Published in Analytics Vidhya

Imam Muhajir Follow

Apr 20, 2019 · 3 min read · Listen

K-Neighbors Regression Analysis in Python

K nearest neighbors is a simple algorithm that stores all available cases and predict the numerical target based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique. **Algorithm** A simple implementation of KNN regression is to calculate the average of the numerical target of the K nearest neighbors. Another approach uses an inverse distance weighted average of the K nearest neighbors. KNN regression uses the same distance functions as KNN classification.

Distance functions

Euclidean $\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$

Manhattan $\sum_{i=1}^{k}|x_i - y_i|$

Minkowski $\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$

The above three distance measures are only valid for continuous variables. In the case of

The prediction using a single neighbor is just the target value of the nearest neighbor.

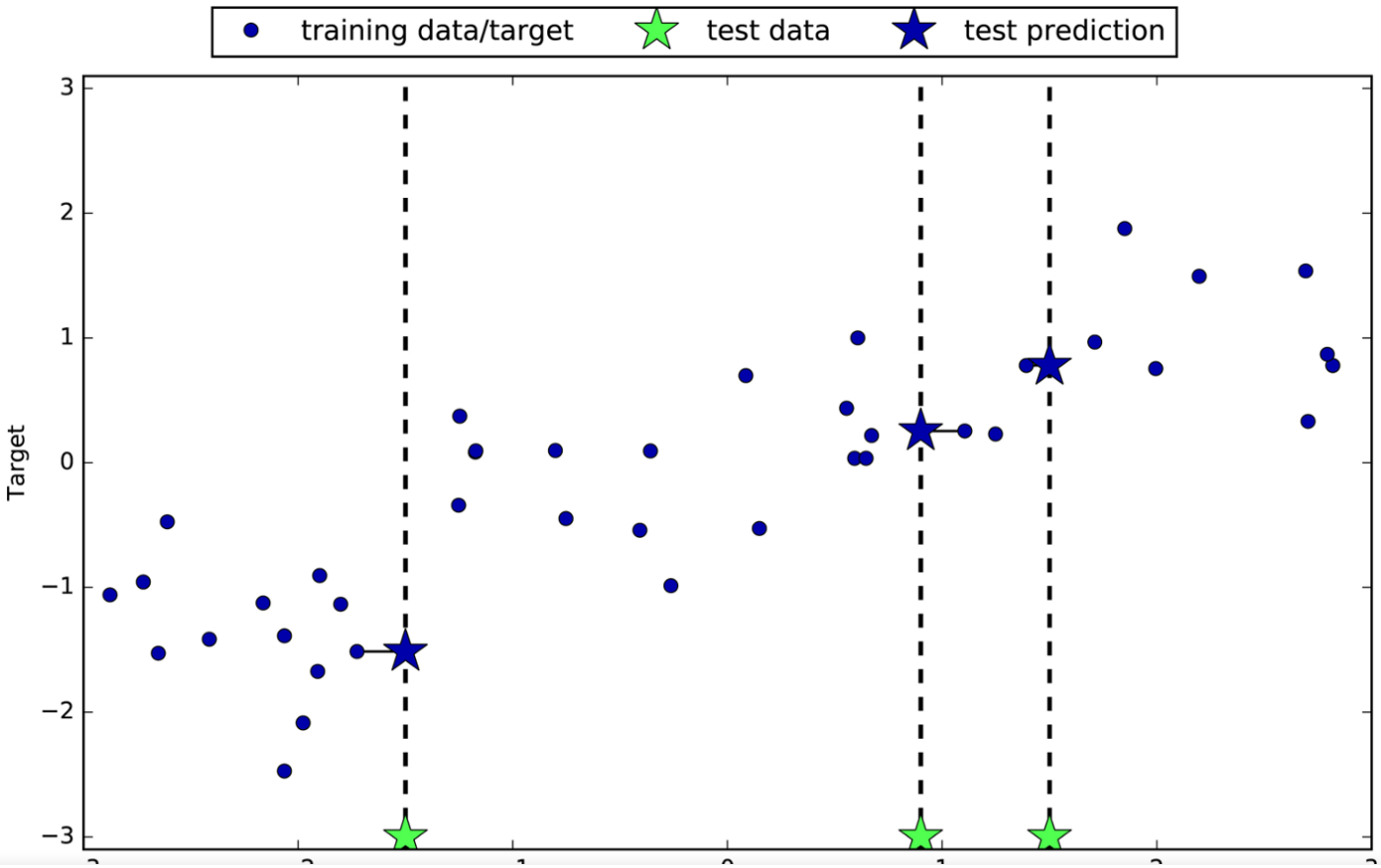
Let's go to hands on, in this article I use the dataset from mglearn, the first step if you don't have package in your note book, install in cmd/anaconda prompt..

```
pip install mglearn
```

