Pattern Recognition

Nearest Neighbors

PHYS 453

Dr Daugherity

ACU
ABILENE CHRISTIAN UNIVERSITY

The Nearest Neighbor

Algorithm:

Classify new point to be the same category as the closest training point

That’s it.

k Nearest Neighbors

Algorithm:

Find k nearest training points and vote

That’s it.

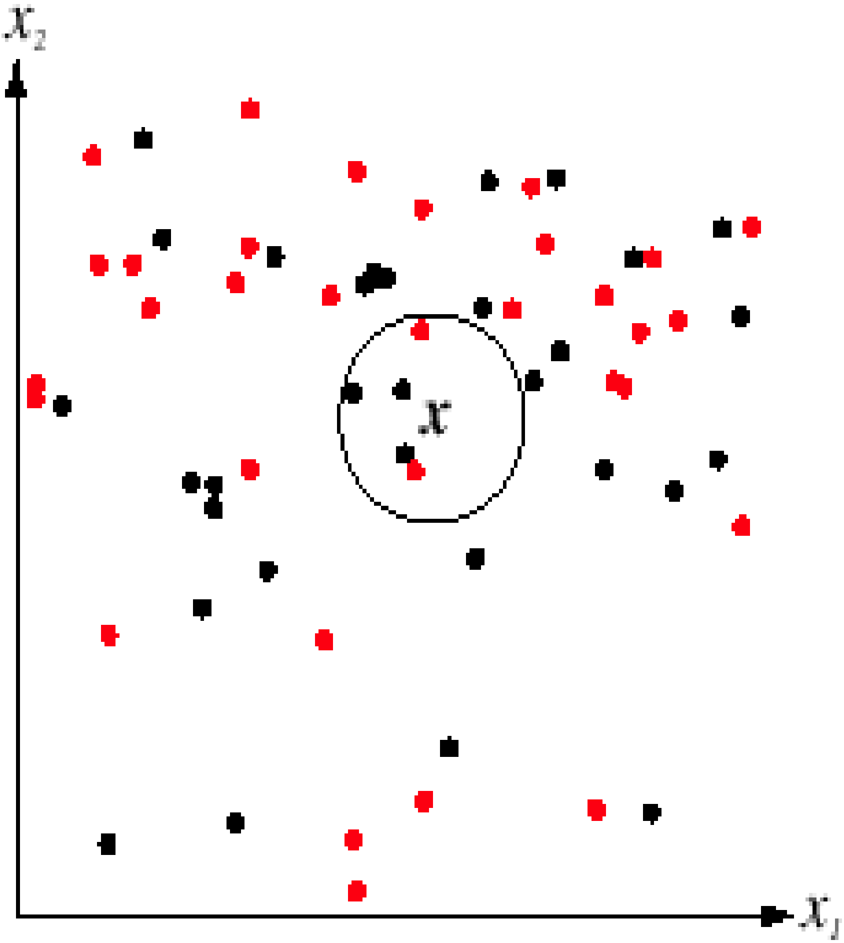
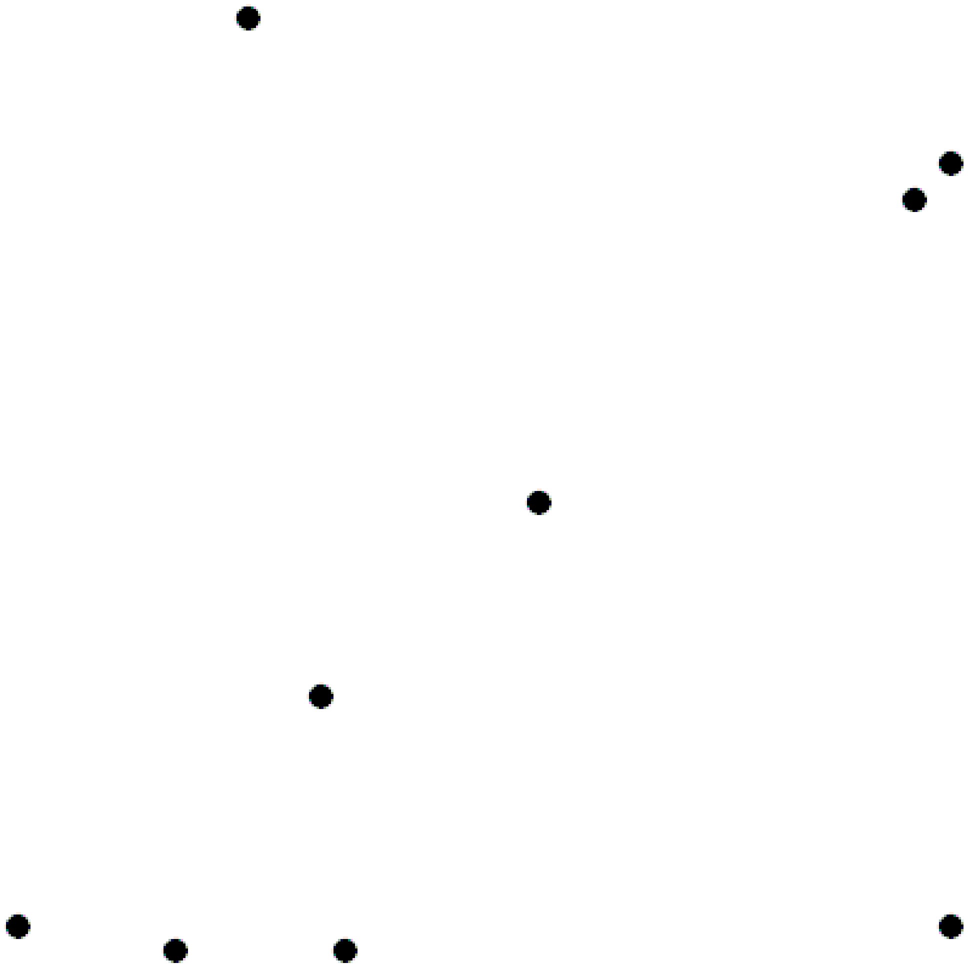


