گزارش قسمت ۱

K-Fold cross validation

علی صالح ۹۷۲۲۲۰۵۳

Cross validation :

ما در حالت عادی برای تست کردن مدلمان پس از ترین شدن با قسمتی از دیتاست (۲۰ درصد) که قبل از ترین کردن جدا کرده بودیم و در فرایند ترین کردن نقشی نداشتند تست می‌کردیم و میزان خطا و دقت را به دست می‌آوردیم.
اما این روش ممکن از مشکلاتی نیز داشته باشد.
یکی از مشکلات این است که ممکن است دیتای تستی که انتخاب می‌کنیم قسمت خاصی از دیتا باشد و این قسمت خاص دقت بالا تر از دقت واقعی یا دقت پایین تر از دقت واقعی به ما دهد.
به طور مثال در همین سوال تخمین متراژ خانه ها فرض کنید که مدل ما مقدار هایی بالاتر از میانگین برای ما پیش بینی میکند حالا اگر همه ی دیتای تستی که انتخاب می‌کنیم به طور اتفاقی بیشتر از میانگین باشد دیتای تست ما دقت بالایی به ما می‌دهد در حالی که دقت واقعی از این اندازه کمتر است.

روش k-fold cross validation به این شکل عمل می‌کند که ما ابتدا همه‌ی دیتا را به k بخش تقسیم می‌کنیم و فرایند آموزش را k بار تکرار می‌کنیم که در هر بار ما یکی از قسمت ها (fold) را به عنوان دیتای تست در نظر می‌گیریم و در پایان ما k درصد و دقت داریم که می‌توانیم برایندی از این درصد و دقت را به عنوان دقت نشان دهیم. با این روش ما هیچوقت امکان ندارد قسمت خاصی از دیتا را به عنوان دیتای تست در نظر بگیریم و به مشکل بالا بخوریم.

البته این روش استفاده های دیگری هم دارد و می‌توان در طول فرایند آموزش و جدای از فرایند تست از k-fold برای ترین کردن هایپر پارامتر ها استفاده کرد.

تابع k-fold :

ابتدا دیتا ست را به صورت رندوم بر می‌زنیم و با گام های n/k روی دیتا ها پیش می‌رویم و هر سری به اندازه ی n/k دیتایی که از i تا i + n/k است را به عنوان دیتای تست و مابقی دیتا هارا دیتای ترین در نظر می‌گیریم و این دیتا ها را به مدلمان می‌دهیم. و در پایان میانگین دقت ها را خروجی می‌دهیم.

این تابع طوری نوشته شده است که بتواند با کدل رگرشنی که در تمرین قبلی نوشته بودیم کار کند و از متود های آن استفاده کند.

```
from sklearn.utils import shuffle


def k_fold(Model, X, y, k=10, epochs=50):
  X, y = shuffle(X, y, random_state=0)
  n = len(y)
  X = np.array([X]).T
  y = np.array(y)
  folds_acc = []
  sum = 0
  j = 1
  for i in range(0, n, n//k):
    trainx = np.concatenate((X[0:i], X[i + n//k: n]), axis=0)
    trainy = np.concatenate((y[0:i], y[i + n//k: n]), axis=0)
    valx = X[i: i + n//k]
    valy = y[i: i + n//k]
    model = Model(epochs=epochs)
    model.fit(trainx, trainy, valx, valy)
    folds_acc.append(model.history['accuracy'][-1])
    print('fold', j, ':', model.history['accuracy'][-1])
    sum += model.history['accuracy'][-1]
    j += 1
  return sum/k
```

کد مدل لینیر رگرشنی که در تمرین قبلی پیاده سازی کردیم:

```
from sklearn.metrics import mean_squared_error

class LinearRegression:

  def __init__(self, learning_rate=0.1, epochs=100, accuracy_rate=0.1):
    self.learning_rate = learning_rate
    self.epochs = epochs
    self.accuracy_rate = accuracy_rate
    self.history = {'loss': [], 'accuracy': []}

  def gradient(self, X, y):
    return 2/X.shape[0] * np.dot(X.T, (np.dot(X, self.weights) - y))

  def fit(self, X, y, Xval, yval):
    train = []
    for i in range(len(X)):
      x = list(X[i])
      x.insert(0, 1)
      train.append(np.array(x))
    train = np.array(train)
    self.weights = np.random.rand((train.shape[1]))
    for i in range(self.epochs + 1):
      pred = self.predict(Xval)
      err = mean_squared_error(yval, pred)
      acc = self.calculate_accuracy(pred, yval)
      self.history['loss'].append(err)
      self.history['accuracy'].append(acc)
      self.weights = self.weights - self.learning_rate * self.gradient(train, y)

  def predict(self, X):
    pred = []
    for i in range(len(X)):
      x = list(X[i])
      x.insert(0, 1)
      pred.append(np.array(x))
    return np.dot(pred, self.weights)

  def calculate_accuracy(self, pred, val):
    right = 0
    for i in range(len(pred)):
      if abs(pred[i] - val[i]) <= val[i] * self.accuracy_rate:
        right += 1
    return right / len(pred)
```

