Clase4 IMA539

Eduardo Castro Thompson

March 28, 2023

1 Clase ejercicios - IMA539

1.1 Ejercicios propuestos:

1. Cargue a un DataFrame el archivo local `iris.data` ¿Qué diferencias se aprecian con el usado en clases?
2. Arme un conjunto de datos para el entrenamiento $(X, y) \mid Dim(X) = 100 \times 2$. Es importante que las muestras no estén ordenadas por clase y que no sean linealmente separables.
3. Compare los métodos `fit()` y `partial_fit()` del modelo `AdalineSGD` ¿Qué diferencia existe entre ellos?
4. Instancie los modelos `Perceptron` y `AdalineSGD` y realice un entrenamiento de ellos con el conjunto de datos generado previamente ¿Cómo son los pesos finales?. Muestre los datos y el umbral de decisión obtenido por los modelos.
5. Si se entrenaran dos modelos idénticos en un conjuntos que contengan las mismas muestras pero ordenadas de forma diferente ¿Los pesos finales obtenidos serán iguales? Mezcle el conjunto generado en 2 y compruebe.

1.2 Librerías, funciones y clases:

[1]:
```
import os

from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

def plot_decision_regions(X, y, classifier, resolution=0.02):
    markers = ('s', 'o', 'x', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
```

```
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0],
                    y=X[y == cl, 1],
                    alpha=0.8,
                    c=colors[idx],
                    marker=markers[idx],
                    label=cl,
                    edgecolor='black')
    plt.show()

class Perceptron(object):
    def __init__(self, eta=0.01, n_iter=50, random_state=1):
        self.eta = eta
        self.n_iter = n_iter
        self.random_state = random_state

    def fit(self, X, y):
        rgen = np.random.RandomState(self.random_state)
        self.w_ = rgen.normal(loc=0.0, scale=0.01, size=1 + X.shape[1])
        self.errors_ = []
        for _ in range(self.n_iter):
            errors = 0
            for xi, target in zip(X, y):
                update = self.eta * (target - self.predict(xi))
                self.w_[1:] += update * xi
                self.w_[0] += update
                errors += int(update != 0.0)
            self.errors_.append(errors)
        return self

    def net_input(self, X):
        return np.dot(X, self.w_[1:] + self.w_[0])

    def predict(self, X):
        return np.where(self.net_input(X) >= 0.0, 1, -1)

class AdalineSGD(object):
    def __init__(self, eta=0.01, n_iter=10, shuffle=True, random_state=None):
        self.eta = eta
        self.n_iter = n_iter
        self.w_initialized = False
        self.shuffle = shuffle
```

```
        self.random_state = random_state

    def fit(self, X, y):
        self._initialize_weights(X.shape[1])
        self.cost_ = []
        for i in range(self.n_iter):
            if self.shuffle:
                X, y = self._shuffle(X, y)
            cost = []
            for xi, target in zip(X, y):
                cost.append(self._update_weights(xi, target))
            avg_cost = sum(cost) / len(y)
            self.cost_.append(avg_cost)
        return self

    def partial_fit(self, X, y):
        if not self.w_initialized:
            self._initialize_weights(X.shape[1])
        if y.ravel().shape[0] > 1:
            for xi, target in zip(X, y):
                self._update_weights(xi, target)
        else:
            self._update_weights(X, y)
        return self

    def _shuffle(self, X, y):
        r = self.rgen.permutation(len(y))
        return X[r], y[r]

    def _initialize_weights(self, m):
        self.rgen = np.random.RandomState(self.random_state)
        self.w_ = self.rgen.normal(loc=0.0, scale=0.01, size=1 + m)
        self.w_initialized = True

    def _update_weights(self, xi, target):
        output = self.activation(self.net_input(xi))
        error = (target - output)
        self.w_[1:] += self.eta * xi.dot(error)
        self.w_[0] += self.eta * error
        cost = 0.5 * error**2
        return cost

    def net_input(self, X):
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def activation(self, X):
        return X
```

```
    def predict(self, X):
        return np.where(self.activation(self.net_input(X)) >= 0.5, 1, -1)
```

1.3 Desarrollo

[2]:
```
path = os.path.join(os.getcwd(), 'iris.data')
df = pd.read_csv(path, header= None)
df
```

