TL01 MNIST

Índice

1 MNIST

Modified NIST (MNIST): corpus de $70\,000$ imágenes 28×28 en gris de dígitos manuscritos

Partición estándar: $60\,000$ primeras muestras para training y $10\,000$ restantes para test

Fuente original: http://yann.lecun.com/exdb/mnist

Tarea muy popular: desde su introducción en los 90, MNIST ha sido muy usado como tarea para comparar técnicas de ML

Tarea "agotada": pues ya se han alcanzado tasas de error muy reducidas, por debajo del 0.1%

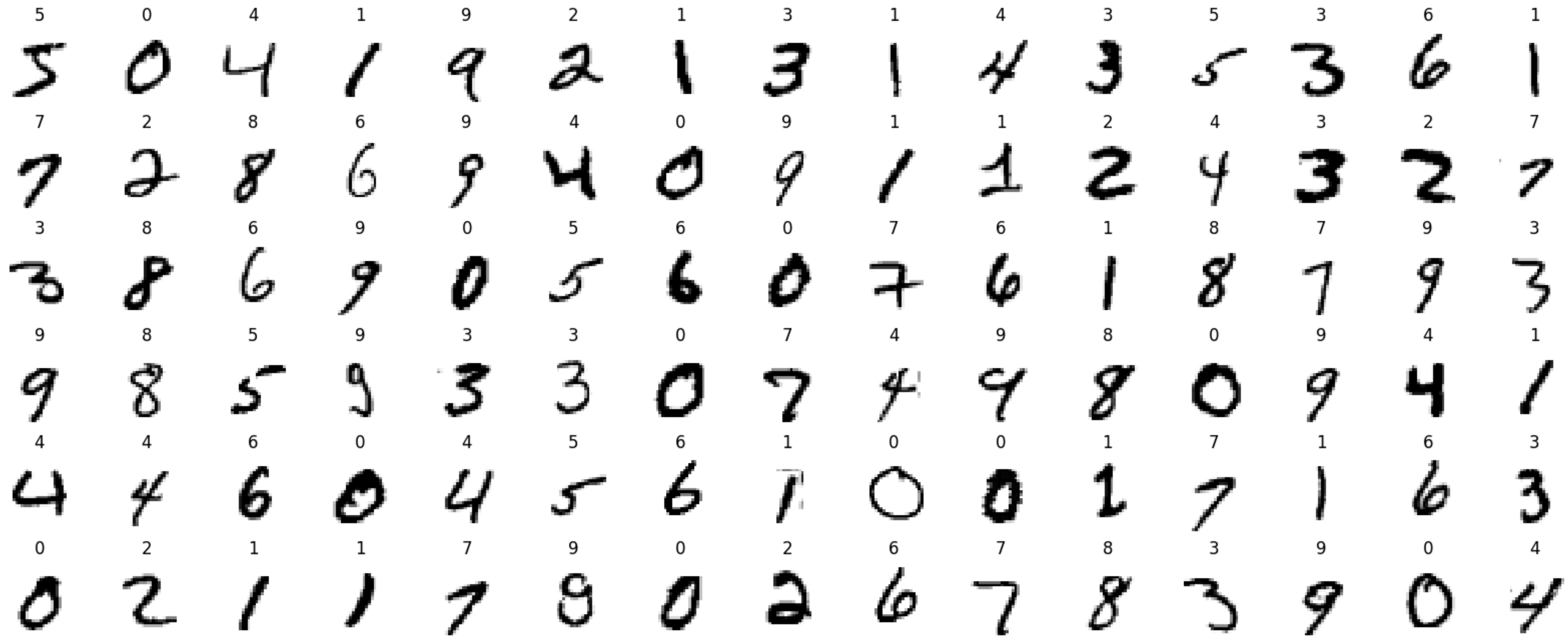
Más info: https://en.wikipedia.org/wiki/MNIST_database

Ejemplo: lectura de MNIST con fetch_openml de sklearn

In [1]:
```
import numpy as np; from sklearn.datasets import fetch_openml
mnist_784_X, mnist_784_y = fetch_openml('mnist_784', version=1, return_X_y=True, as_frame=False, parser='auto')
X_train = mnist_784_X[:60000].astype(np.float32); y_train = mnist_784_y[:60000].astype(np.uint8)
X_test = mnist_784_X[60000:].astype(np.float32); y_test = mnist_784_y[60000:].astype(np.uint8)
X_train /= 255; X_test /= 255 # normalización a [0,1]
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
(60000, 784) (60000,) (10000, 784) (10000,)
```

