INSTITUTO POLITÉCNICO NACIONAL

ESCOM

INTELIGENCIA ARTIFICIAL

LAB 11: REDES NEURONALES

MARTÍNEZ CHÁVEZ JORGE ALEXIS

6CV3

28 NOVIEMBRE 2024

Introducción

En el campo del aprendizaje automático, las redes neuronales artificiales han emergido como una de las herramientas más poderosas para abordar una amplia variedad de tareas de clasificación y regresión. Entre las variantes más conocidas de redes neuronales se encuentran los perceptrones multicapa e utilizan varias capas ocultas para modelar relaciones no lineales entre las características de entrada y las etiquetas de salida por otro lado tenemos el RBF el cual no utiliza las multicapas si no que buscan un heperplano en donde hacer una separación para que se puedan maximizar el margen entre las clase,

Desarrollo

En la primera parte vamos a hacer el clasificador de perceptrón multicapa, lo que primero debemos de hacer es poder cargar el data set, en donde debemos de cargar el documento y separar x y, en donde x son las columnas características del dataset, y es la ultima columna que es la etiqueta que tiene.

```
# Cargar dataset desde CSV
def cargar_dataset(file_path):
    # Cargar el CSV usando pandas
    data = pd.read_csv(file_path)

    # Diagnóstico: revisar las primeras filas
    print("Primeras filas del dataset:")
    print(data.head())

    # Verificar valores nulos
    print("Valores nulos en el dataset:")
    print(data.isnull().sum())

    # Asumir que la última columna es la etiqueta (y) y el resto son las características (X)
    X = data.iloc[:, :-1].values  # todas las columnas excepto la última
    y = data.iloc[:, -1].values   # solo la última columna

    # Si las etiquetas son categóricas, las convertimos en numéricas
    if data.iloc[:, -1].dtype == 'object':  # Si la última columna es categórica
        le = LabelEncoder()
        y = le.fit_transform(y)

    return X, y
```

Ahora debemos de ayudar al procesamiento del machine learning, en donde debemos de hacer un procesamiento para hacer que todos los datos tengas las mismas caracteriscas en donde deben de tener una media de 0 y una deviación estandar de 1, debemos de hacer que todos los datos cumplan esta escala.

```
# Preprocesamiento
def preprocesamiento(X):
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    return X_scaled
```

Ahora debemos de crear el perceptron en donde tiene una capa de 10 neuronas, le damos un maximo de iteraciones de 10000, usamos el algoritmo de Adam para poder optimizar el proceso y hacemos una semilla aleatoria para producir los resultados.

```python
# Perceptrón
def perceptron_model():
    model = MLPClassifier(hidden_layer_sizes=(10,), max_iter=10000, solver='adam', random_state=42)
    return model
```

Ahora creamos los metodos de validación, en hold out dividimos un 70 para el entrenamiento y un 30 para las pruebas, entrenamos el modelo posteriormente realizamos las predicciones y hacemos el calculo de la matriz de consusión y el acurracy.

```python
# Hold-out (70/30)
def validacion_holdout(X, y, model, test_size=0.3):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)
    return accuracy, cm
```

Para cross validation ahora debemos de dividir en 5 casos y realizamos una validación cruzada en cada una de las particiones y volvemos a calcular el acurracy y la matriz de confusión.

```python
# Cross-Validation (K-Fold)
def validacion_kfold(X, y, model, k=5):
    kf = KFold(n_splits=k, shuffle=True, random_state=42)
    scores = cross_val_score(model, X, y, cv=kf)
    accuracy = np.mean(scores)

    # Calcular matriz de confusión para el K-Fold
    model.fit(X, y)  # Entrenamos el modelo con todo el dataset
    y_pred = model.predict(X)
    cm = confusion_matrix(y, y_pred)

    return accuracy, cm
```

Ahora para leave one out, dividimos en subconjuntos y despues cada uno de estos subconjuntos son los que se deben de evaluar, de forma que cada uno de los datos los vamos a ocupar como forma de validación y nuevamente calculamos acurracy y matriz

de consuión.

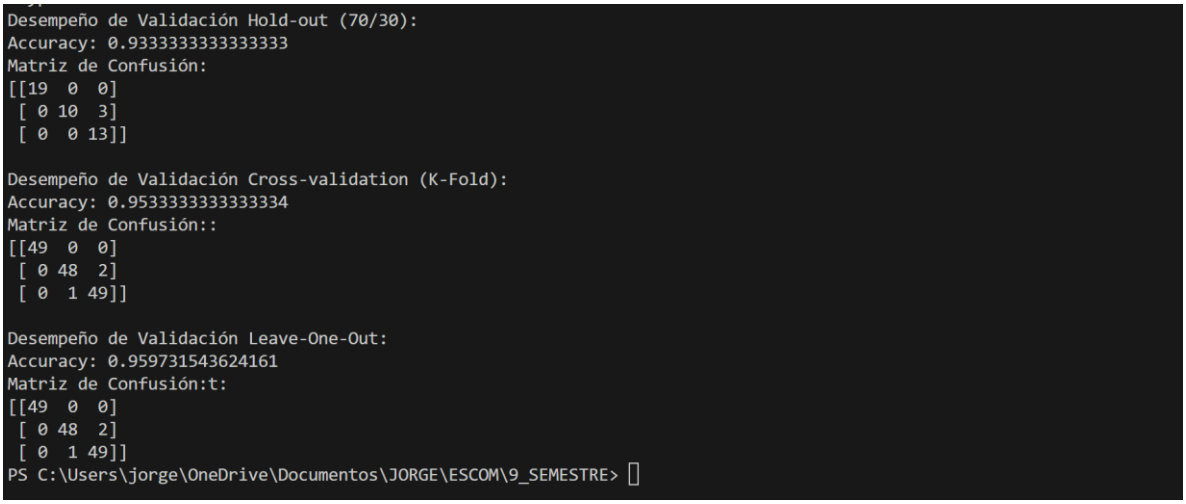
```python
# Leave-One-Out
def validacion_leave_one_out(X, y, model):
    loo = LeaveOneOut()
    scores = cross_val_score(model, X, y, cv=loo)
    accuracy = np.mean(scores)

    # Calcular matriz de confusión para Leave-One-Out
    model.fit(X, y)  # Entrenamos el modelo con todo el dataset
    y_pred = model.predict(X)
    cm = confusion_matrix(y, y_pred)

    return accuracy, cm
```

