k-Nearest Neighbors Classifier

Michela Papandrea

University of Applied Sciences and Arts of Southern Switzerland

michela.papandrea@supsi.ch

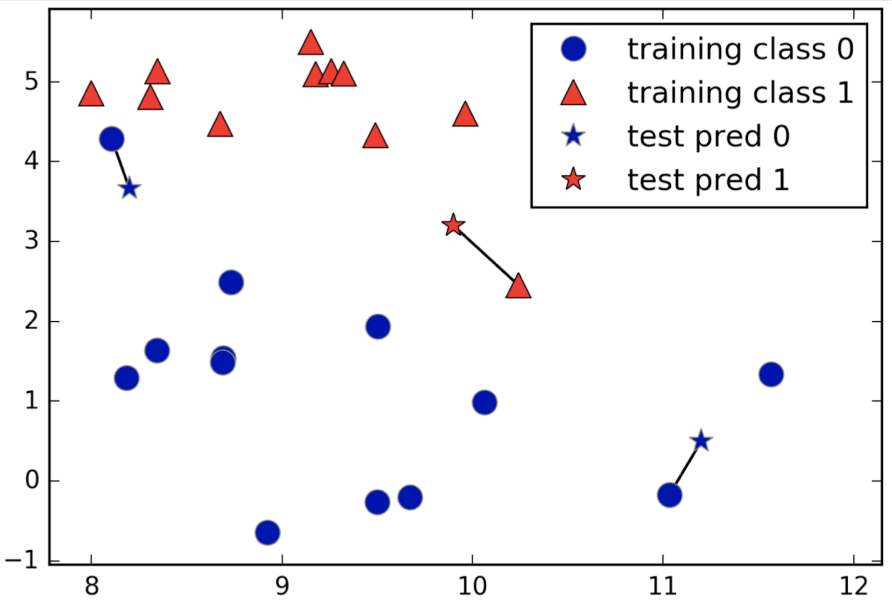
k-Nearest Neighbors

k-NN Algorithm

- simplest ML algorithm
- building the model consists only of storing the training dataset.
- making a prediction for a new data point: consists in finding the closest data points in the training dataset (its *nearest neighbors*)
- *Multiclass classification*: we count how many neighbors belong to each class and predict the most common class.

k-Neighbors classification

- Simplest version: the k-NN algorithm considers exactly one nearest neighbor (the closest training data point): **one-nearest-neighbor**
- The prediction is simply the known output for this training point.

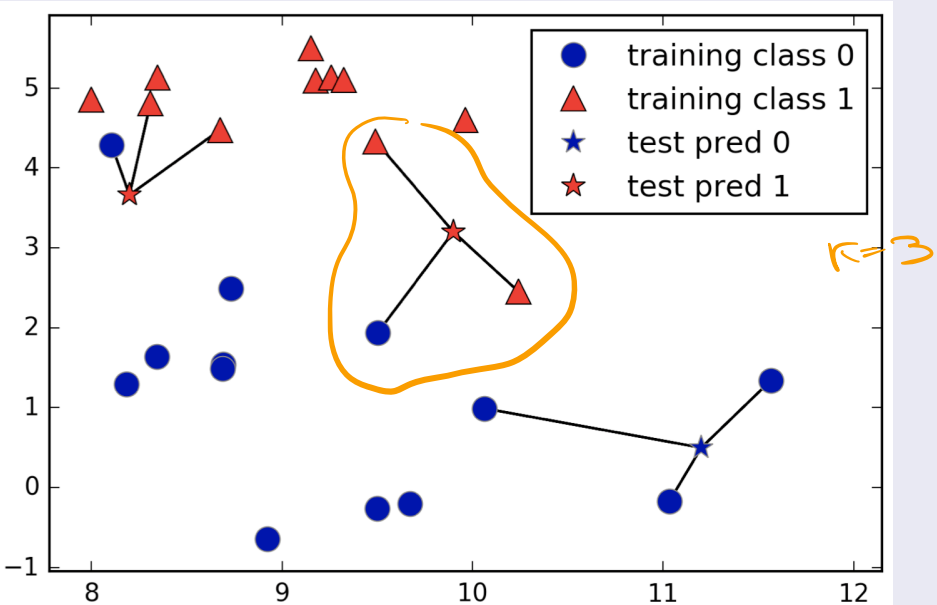


Example: three new data points(stars) in figure connected with the closest point in the training set. The prediction of the one-nearest-neighbor algorithm is the label of that point (shown by the color of the cross).

k-Neighbors classification

k-Nearest Neighbors

- **k-NN**: it considers an arbitrary number, k, of neighbors.
- with k neighbor: *voting* is used to assign a label.
- It assigns the class that is more frequent: the *majority class* among the k-nearest neighbors.



