Implementacion K-Folds

Plata Salinas Eidan Owen

November 8, 2023

Explicacion

A continuación, se describe el código Python que implementa k-folds. De modo que va comparando mejor modelo.

Funcion `CargarDatos`

```

def cargar_datos(ruta_csv):
  data = pd.read_csv(ruta_csv, header=None)
  X = data.iloc[:, :-1].values
  y = data.iloc[:, -1].values
  return X, y
```

Esta función se encarga de leer un archivo CSV sin una fila de encabezado (asumiendo que la primera fila ya contiene datos). Luego, separa los datos en características (X) y etiquetas (y) y los devuelve.

Funcion `kFoldCrossValidation`

```

def k_fold_cross_validation(X, y, k, modelo):
  fold_size = len(X) // k
  indices = np.arange(len(X))
  np.random.shuffle(indices)
  scores = []

  for fold in range(k):
  test_indices = indices[fold * fold_size : (fold + 1) * fold_size]
  train_indices = np.setdiff1d(indices, test_indices)


  X_train, X_test = X[train_indices], X[test_indices]
  y_train, y_test = y[train_indices], y[test_indices]


  modelo_clonado = clone(modelo)
  modelo_clonado.fit(X_train, y_train)


  y_pred = modelo_clonado.predict(X_test)
  score = accuracy_score(y_test, y_pred)
```

```
23    scores.append(score)
24
25    return scores
```

Esta función implementa la validación cruzada k-fold manualmente. Divide los datos en k subconjuntos y realiza k iteraciones de entrenamiento y validación, utilizando un subconjunto diferente como conjunto de prueba en cada iteración y el resto como conjunto de entrenamiento.

Dentro del bucle para cada fold, se clona el modelo proporcionado para evitar la contaminación entre los pliegues y se entrena y evalúa el modelo clonado. Después de las predicciones, se calcula la precisión y se guarda en una lista de scores.

Función `evaluarConKfoldManual`

```
1
2  def evaluar_con_kfold_manual(X, y, k=5):
3    modelos = {
4      'KNN': Pipeline([('scaler', StandardScaler()), ('classifier',
       KNeighborsClassifier(n_neighbors=5))]),
5      'Regresion Logistica': Pipeline([('scaler', StandardScaler()), ('
       classifier', LogisticRegression(random_state=42))]),
6      'SVM': Pipeline([('scaler', StandardScaler()), ('classifier', SVC(
       kernel='linear', random_state=42))]),
7      'Bayesiano': Pipeline([('scaler', StandardScaler()), ('classifier',
        GaussianNB())])
8    }
9
10   resultados = dict()
11
12   for nombre, modelo in modelos.items():
13   scores = k_fold_cross_validation(X, y, k, modelo)
14   resultados[nombre] = scores
15   print(f"{nombre}: {np.mean(scores)} (+/- {np.std(scores)})")
16   mejor_modelo = max(resultados, key=lambda nombre: np.mean(resultados[
       nombre]))
17   print(f"\nEl mejor modelo es {mejor_modelo} con una precision de {np.
       mean(resultados[mejor_modelo])}.")
```

Esta función crea un diccionario de modelos de aprendizaje automático, cada uno envuelto en un Pipeline que primero escala los datos y luego aplica el clasificador. Para cada modelo, utiliza la función k_fold_cross_validation para evaluar su rendimiento utilizando la validación cruzada k-fold y luego imprime el rendimiento promedio y la desviación estándar de las puntuaciones de precisión.

Función `main`

```
1
2 def main(ruta_csv):
3   X, y = cargar_datos(ruta_csv)
4   evaluar_con_kfold_manual(X, y)
```

La función main es el punto de entrada para ejecutar el proceso de evaluación de modelos cuando se ejecuta el script como un programa principal. Carga los datos y llama a la función evaluar_con_kfold_manual.

Conclusión

A través de la función k_fold_cross_validation, he desarrollado en cómo la técnica k-fold divide los datos en partes iguales para entrenar y probar modelos de forma más robusta y confiable. Este enfoque me ha permitido entender mejor la importancia de evaluar el rendimiento del modelo más allá de una simple partición de entrenamiento y prueba, reduciendo la variabilidad y proporcionando una estimación más precisa de cómo el modelo podría desempeñarse en datos no vistos.

Al trabajar con diferentes modelos de clasificación dentro de un pipeline de scikit-learn, incluyendo KNN, regresión logística, SVM y Naive Bayes, he visto cómo el preprocesamiento de datos, como la estandarización, es crucial para algunos algoritmos, especialmente aquellos que son sensibles a la escala de las características. Además, la modularidad y la facilidad de uso de scikit-learn me permitieron intercambiar y evaluar modelos de forma sencilla y directa.

Construir la función evaluar_con_kfold_manual, fue ùtil no solo pude comparar los modelos en términos de precisión media, sino que también pude apreciar las diferencias en la variabilidad de su rendimiento a través de la desviación estándar de las precisiones. Esto me ha dado una visión más completa para determinar cuál es el modelo más adecuado.

A continuacion los resultados.

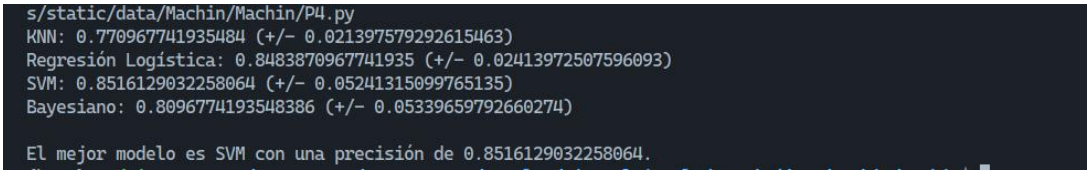

Figure 1: Puntajes del dataset

Codigo completo

```
1
2
3  import pandas as pd
4  import numpy as np
5  from sklearn.preprocessing import StandardScaler
6  from sklearn.neighbors import KNeighborsClassifier
7  from sklearn.linear_model import LogisticRegression
8  from sklearn.svm import SVC
9  from sklearn.naive_bayes import GaussianNB
10 from sklearn.metrics import accuracy_score
11 from sklearn.base import clone
12 from sklearn.pipeline import Pipeline
13
14 def cargar_datos(ruta_csv):
15   data = pd.read_csv(ruta_csv, header=None)
16   X = data.iloc[:, :-1].values
17   y = data.iloc[:, -1].values
18   return X, y
19
20 def k_fold_cross_validation(X, y, k, modelo):
21   fold_size = len(X) // k
22   indices = np.arange(len(X))
23   np.random.shuffle(indices)
24   scores = []
25
26   for fold in range(k):
27   test_indices = indices[fold * fold_size : (fold + 1) * fold_size]
28   train_indices = np.setdiff1d(indices, test_indices)
29
30
31   X_train, X_test = X[train_indices], X[test_indices]
32   y_train, y_test = y[train_indices], y[test_indices]
33
34
35   modelo_clonado = clone(modelo)
36   modelo_clonado.fit(X_train, y_train)
37
38
39   y_pred = modelo_clonado.predict(X_test)
40   score = accuracy_score(y_test, y_pred)
41   scores.append(score)
42
43   return scores
44
45 def evaluar_con_kfold_manual(X, y, k=5):
46   modelos = {
47     'KNN': Pipeline([('scaler', StandardScaler()), ('classifier',
     KNeighborsClassifier(n_neighbors=5))]),
48     'Regresion Logistica': Pipeline([('scaler', StandardScaler()), ('
     classifier', LogisticRegression(random_state=42))]),
49     'SVM': Pipeline([('scaler', StandardScaler()), ('classifier', SVC(
     kernel='linear', random_state=42))]),
50     'Bayesiano': Pipeline([('scaler', StandardScaler()), ('classifier',
      GaussianNB())])
51   }
```

```
52
53    resultados = dict()
54
55    for nombre, modelo in modelos.items():
56    scores = k_fold_cross_validation(X, y, k, modelo)
57    resultados[nombre] = scores
58    print(f"{nombre}: {np.mean(scores)} (+/- {np.std(scores)})")
59    mejor_modelo = max(resultados, key=lambda nombre: np.mean(resultados[
       nombre]))
60    print(f"\nEl mejor modelo es {mejor_modelo} con una precision de {np.
       mean(resultados[mejor_modelo])}.")
61
62 def main(ruta_csv):
63    X, y = cargar_datos(ruta_csv)
64    evaluar_con_kfold_manual(X, y)
65
66 if __name__ == "__main__":
67    main("dataset.csv")
```