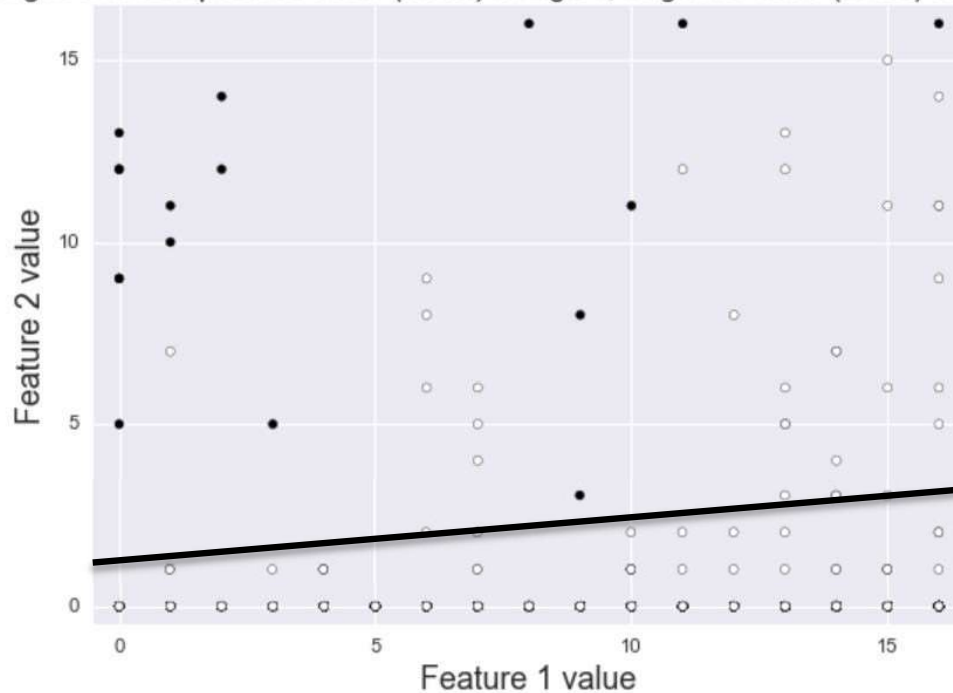
TN = 435	FP = 0
FN = 8	TP = 7

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{7}{7} = 1.00$$

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{7}{15} = 0.47$$

Low Precision, High Recall

digits dataset: positive class (black) is digit 1, negative class (white) all others



TN = 408	FP = 27
FN = 0	TP = 15

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{15}{42} = 0.36$$

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{15}{15} = 1.00$$

Compensación entre precision and recall

- Recall-oriented Tareas de aprendizaje automático:
 - Búsqueda y extracción de información en el descubrimiento legal
 - Detección de tumores
 - A menudo se combina con un experto humano para filtrar falsos positivos

Precision-oriented Tareas de aprendizaje automático:

- Clasificación en motores de búsqueda, sugerencia de consulta
- Clasificación de documentos
- Muchas tareas orientadas al cliente (¡los usuarios recuerdan los fracasos!)

F1-score: Combinando precisión y recuperación en un solo número

$$F_1 = 2 \cdot \frac{\textit{Precision} \cdot \textit{Recall}}{\textit{Precision} + \textit{Recall}} = \frac{2 \cdot TP}{2 \cdot TP + FN + FP}$$

F-score:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \cdot TP}{2 \cdot TP + FN + FP}$$

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{(\beta^2 \cdot \text{Precision}) + \text{Recall}} = \frac{(1 + \beta^2) \cdot TP}{(1 + \beta^2) \cdot TP + \beta \cdot FN + FP}$$

β allows adjustment of the metric to control the emphasis on recall vs precision:

- **Precision-oriented users: $\beta = 0.5$** (false positives hurt performance more than false negatives)
- **Recall-oriented users: $\beta = 2$** (false negatives hurt performance more than false positives)

```

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

print('Accuracy: {:.2f}'.format(accuracy_score(y_test, tree_predicted)))
print('Precision: {:.2f}'.format(precision_score(y_test, tree_predicted)))
print('Recall: {:.2f}'.format(recall_score(y_test, tree_predicted)))
print('F1: {:.2f}'.format(f1_score(y_test, tree_predicted)))