FIGURE 4.15. The k-nearest-neighbor query starts at the test point $\mathbf{x}$ and grows a spherical region until it encloses k training samples, and it labels the test point by a majority vote of these samples. In this $k = 5$ case, the test point $\mathbf{x}$ would be labeled the category of the black points. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

https://en.wikipedia.org/wiki/Voronoi_diagram#/media/File:Voronoi_growth_euclidean.gif



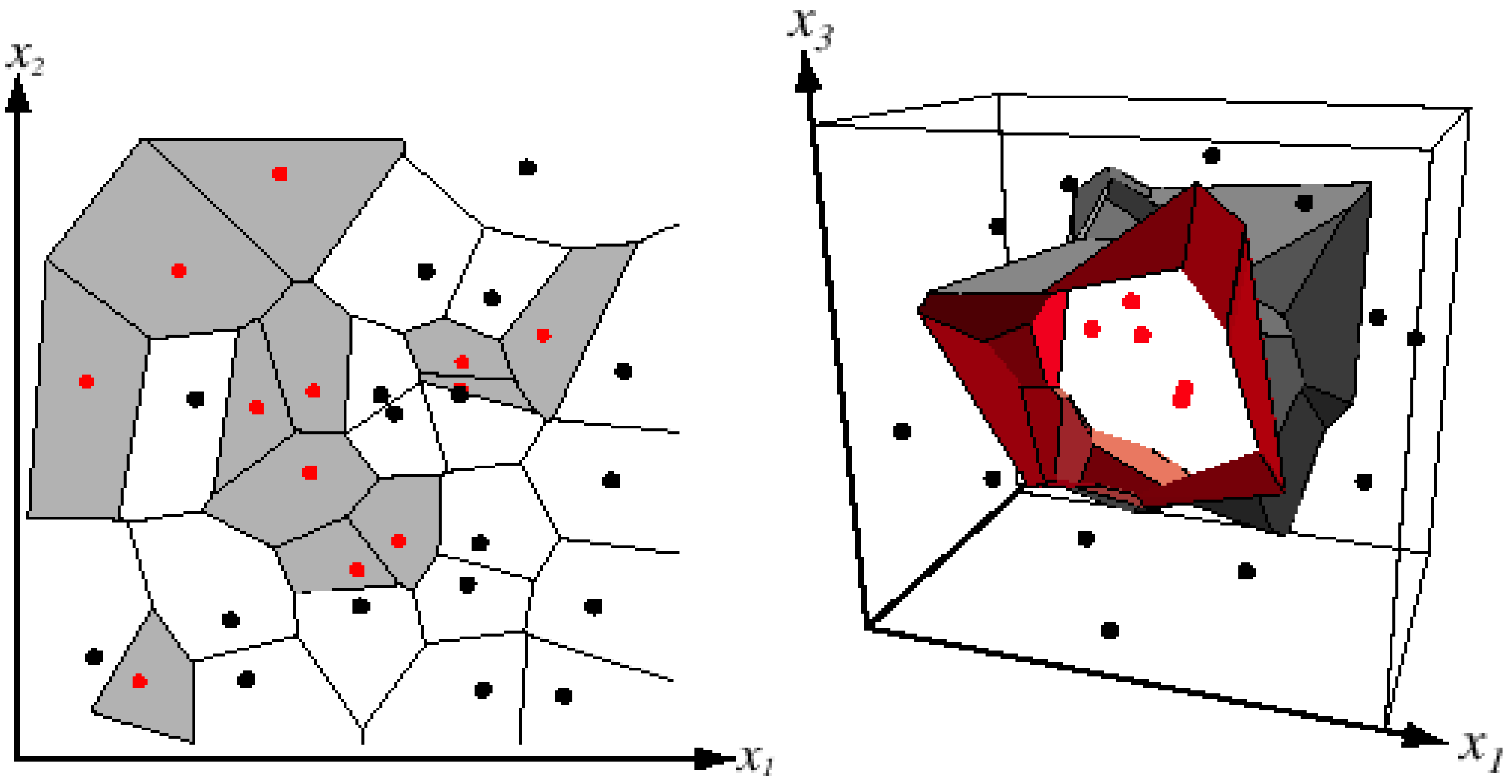
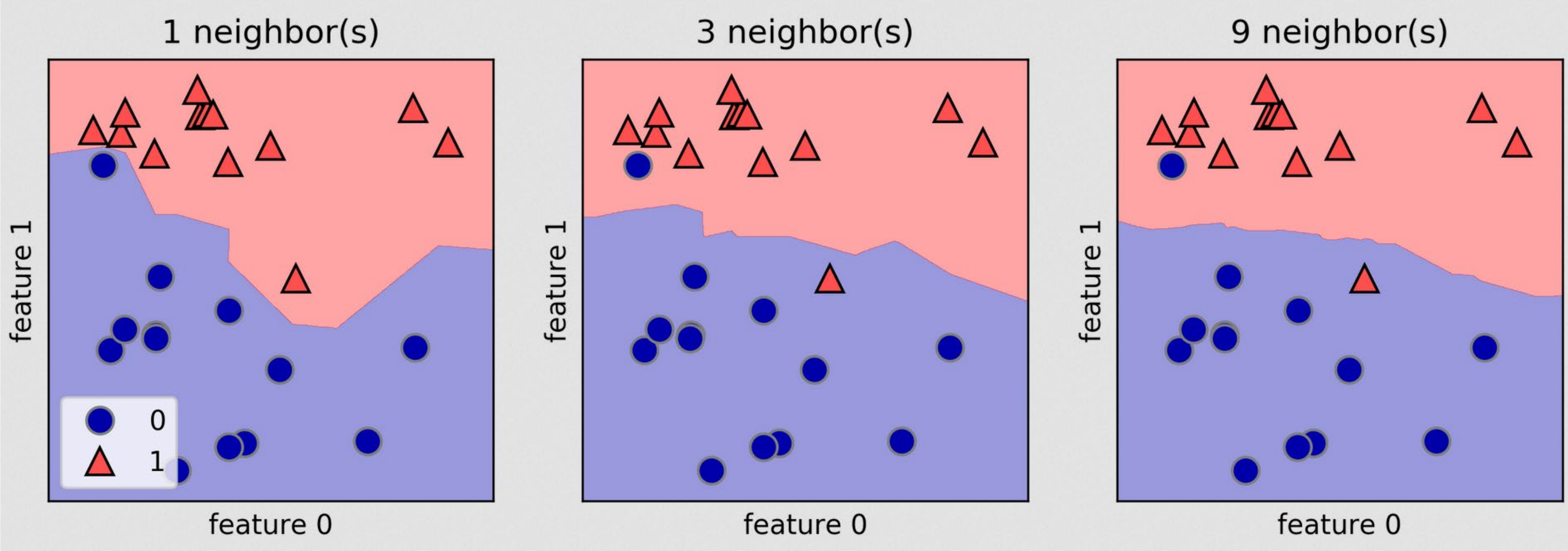