In [2]:
```
import matplotlib.pyplot as plt
nrows = 6; ncols = 15
_, axs = plt.subplots(nrows=nrows, ncols=ncols, figsize=(16, 16*nrows/ncols), constrained_layout=True)
for ax, x, y in zip(axs.flat, X_train, y_train):
    ax.set_axis_off(); image = x.reshape(28, 28); ax.set_title(y)
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="none")
```



2 Modelos lineales generativos

Notación: $\boldsymbol{x} \in \mathbb{R}^D$ y $y \in \{1, \dots, C\}$

Clasificador generativo: expresa **posteriors** en función de **priors** y **densidades condicionales** de las clases, las cuales puede muestrearse para **generar** datos sintéticos

$$p(y = c \mid \boldsymbol{x}, \boldsymbol{\theta}) = \frac{p(\boldsymbol{x} \mid y = c, \boldsymbol{\theta})\, p(y = c, \boldsymbol{\theta})}{\sum_{c'} p(\boldsymbol{x} \mid y = c', \boldsymbol{\theta})\, p(y = c', \boldsymbol{\theta})} \propto p(\boldsymbol{x} \mid y = c, \boldsymbol{\theta})\, p(y = c, \boldsymbol{\theta})$$

Linealidad: la log-posterior suele ser lineal con $\boldsymbol{x}$, aunque en algunos es cuadrática

Ajuste: la máximización de la log-verosimilitud conjunta suele conducir a estimadores fáciles de calcular

- Los priors de las clases se estiman como frecuencias relativas, $\hat{\pi}_c = N_c/N$
- Los parámetros de cada clase se estiman con sus datos de entrenamiento (salvo $\boldsymbol{\Sigma}$ en LDA, que se estima con todos)

Lectura de MNIST:

In [1]:
```
import numpy as np; from sklearn.datasets import fetch_openml
mnist_784_X, mnist_784_y = fetch_openml('mnist_784', version=1, return_X_y=True, as_frame=False, parser='auto')
X_train = mnist_784_X[:60000].astype(np.float32); y_train = mnist_784_y[:60000].astype(np.uint8)
X_test = mnist_784_X[60000:].astype(np.float32); y_test = mnist_784_y[60000:].astype(np.uint8)
X_train /= 255; X_test /= 255 # normalización a [0,1]
```

In [2]:
```
import warnings; warnings.filterwarnings('ignore'); from sklearn.metrics import accuracy_score
```

2.1 Naive Bayes

Clasificador naive Bayes Gaussiano (GNB): $\boldsymbol{\theta}_c = (\boldsymbol{\theta}_{c1}, \dots, \boldsymbol{\theta}_{cD})^t$, $\boldsymbol{\theta}_{cd} = (\mu_{cd}, \sigma_{cd}^2)$, media y varianza de la característica d en c

$$p(\boldsymbol{x} \mid y = c, \boldsymbol{\theta}_c) = \prod_{d=1}^{D} \mathcal{N}(x_d \mid \mu_{cd}, \sigma_{cd}^2)$$

$$\hat{\mu}_{cd} = \frac{1}{N_c} \sum_{n: y_n = c} x_{nd}$$

$$\hat{\sigma}_{cd}^2 = \frac{1}{N_c} \sum_{n: y_n = c} (x_{nd} - \hat{\mu}_{cd})^2$$

Aplicación a MNIST: `var_smoothing` (1e-9 por omisión) fija el porcentaje de la mayor varianza empírica hallada para suavizar varianzas

In [3]:
```
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB(var_smoothing=1e-9).fit(X_train, y_train)
acc = accuracy_score(y_test, clf.predict(X_test))
print(f'La precisión de {clf!s} es {acc:.1%}')
```

```
La precisión de GaussianNB() es 55.6%
```

