Lab4

October 31, 2024

0.0.1 Nearest Neighbour Regression and an Inverted U shape

[1]:
```python
from sklearn.datasets import load_diabetes
diabetes = load_diabetes()
diabetes["data"].shape
```

[1]:
```
(442, 10)
```

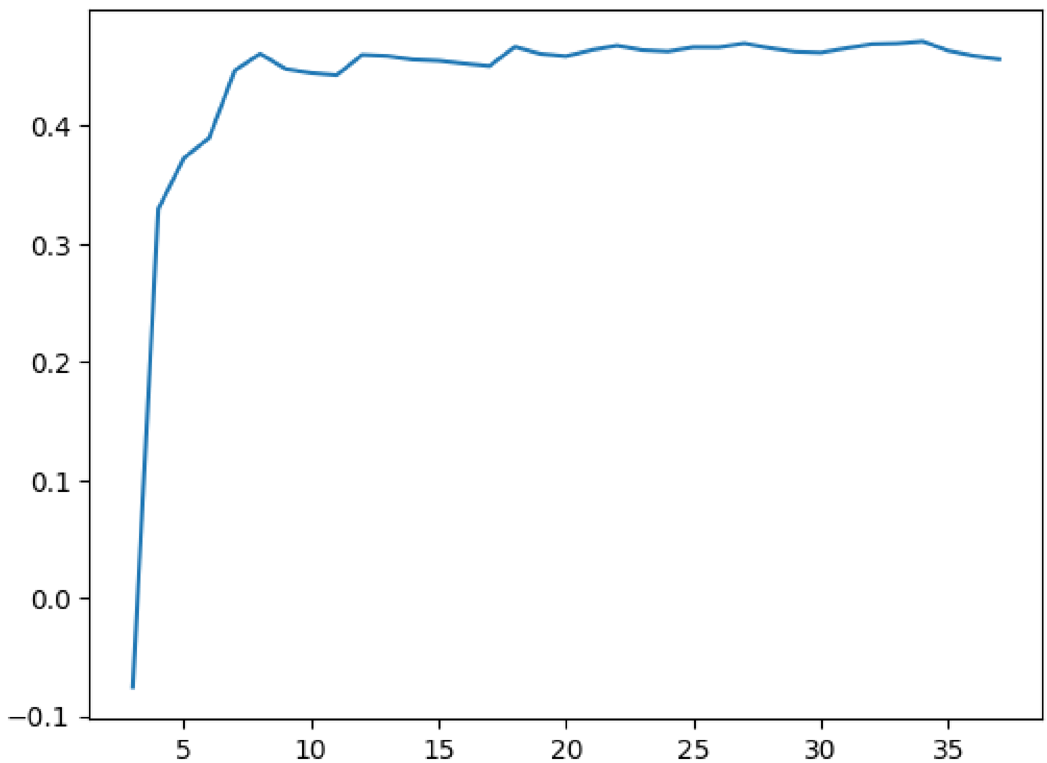
[2]:
```python
print(diabetes.DESCR)
```

```
.. _diabetes_dataset:

Diabetes dataset
----------------

Ten baseline variables, age, sex, body mass index, average blood
pressure, and six blood serum measurements were obtained for each of n =
442 diabetes patients, as well as the response of interest, a
quantitative measure of disease progression one year after baseline.

**Data Set Characteristics:**

:Number of Instances: 442

:Number of Attributes: First 10 columns are numeric predictive values

:Target: Column 11 is a quantitative measure of disease progression one year
after baseline

:Attribute Information:
    - age     age in years
    - sex
    - bmi     body mass index
    - bp      average blood pressure
    - s1      tc, total serum cholesterol
    - s2      ldl, low-density lipoproteins
    - s3      hdl, high-density lipoproteins
    - s4      tch, total cholesterol / HDL
```

```
    - s5      ltg, possibly log of serum triglycerides level
    - s6      glu, blood sugar level

Note: Each of these 10 feature variables have been mean centered and scaled by
the standard deviation times the square root of `n_samples` (i.e. the sum of
squares of each column totals 1).

Source URL:
https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html

For more information see:
Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (2004) "Least
Angle Regression," Annals of Statistics (with discussion), 407-499.
(https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)
```