FIGURE 4.13. In two dimensions, the nearest-neighbor algorithm leads to a partitioning of the input space into Voronoi cells, each labeled by the category of the training point it contains. In three dimensions, the cells are three-dimensional, and the decision boundary resembles the surface of a crystal. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

https://github.com/amueller/introduction_to_ml_with_python/blob/master/02-supervised-learning.ipynb



low bias
high variance

high bias
low variance

Notice how changing k affects the red and blue outliers

BIAS VS VARIANCE

A brief disgression

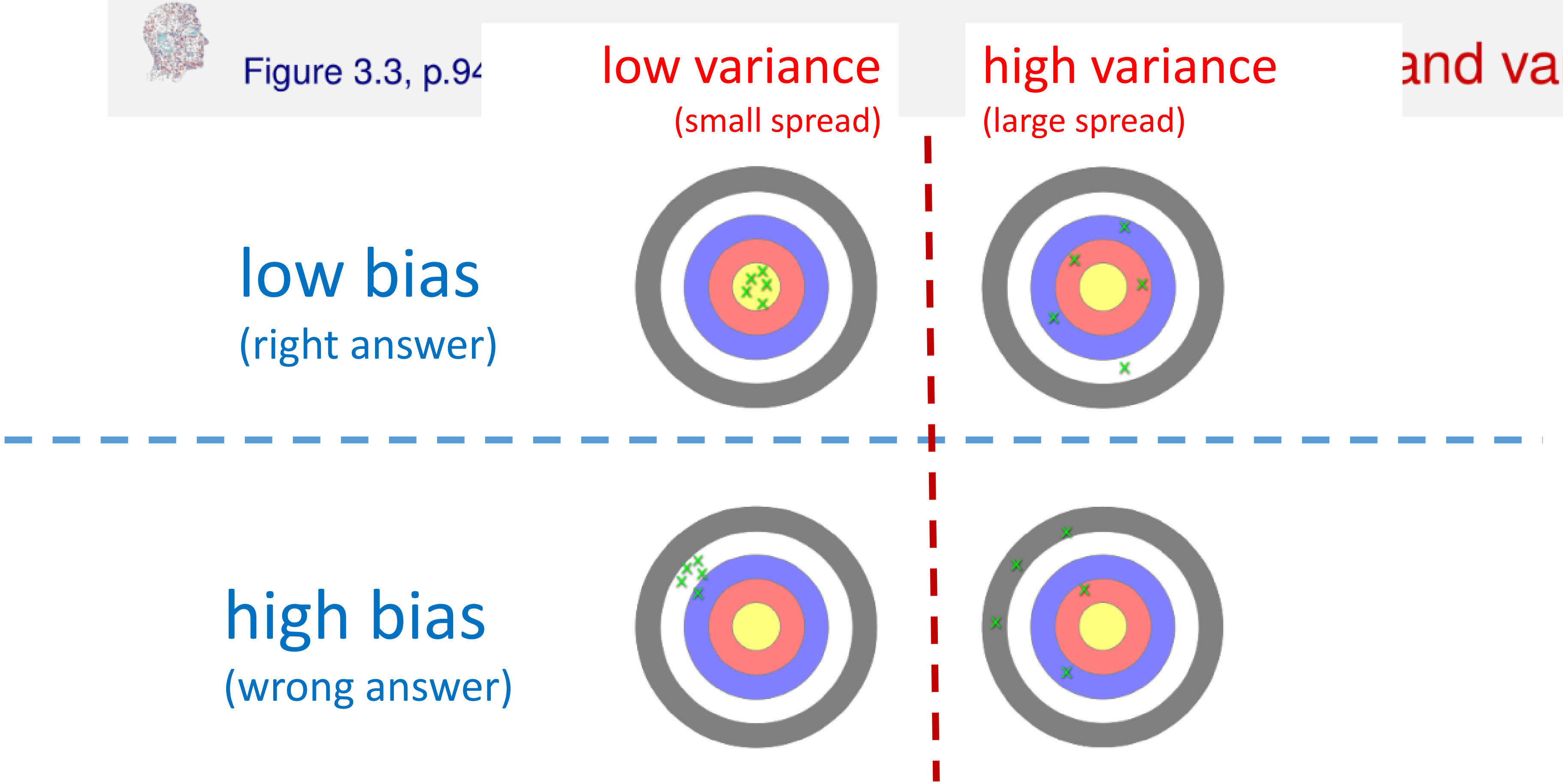
★ Bias and variance I

If we underestimate the number of parameters of the model, we will not be able to decrease the loss to zero, regardless of how much training data we have.

On the other hand, with a larger number of parameters the model will be more dependent on the training sample, and small variations in the training sample can result in a considerably different model.