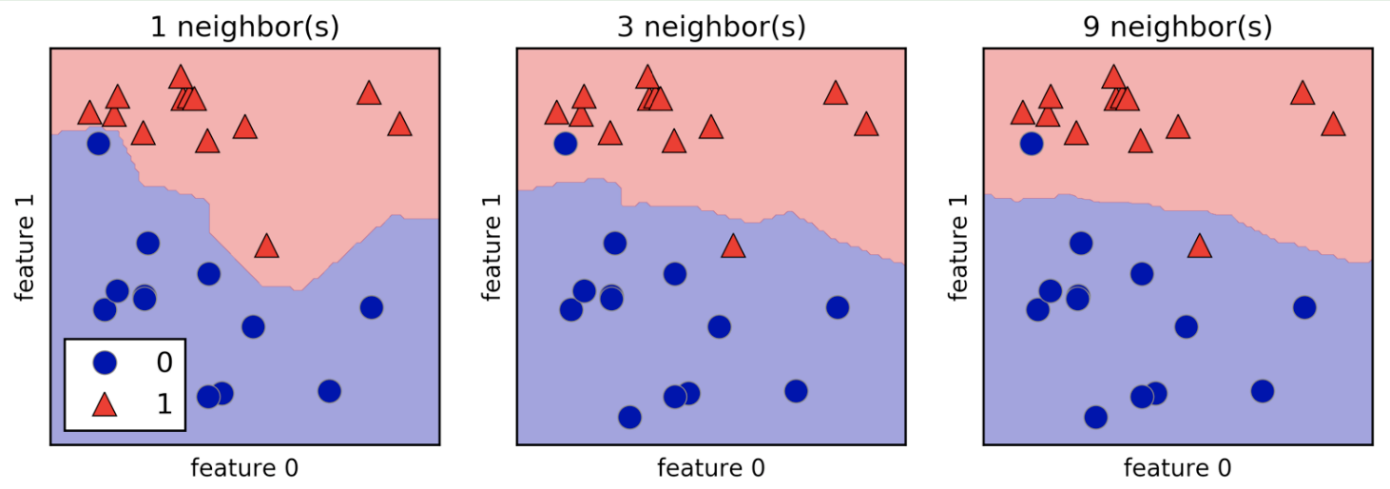
Example: the prediction for the new data point at the top left is not the same as the prediction when we used only one neighbor.

k-Neighbors classification

```
import mglearn
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

X, y = mglearn.datasets.make_forge()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(X_train, y_train)

print("Test set accuracy: {:.2f}".format(clf.score(X_test, y_test)))
```

Analyzing k-Neighbors Classifier

- two-dimensional dataset: we can also illustrate the prediction for all possible test points in the xy-plane.
- plane color represents the class that would be assigned to a point in this region.
- **decision boundary** is clearly visible: it is the line between where the algorithm assigns class 0 versus where it assigns class 1.



k = # samples

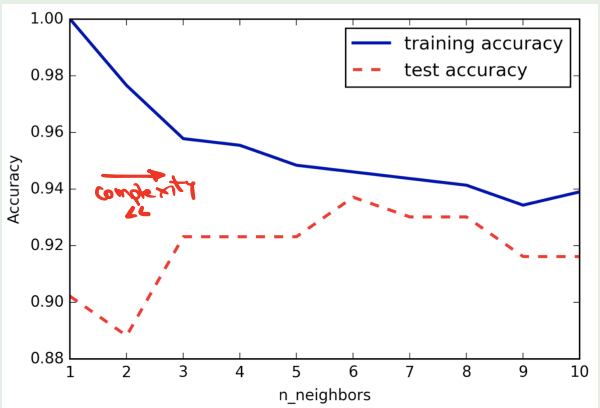
Analyzing k-Neighbors Classifier

- **A single neighbor** $\Rightarrow$ decision boundary follows the training data closely.
- Considering more neighbors leads to a *smoother decision boundary* $\Rightarrow$ simpler model
- Considering **few** neighbors corresponds to *high model complexity*
- Considering **many** neighbors corresponds to *low model complexity*
- Extreme case where the number of neighbors is the number of all data points in the training set $\Rightarrow$ each test point would have exactly the same neighbors (all training points) and all predictions would be the same: the class that is most frequent in the training set.