After that, you can plot k-neighbors regression with n_neighbors = 1.

```
import mglearn
import matplotlib.pyplot as plt

mglearn.plots.plot_knn_regression(n_neighbors=1)
```



Again, this k-neighbors regression just use 1 n_neighbors, you can use more than the single closest neighbor for regression, and the prediction is the average or mean of relevant neighbors. Let us see…

```
mglearn.plots.plot_knn_regression(n_neighbors=3)
```

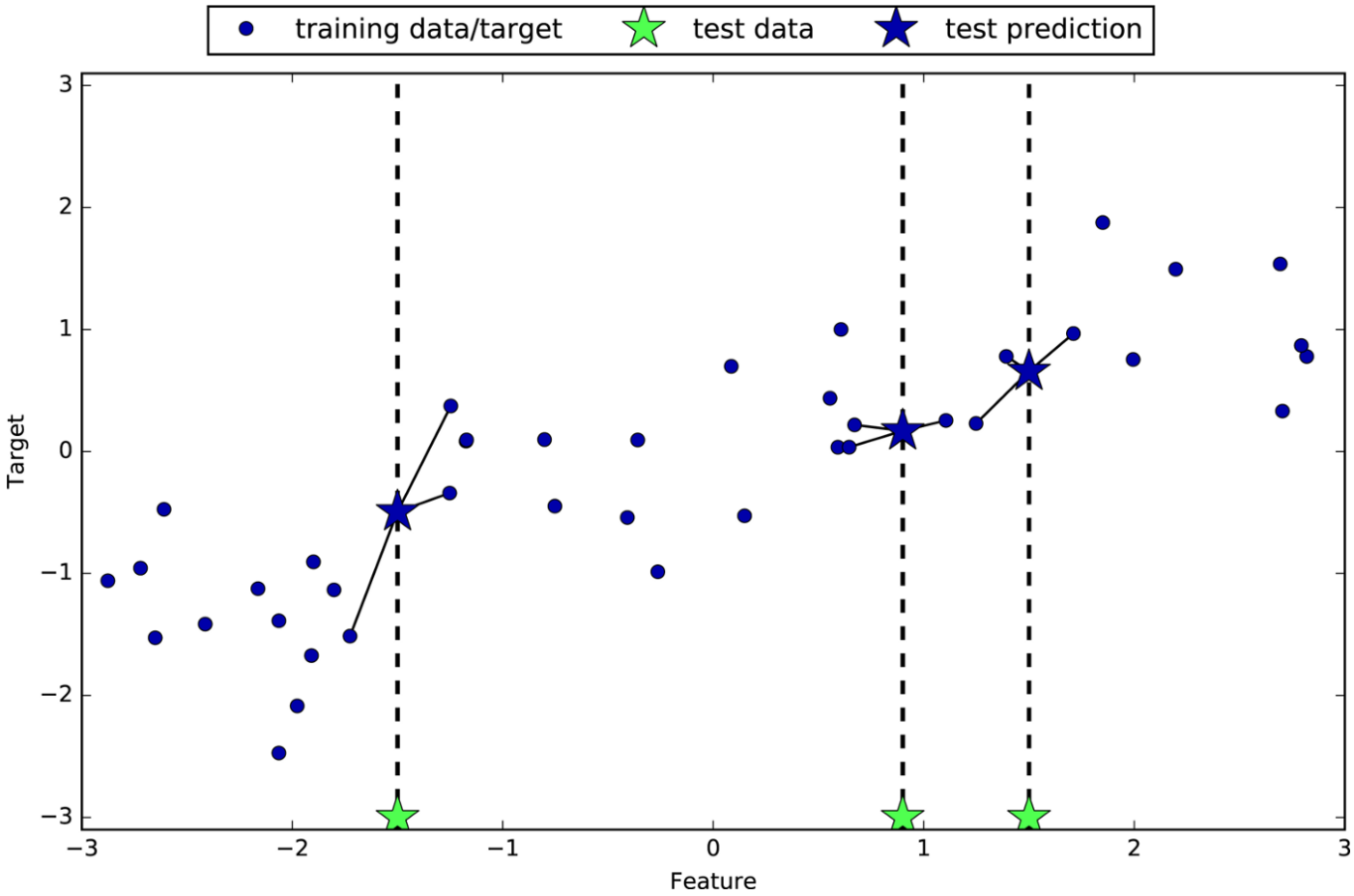


figure 2 . predictions make by three- nearest-neighbors regression on the wave dataset