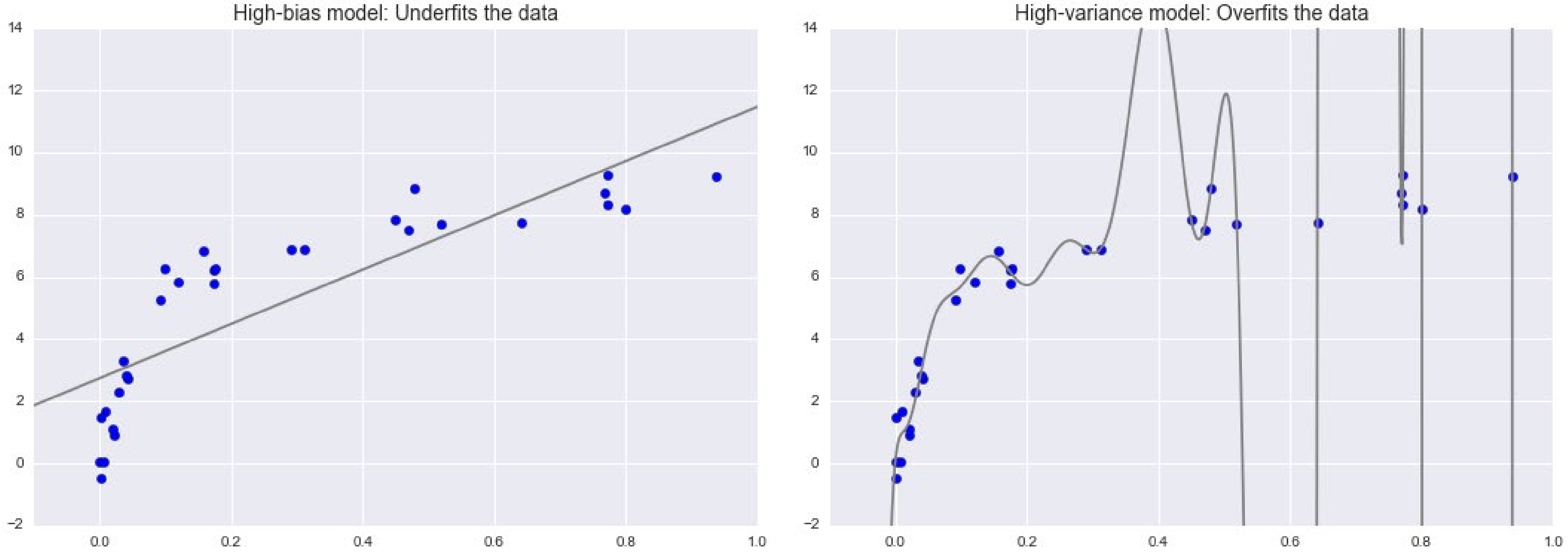
This is sometimes called the *bias–variance dilemma*: a low-complexity model suffers less from variability due to random variations in the training data, but may introduce a systematic bias that even large amounts of training data can't resolve; on the other hand, a high-complexity model eliminates such bias but can suffer non-systematic errors due to variance.

Figure 3.3, p.94 and variance



A dartboard metaphor illustrating the concepts of bias and variance. Each dartboard corresponds to a different learning algorithm, and each dart signifies a different training sample. The top row learning algorithms exhibit low bias, staying close to the bull's eye (the true function value for a particular x) on average, while the ones on the bottom row have high bias. The left column shows low variance and the right column high variance.

http://nbviewer.jupyter.org/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.03-Hyperparameters-and-Model-Validation.ipynb

