CrossValidation

August 8, 2023

1 Cross-Validation for KNN

In this exercise, you will perform k-fold cross-validation when training a KNN classifier. You will use built-in cross-validation tools from scikit-learn. You will train the KNN model on "cell2cell," a telecom company churn prediction data set.

Note: Some of the code cells in this notebook may take a while to run.

1.0.1 Import Packages

Before you get started, import a few packages. Run the code cell below.

[1]:
```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
```

We will also import the scikit-learn `KNeighborsClassifier`, the `train_test_split()` function for splitting the data into training and test sets, and the metric `accuracy_score` to evaluate our model.

[2]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

1.1 Step 1: Load a 'ready-to-fit' Data Set

We will work with the "cell2celltrain" data set. This data set is already preprocessed, with the proper formatting, outliers, and missing values taken care of, and all numerical columns scaled to the [0, 1] interval. One-hot encoding has been performed on all categorical columns. Run the cell below to load the data set and save it to DataFrame `df`.

[3]:
```
filename = os.path.join(os.getcwd(), "data", "cell2celltrain.csv")
df = pd.read_csv(filename, header=0)
```

1.2 Step 2. Create Training and Test Data Sets

1.2.1 a. Create Labeled Examples

Let's obtain columns from our data set to create labeled examples. The code cell below carries out the following steps:

- Gets the `Churn` column from DataFrame `df` and assigns it to the variable `y`. This is our label.
- Gets all other columns from DataFrame `df` and assigns them to the variable `X`. These are our features.

[4]:
```
y = df['Churn']
X = df.drop(columns = 'Churn', axis=1)
```

1.2.2 b. Split Examples Into Training and Test Sets

We will mimic a real-world scenario and split our initial data with the purpose of later performing model selection. Recall that when performing model selection, we split our data into 3 subsets: training, validation, and test. We train on our training set, evaluate on our validation set, make necessary tweaks to the hyperparameters, and continue this process until we are content with our model's performance. We save our test set for the very last and final evaluation of how well our model generalizes to new data.

While we won't perform model selection in this exercise - that is, we won't train and evaluate different KNN models with different values of hyperparameter k - let's get used to splitting 10% of our data and setting it aside as a test set to be used for a final evaluation.

The code cell below splits 10% of our data into a test set.

[5]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10,␣
↪random_state=1234)
```

1.2.3 c. Inspect the Training and Test Data Sets

[6]:
```
print(X_train.shape)
print(X_test.shape)
```

```
(45942, 84)
(5105, 84)
```

[7]:
```
X_train.head()
```

[7]:
```
       CustomerID  ChildrenInHH  HandsetRefurbished  HandsetWebCapable  \
45106     3356966         False               False               True
38896     3310250          True               False               True
29853     3237338         False               False               True
33048     3263222         False               False               True
21061     3165118         False               False               True

       TruckOwner  RVOwner  HomeownershipKnown  BuysViaMailOrder  \
45106       False    False                True             False
38896       False    False                True              True
29853       False    False                True             False
33048       False    False                True              True
21061       False    False               False             False

       RespondsToMailOffers  OptOutMailings  ...  Occupation_Crafts  \
```

```
45106                False          False  ...                0.0
38896                 True          False  ...                0.0
29853                False          False  ...                0.0
33048                 True          False  ...                0.0
21061                False          False  ...                0.0

       Occupation_Homemaker  Occupation_Other  Occupation_Professional  \
45106                   0.0               1.0                      0.0
38896                   0.0               1.0                      0.0
29853                   0.0               1.0                      0.0
33048                   0.0               0.0                      0.0
21061                   0.0               1.0                      0.0

       Occupation_Retired  Occupation_Self  Occupation_Student  Married_False  \
45106                 0.0              0.0                 0.0            0.0
38896                 0.0              0.0                 0.0            0.0
29853                 0.0              0.0                 0.0            1.0
33048                 1.0              0.0                 0.0            0.0
21061                 0.0              0.0                 0.0            0.0

       Married_True  Married_nan
45106           0.0          1.0
38896           0.0          1.0
29853           0.0          0.0
33048           1.0          0.0
21061           0.0          1.0