```

```

Accuracy: 0.95
Precision: 0.79
Recall: 0.60
F1: 0.68

```

```

from sklearn.metrics import classification_report

print(classification_report(y_test, tree_predicted, target_names=['not 1', '1']))

```

	precision	recall	f1-score	support
not 1	0.96	0.98	0.97	407
1	0.79	0.60	0.68	43
avg / total	0.94	0.95	0.94	450

```
]: print('Random class-proportional (dummy)\n',
      classification_report(y_test, y_classprop_predicted, target_names=['not 1', '1']))

print('SVM\n',
      classification_report(y_test, svm_predicted, target_names = ['not 1', '1']))

print('Logistic regression\n',
      classification_report(y_test, lr_predicted, target_names = ['not 1', '1']))

print('Decision tree\n',
      classification_report(y_test, tree_predicted, target_names = ['not 1', '1']))
```

```
Random class-proportional (dummy)
              precision    recall  f1-score   support

 not 1         0.91         0.94         0.92         407
    1         0.19         0.14         0.16          43

 avg / total         0.84         0.86         0.85         450
```

```
SVM
              precision    recall  f1-score   support

 not 1         0.99         0.99         0.99         407
    1         0.88         0.88         0.88          43
```

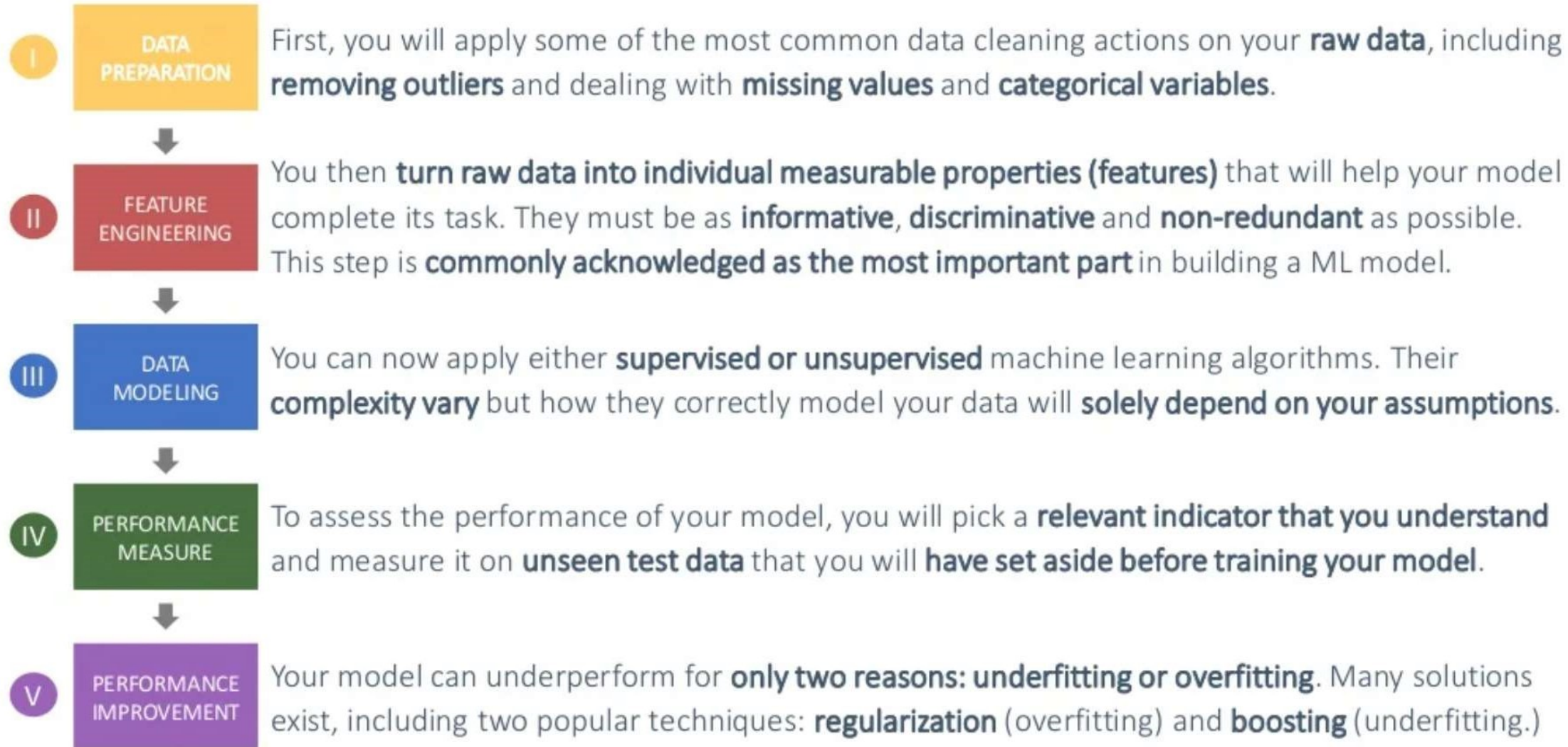
Parámetros de búsqueda de cuadrícula (Grid Search Parameters)

```
sklearn.model_selection.GridSearchCV(estimator, param_grid, scoring=None,  
    n_jobs=None, iid='deprecated', refit=True, cv=None, verbose=0,  
    pre_dispatch='2*n_jobs', error_score=nan, return_train_score=False)
```