Now we can make predict on the test data use knn regresson with n_neightbors = 3

```
from sklearn.neighbors import KNeighborsRegressor
X, y = mglearn.datasets.make_wave(n_samples=40)
```

```
reg = KNeighborsRegressor(n_neighbors=3)

# fit the model using the training data and training targets
reg.fit(X_train, y_train)
```

If you have done the above, you can use your model on test data

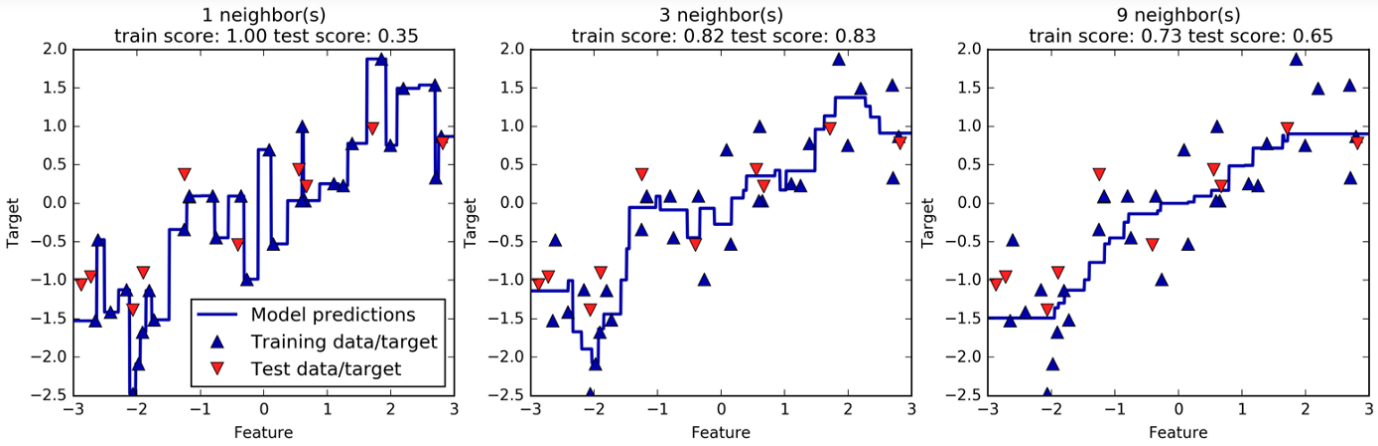
```
print(reg.score(reg.score(X_test, y_test)))
```

out : 0.83

ANALYZING KNEIGHBORS REGRESSOR

We can analyse how accuracy gets affected by n_neighbors: We can use different value 3 n_neighbors, and explain where good value n_neighbors for model.

```
fig, axes = plt.subplots(1, 3, figsize=(15, 4))
# create 1,000 data points, evenly spaced between -3 and 3
line = np.linspace(-3, 3, 1000).reshape(-1, 1)
for n_neighbors, ax in zip([1, 3, 9], axes):
    # make predictions using 1, 3, or 9 neighbors
    reg = KNeighborsRegressor(n_neighbors=n_neighbors)
    reg.fit(X_train, y_train)
    ax.plot(line, reg.predict(line))
    ax.plot(X_train, y_train, '^', c=mglearn.cm2(0),
             markersize=8)
    ax.plot(X_test, y_test, 'v', c=mglearn.cm2(1), markersize=8)
    ax.set_title("{} neighbor(s)\n train score: {:.2f} test
              score: {:.2f}".format(n_neighbors,
              reg.score(X_train, y_train),reg.score(X_test,
              y_test)))
    ax.set_xlabel("Feature")
    ax.set_ylabel("Target")
axes[0].legend(["Model predictions", "Training data/target","Test
    data/target"], loc="best")
```



As we can see from the plot, using only a single neighbor, each point in the training set has an obvious influence on the predictions, and the predicted values go through all of the data points. This leads to a very unsteady prediction. Considering more neighbors leads to smoother predictions, but these do not fit the training data as well.

ref : Andreas C.Muller and Sarah Guido. 2017. Introduction to machine learning with pyhton