[5 rows x 84 columns]
```

1.3 Step 3: Fit a KNN Classifier and Perform k-Fold Cross-Validation

We just created a training set and a test set. We won't use our test set in this exercise, but we will focus on the training set.

We will perform k-fold cross-validation on our training set. K-fold cross-validation is an out-of-sample technique that helps estimate how well your model generalizes to new data.

k-fold cross-validation splits training data into equally sized subsets, or folds (k). We train and test 'k' times, such that each time, we train on k-1 *training* folds and test on 1 *validation* fold. Therefore, every fold will have a chance to serve as a validation set. We then average the resulting accuracies obtained on each of the k iterations to determine the accuracy of the model.

We will use built-in cross-validation tools from `sklearn` that will help us perform a k-fold cross-validation on our KNN model.

First, let's create the KNN model object.

1.3.1 a. Create the KNN Model Object

In the code cell below, using the scikit-learn `KNeighborsClassifier` class, create a KNN model object with a hyperparameter k value of 3 (3 nearest neighbors). Assign the model object to the

variable `model`. We will use this model when performing k-fold cross-validation.

Note: Don't confuse the KNN hyperparameter *k* with the 'k' in k-fold cross-validation; they represent different things.

1.3.2 Graded Cell

The cell below will be graded. Remove the line "raise NotImplementedError()" before writing your code.

[8]:
```
# Create the  KNeighborsClassifier model object
# YOUR CODE HERE
model=KNeighborsClassifier(n_neighbors=3)
```

1.3.3 Self-Check

Run the cell below to test the correctness of your code above before submitting for grading. Do not add code or delete code in the cell. Note: This may take a few minutes to run.

[9]:
```
# Run this self-test cell to check your code;
# do not add code or delete code in this cell
from jn import testKNN

try:
    p, err = testKNN(model)
    print(err)
except Exception as e:
    print("Error!\n" + str(e))
```

```
Correct!
```

1.3.4 b. Train and Evaluate the Model's Performance Using Two K-fold Cross-Validation Approaches

1.3.5 Scikit-Learn's `KFold` Approach

First we will use scikit-learn's `KFold` class. This class helps us partition our training data set into k folds. Let's import the `KFold` class from `sklearn`:

[10]:
```
from sklearn.model_selection import KFold
```

The code cell below is performing 5-fold cross-validation. We use `KFold` to partition our training data (`X_train` and `y_train`) into 5 folds: `KFold` separates our training data into 5 subsets of rows. We then train our KNN model on each subset and evaluate its performance. Therefore, we will train and evaluate our KNN model for a total of 5 times - once per data subset. (Note that we won't change the hyperparameters of our model; our KNN model, with its current configuration of 3 nearest neighbors, will be trained 5 times using different training data subsets.) We will then average the model's accuracy score at each iteration to compute the overall model accuracy.

Examine the code cell below. It is doing the following:

1. Splits the training set into 5 subsets

2. Loops over each subset, and for each data subset it:
    1. Splits the data subset into new training and validation sets
    2. Trains the KNN model on the new training set
    3. Tests the model on the new validation set and computes the accuracy of the model's predictions.
    4. Saves the accuracy score to a list.
3. Computes the average of the 5 accuracy scores.

Run the code cell and inspect the resulting accuracy scores.

[11]:
```
num_folds = 5
folds = KFold(n_splits = num_folds, random_state=None)

acc_scores = []

for train_row_index , test_row_index in folds.split(X_train):

    # our new partition of X_train and X_val
    X_train_new  = X_train.iloc[train_row_index]
    X_val = X_train.iloc[test_row_index]

    # our new partition of y_train and y_val
    y_train_new = y_train.iloc[train_row_index]
    y_val = y_train.iloc[test_row_index]

    model.fit(X_train_new, y_train_new)
    predictions = model.predict(X_val)

    iteration_accuracy = accuracy_score(predictions , y_val)
    acc_scores.append(iteration_accuracy)


for i in range(len(acc_scores)):
    print('Accuracy score for iteration {0}: {1}'.format(i+1, acc_scores[i]))

avg_scores = sum(acc_scores)/num_folds
print('\nAverage accuracy score: {}'.format(avg_scores))
```

```
Accuracy score for iteration 1: 0.6283599956469692
Accuracy score for iteration 2: 0.6305365110458157
Accuracy score for iteration 3: 0.6320200261210275
Accuracy score for iteration 4: 0.6238572050500653
Accuracy score for iteration 5: 0.630387461906835

