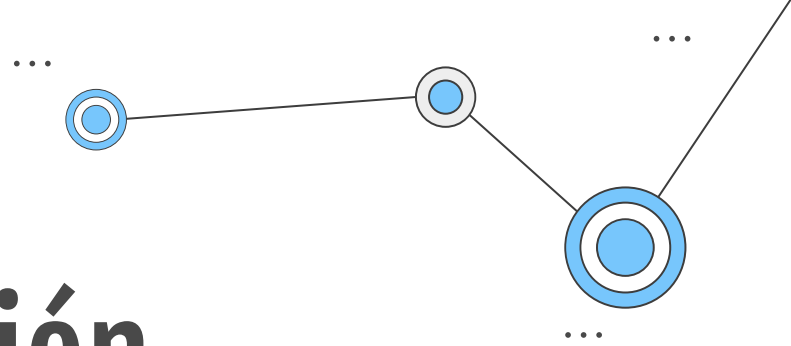


Clasificación



**UNIVERSIDAD
CATÓLICA**
DE CÓRDOBA
JESUITAS

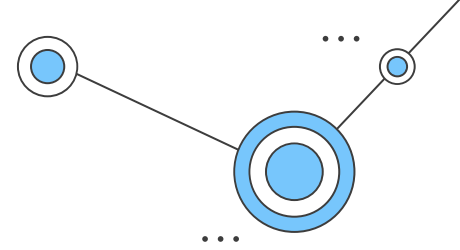
Dr. Francisco Arduh
2023

Dataset MNIST

- 70,000 imágenes pequeñas (28 x 28) de dígitos escritos a mano.
- Cada imagen tiene como etiqueta el número que le corresponde.
- Dataset muy utilizado en la comunidad de machine learning para probar algoritmos de clasificación.



Clasificador Binario

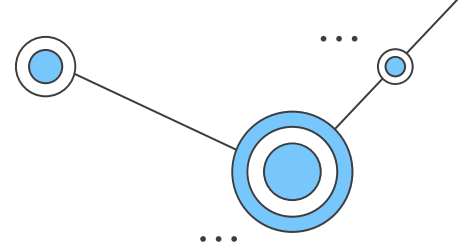


- Simplificamos el problema para detectar sólo un dígito.
- Por ej. Un detector de 5.
- Vamos a utilizar un clasificado “SGD Classifier” que maneja instancias de forma independiente y se puede utilizar para aprendizaje online.

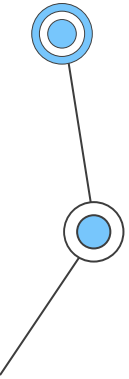
```
>>> y_train_5 = (y_train == 5)
>>> y_test_5 = (y_test == 5)
>>> from sklearn.linear_model import SGDClassifier
>>> sgd_clf = SGDClassifier(random_state=42)
>>> sgd_clf.fit(X_train, y_train_5)
>>> sgd_clf.predict([some_digit])
array([ True])
```



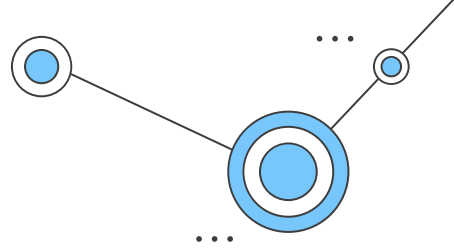
Medición de Rendimiento



- Una parte importante del flujo de trabajo de Machine Learning es la evaluación del desempeño del modelo.
- En el mundo real, por lo general, es imposible tener un clasificador perfecto.
- Debemos entender el problema y entender que es lo importante del mismo, por ej:
 - Si estamos tratando de determinar si un tumor es maligno o benigno, nos interesa que la predicción incorrecta de que un tumor es maligno sea lo más pequeña posible.
 - Si estamos tratando de determinar que transacciones son fraudulentas, podríamos estar interesados en perder la menor cantidad posible de este tipo de transacciones.