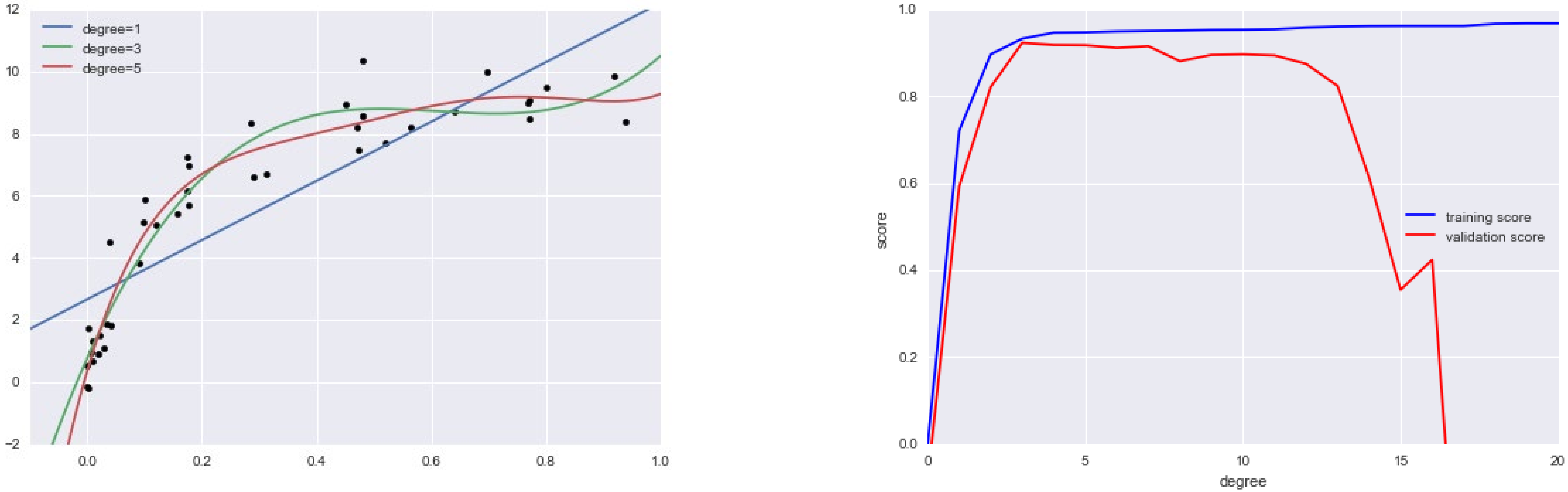


It is clear that neither of these models is a particularly good fit to the data, but they fail in different ways.

The model on the left attempts to find a straight-line fit through the data. Because the data are intrinsically more complicated than a straight line, the straight-line model will never be able to describe this dataset well. Such a model is said to *underfit* the data: that is, it does not have enough model flexibility to suitably account for all the features in the data; another way of saying this is that the model has high *bias*.

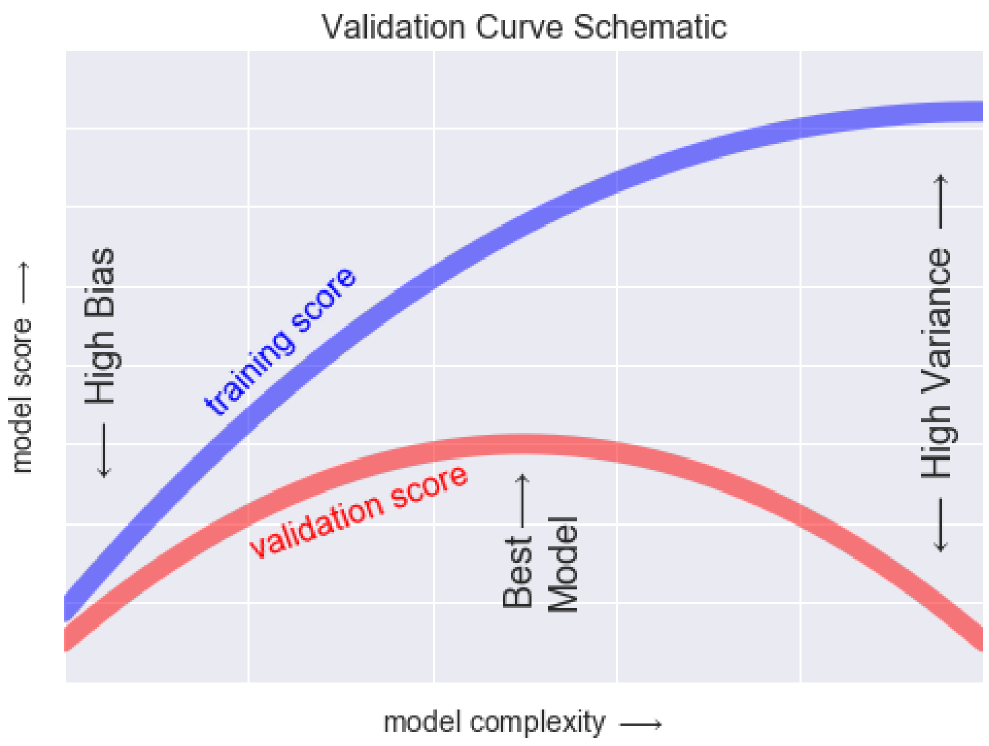
The model on the right attempts to fit a high-order polynomial through the data. Here the model fit has enough flexibility to nearly perfectly account for the fine features in the data, but even though it very accurately describes the training data, its precise form seems to be more reflective of the particular noise properties of the data rather than the intrinsic properties of whatever process generated that data. Such a model is said to *overfit* the data: that is, it has so much model flexibility that the model ends up accounting for random errors as well as the underlying data distribution; another way of saying this is that the model has high *variance*.

http://nbviewer.jupyter.org/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.03-Hyperparameters-and-Model-Validation.ipynb



Fitting data with different degree polynomials. The validation curve shows the n=3 is optimal.

http://nbviewer.jupyter.org/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.03-Hyperparameters-and-Model-Validation.ipynb



The diagram shown here is often called a *validation curve*, and we see the following essential features:

•The training score is everywhere higher than the validation score. This is generally the case: the model will be a better fit to data it has seen than to data it has not seen.