2.2 Análisis discriminante lineal (LDA)

LDA: $\boldsymbol{\theta}_c = (\boldsymbol{\mu}_c^t,\ \text{vec}(\boldsymbol{\Sigma}))^t,$ media de la clase c y matriz de varianzas común para todas las clases

$$p(\boldsymbol{x} \mid y = c, \boldsymbol{\theta}_c) = \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_c, \boldsymbol{\Sigma})$$

$$\hat{\boldsymbol{\mu}}_c = \frac{1}{N_c} \sum_{n:y_n=c} \boldsymbol{x}_n$$

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{N} \sum_c \sum_{n:y_n=c} (\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_c)(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_c)^t$$

Aplicación a MNIST: `tol` (1e-4 por omisión) controla el suavizado de la matriz de varianzas

In [4]:
```python
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
clf = LinearDiscriminantAnalysis(tol=1e-4).fit(X_train, y_train)
acc = accuracy_score(y_test, clf.predict(X_test))
print(f'La precisión de {clf!s} es {acc:.1%}')
```

```
La precisión de LinearDiscriminantAnalysis() es 87.3%
```

2.3 Análisis discriminante cuadrático (QDA)

QDA: $\boldsymbol{\theta}_c = (\boldsymbol{\mu}_c^t, \operatorname{vec}(\boldsymbol{\Sigma}_c))^t$, media y matriz de varianzas de la clase c

$$p(\boldsymbol{x} \mid y = c, \boldsymbol{\theta}_c) = \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$$

$$\hat{\boldsymbol{\mu}}_c = \frac{1}{N_c} \sum_{n:y_n=c} \boldsymbol{x}_n$$

$$\hat{\boldsymbol{\Sigma}}_c = \frac{1}{N_c} \sum_{n:y_n=c} (\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_c)(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_c)^t$$

Aplicación a MNIST: `reg_param` suaviza matrices de varianzas mediante interpolación con $\mathbf{I}$ (0, valor por omisión, no suaviza; 1 identidad); aplicamos QDA con PCA previo para limitar el número de características y ajustamos hiper-parámetros con GridSearchCV

In [5]:
```
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
pca = PCA(); qda = QuadraticDiscriminantAnalysis()
pipe = Pipeline(steps=[("pca", pca), ("qda", qda)])
G = {"pca__n_components": [60, 70, 80, None], "qda__reg_param": [0.01, 0.02, 0.05]}
GS = GridSearchCV(pipe, G, scoring='accuracy', refit=True, cv=5)
acc = GS.fit(X_train, y_train).score(X_test, y_test)
print(f'Precisión: {acc:.1%} con {GS.best_params_}')
```

```
Precisión: 96.6% con {'pca__n_components': 70, 'qda__reg_param': 0.02}
```

3 Modelos lineales discriminativos

Clasificador discriminativo: modela **posteriors** directamente, sin modelar priors ni densidades condicionales

$$p(y = c \mid \boldsymbol{x}; \boldsymbol{\theta}) = \cdots$$

Linealidad: en el caso más sencillo, la log-posterior es lineal con $\boldsymbol{x}$ o alguna transformación de $\boldsymbol{x}$, $\boldsymbol{\phi}(\boldsymbol{x})$

Ajuste: la minimización de la NLL suele conducir a estimadores que se calculan con descenso por gradiente

Lectura de MNIST:

In [1]:
```
import numpy as np; from sklearn.datasets import fetch_openml
mnist_784_X, mnist_784_y = fetch_openml('mnist_784', version=1, return_X_y=True, as_frame=False, parser='auto')
X_train = mnist_784_X[:60000].astype(np.float32); y_train = mnist_784_y[:60000].astype(np.uint8)
X_test = mnist_784_X[60000:].astype(np.float32); y_test = mnist_784_y[60000:].astype(np.uint8)
X_train /= 255; X_test /= 255 # normalización a [0,1]
```

In [2]:
```
import warnings; warnings.filterwarnings('ignore'); from sklearn.metrics import accuracy_score
```

3.1 Regresión logística

Regresión logística: $p(y \mid \boldsymbol{x}, \mathbf{W}) = \operatorname{Cat}(y \mid \boldsymbol{\mu}),\; \boldsymbol{\mu} = \operatorname{Cat}(y \mid \mathcal{S}(\boldsymbol{a})),\; \boldsymbol{a} = \mathbf{W}^t \boldsymbol{x},\; \mathbf{W} \in \mathbb{R}^{D \times C}$