سوال ۱:

حالت ۱- فیچر serviceCharge را به عنوان فیچری که بیشتری کورلیشن دارد را به عنوان ورودی در نظر می‌گیریم و تابع k-fold را با k=10 و k=5 اجرا می‌کنیم.

نتایج:

```
5-fold:
fold 1 : 0.2681048912032732
fold 2 : 0.21898828342942162
fold 3 : 0.229737771991817
fold 4 : 0.2364143574483913
fold 5 : 0.27790589548075134
accuracy: 0.24623023991073087

10-fold:
fold 1 : 0.2518132787799888
fold 2 : 0.22971917426074018
fold 3 : 0.24519248651664496
fold 4 : 0.29060814580621164
fold 5 : 0.2857727357262414
fold 6 : 0.27662265203645153
fold 7 : 0.23976194904221684
fold 8 : 0.2609633624697787
fold 9 : 0.2219825181327878
fold 10 : 0.24121257206620791
accuracy: 0.25436488748372693
```

حالت ۲- چون تابع k-fold ای که نوشته بودیم برای کار با مدل رگرسیونی بود که خودمان در سری قبلی پیاده سازی کرده بودیم برای این قسمت کمی این تابع را تغییر میدهیم. زیرا مدل LinearRegression کتابخانه‌ی sklearn برعکس مدل رگرسیون ما ولیدیشن ندارد و باید ولیدیشن را جدای از مدل بنویسیم.

برای این کار تابع k-fold را تغییر می‌دهیم و به تبدیل به کد روبرو می‌کنیم.

```python
from sklearn.utils import shuffle
from sklearn.metrics import mean_squared_error


def calculate_accuracy(pred, val, accuracy_rate):
  right = 0
  for i in range(len(pred)):
    if abs(pred[i] - val[i]) <= val[i] * accuracy_rate:
      right += 1
  return right / len(pred)

def k_fold(Model, X, y, k=10, epochs=50):
  X, y = shuffle(X, y, random_state=0)
  n = len(y)
  X = np.array([X]).T
  y = np.array(y)
  folds_acc = []
  sum = 0
  j = 1
  for i in range(0, n, n//k):
    trainx = np.concatenate((X[0:i], X[i + n//k: n]), axis=0)
    trainy = np.concatenate((y[0:i], y[i + n//k: n]), axis=0)
    valx = X[i: i + n//k]
    valy = y[i: i + n//k]
    model = Model()
    model.fit(trainx, trainy)
    pred = model.predict(valx)
    acc = calculate_accuracy(pred, valy, 0.1)
    folds_acc.append(acc)
    print('fold', j, ':', acc)
    sum += acc
    j += 1
  return sum/k, folds_acc
```

و مانند قسمت قبلی با دادن serviceCharge به عنوان ورودی به نتایج زیر می‌رسیم:

```
5-fold:
fold 1 : 0.22029012460479822
fold 2 : 0.28673981774223545
fold 3 : 0.21327877998884137
fold 4 : 0.21909986981588248
fold 5 : 0.21428305746698903
accuracy: 0.23073832992374932

10-fold:
fold 1 : 0.2243258322484657
fold 2 : 0.21688673981774223
fold 3 : 0.2161800260368235
fold 4 : 0.2918727915194346
fold 5 : 0.21424586200483542
fold 6 : 0.2213501952761763
fold 7 : 0.22030872233587503
fold 8 : 0.21997396317649245
fold 9 : 0.21294402082945882
fold 10 : 0.2183001673795797
accuracy: 0.2256388320624884
```

