

**Bloque 2**  
**Aprendizaje Automático**

**Práctica 2:**

**Aplicación de los algoritmos de Perceptrón y Regresión Logística a varios datasets**

**2024-2025**

**(Grupo C2)**

# Sesiones de la práctica 2

## Sesión 1 (22 Noviembre):

- familiarizarse con el entorno de trabajo (Google Colab)
- analizar 4 conjuntos de datos (datasets): **iris**, **digits**, **olivetti**, **openml**

## Sesión 2 (29 Noviembre):

- Aplicación del algoritmo del Perceptron a tareas de clasificación: dataset **iris**.
- **Ejercicio 1** : aplicar Perceptrón a **digits** y **olivetti**

## Sesión 3 (5 Diciembre):

- Aplicación de Regresión Logística a tareas de clasificación: dataset **iris**.
- **Ejercicio 2**: Aplicación de Regresión Logística a un dataset de OpenML

## Sesión 4 (**examen 13 Diciembre**):

- Hay que subir la **solución del Ejercicio 1 y Ejercicio 2**
- Se pedirá la aplicación de Regresión Logística para una tarea diferente de OpenML

## Sesión 2: Perceptron (formato de los datos – iris dataset)

```
X = iris.data.astype(np.float16);
```

matriz con las muestras de 4 características: 150 x 4

```
y = iris.target.astype(np.uint) .reshape(-1, 1);
```

vector con las etiquetas de las clases (0,1,2); se convierte a un vector de una sola columna 150 x 1

```
print(X.shape, y.shape, "\n", np.hstack([X, y])[:5, :])
```

np.hstack muestra los vectores /matrices transpuestos, es decir, filas horizontal, columnas vertical

```
(150, 4) (150, 1)
```

```
[[5.1015625 3.5 1.40039062 0.19995117 0.] → muestra; 4 características, clase 0  
[4.8984375 3. 1.40039062 0.19995117 0.]
```

```
....
```

## Sesión 2: Perceptron (algoritmo)

```
def perceptron(X, y, b=0.1, a=1.0, K=200):
    N, D = X.shape;
    Y = np.unique(y);
    C = Y.size;
    W = np.zeros((1+D, C))
    for k in range(1, K+1):           ;;para K iteraciones
        E = 0                        ;;número de muestras mal clasificadas
        for n in range(N):          ;;para N muestras
            xn = np.array([1, *X[n, :]]) ;;conversión a notación homogénea
            cn = np.squeeze(np.where(Y==y[n])) ;;obtiene la clase de xn
            gn = W[:,cn].T @ xn;      ;;multiplica xn por el vector de su clase
            err = False
            for c in np.arange(C):    ;;para C clases
                if c != cn and W[:,c].T @ xn + b >= gn:
                    W[:, c] = W[:, c] - a*xn;
                    err = True
            if err:
                W[:, cn] = W[:, cn] + a*xn;      ;;actualizar vector de pesos
                                                de la clase de la muestra
                E = E + 1
        if E == 0:
            break;
    return W, E, k
```

## Sesión 2: Perceptron (notas aclaratorias al algoritmo)

```
def perceptron(X, y, b=0.1, a=1.0, K=200):
```

`X` : matriz de muestras

`y` : vector de clases

`b=0.1` : valor por defecto del 'margen'

`a=1.0` : valor por defecto del factor de aprendizaje

`K=200` : número de iteraciones por defecto

## Sesión 2: Perceptron (notas aclaratorias al algoritmo)

`N, D = X.shape;`

N filas = N muestras (150)

D columnas = dimensión de las muestras (4)

`Y = np.unique(y);`

array que contiene los elementos únicos de **y**: ([0,1,2])

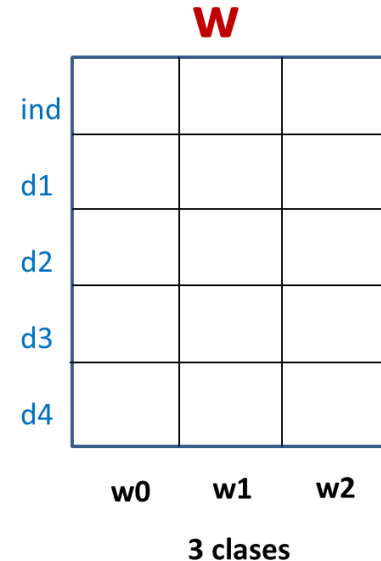
`C = Y.size;`

número de clases: 3

`W = np.zeros((1+D, C))` matriz de pesos

`W[:,cn]` la columna de la clase cn

`W[:,cn].T` la columna de la clase cn (transpuesta),  
es decir, en forma de fila (para poder  
multiplicar por xn)