•For very low model complexity (a high-bias model), the training data is under-fit, which means that the model is a poor predictor both for the training data and for any previously unseen data.

•For very high model complexity (a high-variance model), the training data is over-fit, which means that the model predicts the training data very well, but fails for any previously unseen data.

•For some intermediate value, the validation curve has a maximum. This level of complexity indicates a suitable trade-off between bias and variance.

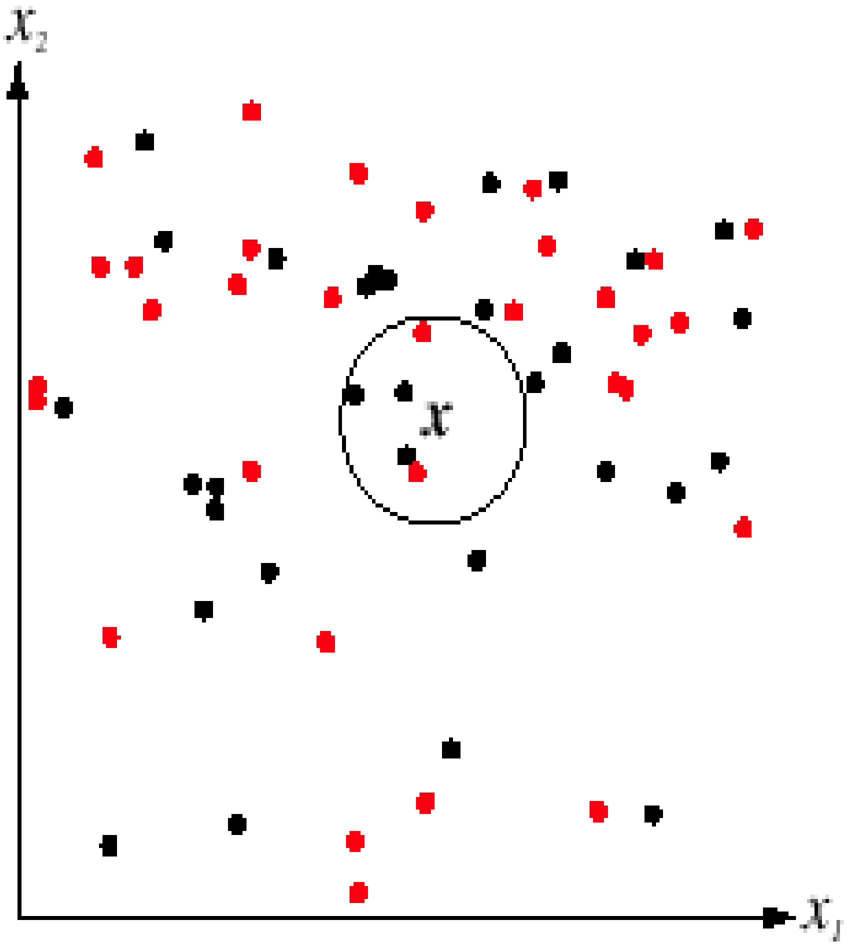
BIAS VS VARIANCE

We now return to our regularly scheduled slides.

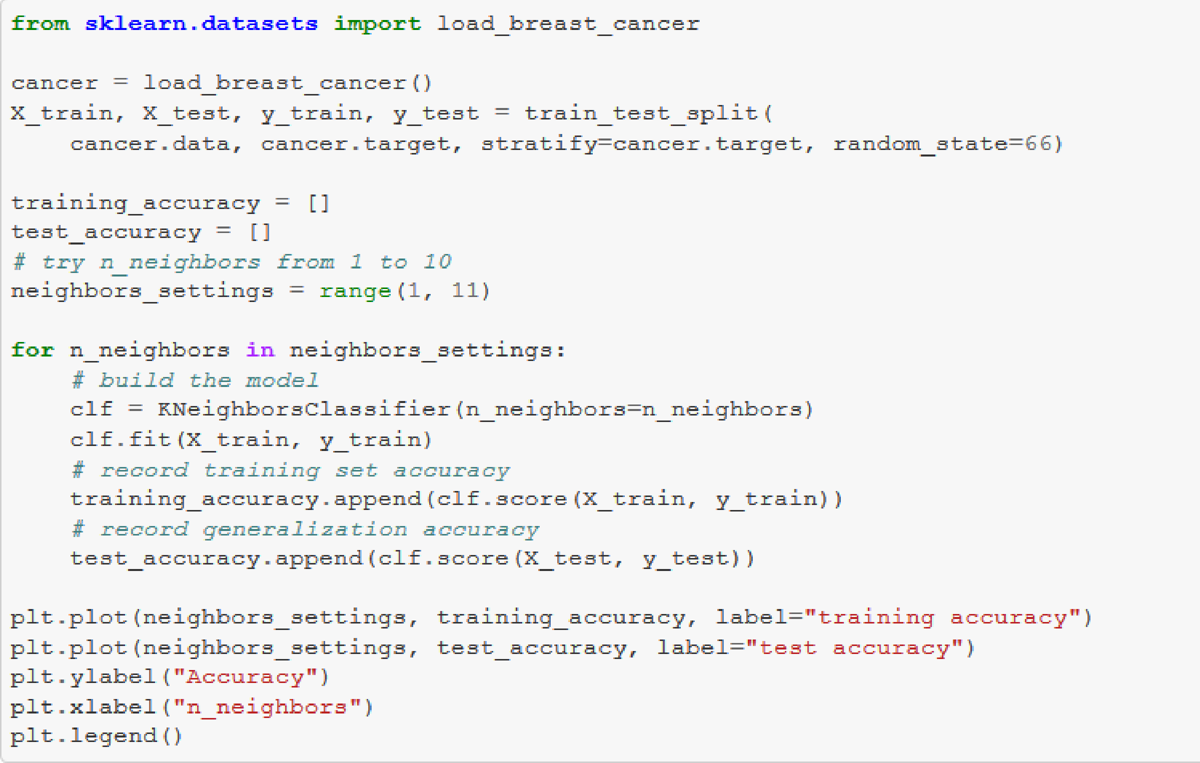
Probability Density Function

Even though the algorithm is simple, it easily generalized to powerful statistics.

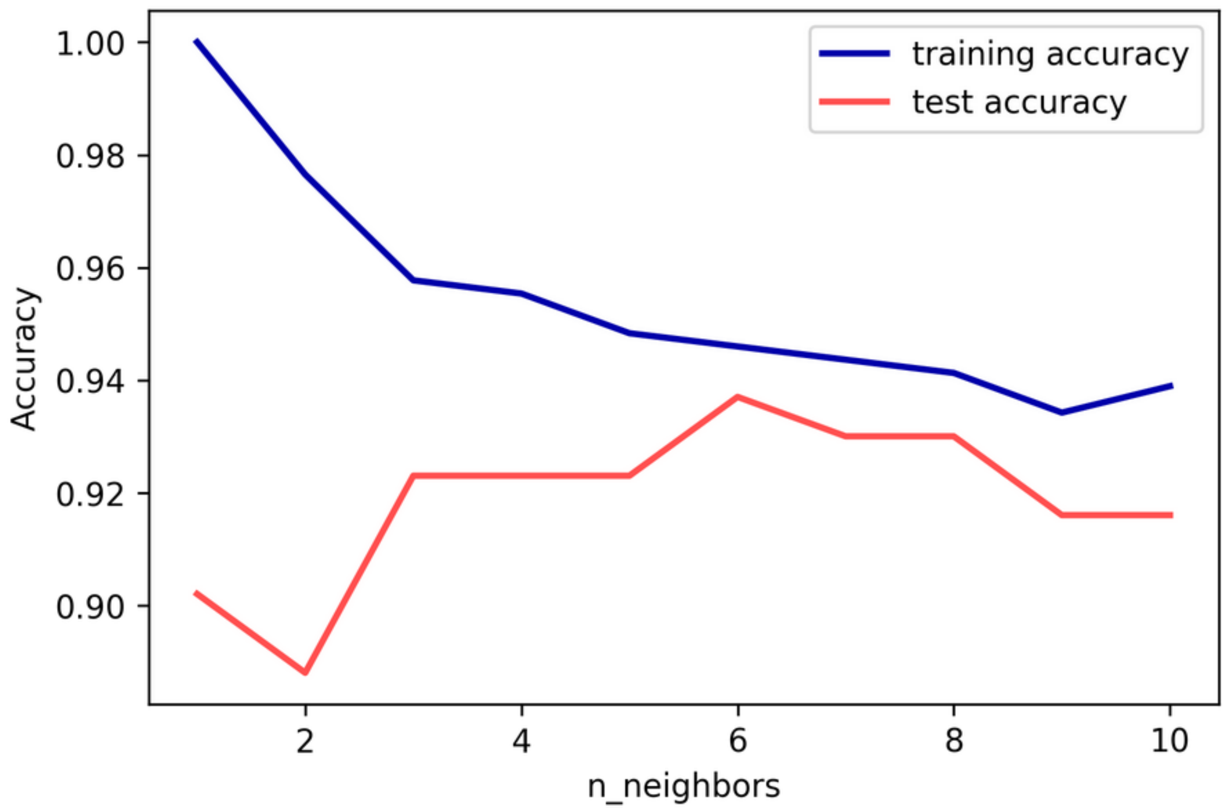
When we find neighbors we are actually estimating the probability density function in that region.