حالت ۳- برای این حالت دو فیچر که بیشترین کولریشن را دارند یعنی 'serviceCharge', 'pricetrend' و دو فیچر که کمترین کورلیشن را دارند یعنی 'geo_plz', 'baseRent' را انتخاب می‌کنیم. و به نتایج روبرو می‌رسیم.

```
5-fold:
(215080, 4)
fold 1 : 0.24800074390924307
(215080, 4)
fold 2 : 0.2945880602566487
(215080, 4)
fold 3 : 0.2405430537474428
(215080, 4)
fold 4 : 0.25419378835782036
(215080, 4)
fold 5 : 0.24681048912032733
accuracy: 0.2568272270782964

10-fold:
(241965, 4)
fold 1 : 0.25028826483169053
(241965, 4)
fold 2 : 0.24757299609447647
(241965, 4)
fold 3 : 0.24820531895108797
(241965, 4)
fold 4 : 0.2956667286591036
(241965, 4)
fold 5 : 0.24459735912218708
(241965, 4)
fold 6 : 0.24708945508647945
(241965, 4)
fold 7 : 0.24961874651292543
(241965, 4)
fold 8 : 0.25437976566858844
(241965, 4)
fold 9 : 0.24429979542495817
(241965, 4)
fold 10 : 0.25170169239352796
accuracy: 0.2533420122745025
```

حالت ۴- تابع k-fold ما توانایی دادن هر تعداد فیچر را دارد. این بار همه فیچر ها را باهم به مدلمان می‌دهیم. و به نتایج زیر می‌رسیم.

```
5-fold:
fold 1 : 0.5396503626557559
fold 2 : 0.42356332527431656
fold 3 : 0.4826855123674912
fold 4 : 0.569964664310954
fold 5 : 0.47703180212014135
accuracy: 0.4985791333457318

10-fold:
fold 1 : 0.5484098939929328
fold 2 : 0.49856797470708575
fold 3 : 0.49302585084619677
fold 4 : 0.44493211828156964
fold 5 : 0.4941789101729589
fold 6 : 0.5032917984005951
fold 7 : 0.5738887855681607
fold 8 : 0.5014692207550678
fold 9 : 0.49667100613725124
fold 10 : 0.49577831504556447
accuracy: 0.5050213873907383
```

حالت ۵- همه فیچر ها با مدل Ridge با alpha=1.0 را به تابع می‌دهیم و به نتیجه زیر می‌رسیم.

```
5-fold:
fold 1 : 0.5368607029942347
fold 2 : 0.48043518690719733
fold 3 : 0.4825739259810303
fold 4 : 0.5648130928026781
fold 5 : 0.4737957969127766
accuracy: 0.5076957411195834

10-fold:
fold 1 : 0.5449135205504928
fold 2 : 0.4956667286591036
fold 3 : 0.4894922819416031
fold 4 : 0.5032546029384415
fold 5 : 0.49429049655941976
fold 6 : 0.4999442068067696
fold 7 : 0.5694997210340339
fold 8 : 0.49689417891017296
fold 9 : 0.49328621908127207
fold 10 : 0.4915008368978985
accuracy: 0.5078742793379207
```

حالت ۶- همه فیچر ها با مدل Lasso با alpha=1.0 را به تابع می‌دهیم و به نتیجه زیر می‌رسیم.

```
5-fold:
fold 1 : 0.2075692765482611
fold 2 : 0.20610005579319324
fold 3 : 0.20727171285103219
fold 4 : 0.20636042402826854
fold 5 : 0.2040915008368979
accuracy: 0.2062785940115306

10-fold:
fold 1 : 0.21175376604054305
fold 2 : 0.2026036823507532
fold 3 : 0.20200855495629533
fold 4 : 0.210786684024549
fold 5 : 0.2081830016737958
fold 6 : 0.2066951831876511
fold 7 : 0.20636042402826854
fold 8 : 0.20505858285289194
fold 9 : 0.20104147294030128
fold 10 : 0.2077366561279524
accuracy: 0.2062228008183002
```

سوال۲

بین مدل رگرسیون و رگرسیون Ridge تفاوت خاصی مشاهده نشد و هر دو در هر دو حالت fold-5 و fold-10 حدود ۵۰ درصد دقت را نمایش دادند اما رگرسیون Lasso درصد خیلی کمتری (حدود ۲۰ درصد) به ما داد.

(خروجی حالت ۴ و ۵ و ۶)