Average accuracy score: 0.6290322399541426
```

1.3.6 Scikit-Learn's `cross_val_score` Approach

Scikit-learn has another method of performing k-fold cross-validation that performs the steps above in one line of code. This is the `cross_val_score()` function. It allows you to specify the number of folds 'k'. It automatically and repeatedly subsamples the training and validation sets out of the initial training data and computes the accuracy scores when testing on these validation sets. It returns one accuracy score per iteration ('k' iterations in total).

First let's import the function from `sklearn`:

[12]:
```
from sklearn.model_selection import cross_val_score
```

Inspect the online documentation for the `cross_val_score()` function to learn how to use it.

In the code cell below, use the `cross_val_score()` function to perform a 5-fold cross-validation for our KNN model and obtain a list of accuracy scores.

Call the function with the following arguments:

1. Your model object `model`.
2. Your training data `X_train` and `y_train`
3. Specify the number of folds using the parameter `cv` (`cv=5`).

Save the results to the list `accuracy_scores`.

Note that for a classifier, the function `cross_val_score()` uses the scikit-learn `accuracy_score` metric by default. If you wanted to use a different evaluation metric, you could specify so using the parameter `scoring` (e.g., `scoring = 'accuracy'`). Since we are interested in obtaining accuracy scores, you do not have to specify this.

Note: This may take a little while to run.

1.3.7 Graded Cell

The cell below will be graded. Remove the line "raise NotImplementedError()" before writing your code.

[13]:
```
# Use function cross_val_score() to perform 5-fold cross-validation on the␣
 ↪training data and
# save the result to variable accuracy_scores
print('Running Cross-Validation...')

# YOUR CODE HERE
accuracy_scores=cross_val_score(model,X_train,y_train,cv=5)

print('Done')

# Print the accuracy scores
print('Accuracies for the five training/test iterations on the validation sets:
 ↪')
print(accuracy_scores)
```

```
Running Cross-Validation...
Done
Accuracies for the five training/test iterations on the validation sets:
[0.62890412 0.63075416 0.63006095 0.62483674 0.62940792]
```

1.3.8 Self-Check

Run the cell below to test the correctness of your code above before submitting for grading. Do not add code or delete code in the cell. Note: This may take a little while to run.

[14]:
```
# Run this self-test cell to check your code;
# do not add code or delete code in this cell
from jn import testCrossVal

try:
    p, err = testCrossVal(df, accuracy_scores)
    print(err)
except Exception as e:
    print("Error!\n" + str(e))
```

```
Running Test...
Test Complete. See results below:


Correct!
```

The `cross_val_score()` function returns a NumPy array. In the code cell below, use NumPy's `mean()` and `std()` methods on array `accuracy_scores` to find the average accuracy score for our model and to see the degree of variance among each score.

Save the mean to variable `acc_mean` and save the standard deviation to the variable `acc_std`.

1.3.9 Graded Cell

The cell below will be graded. Remove the line "raise NotImplementedError()" before writing your code.

[15]:
```
# Find the mean accuracy score and save to variable 'acc_mean'
# YOUR CODE HERE
acc_mean=np.mean(accuracy_scores)
print('The mean accuracy score across the five iterations:')
print(acc_mean)


# Find the standard deviation of the accuracy score and save to variable␣
 ↪'acc_std'
# YOUR CODE HERE
acc_std=np.std(accuracy_scores)

# Print the standard deviation of the accuracy scores using the std() method to␣
 ↪see the degree of variance.
print('The standard deviation of the accuracy score across the five iterations:
 ↪')
print(acc_std)
```

```
The mean accuracy score across the five iterations:
0.6287927806206556
The standard deviation of the accuracy score across the five iterations:
0.0020734655802685143
```

1.3.10 Self-Check

Run the cell below to test the correctness of your code above before submitting for grading. Do not add code or delete code in the cell. Note: This may take a little while to run.

[16]:
```
# Run this self-test cell to check your code;
# do not add code or delete code in this cell
from jn import testMeanStd

try:
    p, err = testMeanStd(df, acc_mean, acc_std, accuracy_scores)
    print(err)
except Exception as e:
    print("Error!\n" + str(e))
```

```
Running Test...
Test Complete. See results below:


Correct!
```

You'll notice that the five resulting accuracy scores are good, and the standard deviation among the scores are low, indicating that our model performs well.