NLL: $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\},\; \operatorname{NLL}(\mathbf{W}) = \frac{1}{N} \sum_n \ell(\boldsymbol{y}_n, \hat{\boldsymbol{y}}_n)$ con $\hat{\boldsymbol{y}}_n = \boldsymbol{\mu}_n = \mathcal{S}(\boldsymbol{a}_n),\; \boldsymbol{a}_n = \mathbf{W}^t \boldsymbol{x}_n$

$$\frac{\partial \operatorname{NLL}(\mathbf{W})}{\partial \mathbf{W}} = \frac{1}{N} (\boldsymbol{\mathcal{S}}(\mathbf{X}\mathbf{W}) - \mathbf{Y})^t \mathbf{X} \quad \text{con } \boldsymbol{\mathcal{S}} \text{ aplicada por filas, } \mathbf{Y} = [\boldsymbol{y}_1, \dots, \boldsymbol{y}_N]^t$$

Descenso por gradiente: $$\mathbf{W}_{i+1} = \mathbf{W}_i - \eta_i \left.\frac{\partial \operatorname{NLL}}{\partial \mathbf{W}^t}\right|_{\mathbf{W}_i} \qquad \text{con} \qquad \frac{\partial \operatorname{NLL}}{\partial \mathbf{W}^t} = \frac{1}{N} \mathbf{X}^t (\boldsymbol{\mathcal{S}}(\mathbf{X}\mathbf{W}) - \mathbf{Y})$$

Aplicación a MNIST: con des-regularización `C` (1 por omisión; próximo a cero máxima regularización), `solver`, `tol` y `max_iter`

In [3]:
```
from sklearn.linear_model import LogisticRegression; from sklearn.model_selection import GridSearchCV
G = {"solver": ["lbfgs"], "tol": [1e-4], "C": [1], "max_iter": [100]}
GS = GridSearchCV(LogisticRegression(random_state=23), G, scoring='accuracy', refit=True, cv=5)
acc = GS.fit(X_train, y_train).score(X_test, y_test)
print(f'Precisión: {acc:.1%} con {GS.best_params_}')
```

```
Precisión: 92.6% con {'C': 1, 'max_iter': 100, 'solver': 'lbfgs', 'tol': 0.0001}
```

3.2 Ingeniería de características

Propósito: encontrar alguna transformación de los datos, $\boldsymbol{x}$, $\boldsymbol{\phi}(\boldsymbol{x})$, que linearice un problema de clasificación con clases (datos) no linealmente separables

PolynomialFeatures: añade características polinómicas hasta un grado dado (2 por omisión)

Número de características polinómicas: puede limitarse con PCA previo

Aplicación a MNIST: con PCA previo y ajuste de hiper-parámetros (con GridSearchCV)

In [4]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.pipeline import Pipeline
pca = PCA(); poly = PolynomialFeatures(); scaler = StandardScaler(); logreg = LogisticRegression()
pipe = Pipeline(steps=[("pca", pca), ("poly", poly), ("scaler", scaler), ("logreg", logreg)])
G = {"pca__n_components": [70], "poly__degree": [2], "logreg__C": [1]}
GS = GridSearchCV(pipe, G, scoring='accuracy', refit=True, cv=5)
acc = GS.fit(X_train, y_train).score(X_test, y_test)
print(f'Precisión: {acc:.1%} con {GS.best_params_}')
```