Midiendo "Accuracy" usando validación cruzada



Se define accuracy como:

$$accuracy = \frac{\text{predicciones correctas}}{\text{total de predicciones}}$$

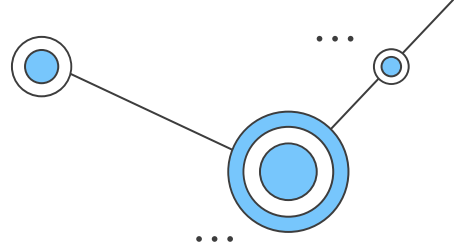
Es posible utilizar `cross_val_score()` para evaluar nuestro clasificador.

```
>>> from sklearn.model_selection import cross_val_score
>>> cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([0.96355, 0.93795, 0.95615])
```

El accuracy de un resultado por encima del 93%.



Midiendo "Accuracy" en Dummy Classifier



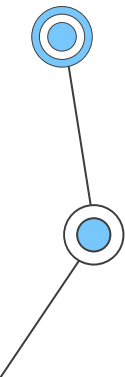
Utilizando un clasificador que siempre predice la clase más frecuente (en este caso no 5)

```
>>> from sklearn.dummy import DummyClassifier  
>>> never_5_clf = DummyClassifier(strategy="most_frequent")
```

Mide el accuracy

```
>>> cross_val_score(never_5_clf, X_train, y_train_5, cv=3, scoring="accuracy")  
array([0.91125, 0.90855, 0.90915])
```

Esto da más del 90%! No siempre es buena idea usar accuracy como métrica.



Implementación de Validación cruzada sin `cross_val_score()`

```
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone

skfolds = StratifiedKFold(n_splits=3, random_state=42)

for train_index, test_index in skfolds.split(X_train, y_train_5):
    clone_clf = clone(sgd_clf)
    X_train_folds = X_train[train_index]
    y_train_folds = y_train_5[train_index]
    X_test_fold = X_train[test_index]
    y_test_fold = y_train_5[test_index]

    clone_clf.fit(X_train_folds, y_train_folds)
    y_pred = clone_clf.predict(X_test_fold)
    n_correct = sum(y_pred == y_test_fold)
    print(n_correct / len(y_pred)) # prints 0.9502, 0.96565, and 0.96495
```

Matriz de confusión

Para computar, la matriz de confusión es necesario tener las predicciones y la etiqueta real de cada instancia.

Actual \ Predicted	Negative	Positive
Negative	8 3 9 7 2	6
Positive	5 5	5 5 5

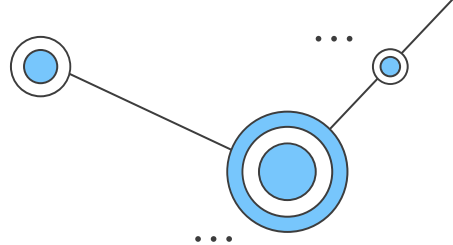
TPNFP

Precision (e.g., 3 out of 4)

Recall (e.g., 3 out of 5)

Notación: TP: verdadero positivo. TN: verdadero negativo. FP: falso positivo. FN: falso negativo.

Matriz de confusión en código



```
>>> from sklearn.model_selection import cross_val_predict
>>> from sklearn.metrics import confusion_matrix
>>>
>>> y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)
>>> confusion_matrix(y_train_5, y_train_pred)
array([[53057, 1522],
       [ 1325, 4096]])
```