[3]:
```python
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
import numpy as np
import matplotlib.pyplot as plt

X_train, X_test, Y_train, Y_test = train_test_split(diabetes['data'],
 ↪diabetes['target'], random_state=42)

K_max = 35
results = np.empty(K_max)
for k in range(K_max):
    knn = KNeighborsRegressor(n_neighbors = k+1)
    knn.fit(X_train, Y_train)
    results[k] = knn.score(X_test, Y_test)
%matplotlib inline
plt.plot(np.arange(K_max)+3, results)
```

[3]:
```
[<matplotlib.lines.Line2D at 0x775d175f00d0>]
```



[4]:
```
help(KNeighborsRegressor.score)
```

```
Help on function score in module sklearn.base:

score(self, X, y, sample_weight=None)
    Return the coefficient of determination of the prediction.

    The coefficient of determination :math:`R^2` is defined as
    :math:`(1 - \frac{u}{v})`, where :math:`u` is the residual
    sum of squares ``((y_true - y_pred)** 2).sum()`` and :math:`v`
    is the total sum of squares ``((y_true - y_true.mean()) ** 2).sum()``.
    The best possible score is 1.0 and it can be negative (because the
    model can be arbitrarily worse). A constant model that always predicts
    the expected value of `y`, disregarding the input features, would get
    a :math:`R^2` score of 0.0.

    Parameters
    ----------
    X : array-like of shape (n_samples, n_features)
        Test samples. For some estimators this may be a precomputed
        kernel matrix or a list of generic objects instead with shape
        ``(n_samples, n_samples_fitted)``, where ``n_samples_fitted``
```

```
        is the number of samples used in the fitting for the estimator.

    y : array-like of shape (n_samples,) or (n_samples, n_outputs)
        True values for `X`.

    sample_weight : array-like of shape (n_samples,), default=None
        Sample weights.

    Returns
    -------
    score : float
        :math:`R^2` of ``self.predict(X)`` w.r.t. `y`.

    Notes
    -----
    The :math:`R^2` score used when calling ``score`` on a regressor uses
    ``multioutput='uniform_average'`` from version 0.23 to keep consistent
    with default value of :func:`~sklearn.metrics.r2_score`.
    This influences the ``score`` method of all the multioutput
    regressors (except for
    :class:`~sklearn.multioutput.MultiOutputRegressor`).
```

0.0.2 Using cross-validation to get an inverted U-shaped curve

[5]:
```python
from sklearn.model_selection import cross_val_score
knn = KNeighborsRegressor(n_neighbors=3)
cross_val_score(knn, X_train, Y_train)
```

[5]:
```
array([0.28743721, 0.24283699, 0.32312734, 0.32108514, 0.12845082])
```

[6]:
```python
from sklearn.utils import shuffle
X, Y = shuffle(diabetes["data"], diabetes["target"], random_state=42)
print(cross_val_score(knn, X, Y))
```

