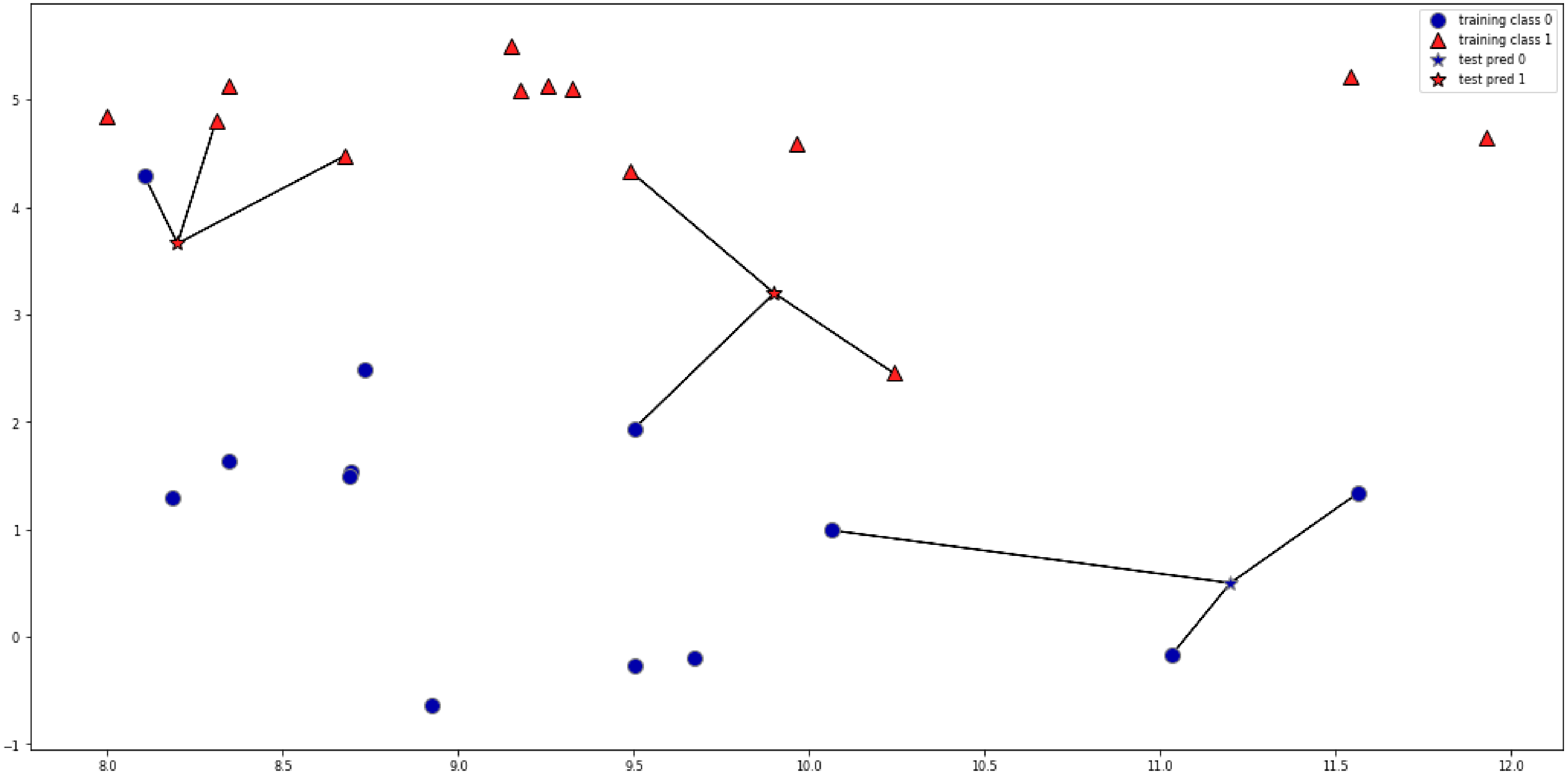
Recap: k-Nearest Neighbor

- Building the model consists only of storing the training dataset.
- To make a prediction, the algorithm finds the *k* closest data points in the training dataset
    - Classification: predict the most frequent class of the k neighbors
    - Regression: predict the average of the values of the k neighbors
    - Both can be weighted by the distance to each neighbor
- Main hyper-parameters:
    - Number of neighbors (k). Acts as a regularizer.
    - Choice of distance function (e.g. Euclidean)
    - Weighting scheme (uniform, distance,...)
- Model:
    - Representation: Store training examples (e.g. in KD-tree)
    - Typical loss functions:
        - Classification: Accuracy (Zero-One Loss)
        - Regression: Root mean squared error
    - Optimization: None (no model parameters to tune)