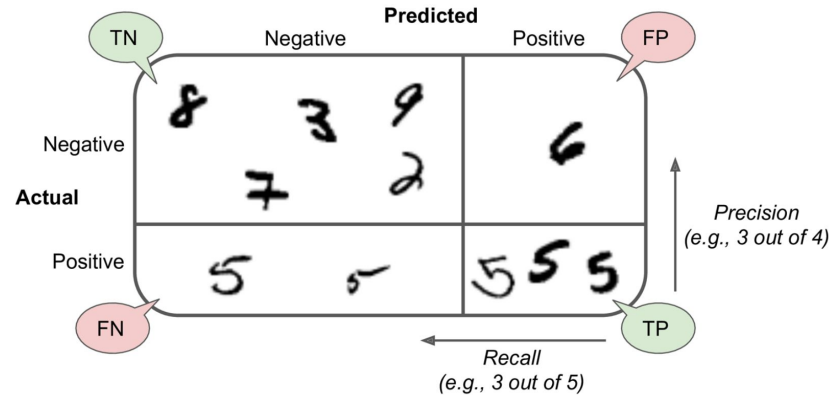


“Precision” y “recall”

A partir de la matriz de confusión puede obtenerse métricas más concisas

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

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



A confusion matrix diagram illustrating precision and recall using handwritten digits. The matrix is divided into four quadrants: True Negative (TN), False Positive (FP), False Negative (FN), and True Positive (TP). The rows represent the 'Actual' class and the columns represent the 'Predicted' class.

	Predicted	
	Negative	Positive
Negative	8, 3, 9 7, 2	6
Positive	5, 5	5, 5, 5

Callouts and Metrics:

- TN (True Negative):** Top-left quadrant (green circle).
- FP (False Positive):** Top-right quadrant (pink circle).
- FN (False Negative):** Bottom-left quadrant (pink circle).
- TP (True Positive):** Bottom-right quadrant (green circle).
- Precision:** Indicated by an upward arrow on the right, labeled "Precision (e.g., 3 out of 4)".
- Recall:** Indicated by a leftward arrow at the bottom, labeled "Recall (e.g., 3 out of 5)".

“Precision” y “recall” en código

```
>>> from sklearn.metrics import precision_score, recall_score
>>> precision_score(y_train_5, y_train_pred) # == 4096 / (4096 + 1522)
0.7290850836596654
>>> recall_score(y_train_5, y_train_pred) # == 4096 / (4096 + 1325)
0.7555801512636044
```

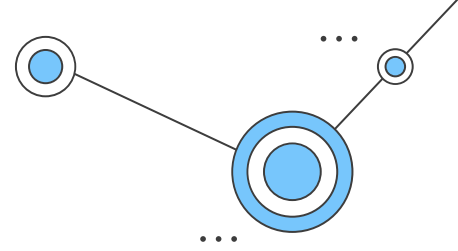
- Algunas veces es conveniente combinar precision y recall en una sola métrica llamada F1 score.
- F1 score se define como la media armónica entre el recall y la precision:

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

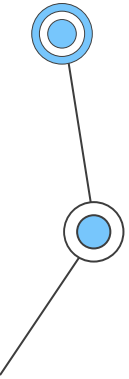
Para computarlo en código:

```
>>> from sklearn.metrics import f1_score
>>> f1_score(y_train_5, y_train_pred)
0.7420962043663375
```