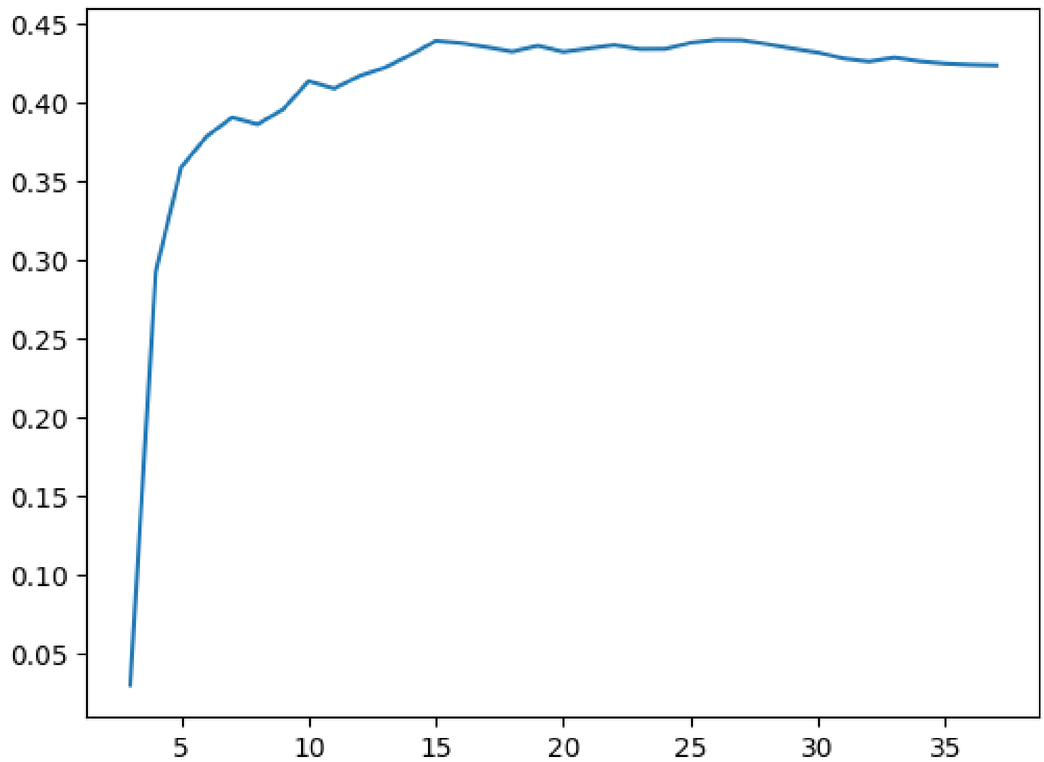
```
[0.36498737 0.32300369 0.26748019 0.43230668 0.40572547]
```

[7]:
```python
knn.fit(X_train, Y_train)
knn.score(X_test, Y_test)
```

[7]:
```
0.37222167132521977
```

[8]:
```python
K_max = 35
for k in range(K_max):
    knn = KNeighborsRegressor(n_neighbors=k+1)
    results[k] = np.mean(cross_val_score(knn, X, Y))
plt.plot(np.arange(K_max)+3, results)
```

[8]: [<matplotlib.lines.Line2D at 0x775d56ce6710>]