[2]:

	0	1	2	3	4
0	5.4	3.9	1.7	0.4	Iris-setosa
1	6.3	2.7	4.9	1.8	Iris-virginica
2	6.9	3.2	5.7	2.3	Iris-virginica
3	6.9	3.1	5.4	2.1	Iris-virginica
4	4.8	3.1	1.6	0.2	Iris-setosa
..	…	…	…	…	…
145	5.0	3.5	1.3	0.3	Iris-setosa
146	7.3	2.9	6.3	1.8	Iris-virginica
147	6.5	2.8	4.6	1.5	Iris-versicolor
148	5.6	2.8	4.9	2.0	Iris-virginica
149	5.1	2.5	3.0	1.1	Iris-versicolor

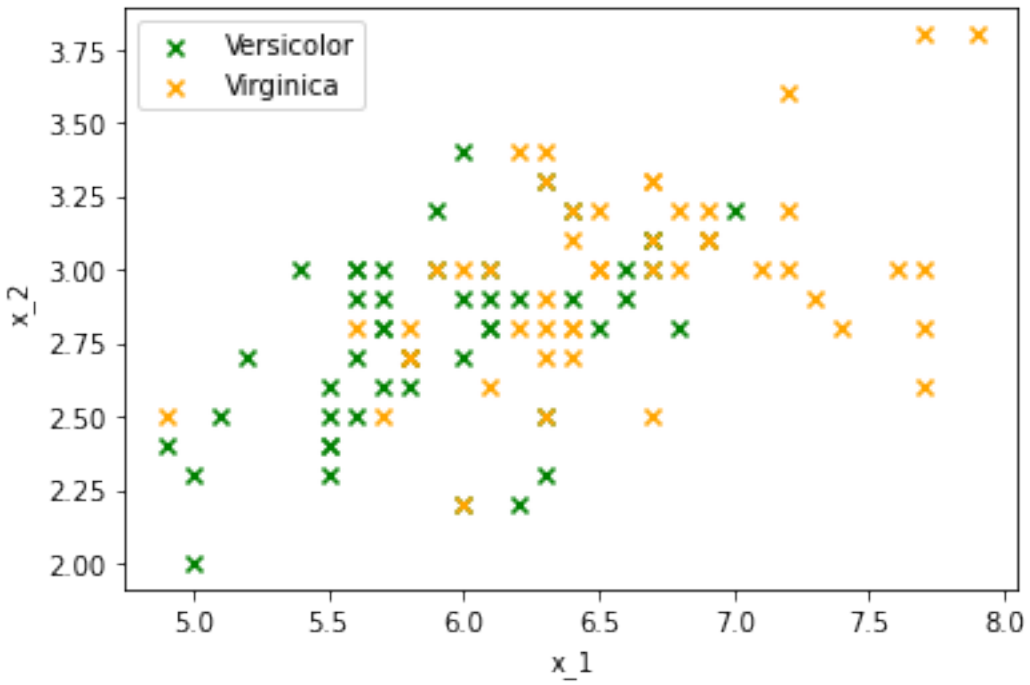
[150 rows x 5 columns]

Armado de la BD

[3]:
```
df = df[df[4] != 'Iris-setosa'].reset_index(drop= True)
```

[4]:
```
df = df.replace({'Iris-versicolor': 1, 'Iris-virginica': -1})
```

[5]:
```
# Comprobación de si son linealmente separables
idx = [0, 1]
plt.scatter(df[df[4] != -1][idx[0]], df[df[4] != -1][idx[1]],
            color= 'green', marker= 'x', label= 'Versicolor')
plt.scatter(df[df[4] == -1][idx[0]], df[df[4] == -1][idx[1]],
            color= 'orange', marker= 'x', label= 'Virginica')

plt.xlabel('x_1')
plt.ylabel('x_2')
plt.legend(loc='upper left')
plt.show()
```



[19]:
```
X, y = df.iloc[:, idx].values, df[4].values
```

Preprocesamiento y entrenamiento

[20]:
```
X_std = np.copy(X)
X_std[:, 0] = (X[:, 0] - X[:, 0].mean()) / X[:, 0].std()
X_std[:, 1] = (X[:, 1] - X[:, 1].mean()) / X[:, 1].std()
```