Complexity VS generalization

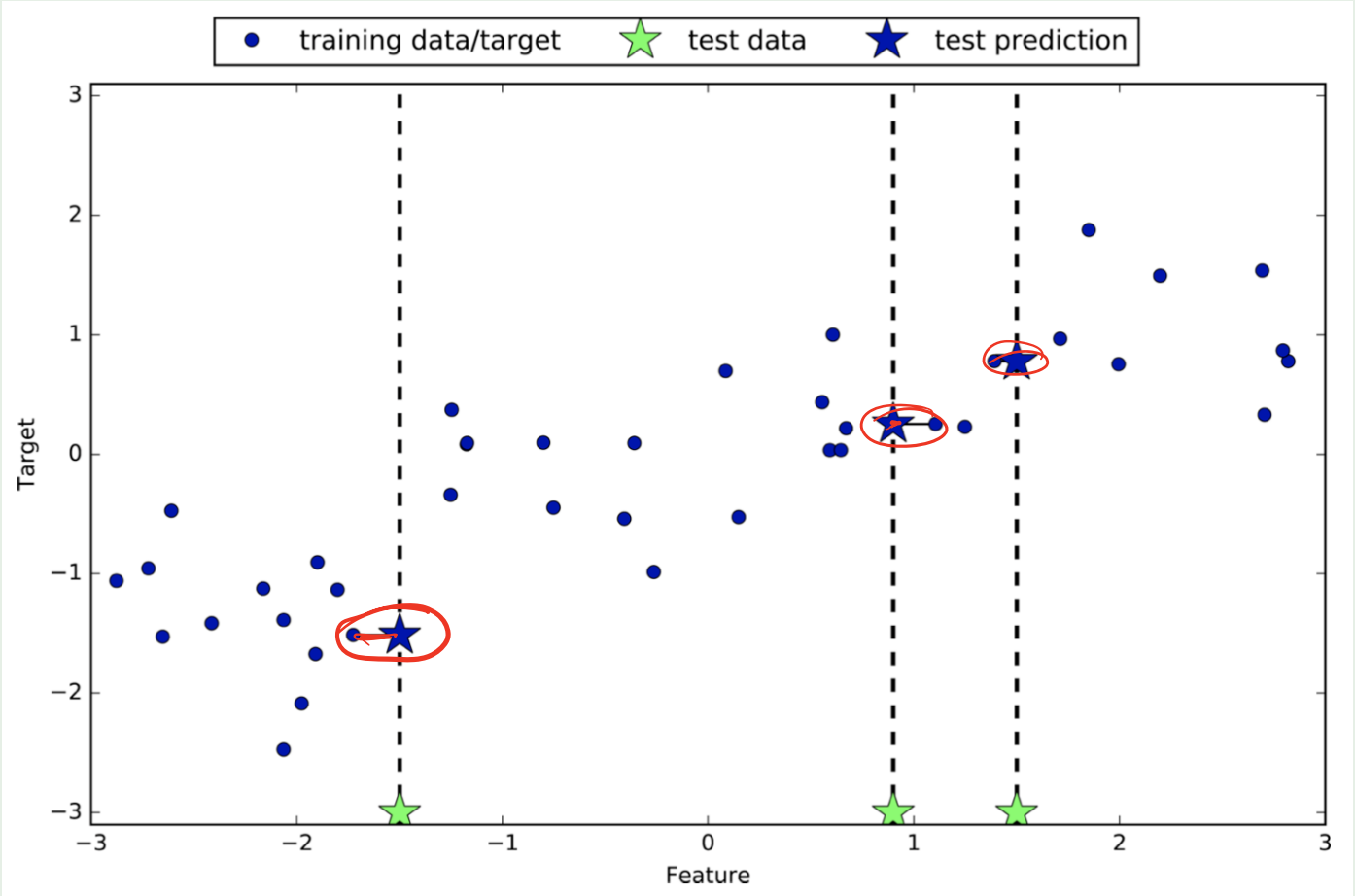
Breast cancer dataset classification accuracy with increasing k