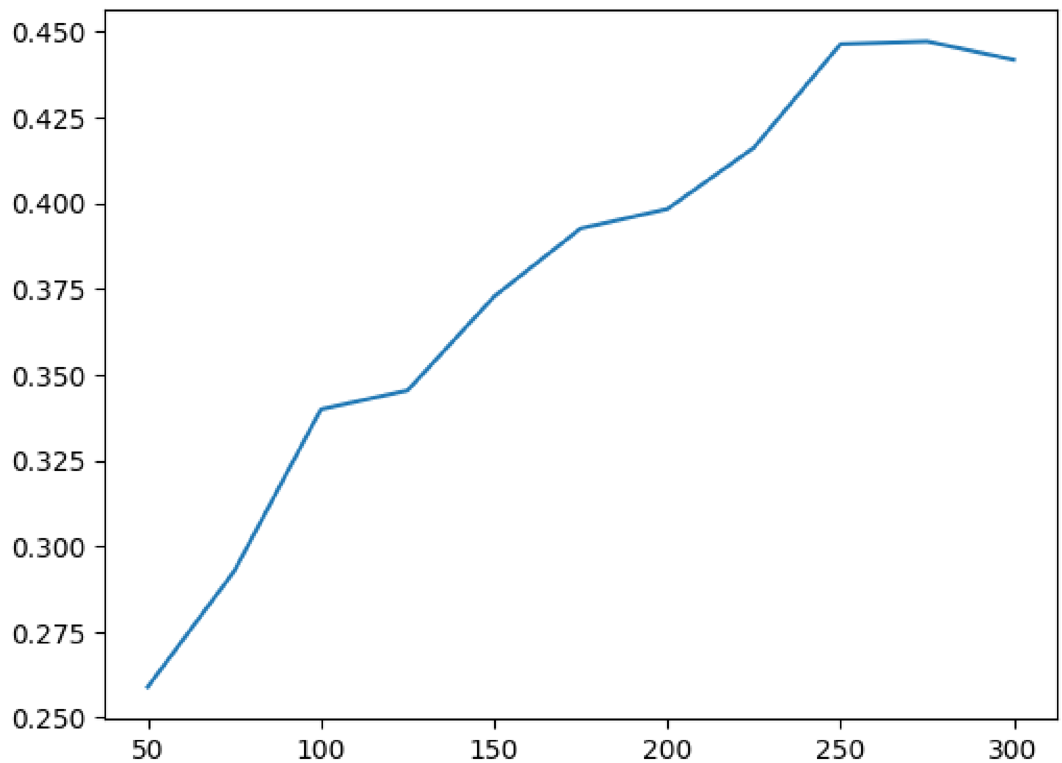


Exercises 1. np.mean takes all accuracies of each fold in cross_val_score() and performs and average calculation on all the values to give a singular value that can be graphed. 2. For both calculations, the optimal K-value floats around the 20-30 range. It may even be argued that 10 would suffice. At large values of K, the accuracy of the predictor drops off drastically. Become less and less accurate as more neighbours are taken in to account.

0.0.3 Learning Curves

[9]:
```
knn = KNeighborsRegressor(n_neighbors=10)
train_sizes = np.array([50, 75, 100, 125, 150, 175, 200, 225, 250, 275, 300])
results = np.empty(train_sizes.size)
for k in range(train_sizes.size):
    X_train, X_test, Y_train, Y_test = train_test_split(diabetes["data"],␣
 ↪diabetes["target"], train_size = train_sizes[k], random_state=42)
    knn.fit(X_train, Y_train)
    results[k] = knn.score(X_test, Y_test)
plt.plot(train_sizes, results)
```

[9]: [<matplotlib.lines.Line2D at 0x775d15480fd0>]



Exercises 3. The accuracy is more of a positive linear graph. 4. The size of the test set would be 22. If only test_size was specified, train_size would be the complement of test_size on the size of the dataset. You can specify both and they do not have to add up to the size of the dataset.

0.0.4 Value at Risk

[10]:
```python
n = 99
L = 10**6 * np.random.random((n)) - 10**6/2
print(L)
```

```
[  65922.1999099  -457895.48303754 -304008.33917592 -429073.94147953
  244773.13641742  162562.10395629  157284.30675486  391385.15082257
  192708.41663258   13040.23147988 -433850.03934718  285701.32668387
 -486662.98980427  232412.65500513 -121506.32124583 -364612.57210957
 -203453.32714741   40492.00470941 -181498.32415667 -367959.60754015
  491974.48514982  348590.08959706   83715.48769727 -429896.53107894
 -214389.5663539   308247.26395334  153371.30028126  298372.73381471
  306771.67351612   46181.67344741  456689.68855559 -150105.9291947
   33787.52321007   56303.82844309 -451075.18339606   94139.74220831
  -73917.66745356   20847.95272094  -34879.41659521 -277597.05256904
  -74544.5264068    35580.46889994 -248303.37019428 -354439.67999264
  294796.21704481 -405385.50556517 -368143.84025485 -227120.03776037
  259567.15500801  268444.61212572  406225.85472881  -32258.32823494
```

```
  54239.29303335  191144.01111077  -33622.23054772  -84944.78035642
-120958.61536565  192145.80445293  276513.62853419  -33760.34530748
-309980.43714813  -23396.50188935 -183687.5414733   279041.78391859
-375662.17238922  283490.87306095 -488979.25340934  449128.39222244
-140162.44210875  311930.18334797  423021.67302289 -114468.13894916
-205882.25156318   54897.69299371  380238.4979516  -262608.6718197
  41214.41681512 -307528.96924808  489255.08812563 -279209.89837034
-463937.64057674  121540.16753583  480503.20749935  249021.8092474
 -62316.26940698  301265.51514182 -429167.67684511  -59891.13786629
  22015.7630486   -46498.74135347   32842.97636588 -266627.00162274
 -46468.41455567 -388130.11064565 -440471.37325647 -401955.63997069
-151889.74822131  116529.81667765  492735.64131887]
```

