Hoja informativa: Entrenamiento del descenso de gradiente

Práctica

```
# Descenso de gradiente estocástico

class SGDLinearRegression:
    def __init__(self, step_size, epochs, batch_size):
        self.step_size = step_size
        self.epochs = epochs
        self.batch_size = batch_size

    def fit(self, train_features, train_target):
        X = np.concatenate((np.ones((train_features.shape[0], 1)), train_features), axis=
1)
        y = train_target
        w = np.zeros(X.shape[1])

        for _ in range(self.epochs):
            batches_count = X.shape[0] // self.batch_size
            for i in range(batches_count):
                begin = i * self.batch_size
                end = (i + 1) * self.batch_size
                X_batch = X[begin:end, :]
                y_batch = y[begin:end]

                gradient = 2 * X_batch.T.dot(X_batch.dot(w) - y_batch) / X_batch.shape[0]

                w -= self.step_size * gradient

        self.w = w[1:]
        self.w0 = w[0]

    def predict(self, test_features):
        return test_features.dot(self.w) + self.w0
```

```
# Regresión de cresta

class RidgeRegression:
    def __init__(self, step_size, epochs, batch_size, reg_weight):
        self.step_size = step_size
        self.epochs = epochs
```

```
        self.batch_size = batch_size
        self.reg_weight = reg_weight

    def fit(self, train_features, train_target):
        X = np.concatenate((np.ones((train_features.shape[0], 1)), train_features), axis=
1)
        y = train_target
        w = np.zeros(X.shape[1])

        for _ in range(self.epochs):
            batches_count = X.shape[0] // self.batch_size
            for i in range(batches_count):
                begin = i * self.batch_size
                end = (i + 1) * self.batch_size
                X_batch = X[begin:end, :]
                y_batch = y[begin:end]

                gradient = 2 * X_batch.T.dot(X_batch.dot(w) - y_batch) / X_batch.shape[0]
                reg = 2 * w.copy()
                reg[0] = 0
                gradient += self.reg_weight * reg

                w -= self.step_size * gradient

        self.w = w[1:]
        self.w0 = w[0]

    def predict(self, test_features):
        return test_features.dot(self.w) + self.w0
```

Teoría

Un **lote** es una pequeña parte del conjunto de entrenamiento

Se llama al **Inicializador de clase** cuando se crea una instancia de una clase.

Regularización es un método que agrega limitaciones adicionales a las condiciones para reducir el sobreajuste

Una **red neuronal** es un modelo que consta de muchos modelos simples