1. **estimator**: Pass the model instance for which you want to check the hyperparameters.
2. **params_grid**: the dictionary object that holds the hyperparameters you want to try
3. **scoring**: evaluation metric that you want to use, you can simply pass a valid string/ object of evaluation metric
4. **cv**: number of cross-validation you have to try for each selected set of hyperparameters
5. **verbose**: you can set it to 1 to get the detailed print out while you fit the data to GridSearchCV
6. **n_jobs**: number of processes you wish to run in parallel for this task if it -1 it will use all available processors.

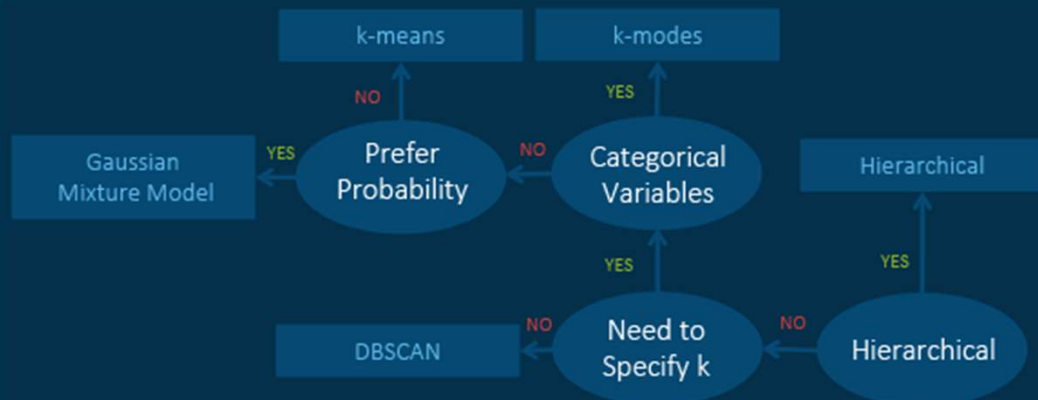
BUILDING A MACHINE LEARNING MODEL: SUMMARY

Pulsa Esc para salir del modo de pantalla completa

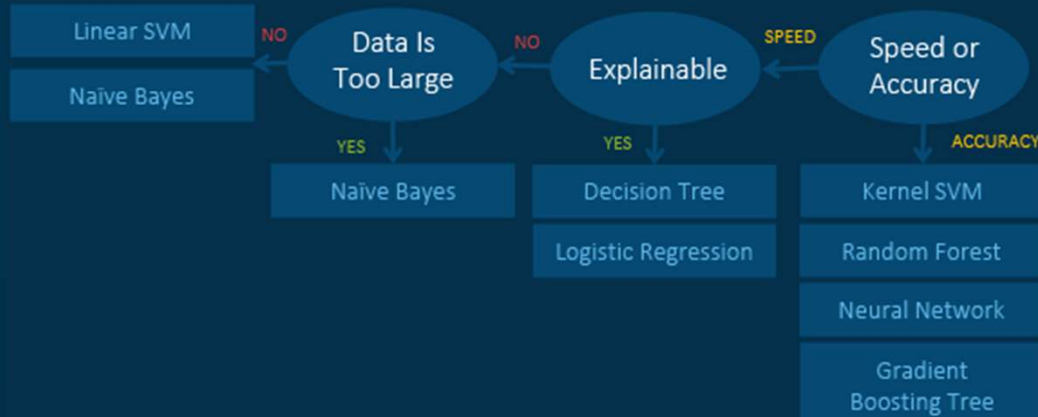


Machine Learning Algorithms Cheat Sheet

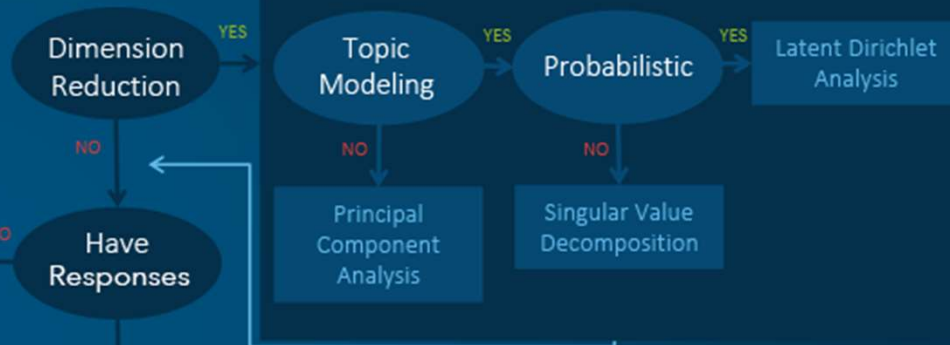
Unsupervised Learning: Clustering



Supervised Learning: Classification

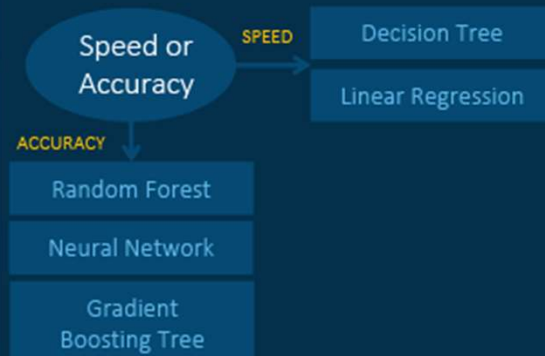


START



Unsupervised Learning: Dimension Reduction

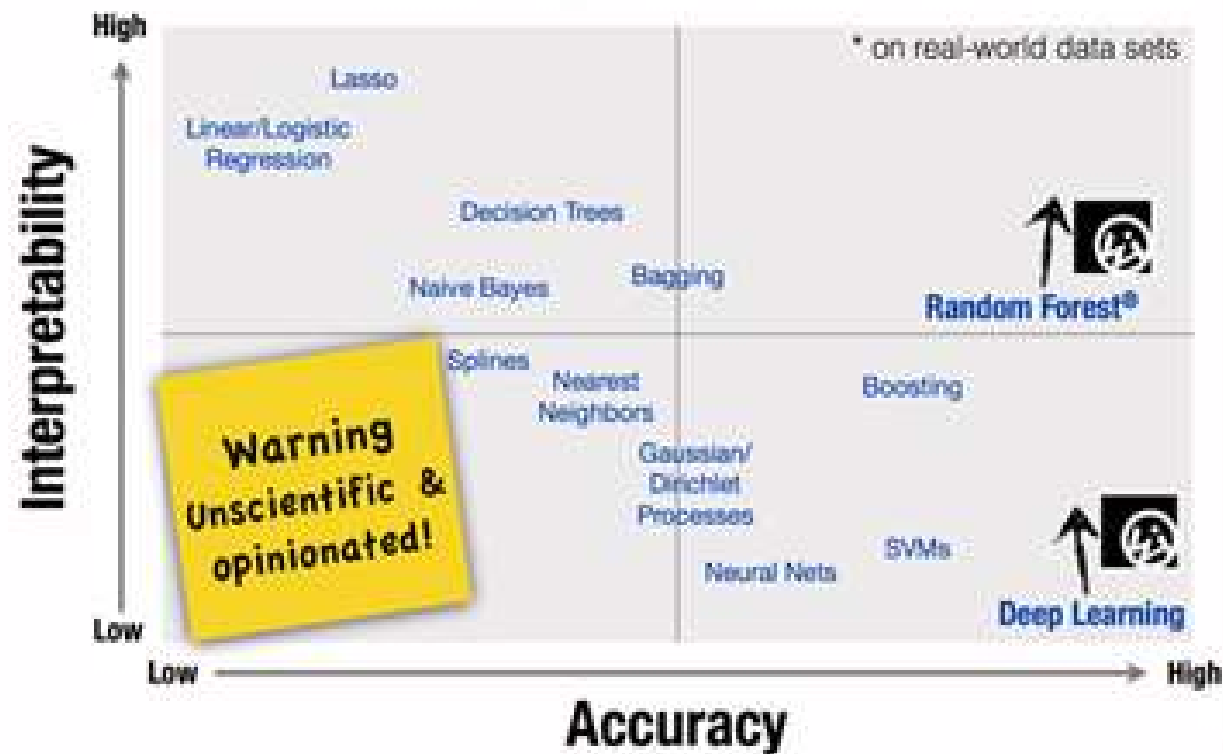
Supervised Learning: Regression



ESTRATEGIA GENERAL

- Decidir la métrica
- Cuando se trabaja con un nuevo conjunto de datos, en general es una buena idea comenzar con un modelo simple, como un modelo lineal, Bayes o K-nn y ver hasta dónde puede llegar. Después de comprender más sobre los datos, puede considerar pasar a un algoritmo que pueda crear modelos más complejos, como bosques aleatorios, XGBoost, SVM para acabar redes neuronales/deep learning.
- La mayoría de los algoritmos presentados anteriormente tienen variantes de clasificación y regresión, y todos los algoritmos de clasificación admiten la clasificación binaria y multiclase.



ML Algorithmic Trade-Off



Cuando aplicar los modelos

- **K Vecinos más cercanos:** para conjuntos de datos pequeños, buenos como línea de base, fáciles de explicar.
- **Modelos lineales:** Opta como primer algoritmo a probar, bueno para conjuntos de datos muy grandes, bueno para datos de muy alta dimensión.
- **Árboles de decisión:** Muy rápidos, no necesitan escalado de los datos, se pueden visualizar y explicar fácilmente.
- **Bosques aleatorios:** Casi siempre se comportan mejor que un solo árbol de decisión, muy robustos y potentes. No es necesario escalar los datos. No es bueno para datos dispersos de dimensiones muy altas.
- **Árboles de decisión potenciados por gradiente** (Gradient boosted decision trees) : a menudo son ligeramente más precisos que los bosques aleatorios. Más lento de entrenar, pero más rápido de predecir que el bosque aleatorio y más pequeño en memoria. Necesita más ajuste de parámetros que el bosque aleatorio.
- **Máquinas vectoriales de soporte:** Potente para conjuntos de datos de tamaño mediano de características con significado similar. Necesita escalado de datos, sensible a los parámetros.
- **NN/DL:** Grandes cantidades de datos, costosos, nula explicabilidad, buenos resultados.



#AIskills4all |  @AIskillsEU |  [linkedin.com/AIskillsEU](https://www.linkedin.com/AIskillsEU)



www.aiskills.eu

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.