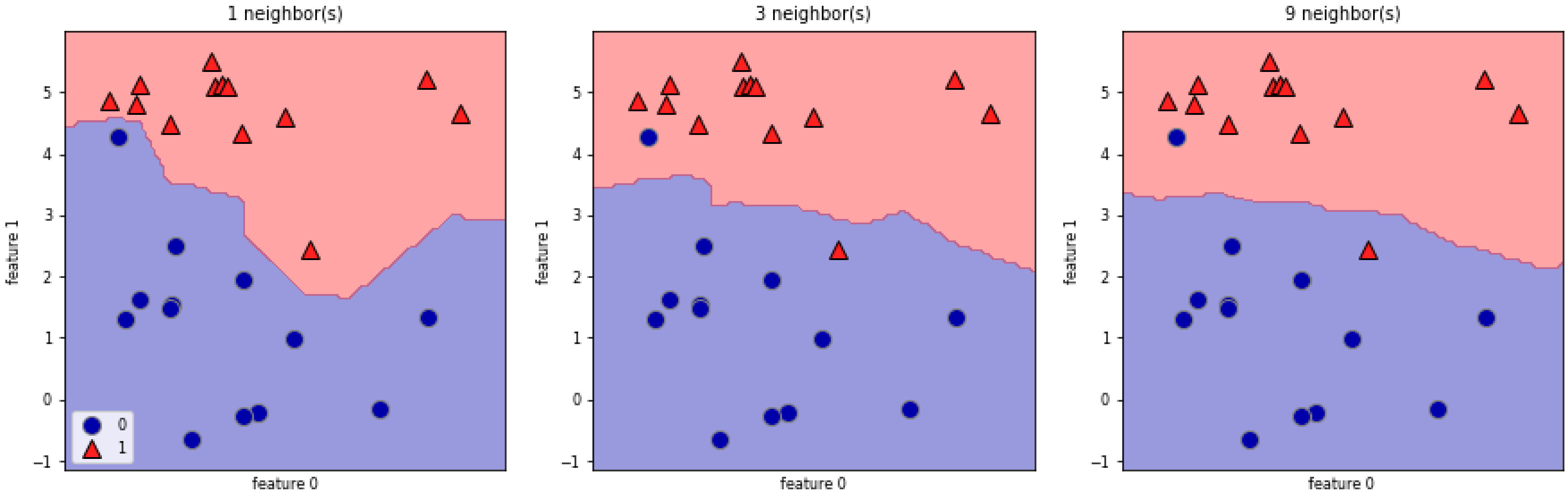
k-Nearest Neighbor Classification

k=1: look at nearest neighbor only: likely to overfit
k>1: do a vote and return the majority (or a confidence value for each class)