```
Precisión: 97.8% con {'logreg__C': 1, 'pca__n_components': 70, 'poly__degree': 2}
```

4 Random forests

Ensamble de árboles: reduce la varianza de los árboles promediando M modelos base

Ensamble en clasificación: la salida se decide por el **método comité,** esto es, por voto mayoritario

Bagging: los modelos base se ajustan con diferentes versiones de los datos, obtenidas por boostraping

Random forests: variante de bagging con aleatorización, no solo de datos, sino también de variables de entrada

Aplicación a MNIST: con `n_estimators` y `max_depth` ajustados mediante GridSearchCV

In [1]:
```
import numpy as np; from sklearn.datasets import fetch_openml
mnist_784_X, mnist_784_y = fetch_openml('mnist_784', version=1, return_X_y=True, as_frame=False, parser='auto')
X_train = mnist_784_X[:60000].astype(np.float32); y_train = mnist_784_y[:60000].astype(np.uint8)
X_test = mnist_784_X[60000:].astype(np.float32); y_test = mnist_784_y[60000:].astype(np.uint8)
X_train /= 255; X_test /= 255 # normalización a [0,1]
```

In [2]:
```
import warnings; warnings.filterwarnings('ignore'); from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier; from sklearn.model_selection import GridSearchCV
clf = RandomForestClassifier(random_state=23)
G = {"n_estimators": [100, 200, 300, 400, 500], "max_depth": [None]}
GS = GridSearchCV(clf, G, scoring='accuracy', refit=True, cv=5, verbose=1)
acc = GS.fit(X_train, y_train).score(X_test, y_test)
print(f'Precisión: {acc:.1%} con {GS.best_params_}')
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
Precisión: 97.2% con {'max_depth': None, 'n_estimators': 400}
```

5 Boosting

Modelo aditivo de funciones base adaptativas: ensamble visto como suma de modelos base

Boosting (FSAM): minimiza el riesgo empírico mediante ajuste secuencial de modelos base

Gradient boosting: FSAM visto como descenso por gradiente para un problema de minimización en un espacio funcional

Aplicación a MNIST: con `max_depth` ajustado mediante GridSearchCV

In [1]:
```
import numpy as np; from sklearn.datasets import fetch_openml
mnist_784_X, mnist_784_y = fetch_openml('mnist_784', version=1, return_X_y=True, as_frame=False, parser='auto')
X_train = mnist_784_X[:60000].astype(np.float32); y_train = mnist_784_y[:60000].astype(np.uint8)
X_test = mnist_784_X[60000:].astype(np.float32); y_test = mnist_784_y[60000:].astype(np.uint8)
X_train /= 255; X_test /= 255 # normalización a [0,1]
```

In [2]:
```
import warnings; warnings.filterwarnings('ignore'); from sklearn.metrics import accuracy_score
from sklearn.ensemble import HistGradientBoostingClassifier; from sklearn.model_selection import GridSearchCV
clf = HistGradientBoostingClassifier(random_state=23)
G = {"max_depth": [4, 8, 16, None]}
GS = GridSearchCV(clf, G, scoring='accuracy', refit=True, cv=5, verbose=1)
acc = GS.fit(X_train, y_train).score(X_test, y_test)
print(f'Precisión: {acc:.1%} con {GS.best_params_}')
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
Precisión: 97.9% con {'max_depth': None}
```

6 Ejercicio: Fashion-MNIST

Fashion-MNIST: corpus de $70\,000$ imágenes 28×28 en gris de 10 prendas de ropa

Partición estándar: $60\,000$ primeras muestras para training y $10\,000$ restantes para test

Fuente original: https://github.com/zalandoresearch/fashion-mnist

Formato idéntico al de MNIST: se publicó en 2017 como tarea continuadora de la ya agotada MNIST

Más info: https://en.wikipedia.org/wiki/Fashion_MNIST

Ejercicio: realiza experimentos como los anteriores para obtener la máxima precisión posible

6.1 Lectura de Fashion-MNIST

In [1]:
```python
import numpy as np
from sklearn.datasets import fetch_openml
fashion_mnist_X, fashion_mnist_y = fetch_openml('Fashion-MNIST', return_X_y=True, as_frame=False, parser='auto')
X_train = fashion_mnist_X[:60000].astype(np.float32); y_train = fashion_mnist_y[:60000].astype(np.uint8)
X_test = fashion_mnist_X[60000:].astype(np.float32); y_test = fashion_mnist_y[60000:].astype(np.uint8)
X_train /= 255; X_test /= 255 # normalización a [0,1]
labels = ('T-Shirt', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle Boot')
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
(60000, 784) (60000,) (10000, 784) (10000,)
```

In [2]:
```python
import matplotlib.pyplot as plt
nrows = 2; ncols = 10
_, axs = plt.subplots(nrows=nrows, ncols=ncols, figsize=(16, 16*nrows/ncols), constrained_layout=True)
for ax, x, y in zip(axs.flat, X_train, y_train):
    ax.set_axis_off(); image = x.reshape(28, 28); ax.set_title(labels[y])
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="none")
```