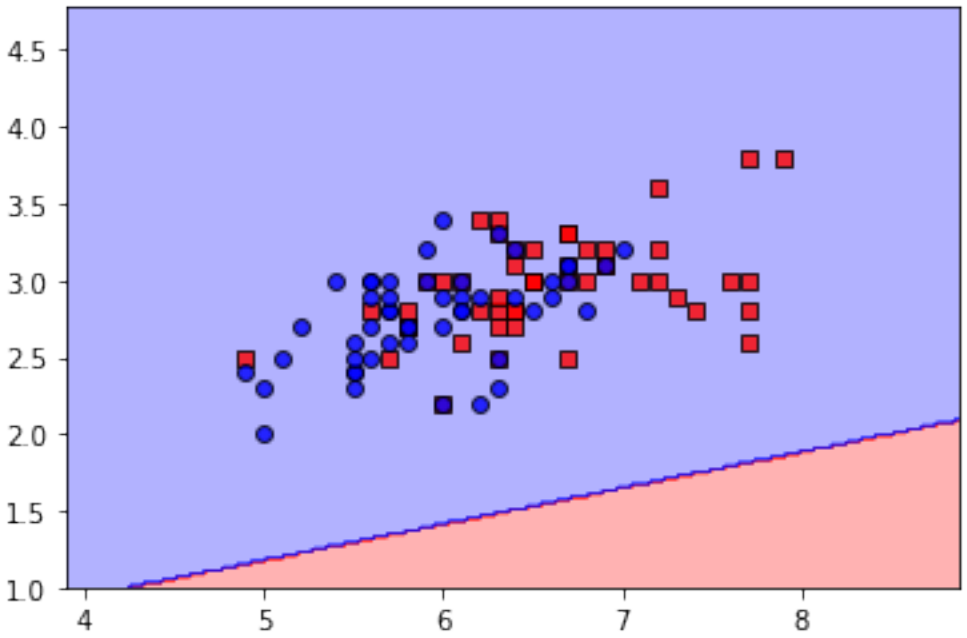
[21]:
```
percep = Perceptron(n_iter= 20, eta= 0.01, random_state= 18675)
print("Perceptron:")
percep.fit(X, y)
print(f'Pesos entrenando en set 1: {percep.w_}')

adaSGD = AdalineSGD(n_iter= 20, eta= 0.01, random_state= 18675)
print("Adaline SDG")
adaSGD.fit(X_std, y)
print(f'Pesos entrenando en set 1: {adaSGD.w_}')
```

```
Perceptron:
Pesos entrenando en set 1: [ 0.72299876 -0.83444278 -0.24927081]
Adaline SDG
Pesos entrenando en set 1: [ 0.00929995 -0.46895998 -0.04236348]
```

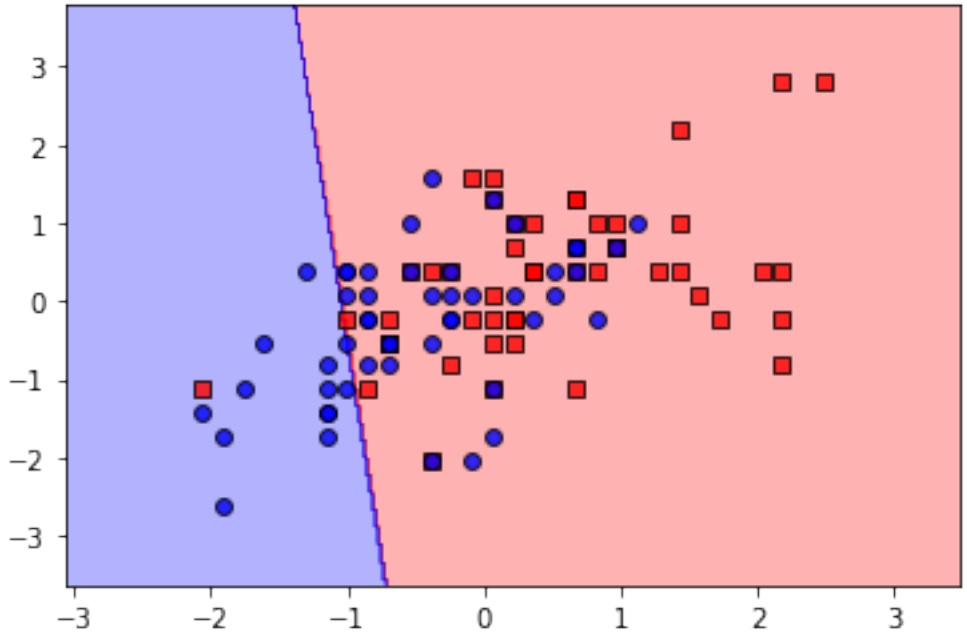
[22]:
```
print("Perceptron:")
plot_decision_regions(X, y, classifier= percep)
print("Adaline SGD:")
```

```
plot_decision_regions(X_std, y, classifier= adaSGD)
```