Uso de F1 score

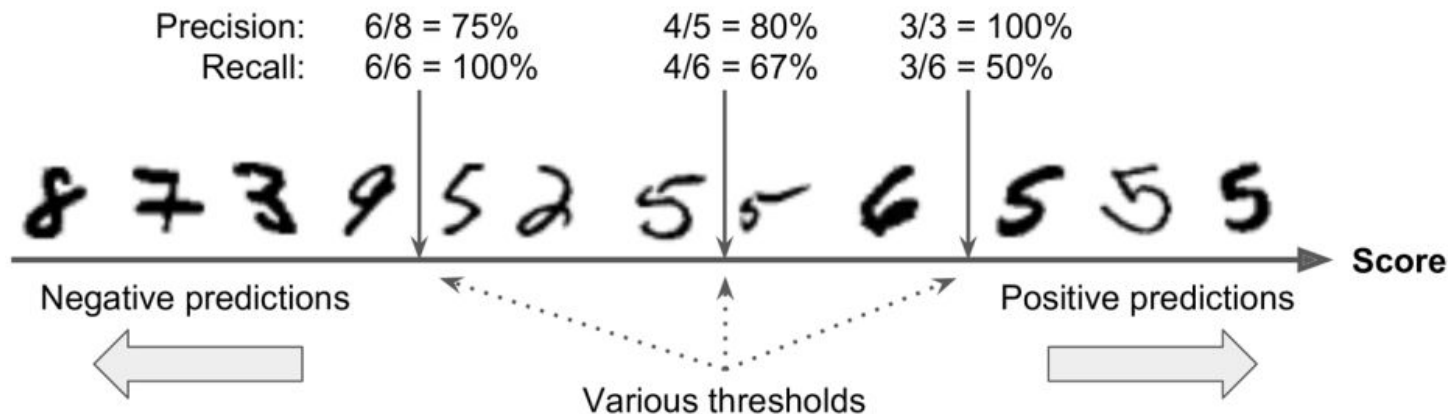


- F1 score nos proveerá un clasificador con recall y precision similar, aunque no es siempre lo que queremos. Ej:
 - Detectar videos seguros para niños: es preferible descartar buenos videos (bajo recall), pero descartar los malos (alta precision)
 - Clasificador para detectar “mecheras” en una tienda es importante que nuestro clasificador tenga un alto recall.



Precision/recall trade off

Tomemos el clasificador **SDGClassifier**, para cada instancia computa un puntaje basado en su función de decisión.

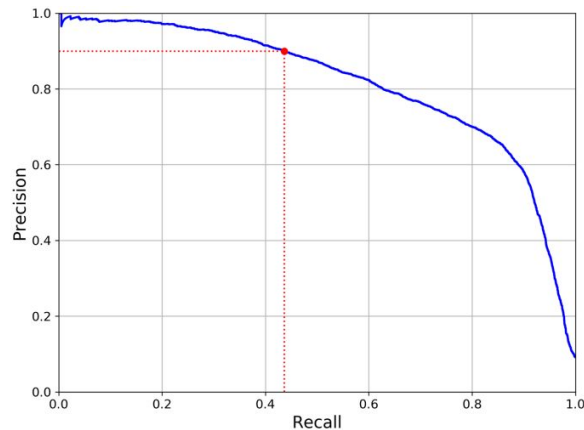
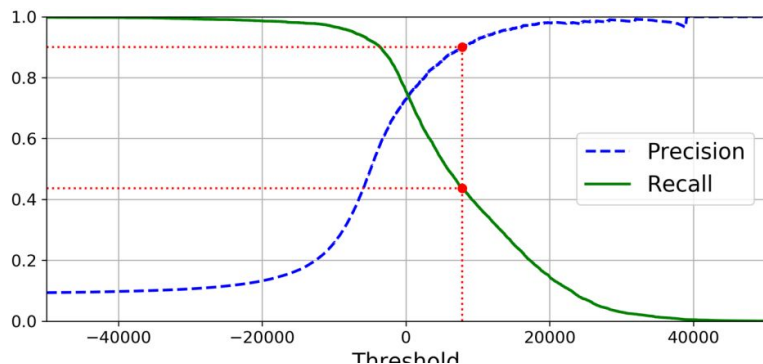


Precision/recall trade off en código

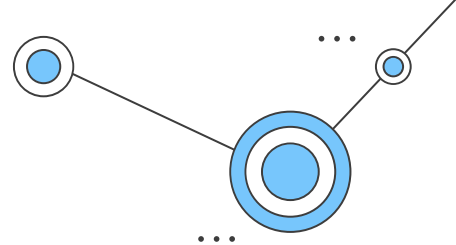
```
from sklearn.metrics import precision_recall_curve
```

```
precisions, recalls, thresholds = precision_recall_curve(y_train_5, y_scores)
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall")
    [...] # highlight the threshold and add the legend, axis label, and grid
```

```
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.show()
```



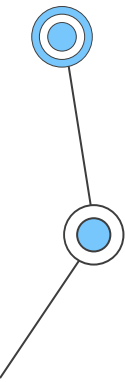
Curva ROC



- La curva ROC (receiver operating characteristic) es otra métrica usada en clasificación binaria.
- Es similar a la curva precision/recall, pero utiliza las siguientes métricas:
 - True positive rate: otro nombre para recall.
 - False positive rate: se define como la tasa de instancia negativa que han sido clasificadas como positivas.