Analysis

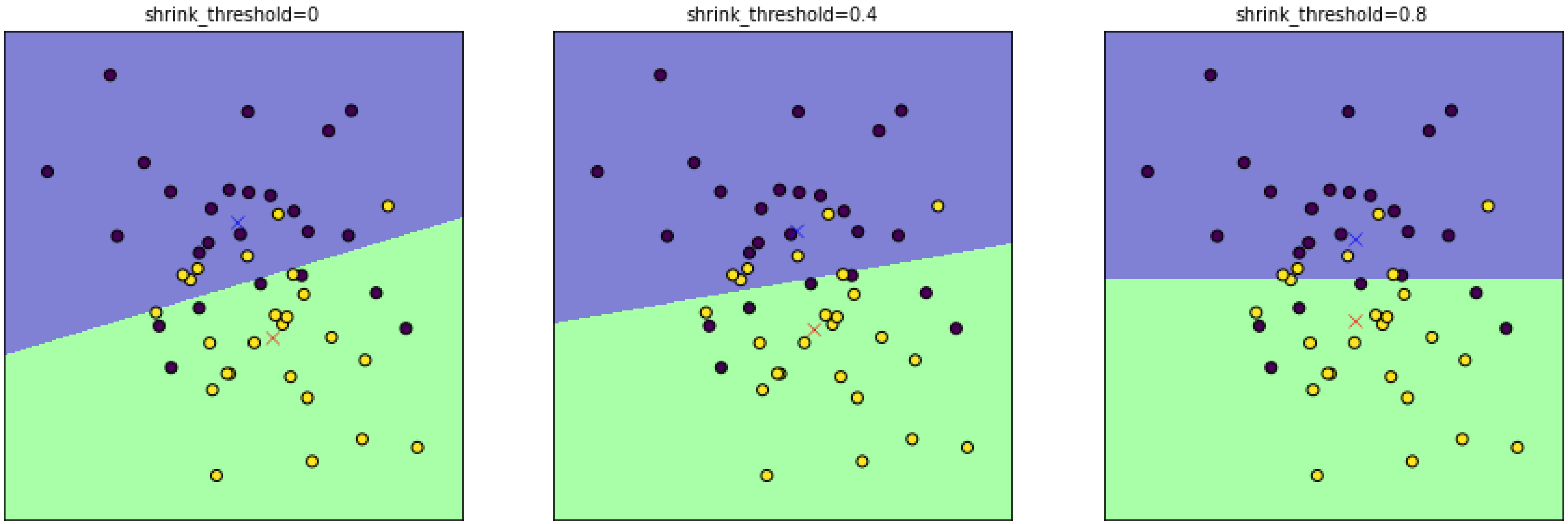
We can plot the prediction for each possible input to see the *decision boundary*



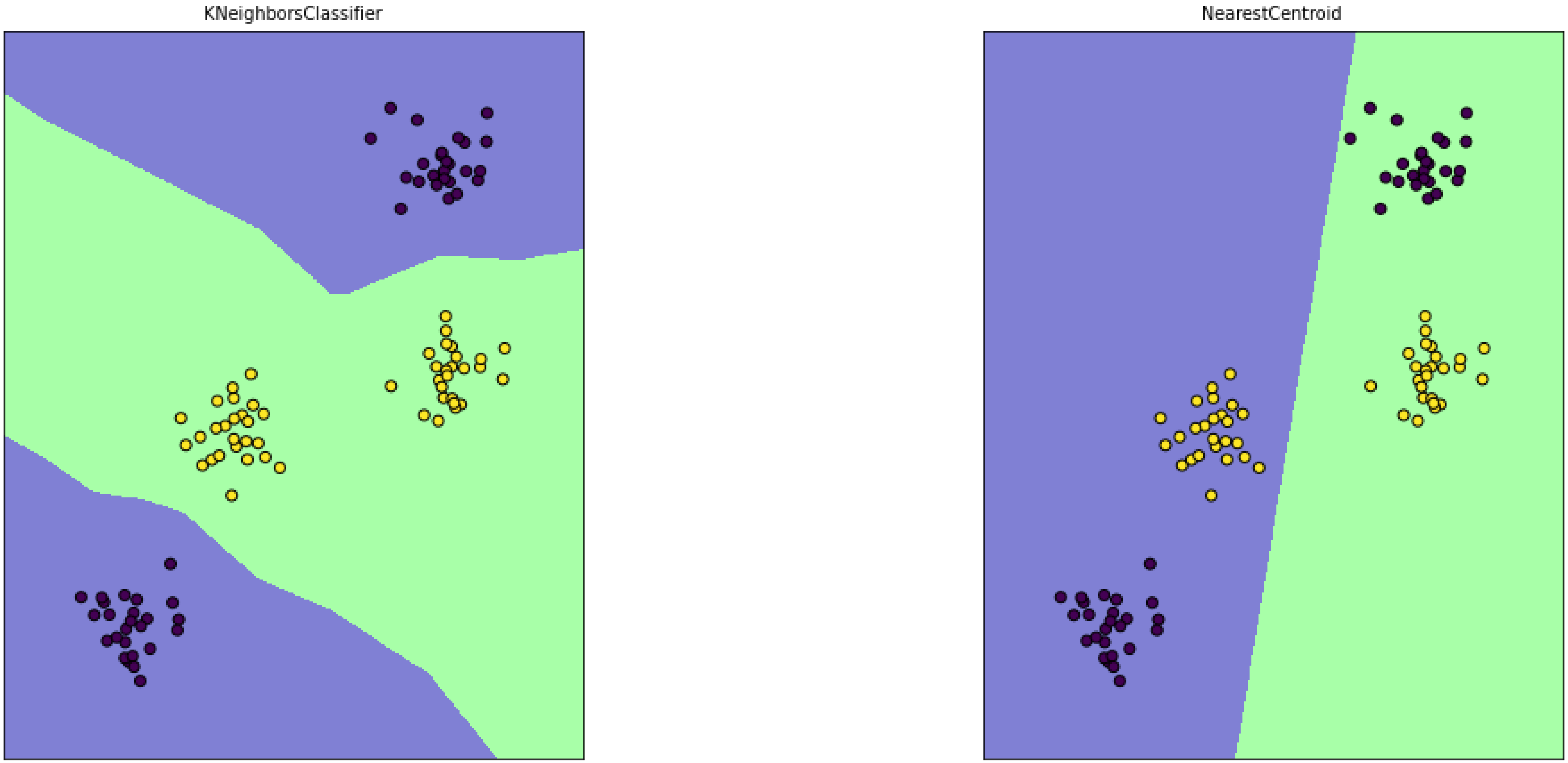
Using few neighbors corresponds to high model complexity (left), and using many neighbors corresponds to low model complexity and smoother decision boundary (right).