- We can recognize some of the characteristics of overfitting and underfitting
- *single nearest neighbor*
  - the prediction on the training set is perfect
  - the test set accuracy is low
  - the model is too complex
- *when more neighbors are considered*:
  - the model becomes simpler and the training accuracy drops
  - the test set accuracy is higher
  - when neighbors=10, the model is too simple and performance is even worse
- The best performance is somewhere in the middle, using around six neighbors.
- It is good to keep the scale of the plot in mind (worst performance is 88% accuracy)



k-neighbors regression

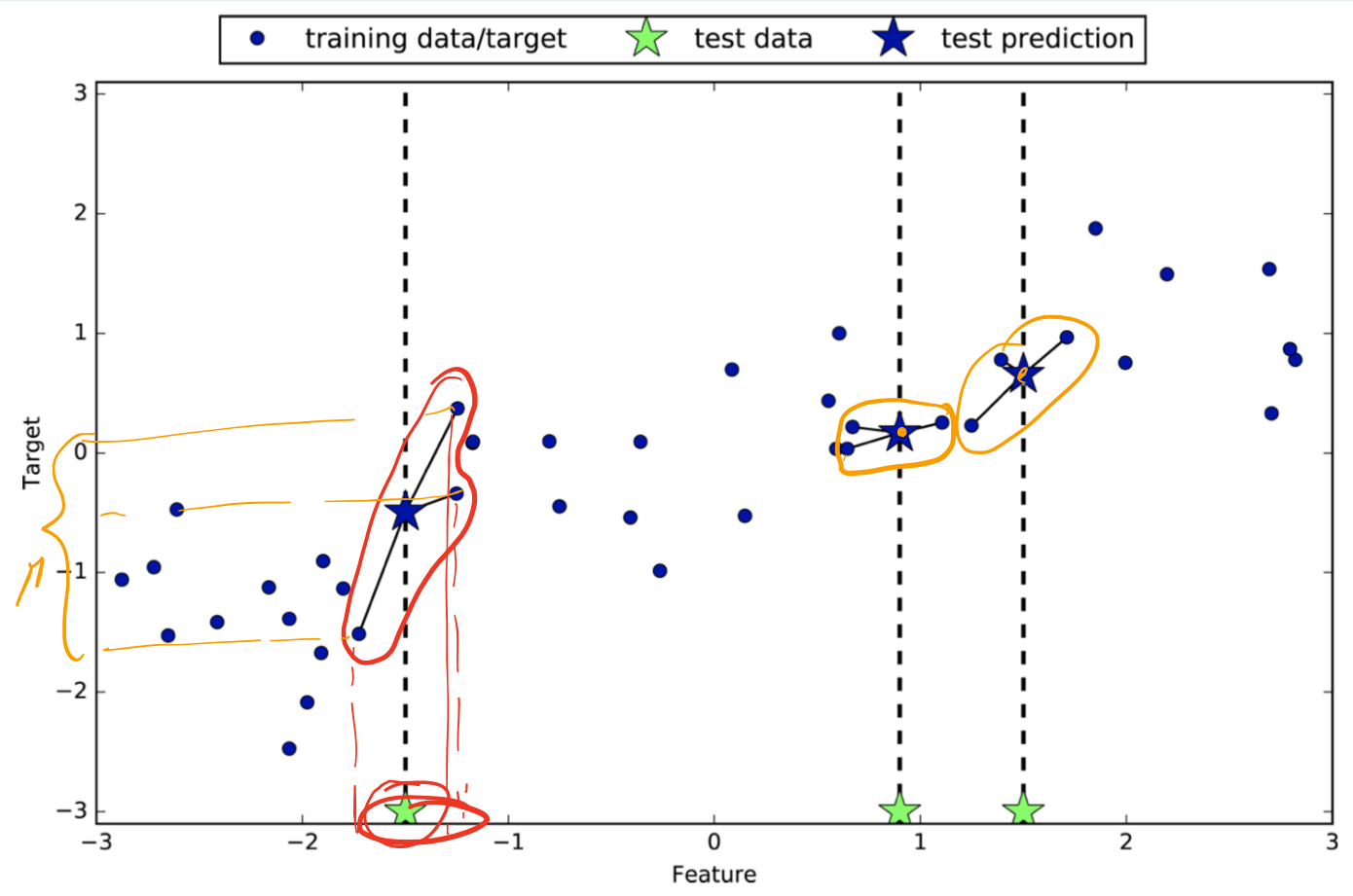
Example: 1-NN Regression on `wave` dataset



k-neighbors regression

Example: 3-NN Regression on `wave` dataset

When using multiple nearest neighbors, the prediction is the average, or mean, of the relevant neighbors



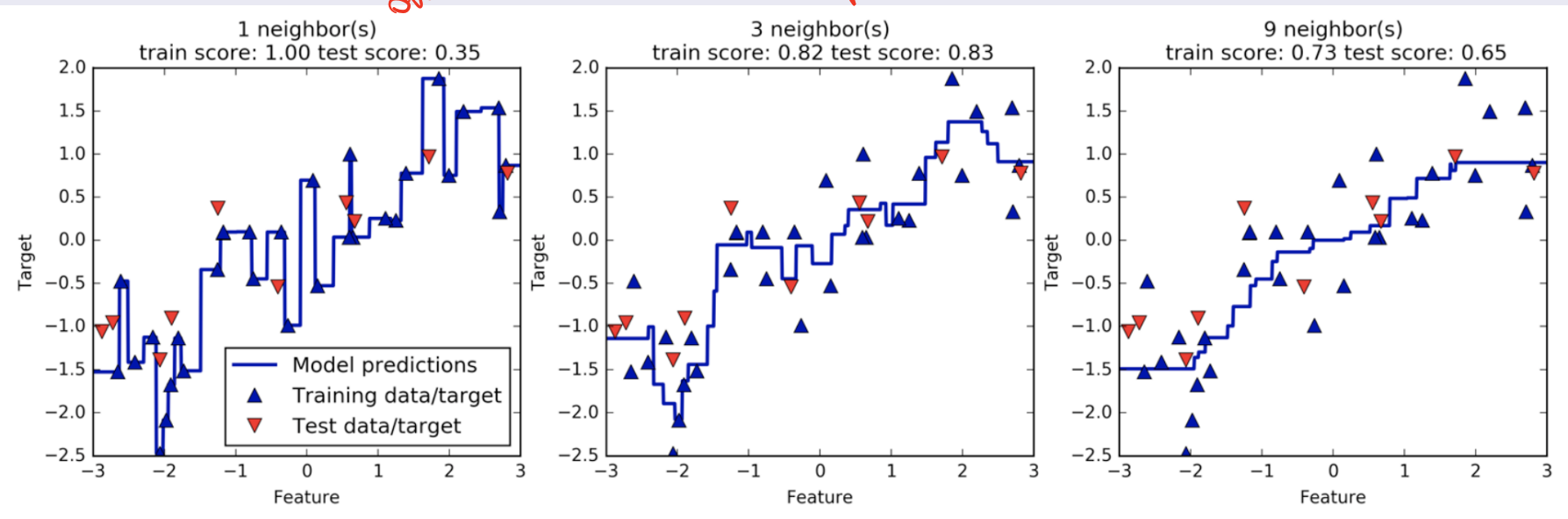
k-neighbors regression

```
from sklearn.neighbors import KNeighborsRegressor

X, y = mglearn.datasets.make_wave(n_samples=40)

# split the wave dataset into a training and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# instantiate the model and set the number of neighbors to consider to 3
reg = KNeighborsRegressor(n_neighbors=3)

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

print("Test set R^2: {:.2f}".format(reg.score(X_test, y_test)))
```

```
Test set R^2: 0.83
```

- R^2 score is the coefficient of determination
- it is a measure of goodness of a prediction for a regression model
- yields a score between 0 and 1.
- a value of 1 corresponds to a perfect prediction
- a value of 0 corresponds to a constant model that just predicts the mean of the training set responses

Analyzing k-Neighbors Regressor



- For a one-dimensional dataset, we can see what the predictions look like for all possible feature value
- 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.

Strengths, weaknesses, and parameters

Parameters

- **Number of neighbors**: this parameter needs to be adjusted, however, very common values are 3 or 5 neighbors.
- **Distance metric**: Euclidean distance is frequently used, which works well in many settings. Hovever, it is possible to choose a different metric (OOS).

Strenghts and weaknesses

- the model is very easy to understand
- often provides reasonable performance without a lot of adjustments.
- Using this algorithm is a good baseline method to try before considering more advanced techniques.
- Building the nearest neighbors model is usually very fast, but when your training set is very large (either in number of features or in number of samples) prediction can be slow.
- When using the k-NN algorithm, it's important to preprocess your data.
- This approach often does not perform well on datasets with many features (hundreds or more), and it does particularly badly with datasets where most features are 0 most of the time (*sparse datasets*).

References

Andreas C. Müller & Sarah Guido (2017)

Introduction to Machine Learning with Python

Chapter 2: Supervised Learning (pp. 37 – 46)

Published by O'Reilly Media, Inc..