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

- A TPR se lo puede llamar *sensitivity* y al FPR también se lo denomina como *1-specificity*.



Curva ROC en código

```
from sklearn.metrics import roc_curve
```

```
fpr, tpr, thresholds = roc_curve(y_train_5, y_scores)
```

```
def plot_roc_curve(fpr, tpr, label=None):
```

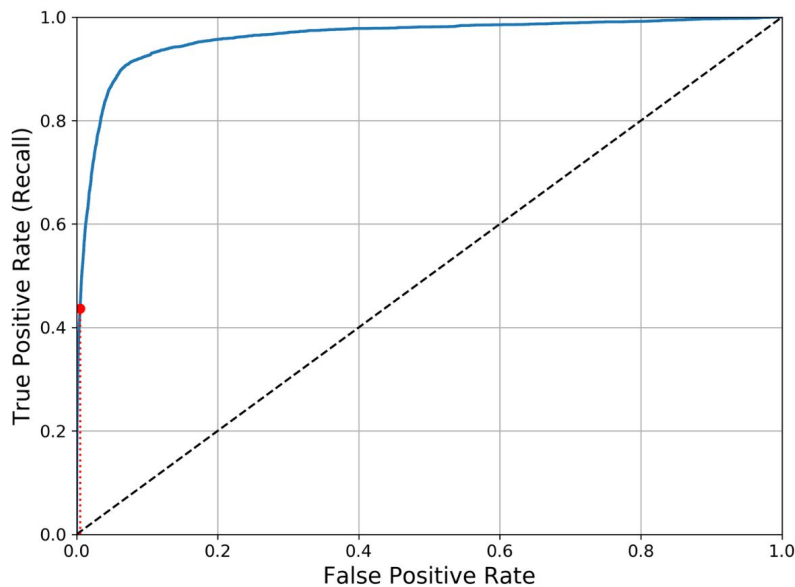
```
    plt.plot(fpr, tpr, linewidth=2, label=label)
```

```
    plt.plot([0, 1], [0, 1], 'k--') # Dashed diagonal
```

```
    [...] # Add axis labels and grid
```

```
plot_roc_curve(fpr, tpr)
```

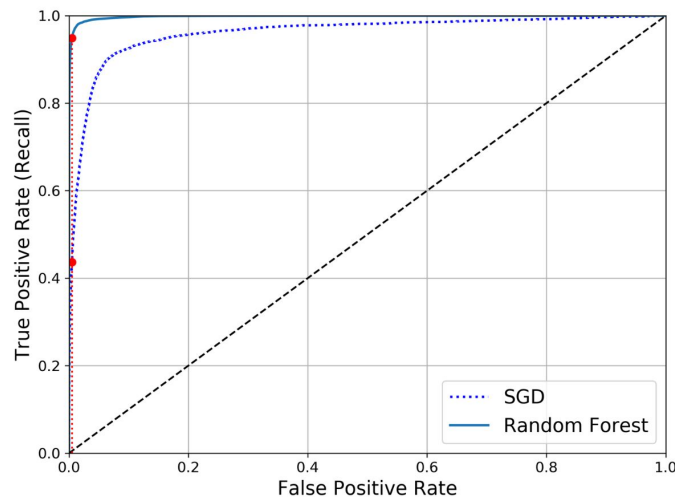
```
plt.show()
```



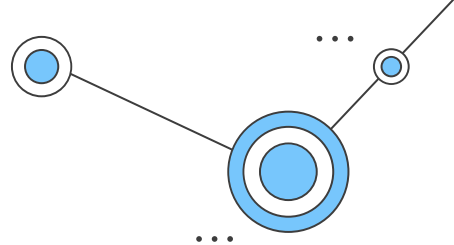
Curva ROC para comparar modelos

```
from sklearn.ensemble import RandomForestClassifier
```