[11]:
```
sorted_L = np.sort(L)
VaR = sorted_L[-int(np.floor((n+1)/5))]
print(VaR)
```

```
279041.78391859063
```

[12]:
```
import math
def VaR(L):
    if L.size >= 4:
        return np.sort(L)[-int(np.floor((n+1)/5))]
    else:
        return math.inf
```

0.0.5 Validity of conformal prediction: an empirical test

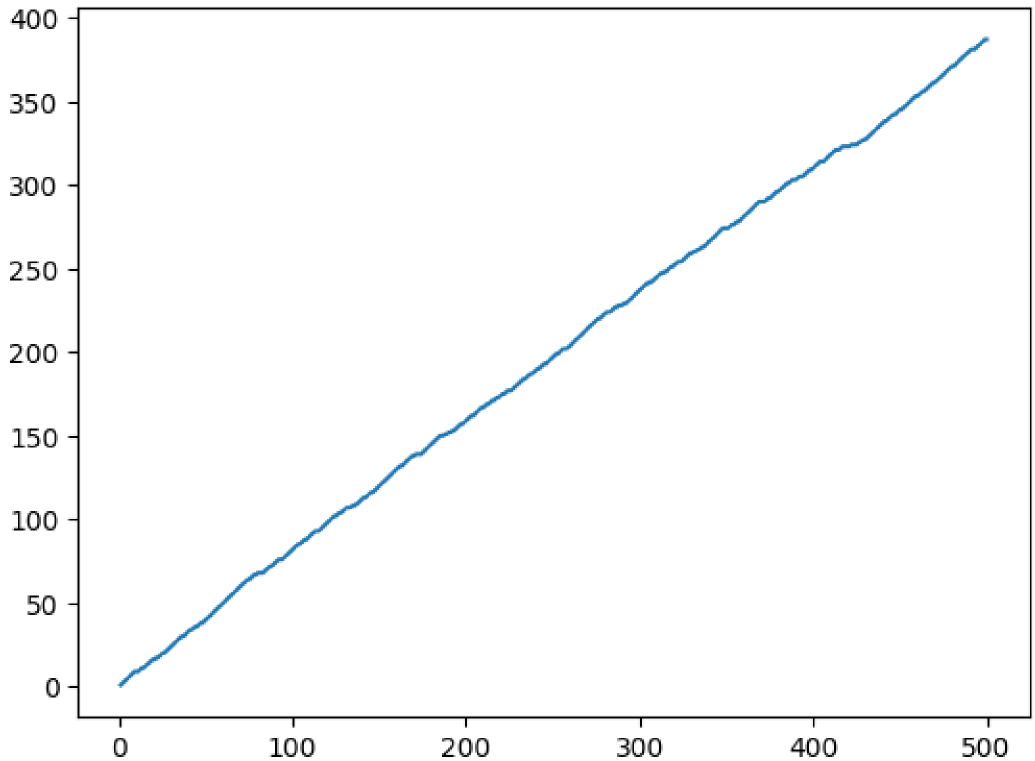
[13]:
```
N = 500
L = 10**6 * np.random.random_sample((N)) - 10**6/2
```