Nearest Shrunken Centroid

- Nearest Centroid: Represents each class by the centroid of its members.
    - Parameteric model (while kNN is non-parametric)
- Regularization is possible with the `shrink_threshold` parameter
    - Shrinks (scales) each feature value by within-class variance of that feature
    - Soft thresholding: if feature value falls below threshold, it is set to 0
    - Effectively removes (noisy) features



Note: Nearest Centroid suffers when the data is not 'convex'



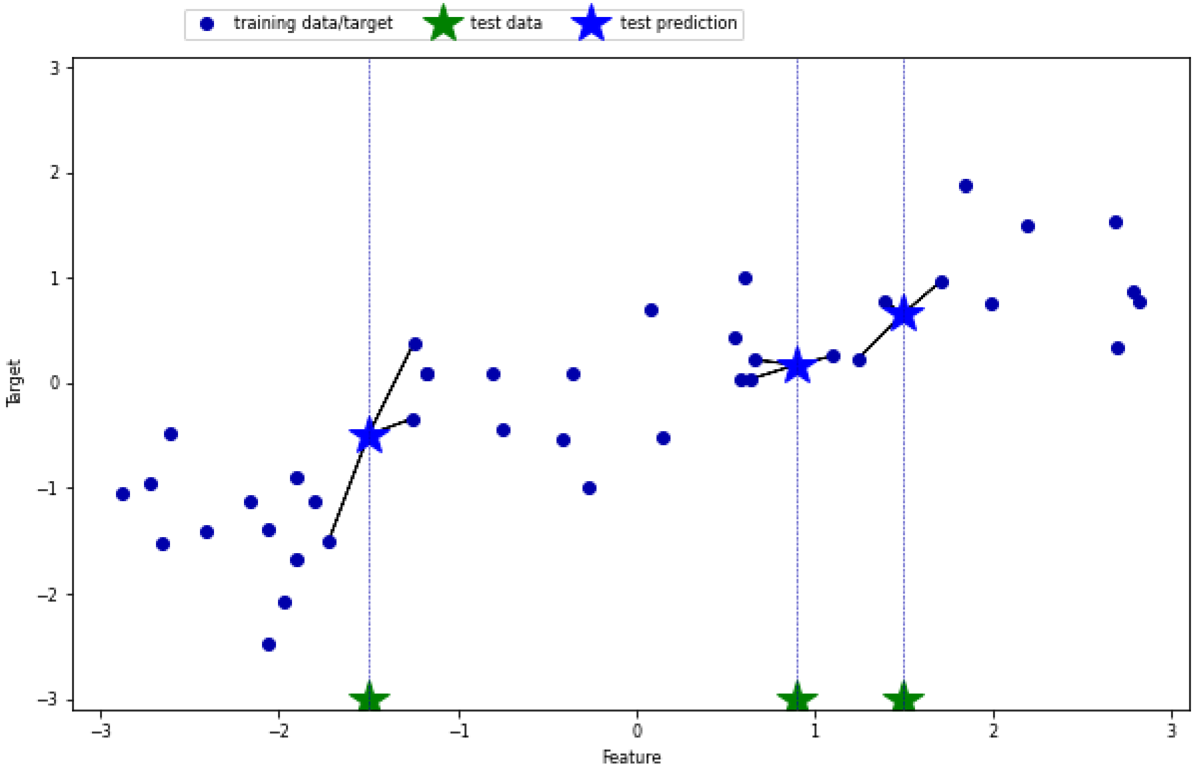
Scalability

With n = nr examples and p = nr features

- Nearest shrunken threshold
    - Fit: $O(n * p)$
    - Memory: $O(nrclasses * p)$
    - Predict: $O(nrclasses * p)$
- Nearest neighbors (naive)
    - Fit: 0
    - Memory: $O(n * p)$
    - Predict: $O(n * p)$
- Nearest neighbors (with KD trees)