Por ultmo cargamos la ruta en donde se encuentra el dataset que vamos a analizar y por ultimo imprimos los resultados obtenidos.

```python
# Ejecutar el código
if __name__ == "__main__":
    # Cargar dataset desde el archivo CSV
    file_path = "C:/Users/jorge/OneDrive/Documentos/JORGE/ESCOM/9_SEMESTRE/IA/LABORATORIO/10_KNN_BAY
    X, y = cargar_dataset(file_path)
    X_scaled = preprocesamiento(X)

    # Crear el modelo de perceptrón
    model = perceptron_model()

    # Validación Hold-out (70/30)
    print("Desempeño de Validación Hold-out (70/30):")
    accuracy_holdout, cm_holdout = validacion_holdout(X_scaled, y, model)
    print(f"Accuracy: {accuracy_holdout}")
    print("Matriz de Confusión:")
    print(cm_holdout)

    # Validación Cross-validation (K-Fold)
    print("\nDesempeño de Validación Cross-validation (K-Fold):")
    accuracy_kfold, cm_kfold = validacion_kfold(X_scaled, y, model)
    print(f"Accuracy: {accuracy_kfold}")
    print("Matriz de Confusión::")
    print(cm_kfold)

    # Validación Leave-One-Out
    print("\nDesempeño de Validación Leave-One-Out:")
    accuracy_loo, cm_loo = validacion_leave_one_out(X_scaled, y, model)
    print(f"Accuracy: {accuracy_loo}")
    print("Matriz de Confusión:t:")
    print(cm_loo)
```

Obtendremos el siguiente resultado, en donde hold out nos dice que para la clase 1 tuvimos un accuracy de 93% en donde en la clase 1 tuvimos 19 instancias en donde hubo 0 errores, en la clase 2 tuvimos 10 instancias en donde hubo 3 errores de

confusión con la clase 3, y por ultimo en la clase 3 tuvimos 13 instancias en donde hubo 0 errores. Para cross validation tuvimos un accuracy de 95% en donde en clase 1 tuvimos 45 instancias con 0 errores, en la clase 2 tuvimos 48 instancias con 2 errores de confusión con la clase 3, por ultimo en la clase 3 tuvimos 49 instancias con 1 error de confusión con la clase2. Para leave one out en la clase 1 tuvimos 49 instancias con 0 errores, para la clase 2 48 instancias con 2 errores de confusión con la clase 3, para la clase 3 tuvimos 49 instancias con 1 error de confusión con la clase 2, lo que nos quiere decir que se tuvo un accuracy del 95%.



Para esta segunda ocupamos la mayor parte del código anterior, lo único que vamos a modificar es la parte del clasificador, para ellos agregamos el código correspondiente a la red neuronal RBF, en donde se crea y retorna un clasificador SVM con kernel RBF en donde se utilizara un kernel de base radial y se le asigna el parámetro random_state=42 en donde nos aseguramos de que los resultados sean reproducibles.

```
# Clasificador RBF (SVM con kernel RBF)
def rbf_model():
    model = SVC(kernel='rbf', random_state=42)  # Usamos el kernel RBF
    return model
```

Los resultados que obtenemos son: Para Hold Out tenemos un accuracy del 93% en donde la clase 1 tuvo 19 instancia con 0 errores, la clase 2 tuvo 11 instancias con 2 errores de confusión con la clase 3, la clase 3 tuvo 12 instancias y tuvo 1 error con de confusión con la clase 2. Para Cross Validation se tuvo un accuracy del 96% en donde la clase 1 tuvo 49 instancias con 0 errores, en la clase 2 se tuvieron 48 instancias con 2 errores de confusión con la clase 2, en la clase 3 se tuvieron 48 instancias con 2 errores de confusión con la clase 2. Para Leave One Out se tuvo un accuracy del 96%, en donde en la clase 1 se tuvo 49 instancias con 0 errores, en la clase 2 se tuvo 48 instancias con 2 errores de confusión con la clase 3 y en la clase 3 se tuvo de la misma forma 48 instancias pero con 2 errores de confusión con la clase 2.

```
Desempeño de Validación Hold-out (70/30):
Accuracy: 0.9333333333333333
Matriz de Confusión:
[[19  0  0]
 [ 0 11  2]
 [ 0  1 12]]

Desempeño de Validación Cross-validation (K-Fold):
Accuracy: 0.9666666666666668
Matriz de Confusión:
[[49  0  0]
 [ 0 48  2]
 [ 0  2 48]]

Desempeño de Validación Leave-One-Out:
Accuracy: 0.9664429530201343
Matriz de Confusión:
[[49  0  0]
 [ 0 48  2]
 [ 0  2 48]]
PS C:\Users\jorge\OneDrive\Documentos\JORGE\ESCOM\9_SEMESTRE>
```

Enlace

Se agrega en enlace correspondiente a este práctica en donde se encuentra en la carpeta “11_REDES_NEURONALES”
https://github.com/Jorge300403/IA_6CV3_MartinezChavez/tree/main/LAB_11_REDES_NEURONALES