[14]:
```
successes = np.empty(N)
for n in range(N):
    V = VaR(L[:n])
    if L[n] <= V:
        successes[n] = 1
    else:
        successes[n] = 0
print(np.mean(successes))
```

```
0.774
```

[15]:
```
plt.plot(np.arange(N)+1, np.cumsum(successes))
```

[15]:
```
[<matplotlib.lines.Line2D at 0x775d1556bf10>]
```

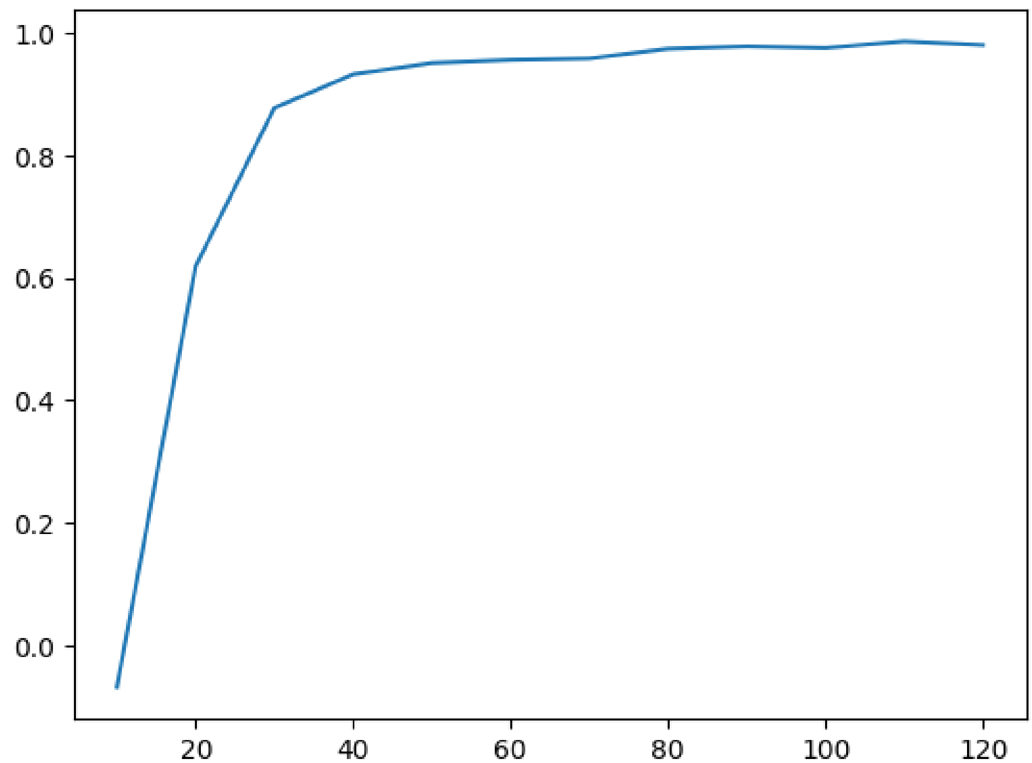


Exercise 5. cumsum is a cumulative sum where it takes the sum of all previous entries and current entry.

0.0.6 One more exercise

[16]:
```python
from sklearn.datasets import load_iris

iris = load_iris()
K_max = 100
results = np.empty(K_max)
knn = KNeighborsRegressor(n_neighbors=10)
train_sizes = np.array([10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120])
results = np.empty(train_sizes.size)
for k in range(train_sizes.size):
    X_train, X_test, Y_train, Y_test = train_test_split(iris["data"],␣
 ↪iris["target"], train_size = train_sizes[k], random_state=42)
    knn.fit(X_train, Y_train)
    results[k] = knn.score(X_test, Y_test)
plt.plot(train_sizes, results)
```

[16]: [<matplotlib.lines.Line2D at 0x775d1559d950>]


1.0
0.8
0.6
0.4
0.2
0.0
20
40
60
80
100
120