    - Fit: $O(p * nlogn)$
    - Memory: $O(n * p)$
    - Predict: $O(k * logn)$

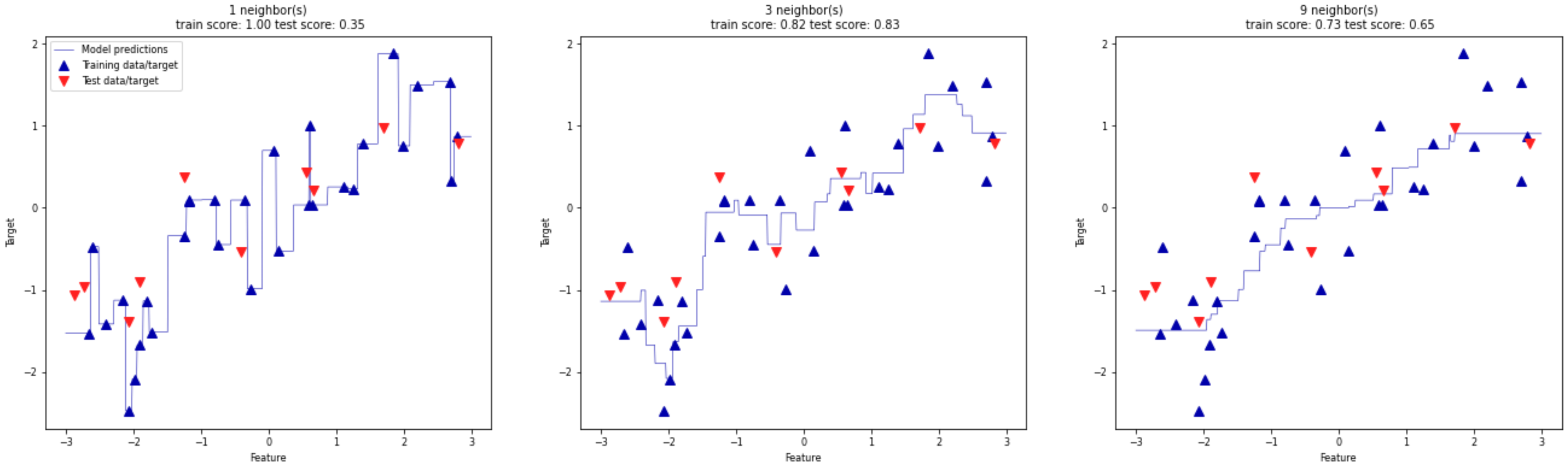
k-Neighbors Regression

k=1: return the target value of the nearest neighbor (overfits easily)
k>1: return the *mean* of the target values of the *k* nearest neighbors



Analysis

We can again output the predictions for each possible input, for different values of *k*.



We see that again, a small *k* leads to an overly complex (overfitting) model, while a larger *k* yields a smoother fit.

kNN: Strengths, weaknesses and parameters

- Easy to understand, works well in many settings
- Training is very fast, predicting is slow for large datasets
- Bad at high-dimensional and sparse data (curse of dimensionality)
- Nearest centroid is a useful parametric alternative, but only if data is (near) linearly separable.