Perceptron:

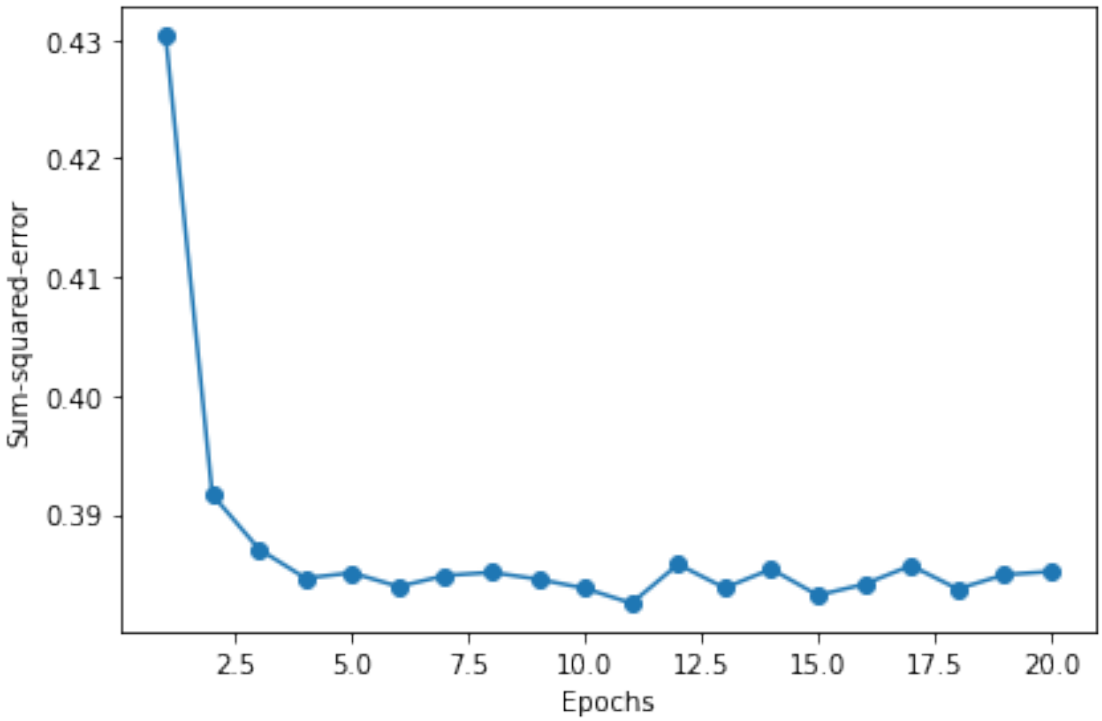


Adaline SGD:

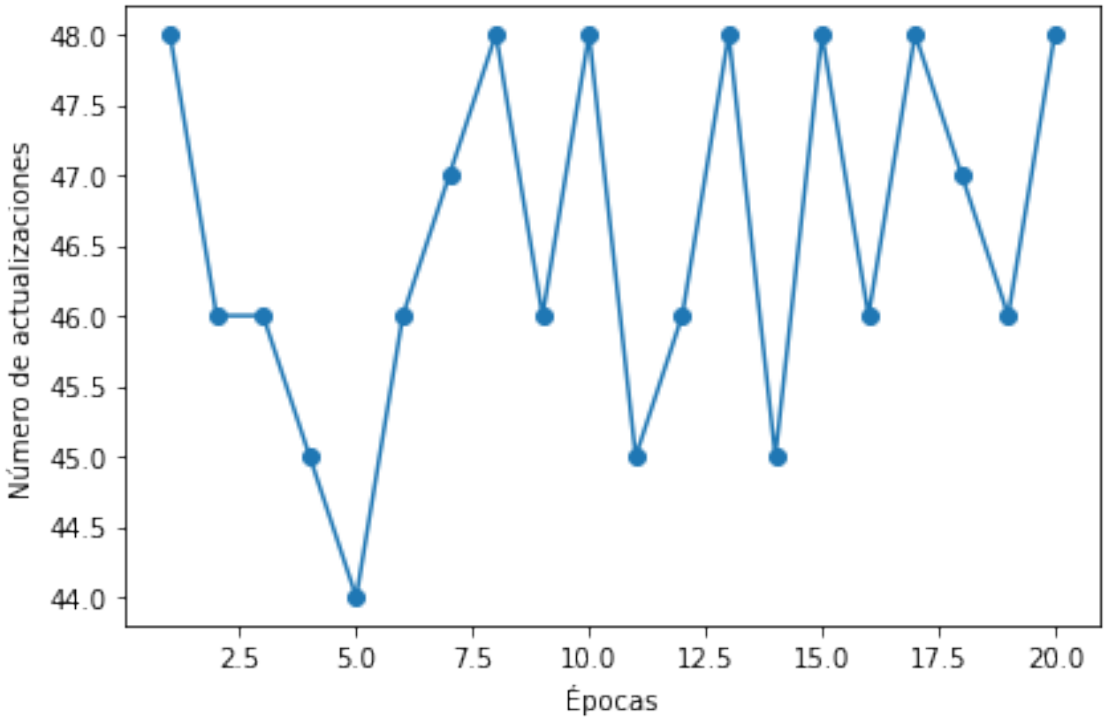


[23]:
```
print('AdalineSGD')
plt.plot(range(1, len(adaSGD.cost_) + 1), adaSGD.cost_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Sum-squared-error')
plt.tight_layout()
plt.show()

print('Perceptron')
plt.plot(range(1, len(percep.errors_) + 1), percep.errors_, marker='o')
plt.xlabel('Épocas')
plt.ylabel('Número de actualizaciones')
plt.tight_layout()
plt.show()
```

```
AdalineSGD
```




```
Perceptron
```



[24]:
```
newIndxs = np.random.permutation(y.shape[0])
newY = y[newIndxs]
newX = X[newIndxs]
```

[25]:
```
newIndxs
```

[25]:
```
array([66, 40, 61, 63, 33, 26, 84, 53, 20, 18, 62, 99, 23, 30, 86, 74, 24,
       88, 50, 80, 11, 43, 52, 77, 32,  1,  6, 28, 47, 87, 81, 93, 51, 73,
        7, 13, 29, 85,  9, 56, 91, 76, 39, 65,  2, 16, 68, 79, 17, 83, 14,
       44, 37, 25, 89,  5, 94, 96, 49, 45, 46, 42, 12, 82, 35, 19, 31, 90,
       59,  8, 22, 38, 72, 34, 58, 71, 97, 60, 95, 21, 57, 15, 75, 41, 36,
        0, 27, 92, 67, 70,  4, 55, 78, 10, 54, 69,  3, 64, 98, 48])
```

[26]:
```
adaSGD1 = AdalineSGD(n_iter= 5, eta= 0.01, random_state= 18675)
adaSGD2 = AdalineSGD(n_iter= 5, eta= 0.01, random_state= 18675)

adaSGD1.fit(X, y)
adaSGD2.fit(newX, newY)

print(f'Pesos para el modelo entrenado en el conjunto original: {adaSGD1.w_}')
print(f'Pesos para el modelo entrenado en el conjunto mezclado: {adaSGD2.w_}')
```

```
Pesos para el modelo entrenado en el conjunto original: [ 0.24382824 -0.11941268
0.11590873]
```

```
Pesos para el modelo entrenado en el conjunto mezclado: [ 0.23368668 -0.18216013
0.11585527]
```

[ ]: