Instituto Politecnico Nacional

Escuela Superior de Computo

Ingeniería en Sistemas Computacionales

Sexto semestre

Grupo: 6CV3

Inteligencia Artificial

Practica 10: Clasificadores Naive Bayes y KNN

Presentado por:

Flores Vicencio Marco Isaí

Pérez Gonzalez Jesús

Reyes Núñez Sebastián

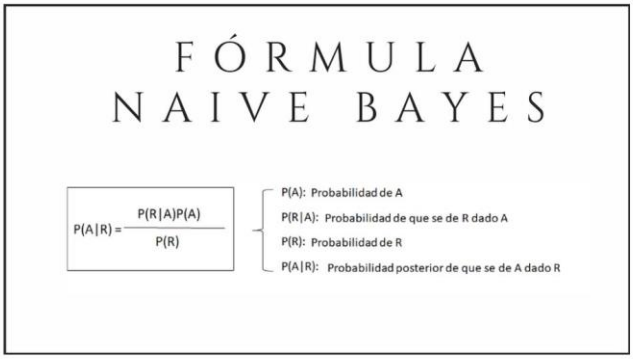
Fecha: 22/11/2024

Indice

Introducción

1. Clasificador Naive Bayes

Naive Bayes es un modelo probabilístico basado en el Teorema de Bayes, el cual se usa para clasificación. El modelo asume que las características (atributos) son independientes entre sí, lo que simplifica enormemente los cálculos, ya que no es necesario modelar las interacciones entre las variables. A pesar de esta suposición simplificadora, el clasificador Naive Bayes ha mostrado ser muy efectivo en una amplia gama de problemas de clasificación, especialmente en problemas con muchas características, como la clasificación de texto (spam, por ejemplo) y otros tipos de datos tabulares. El desempeño del modelo depende de la calidad de las características y la adecuación de la suposición de independencia condicional.



2. K-Nearest Neighbors (KNN)

KNN es un clasificador basado en la idea de que las muestras similares estarán cerca en el espacio de características. Para clasificar una nueva muestra, el algoritmo busca los "K" vecinos más cercanos en el conjunto de entrenamiento y asigna la clase más frecuente entre esos vecinos. El valor de K es un parámetro clave que influye en el rendimiento del modelo. Si K es muy pequeño, el modelo puede ser susceptible a ruido, mientras que un K grande puede hacer que el modelo pierda sensibilidad a las diferencias finas entre clases.

3. Métodos de Validación

- *Hold-Out: Consiste en dividir el dataset en dos subconjuntos: un conjunto de entrenamiento (70%) y uno de prueba (30%). Esta es la técnica de validación más simple, pero puede ser susceptible a la variabilidad dependiendo de cómo se realiza la división de los datos.*

- *10-Fold Cross-Validation: En este método, el dataset se divide en 10 subconjuntos (o "folds"). El modelo se entrena usando 9 de esos subconjuntos y se prueba con el subconjunto restante. Este proceso se repite 10 veces, cada vez con un subconjunto diferente como conjunto de prueba. Finalmente, se promedia el rendimiento del modelo. Esta técnica es más robusta que el Hold-Out porque utiliza todo el conjunto de datos tanto para entrenamiento como para prueba.*

- *Leave-One-Out Cross-Validation (LOO-CV): Es un caso extremo de K-fold cross-validation en el que el número de "folds" es igual al número de muestras del dataset. Es decir, en cada iteración se entrena el modelo con todos los datos excepto uno, y se prueba con esa muestra. Aunque es más exhaustiva, esta técnica puede ser muy costosa en términos de tiempo computacional si el conjunto de datos es grande.*

4. Métricas de Evaluación

- *Accuracy (Exactitud): Es la proporción de predicciones correctas entre el total de predicciones. Es útil cuando las clases están balanceadas, pero puede no ser indicativa del desempeño real si las clases están desbalanceadas.*

- *Matriz de Confusión: Es una herramienta que permite visualizar el desempeño del clasificador en términos de falsos positivos, falsos negativos, verdaderos positivos y verdaderos negativos. De ella se derivan otras métricas como la precisión, recall, y F1-score, que pueden ser más informativas cuando se trata de clases desbalanceadas.*

Desarrollo

Para el desarrollo de esta práctica, se utilizaron 3 datasets, Iris, Seeds y Pima-Indians Diabetes, en los que se implementara los clasificadores KNN y Naive-Bayes y con cada uno se usaran los métodos de validación Hold Out 70/30, 10-Fold Cross-Validation y Leave-One-Out para verificar el Accuracy y la matriz de confusión para cada caso. Por lo que empezaremos por las bibliotecas a utilizar.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, KFold, LeaveOneOut
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
```

Con estas bibliotecas podemos hacer uso de implementaciones de KNN, Naive Bayes, etc, ya implementadas.

Para leer el dataset, usamos una función que valida que nombre fue el solicitado para leer el archivo desde la carpeta de Datasets, y lee el csv para almacenarlo y posteriormente ser procesado:

```
def load_dataset(dataset_name):
    if dataset_name == "iris":
        file_path = "Datasets/iris.data"
        data = pd.read_csv(file_path, header=None)
        X = data.iloc[:, :-1]
        y = data.iloc[:, -1]
        # Convertir etiquetas categóricas a numéricas
        y = y.astype('category').cat.codes
    elif dataset_name == "pima-diabetes":

        file_path = "Datasets/pima-indians-diabetes.csv"
        data = pd.read_csv(file_path, header=None)
        X = data.iloc[:, :-1]
        y = data.iloc[:, -1]
    elif dataset_name == "seeds":

        file_path = "Datasets/seeds_dataset.txt"
        data = pd.read_csv(file_path, sep="\t", header=None)
        X = data.iloc[:, :-1]
        y = data.iloc[:, -1] - 1
    else:
        raise ValueError("Dataset desconocido")
    return X.to_numpy(), y.to_numpy()
```

Utilizamos el clasificador Naive Bayes basado en la distribución Gaussiana (supone que los datos siguen una distribución normal). Por lo que creamos un modelo (GaussianNB). Y entrena el modelo con los datos de entrenamiento (X_train, y_train), finalmente predice las etiquetas de los datos de prueba (X_test) y devuelve las predicciones.

```
# Clasificador Naive Bayes
def naive_bayes(X_train, X_test, y_train, y_test):
    model = GaussianNB()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return y_pred
```

Para el clasificador KNN utilizamos el algoritmo K-Nearest Neighbors, que clasifica un dato según los 3 vecinos más cercanos en el espacio de características. Creamos un modelo KNN con 3 vecinos, entrena el modelo con los datos de entrenamiento, predice las etiquetas para los datos de prueba y devuelve las predicciones.

```
# Clasificador KNN
def knn(X_train, X_test, y_train, y_test, k=3):
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return y_pred
```

Tambiem, con la función para evaluar el desempeño, calculamos métricas de desempeño para evaluar el modelo con calcular la precisión (accuracy_score), que mide la proporción de predicciones correctas, calcula la matriz de confusión, que muestra el conteo de verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos y devuelve la precisión y la matriz de confusión.

```
# Evaluación del desempeño
def evaluate_model(y_test, y_pred):
    acc = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)
    return acc, conf_matrix
```

Para los clasificadores: empezamos con Validación Hold-Out el cual divide el conjunto de datos en entrenamiento (70%) y prueba (30%) usando el clasificador especificado (naive_bayes o KNN).En donde divide los datos usando train_test_split, entrena y predice usando el clasificador seleccionado, evalúa el desempeño con evaluate_model y devuelve la precisión y la matriz de confusión.

```
# Validación Hold-Out
def hold_out(X, y, classifier, k=None):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
    if classifier == "naive_bayes":
        y_pred = naive_bayes(X_train, X_test, y_train, y_test)
    elif classifier == "knn":
        y_pred = knn(X_train, X_test, y_train, y_test, k=k)
    return evaluate_model(y_test, y_pred)
```

Ahora para 10-fold Cross

Dividimos los datos en 10 subconjuntos (folds), entrenamos en 9 y prueba en 1, repitiendo para cada fold, creamos un objeto KFold para dividir los datos y para cada fold:

- Entrena con los datos de entrenamiento del fold.
- Predice las etiquetas del fold de prueba.
- Calcula la precisión y la almacena.

```
def k_fold_validation(X, y, classifier, k=None):
    kf = KFold(n_splits=10, shuffle=True, random_state=42)
    accuracies = []
    for train_idx, test_idx in kf.split(X):
        X_train, X_test = X[train_idx], X[test_idx]
        y_train, y_test = y[train_idx], y[test_idx]
        if classifier == "naive_bayes":
            y_pred = naive_bayes(X_train, X_test, y_train, y_test)
        elif classifier == "knn":
            y_pred = knn(X_train, X_test, y_train, y_test, k=k)
        acc, _ = evaluate_model(y_test, y_pred)
        accuracies.append(acc)
    return np.mean(accuracies)
```

Finalmente para Leave-One-Out usamos un solo dato como prueba y el resto como entrenamiento, repitiendo para cada muestra los siguientes pasos:

1. Dividir los datos en un conjunto de entrenamiento y uno de prueba para cada dato.
2. Entrenar el modelo y realiza predicciones para el dato de prueba.
3. Calcular la precisión para cada iteración.
4. Devolver la precisión promedio.

```python
# Leave-One-Out
def leave_one_out_validation(X, y, classifier, k=None):
    loo = LeaveOneOut()
    accuracies = []
    for train_idx, test_idx in loo.split(X):
        X_train, X_test = X[train_idx], X[test_idx]
        y_train, y_test = y[train_idx], y[test_idx]
        if classifier == "naive_bayes":
            y_pred = naive_bayes(X_train, X_test, y_train, y_test)
        elif classifier == "knn":
            y_pred = knn(X_train, X_test, y_train, y_test, k=k)
        acc, _ = evaluate_model(y_test, y_pred)
        accuracies.append(acc)
    return np.mean(accuracies)
```

En la función principal hacemos todos los pasos mencionados anteriormente para cada Dataset que introducimos, y llamamos a las funciones que generan el resultado de cada validación y lo imprimimos para poder visualizar el resultado de cada uno

```python
datasets = ["iris", "pima-diabetes", "seeds"]
for dataset_name in datasets:
    print(f"Dataset: {dataset_name}")
    X, y = load_dataset(dataset_name)

    print("Hold-Out Validation (Naive Bayes):", hold_out(X, y, "naive_bayes"))
    print("10-Fold Cross-Validation (Naive Bayes):", k_fold_validation(X, y, "naive_bayes"))
    print("Leave-One-Out (Naive Bayes):", leave_one_out_validation(X, y, "naive_bayes"))

    print("Hold-Out Validation (KNN):", hold_out(X, y, "knn", k=5))
    print("10-Fold Cross-Validation (KNN):", k_fold_validation(X, y, "knn", k=5))
    print("Leave-One-Out (KNN):", leave_one_out_validation(X, y, "knn", k=5))
    print()
```

Resultados

Dataset: Iris

```
Dataset: iris
Hold-Out Validation (Naive Bayes): (0.9777777777777777, array([[19,  0,  0],
       [ 0, 12,  1],
       [ 0,  0, 13]]))
10-Fold Cross-Validation (Naive Bayes): 0.9600000000000002
Leave-One-Out (Naive Bayes): 0.9533333333333334
Hold-Out Validation (KNN): (1.0, array([[19,  0,  0],
       [ 0, 13,  0],
       [ 0,  0, 13]]))
10-Fold Cross-Validation (KNN): 0.9733333333333334
Leave-One-Out (KNN): 0.9666666666666667
```

Hold-Out Validation

- Accuracy: 0.9778 (97.78%).

Matriz de confusión:

19	0	0	#Clase 1: 19 predicciones correctas, 0 incorrectas
0	12	1	#Clase 2: 12 predicciones correctas, 1 incorrecta (confundida con clase 3)
0	0	13	#Clase 3: 13 predicciones correctas, 0 incorrectas

10-Fold Cross-Validation

- Accuracy: 0.96 (96.00%). Este es el promedio de las accuracies obtenidas en 10 divisiones de los datos.

Leave-One-Out

- Accuracy: 0.9533 (95.33%). Evaluación individual de cada muestra como conjunto de prueba.

Dataset: Pima-Indians-Diabetes

```
Dataset: pima-diabetes
Hold-Out Validation (Naive Bayes): (0.7445887445887446, array([[119,  32],
       [ 27,  53]]))
10-Fold Cross-Validation (Naive Bayes): 0.7512303485987697
Leave-One-Out (Naive Bayes): 0.75390625
Hold-Out Validation (KNN): (0.6883116883116883, array([[114,  37],
       [ 35,  45]]))
10-Fold Cross-Validation (KNN): 0.6992139439507861
Leave-One-Out (KNN): 0.71484375
```

Hold-Out Validation

Naive Bayes:

- Accuracy: 0.7446 (74.46%).

Matriz de confusión:

119	32	#Clase Negativa: 119 correctas, 32 incorrectas
27	53	# Clase Positiva: 53 correctas, 27 incorrectas.

Se puede observar que este modelo puede confundirse al clasificar a los positivos del dataset.

KNN:

- Accuracy: 0.6883 (68.83%).

Matriz de confusión:

114	37	#Clase Negativa: 14 correctas, 37 incorrectas
35	45	# Clase Positiva: 35 correctas, 45 incorrectas.

Y para este dataset, el clasificador KNN tiene menor precisión comparado al Naive Bayes

10-Fold Cross-Validation

- Naive Bayes: 75.12%. Similar al resultado de Hold-Out.
- KNN: 69.92%. Consistente con el resultado de Hold-Out.

Leave-One-Out

- Naive Bayes: 75.39%.
- KNN: 71.48%. KNN mejora con este método debido al mayor tamaño de los datos de entrenamiento.

Dataset: Seeds

```
Dataset: seeds
Hold-Out Validation (Naive Bayes): (0.873015873015873, array([[16,  1,  3],
       [ 2, 19,  0],
       [ 2,  0, 20]]))
10-Fold Cross-Validation (Naive Bayes): 0.9095238095238095
Leave-One-Out (Naive Bayes): 0.9047619047619048
Hold-Out Validation (KNN): (0.8888888888888888, array([[17,  0,  3],
       [ 1, 20,  0],
       [ 3,  0, 19]]))
10-Fold Cross-Validation (KNN): 0.8904761904761905
Leave-One-Out (KNN): 0.8761904761904762
```

Hold-Out Validation

Naive Bayes:

- Accuracy: 87.30%.

Matriz de confusión

16	1	3	#Clase 1: 16 predicciones correctas, 4 incorrectas
2	19	0	#Clase 2: 19 predicciones correctas, 2 incorrecta (confundida con clase 1)
2	0	20	#Clase 3: 20 predicciones correctas, 2 incorrectas

KNN:

- Accuracy: 88.89%.

Matriz de confusión:

17	0	3	#Clase 1: 17 predicciones correctas, 3 incorrectas
1	20	0	#Clase 2: 20 predicciones correctas, 1 incorrecta (confundida con clase 1)
3	0	19	#Clase 3: 19 predicciones correctas, 3 incorrectas

Podemos observar un mejor desempeño que con Naive-Bayes con KNN

10-Fold Cross-Validation

- Naive Bayes: 90.95%. Buen desempeño en general.
- KNN: 89.05%. Ligeramente inferior.

Leave-One-Out

- Naive Bayes: 90.48%.
- KNN: 87.62%. KNN pierde un poco de precisión comparado con 10-Fold.

Tabla Comparativa de Resultados

Dataset	Validation	Naive Bayes	KNN
Iris	Hold-Out	97.78%	100%
	10-Fold	96.00%	97.33%
	Leave-One-Out	95.33%	96.67%
Pima Diabetes	Hold-Out	74.46%	68.83%
	10-Fold	75.12%	69.92%
	Leave-One-Out	75.39%	71.48%
Seeds	Hold-Out	87.30%	88.89%
	10-Fold	90.95%	89.05%
	Leave-One-Out	90.48%	87.62%

Conclusión

El desarrollo de estos ejercicios de análisis ha permitido explorar dos algoritmos de clasificación populares, Naive Bayes y KNN, aplicándolos a tres dataset diferentes (Iris, Pima Diabetes, y Seeds) mediante tres métodos de validación (Hold-Out, 10-Fold Cross-Validation, y Leave-One-Out). A través de este análisis, hemos obtenido varias conclusiones:

1. **Naive Bayes** es un clasificador efectivo, especialmente cuando las características son independientes entre sí, como lo demuestra su buen desempeño en la mayoría de los datasets. Además, su simplicidad lo hace útil incluso con datos limitados.
2. **KNN** mostró un desempeño muy alto en datasets como Iris, pero en el caso de Pima Diabetes, su desempeño fue más modesto. La elección del valor de **K** es crucial y puede afectar significativamente el rendimiento del modelo.
3. Los **métodos de validación** revelaron que técnicas como el **10-Fold Cross-Validation** y **Leave-One-Out** proporcionan una evaluación más robusta del modelo, ayudando a mitigar el sesgo que puede ocurrir con el Hold-Out, el cual es más sensible a la partición aleatoria de los datos.
4. La **métrica de accuracy** es útil, pero debe complementarse con la **matriz de confusión** para obtener una comprensión más completa del rendimiento del modelo, especialmente en datasets con clases desbalanceadas, como en el caso de Pima Diabetes.

Referencias

- ¿Qué es KNN? (2024, October 24). *Ibm.com*. https://www.ibm.com/mx-es/topics/knn
- (N.d.). Datacamp.com. Retrieved November 22, 2024, from https://datacamp.com/es/tutorial/naive-bayes-scikit-learn
- Allibhai, J. (2018, October 3). Hold-out vs. Cross-validation in Machine Learning. Medium. https://medium.com/@jaz1/holdout-vs-cross-validation-in-machine-learning-7637112d3f8f