




22 DE FEBRERO DE 2024

EJERCICIO 4
PRACTICA 1

CRUZ GALLEGOS RAMSÉS AARÓN
MORALES ZEPEDA IVAN YUTLANIH
SEM. DE SOL. DE PROBLEMAS DE INTELIGENCIA ARTIFICIAL II
SECCIÓN: D05



Contenido

| | |
|---------------------|---|
| Introducción | 2 |
| Desarrollo | 2 |
| Resultados..... | 5 |
| Conclusiones: | 6 |

Introducción

Leave-k-Out (LKO) y el Leave-One-Out (LOO) son técnicas de validación cruzada utilizadas para evaluar el rendimiento de un modelo. Leave-k-Out (LKO) divide los datos en k subconjuntos iguales para evaluar el modelo en múltiples iteraciones, mientras que Leave-One-Out (LOO) es una variante donde k es igual al número total de muestras. Ambas técnicas son muy útiles para evaluar el rendimiento de un MLP entrenado con retropropagación.

Desarrollo

El código implementa un MLP utilizando la biblioteca NumPy en Python. El MLP se entrena utilizando el algoritmo de retropropagación con las técnicas de validación cruzada LKO y LOO para evaluar su rendimiento.

Inicialización del MLP: Se define una clase MLP que inicializa los pesos y sesgos de la red neuronal de acuerdo con la arquitectura especificada.

```
class MLP:
    def __init__(self, layers):
        self.layers = layers
        self.weights = [np.random.randn(layers[i], layers[i + 1]) for i
in range(len(layers) - 1)]
        self.bias = [np.zeros((1, layers[i + 1])) for i in
range(len(layers) - 1)]
```

Propagación hacia adelante: Se implementa el método forward que realiza la propagación hacia adelante para calcular las activaciones de cada capa de la red neuronal.

```
def forward(self, X):
    self.activations = [X]
    self.z_values = []

    for i in range(len(self.layers) - 1):
        z = np.dot(self.activations[-1], self.weights[i]) +
self.bias[i]
        a = self.softmax(z) if i == len(self.layers) - 2 else
self.activation(z)
```

```
self.z_values.append(z)
self.activations.append(a)
```

Retropropagación: Se implementa el método `backward` que realiza la retropropagación para ajustar los pesos y sesgos de la red neuronal utilizando el algoritmo de descenso de gradiente.

```
def backward(self, X, y, learning_rate):
    errors = [y - self.activations[-1]]
    deltas = [errors[-1]]

    for i in range(len(self.layers) - 2, 0, -1):
        error = deltas[-1].dot(self.weights[i].T)
        delta = error *
self.activation_derivative(self.activations[i])
        errors.append(error)
        deltas.append(delta)

    for i in range(len(self.layers) - 2, -1, -1):
        self.weights[i] += np.dot(self.activations[i].T,
deltas[len(self.layers) - 2 - i]) * learning_rate
        self.bias[i] += np.sum(deltas[len(self.layers) - 2 - i],
axis=0, keepdims=True) * learning_rate
```

Entrenamiento del MLP: Se implementa el método `train` que entrena el MLP en un conjunto de datos de entrada durante un número especificado de épocas.

```
def train(self, X, y, epochs, learning_rate):
    for _ in range(epochs):
        self.forward(X)
        self.backward(X, y, learning_rate)
```

Predicción del MLP: Se implementa el método `predict` que realiza predicciones utilizando el MLP entrenado.

```
def predict(self, X):
    self.forward(X)
    return self.activations[-1]
```

Evaluación con LKO y LOO: Se implementan los métodos `evaluate_lko` y `evaluate_loo` para evaluar el rendimiento del MLP utilizando las técnicas de validación cruzada LKO y LOO, respectivamente.

```
def evaluate_lko(self, X, y, k):
    lko = KFold(n_splits=k)
    accuracies = []

    for train_index, test_index in lko.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        self.train(X_train, y_train, epochs=1000,
learning_rate=0.2)
        y_pred_onehot = self.predict(X_test)
        y_pred = np.argmax(y_pred_onehot, axis=1)
        y_true = np.argmax(y_test, axis=1)
        accuracies.append(accuracy_score(y_true, y_pred))
    return np.mean(accuracies), np.std(accuracies)
```

```
def evaluate_loo(self, X, y):
    loo = LeaveOneOut()
    accuracies = []

    for train_index, test_index in loo.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        self.train(X_train, y_train, epochs=1000,
learning_rate=0.2)
        y_pred_onehot = self.predict(X_test)
        y_pred = np.argmax(y_pred_onehot, axis=1)
        y_true = np.argmax(y_test, axis=1)
        accuracies.append(accuracy_score(y_true, y_pred))
    return np.mean(accuracies), np.std(accuracies)
```

Resultados

Los resultados obtenidos del entrenamiento y evaluación utilizando las técnicas de validación cruzada LKO y LOO fueron:

Leave-k-Out (LKO):

Error Esperado: 0.2466666666666667

Precisión Promedio: 0.7533333333333333

Desviación Estándar: 0.06182412330330473

Leave-One-Out (LOO):

Error Esperado: 0.30666666666666664

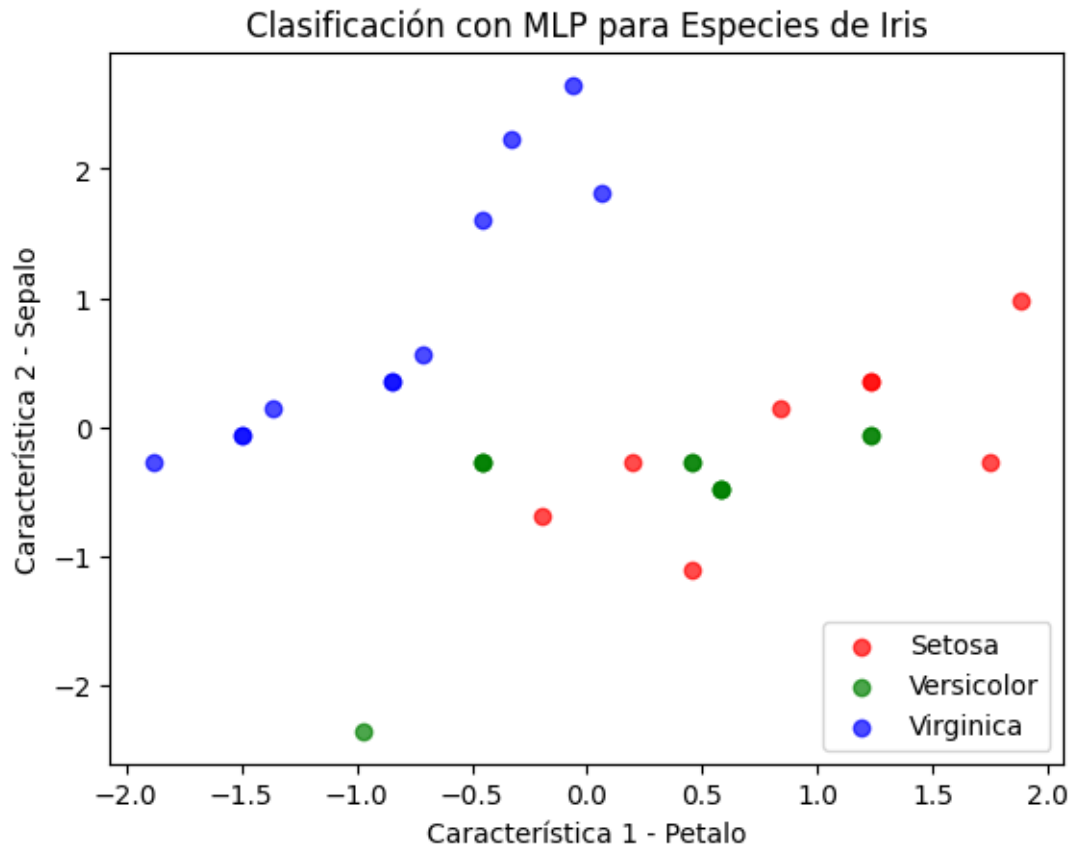
Precisión Promedio: 0.6933333333333334

Desviación Estándar: 0.4611097724210822

Las predicciones del modelo para las especies de iris en el conjunto de datos de prueba son las siguientes:

- 1: Predicción=Virginica, Especie real=Versicolor
- 2: Predicción=Setosa, Especie real=Setosa
- 3: Predicción=Setosa, Especie real=Setosa
- 4: Predicción=Setosa, Especie real=Setosa
- 5: Predicción=Setosa, Especie real=Setosa
- 6: Predicción=Virginica, Especie real=Versicolor
- 7: Predicción=Virginica, Especie real=Virginica
- 8: Predicción=Virginica, Especie real=Versicolor
- 9: Predicción=Setosa, Especie real=Setosa
- 10: Predicción=Setosa, Especie real=Setosa
- 11: Predicción=Setosa, Especie real=Setosa
- 12: Predicción=Virginica, Especie real=Versicolor
- 13: Predicción=Setosa, Especie real=Versicolor
- 14: Predicción=Virginica, Especie real=Virginica
- 15: Predicción=Virginica, Especie real=Virginica
- 16: Predicción=Virginica, Especie real=Versicolor
- 17: Predicción=Virginica, Especie real=Versicolor
- 18: Predicción=Virginica, Especie real=Virginica
- 19: Predicción=Setosa, Especie real=Setosa
- 20: Predicción=Setosa, Especie real=Setosa
- 21: Predicción=Virginica, Especie real=Virginica
- 22: Predicción=Virginica, Especie real=Virginica
- 23: Predicción=Virginica, Especie real=Versicolor
- 24: Predicción=Setosa, Especie real=Setosa
- 25: Predicción=Virginica, Especie real=Versicolor
- 26: Predicción=Virginica, Especie real=Versicolor
- 27: Predicción=Virginica, Especie real=Versicolor

28: Predicción=Setosa, Especie real=Setosa
29: Predicción=Virginica, Especie real=Virginica
30: Predicción=Virginica, Especie real=Virginica



Existe una ligera diferencia en el error esperado y la precisión promedio entre LKO y LOO, ambos mostraron un rendimiento satisfactorio en la clasificación. Las desviaciones estándar muestran cierta variabilidad en el rendimiento del modelo entre las diferentes iteraciones de validación cruzada. Sin embargo, en general, el MLP logra clasificar correctamente las especies de iris en el conjunto de datos de prueba.

Conclusiones:

La implementación del MLP utilizando las validaciones LKO y LOO obtuvieron un buen resultado al clasificar las especies de iris del conjunto de datos "irisbin.csv". A pesar de que los resultados muestran cierta variabilidad, el modelo MLP junto con las técnicas de validación cruzada, demostraron ser útiles para esta clasificación de especies de iris en el conjunto de datos.