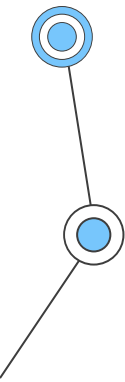
```
forest_clf = RandomForestClassifier(random_state=42)
y_proba_forest = cross_val_predict(forest_clf, X_train, y_train_5, cv=3,
                                   method="predict_proba")
y_scores_forest = y_proba_forest[:, 1] # score = proba of positive class
fpr_forest, tpr_forest, thresholds_forest = roc_curve(y_train_5, y_scores_forest)
plt.plot(fpr, tpr, "b:", label="SGD")
plot_roc_curve(fpr_forest, tpr_forest, "Random Forest")
plt.legend(loc="lower right")
plt.show()
```



Clasificación multiclase



- ¿Qué pasa si queremos distinguir más de dos clases?
- Algunos algoritmos son capaces de manejar multiclase naturalmente (Random Forest, SGD Classifier, Naive Bayes), mientras que otros no (Logistic regression, Support Vector Machine)
- Es posible construir clasificadores multiclase a partir de clasificadores binarios:
 - One versus the rest (OvR): entreno clasificadores para detectar un dígito.
 - One versus one (OvO): entreno clasificadores para distinguir entre dos clases (por ej: 1 vs 2)



Clasificación multiclase en Sklearn.

En caso que el clasificador solo pueda manejar clasificación binaria se pueden utilizar las clases **OneVsOneClassifier** o **OneVsRestClassifier**.

```
>>> from sklearn.multiclass import OneVsRestClassifier
>>> ovr_clf = OneVsRestClassifier(SVC())
>>> ovr_clf.fit(X_train, y_train)
```

En el caso de **SDGClassifier** no es necesario las funciones anteriores porque puede manejar clasificación multiclase.

```
>>> sgd_clf.fit(X_train, y_train)
>>> cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring="accuracy")
array([0.8489802 , 0.87129356, 0.86988048])
```

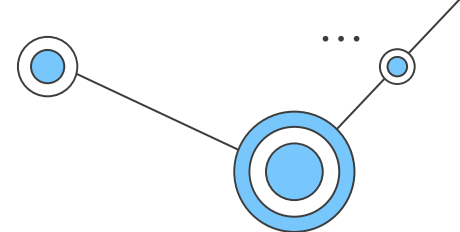
Usando scaler

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler()
>>> X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
>>> cross_val_score(sgd_clf, X_train_scaled, y_train, cv=3, scoring="accuracy")
array([0.89707059, 0.8960948 , 0.90693604])
```

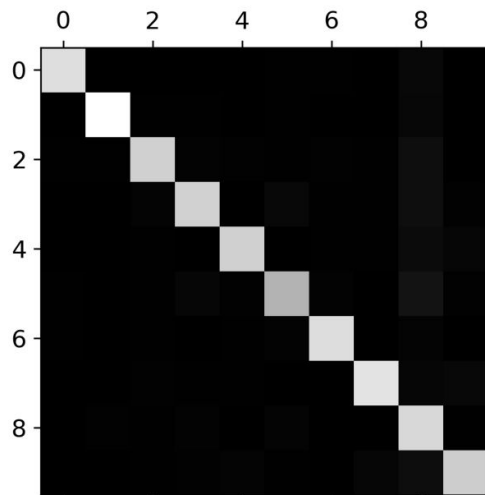
Análisis de error