`xn = np.array([1, *X[n, :]])`

convierte una muestra xn a formato homogéneo  
(añadir 1 al vector de características)

## Sesión 2

### Ejercicio 1 (entregar día del examen): aplicar el Perceptron a los datasets **digits** y **olivetti**

- Aplica el algoritmo Perceptron a los dos datasets para diferentes valores de  $a$  (factor de aprendizaje),  $b$  (margen) y  $K$  (número de iteraciones)

- Muestra los resultados del siguiente modo:

ejemplo de resultados para el **dataset iris**

#	a	b	K	E	k	Ete
#-----	-----	-----	-----	-----	-----	-----
	0.10	0.00	200	6	200	0.133
	0.10	0.01	200	2	200	0.167
	0.10	0.10	200	8	200	0.100
	0.10	1.00	200	10	200	0.033
	0.10	10.00	200	13	200	0.100
	0.10	100.00	200	29	200	0.000
	0.10	1000.00	200	82	200	0.133
	1.00	0.00	200	6	200	0.133
	1.00	0.01	200	6	200	0.133
	1.00	0.10	200	2	200	0.167
	1.00	1.00	200	8	200	0.100
	1.00	10.00	200	10	200	0.033
	1.00	100.00	200	13	200	0.100
	1.00	1000.00	200	29	200	0.000
	10.00	0.00	200	6	200	0.133
	10.00	0.01	200	6	200	0.133
	10.00	0.10	200	6	200	0.133
	10.00	1.00	200	8	200	0.100
	10.00	10.00	200	10	200	0.033
	10.00	100.00	200	13	200	0.100
	10.00	1000.00	200	29	200	0.000

- Indica cuál crees que es la mejor combinación de ' $a$ ', ' $b$ ' y ' $K$ ' para cada dataset (escribe la respuesta en el mismo cuaderno en una caja de texto)
- Sube el cuaderno con el output de la ejecución y la respuesta a la tarea que se creará para el examen

## Sesión 3: Regresión Logística (RL)

### Solver

La función `LogisticRegression` de `sklearn` ofrece la posibilidad de utilizar varios solvers:

**liblinear**: buena opción para datasets pequeños; se aplica para problemas de clasificación binaria o *one-versus-rest* para problemas multi-clase.

**sag, saga**: buenas opciones para datasets grandes

**sag, saga, lbfgs, newton-cg**: solo estos solvers manejan pérdida multinomial para problemas multi-clase

**newton-cholesky**: buena opción cuando número de muestras  $\gg$  número de características; se aplica para problemas de clasificación binaria o *one-versus-rest* para problemas multi-clase.



## Sesión 3: Regresión Logística (RL)

### Solver (RECOMENDACIONES)

Utilizar **sag**, **saga**, **lbfgs**, **newton-cg** que son para problemas multi-clase

La librería **sklearn** utiliza **lbfgs** por defecto

Se puede priorizar **sag** o **saga** para datasets grandes

## Sesión 3: Regresión Logística (RL)

### Tolerancia

Umbral de tolerancia (tol) para el criterio de parada (para acabar el entrenamiento)

El algoritmo RL dejará de buscar un mínimo una vez se alcanza cierta tolerancia, es decir, cuando se está suficientemente cerca del objetivo.