```
>>> y_train_pred = cross_val_predict(sgd_clf, X_train_scaled, y_train, cv=3)
>>> conf_mx = confusion_matrix(y_train, y_train_pred)
>>> conf_mx
array([[5578,    0,   22,    7,    8,   45,   35,    5,  222,    1],
       [    0, 6410,   35,   26,    4,   44,    4,    8,  198,   13],
       [   28,   27, 5232,  100,   74,   27,   68,   37,  354,   11],
       [   23,   18,  115, 5254,    2,  209,   26,   38,  373,   73],
       [   11,   14,   45,   12, 5219,   11,   33,   26,  299,  172],
       [   26,   16,   31,  173,   54, 4484,   76,   14,  482,   65],
       [   31,   17,   45,    2,   42,   98, 5556,    3,  123,    1],
       [   20,   10,   53,   27,   50,   13,    3, 5696,  173,  220],
       [   17,   64,   47,   91,    3,  125,   24,   11, 5421,   48],
       [   24,   18,   29,   67,  116,   39,    1,  174,  329, 5152]])
```

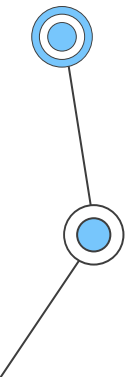
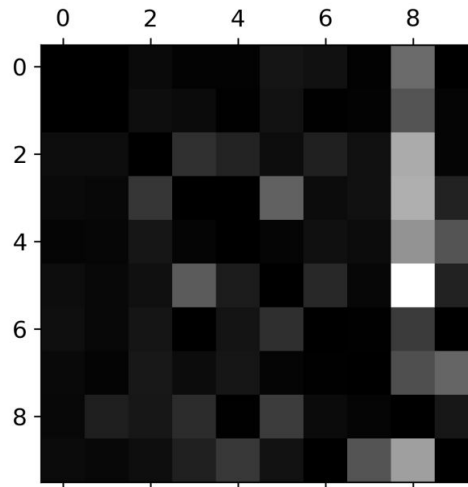
Análisis de error: Gráfico



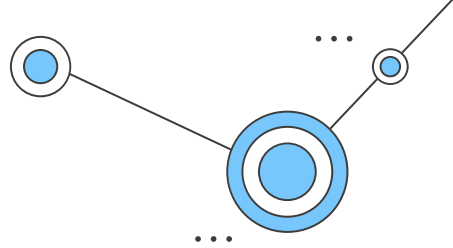
```
plt.matshow(conf_mx, cmap=plt.cm.gray)
plt.show()
```



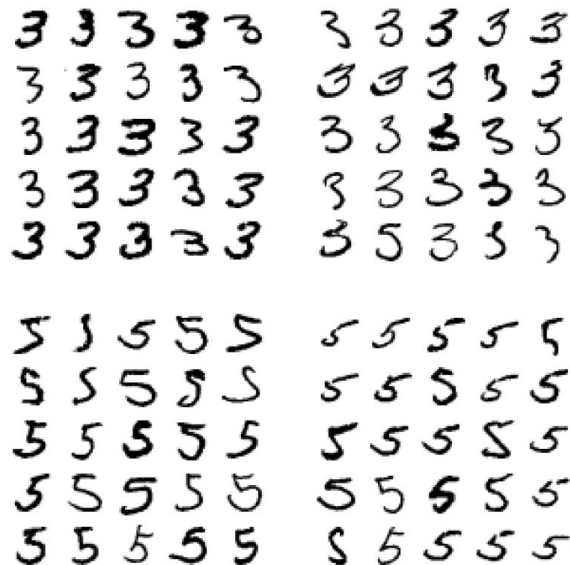
```
row_sums = conf_mx.sum(axis=1, keepdims=True) ...
norm_conf_mx = conf_mx / row_sums
np.fill_diagonal(norm_conf_mx, 0)
plt.matshow(norm_conf_mx, cmap=plt.cm.gray)
plt.show()
```



Análisis de error: Gráfico



```
cl_a, cl_b = 3, 5
X_aa = X_train[(y_train == cl_a) & (y_train_pred == cl_a)]
X_ab = X_train[(y_train == cl_a) & (y_train_pred == cl_b)]
X_ba = X_train[(y_train == cl_b) & (y_train_pred == cl_a)]
X_bb = X_train[(y_train == cl_b) & (y_train_pred == cl_b)]
plt.figure(figsize=(8,8))
plt.subplot(221); plot_digits(X_aa[:25], images_per_row=5)
plt.subplot(222); plot_digits(X_ab[:25], images_per_row=5)
plt.subplot(223); plot_digits(X_ba[:25], images_per_row=5)
plt.subplot(224); plot_digits(X_bb[:25], images_per_row=5)
plt.show()
```



Clasificación de múltiples etiquetas

Por ej: Se quiere saber si un número es mayor a 7 e impar.

```
from sklearn.neighbors import KNeighborsClassifier

y_train_large = (y_train >= 7)
y_train_odd = (y_train % 2 == 1)
y_multilabel = np.c_[y_train_large, y_train_odd]

knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_multilabel)
```

```
KNeighborsClassifier()
```

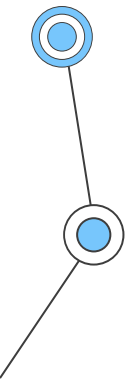
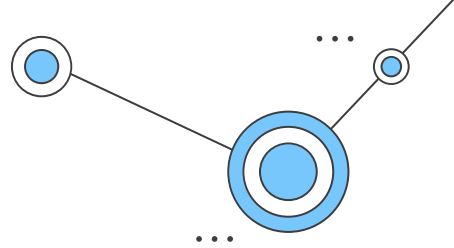
```
knn_clf.predict([some_digit])
```

```
array([[False,  True]])
```

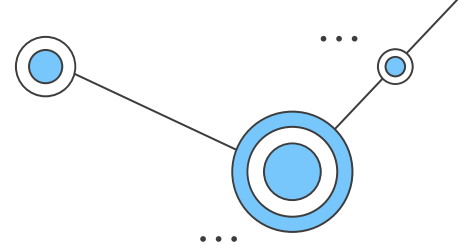
Warning: the following cell may take a very long time (possibly hours depending on your hardware).

```
y_train_knn_pred = cross_val_predict(knn_clf, X_train, y_multilabel, cv=3)
f1_score(y_multilabel, y_train_knn_pred, average="macro")
```

```
0.976410265560605
```



Clasificación de múltiples salidas



Ejemplo: se quiere limpiar el ruido de la imagen.

```
noise = np.random.randint(0, 100, (len(X_train), 784))
X_train_mod = X_train + noise
noise = np.random.randint(0, 100, (len(X_test), 784))
X_test_mod = X_test + noise
y_train_mod = X_train
y_test_mod = X_test
```

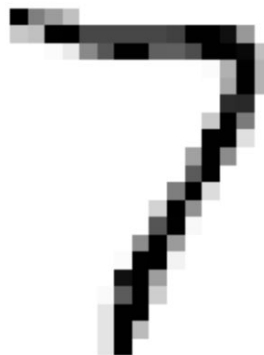
```
some_index = 0
plt.subplot(121); plot_digit(X_test_mod[some_index])
plt.subplot(122); plot_digit(y_test_mod[some_index])
save_fig("noisy_digit_example_plot")
plt.show()
```

Saving figure noisy_digit_example_plot

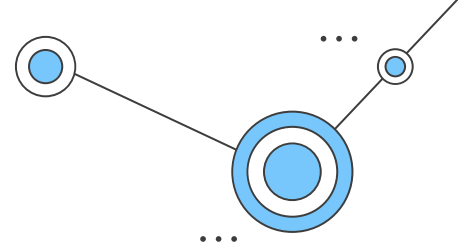


```
knn_clf.fit(X_train_mod, y_train_mod)
clean_digit = knn_clf.predict([X_test_mod[some_index]])
plot_digit(clean_digit)
save_fig("cleaned_digit_example_plot")
```

Saving figure cleaned_digit_example_plot



Consideraciones finales



- El clasificador perfecto, por lo general, no es alcanzable.
- Entender qué es lo más importante y a partir de esto utilizar la métrica que mejor se ajuste a esto.
- Entender las posibles fallas de nuestro algoritmo.