Por defecto,  $\text{tol} = 0.0001$  ( $1e^{-4}$ )

## Sesión 3: Regresión Logística (RL)

### Regularización

añade una penalización para evitar el sobreajuste (overfitting), es decir, para conseguir que el algoritmo generalice bien en conjuntos de test

penaliza los coeficientes altos de la función lineal (sin incluir el término independiente)

cuando una característica de las muestras solo ocurre en una clase (y en el resto de clases los valores de dicha característica son 0) el algoritmo de RL asignará coeficientes muy altos a dicha característica. En este caso, el modelo probablemente se ajustará demasiado al conjunto de entrenamiento.

$$\operatorname{argmin}_{\mathbf{W}} \text{NLL}(\mathbf{W}) + \lambda R(\mathbf{W})$$


parámetro  $\lambda \in \mathbb{R}^{>0}$

término de regularización

## Sesión 3: Regresión Logística (RL)

### Regularización

Hay varios tipos de regularización  $R(\mathbf{W})$ . La librería sklearn aplica por defecto una regularización llamada regularización L2 o Gauss.

$\lambda = \frac{1}{C}$   parámetro C que utiliza la función LogisticRegression de sklearn

- **Por defecto, C=1.0**
- Para valores muy altos de C,  $\lambda$  es muy pequeño y por tanto se aplica mínima regularización lo cual puede conllevar una posibilidad de sobre-ajuste al conjunto de entrenamiento.
- Para valores próximos a 0,  $\lambda$  es muy grande y por tanto se aplica máxima regularización lo cual puede conllevar una posibilidad de sub-ajuste al conjunto de entrenamiento.

## Sesión 3: Regresión Logística (RL)

### Early stopping

Máximo número de iteraciones que se ejecutará el *solver* de la función LogisticRegression.

La librería sklearn utiliza `max_iter=100` por defecto

## Sesión 3: Regresión Logística (RL)

### Parámetros a controlar:

```
clf = LogisticRegression(random_state=23,  
    solver=solver,  
    tol=tol,  
    C=C,  
    max_iter=max_iter).fit(X_train, y_train)
```

## Sesión 3

### Aplicar el algoritmo de Regresión Logística al dataset de iris

- para diferentes valores de solver, tolerancia, C y max\_iter
- Muestra los resultados del siguiente modo:

ejemplo  
de  
resultados  
para el  
**dataset**  
**iris**

#	Iter	solver	C	tol	Etr	Ete
#	-----					
	10	lbfgs	0.010	0.0001	20.83%	13.33%
	10	lbfgs	0.010	0.0100	20.83%	13.33%
	10	lbfgs	0.010	1.0000	68.33%	60.00%
	10	lbfgs	0.010	100.0000	68.33%	60.00%
	10	lbfgs	0.010	10000.0000	68.33%	60.00%
#	-----					
	10	lbfgs	0.100	0.0001	4.17%	0.00%
	10	lbfgs	0.100	0.0100	4.17%	0.00%
	10	lbfgs	0.100	1.0000	68.33%	60.00%
	10	lbfgs	0.100	100.0000	68.33%	60.00%
	10	lbfgs	0.100	10000.0000	68.33%	60.00%
#	-----					
	10	lbfgs	1.000	0.0001	3.33%	0.00%
	10	lbfgs	1.000	0.0100	3.33%	0.00%
	10	lbfgs	1.000	1.0000	68.33%	60.00%

- Indica cuál crees que es la mejor combinación de iteraciones, solver, C y tolerancia (escribe la respuesta en el mismo cuaderno en una caja de texto)

## Sesión 3

### Ejercicio 2 (entregar día del examen):

aplicar el algoritmo de Regresión Logística un dataset de OpenML

- data\_id=40670, dna
- Aplica el algoritmo RL a dicho dataset para diferentes valores de solver, tolerancia, C y max\_iter
- Muestra los resultados del mismo modo que para el dataset. Para facilitar el análisis, si no se quiere imprimir los resultados de todas las combinaciones, se puede mostrar solo los resultados de las combinaciones cuyos errores no sean superiores a un 5% respecto a la solución encontrada con los parámetros por defecto o bien respecto a la mejor solución encontrada hasta el momento, tanto para el error de entrenamiento como para el error de test.
- Indica cuál crees que es la mejor combinación de iteraciones, solver, C y tolerancia (escribe la respuesta en el mismo cuaderno en una caja de texto)
- **Sube el cuaderno con el output de la ejecución y la respuesta a la tarea que se creará para el examen.**



**Bloque 2**  
**Aprendizaje Automático**

**Práctica 2:**

**Aplicación de los algoritmos de Perceptrón y Regresión Logística a varios datasets**

**2024-2025**

**(Grupo C2)**