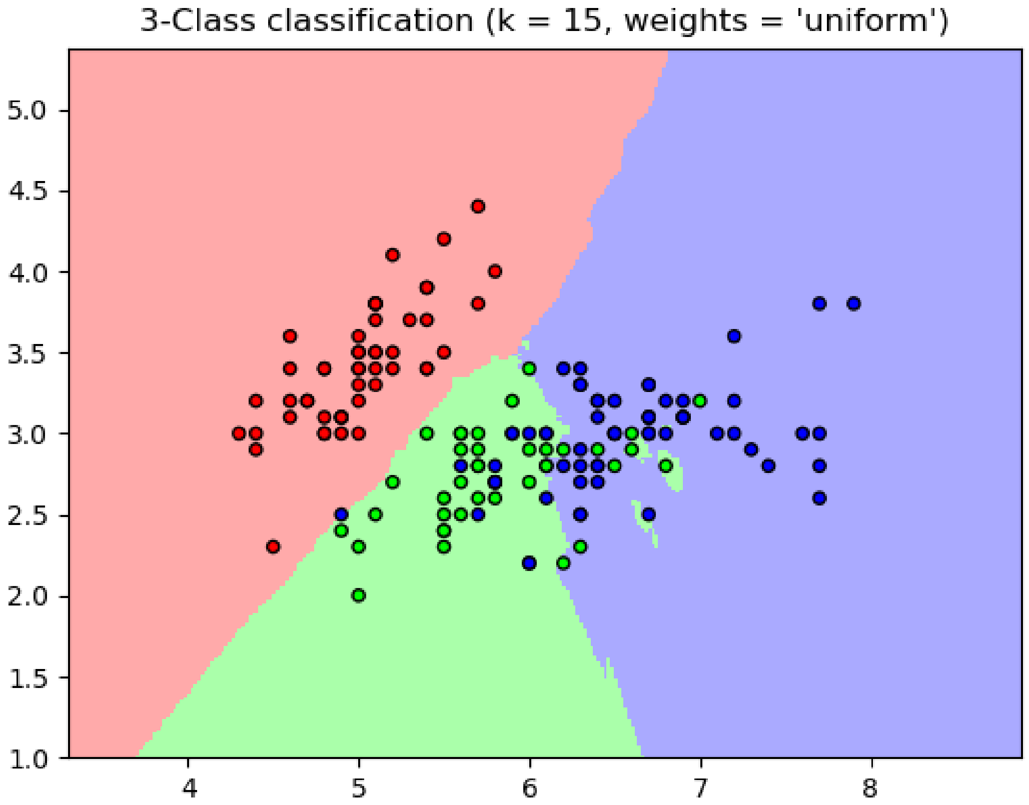
https://github.com/amueller/introduction_to_ml_with_python/blob/master/02-supervised-learning.ipynb



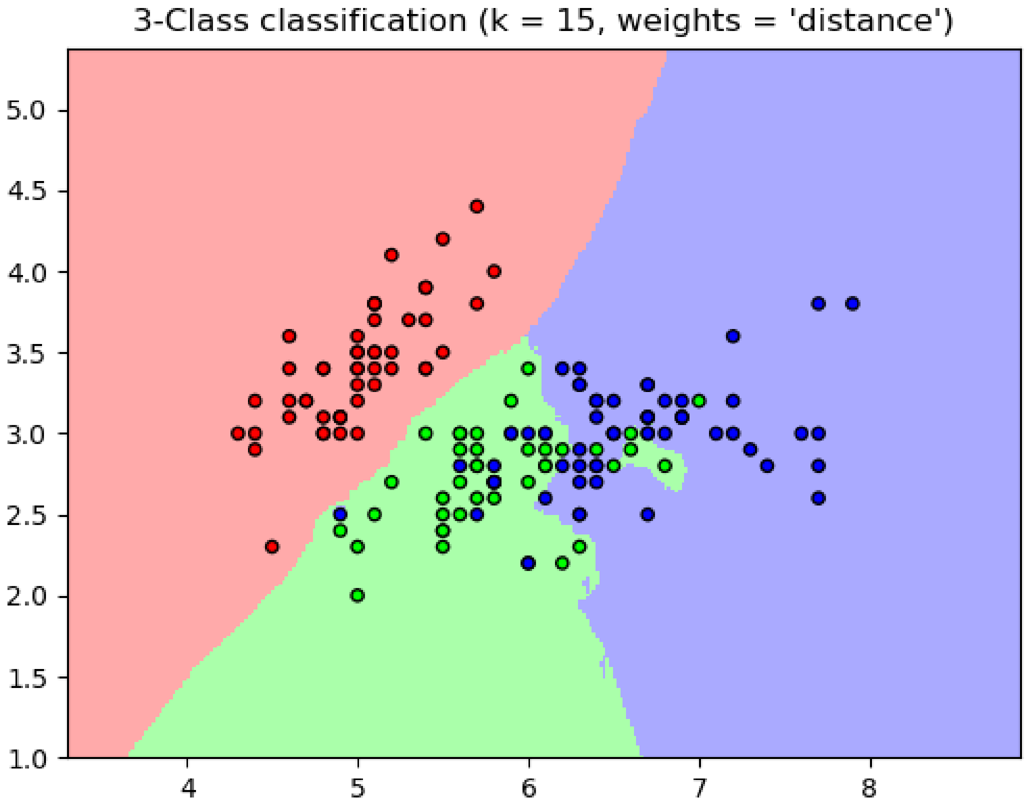
```
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=66)

training_accuracy = []
test_accuracy = []
# try n_neighbors from 1 to 10
neighbors_settings = range(1, 11)

for n_neighbors in neighbors_settings:
    # build the model
    clf = KNeighborsClassifier(n_neighbors=n_neighbors)
    clf.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(clf.score(X_train, y_train))
    # record generalization accuracy
    test_accuracy.append(clf.score(X_test, y_test))

plt.plot(neighbors_settings, training_accuracy, label="training accuracy")
plt.plot(neighbors_settings, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend()
```



Increasing k decreases training accuracy but gives a maximum in test accuracy http://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html#sphx-glr-auto-examples-neighbors-plot-classification-py



All k neighbors get one vote



Closer neighbors get extra votes

Distance

“Distance” can mean anything we want it to mean:

- Euclidean: $dist = \sqrt{\sum(x_i - y_i)^2}$
- Manhattan: $dist = \sum| x_i - y_i|^2$
- Hamming: $dist = \# \, bits \; to \; flip \; to \; change \; x \; into \; y$

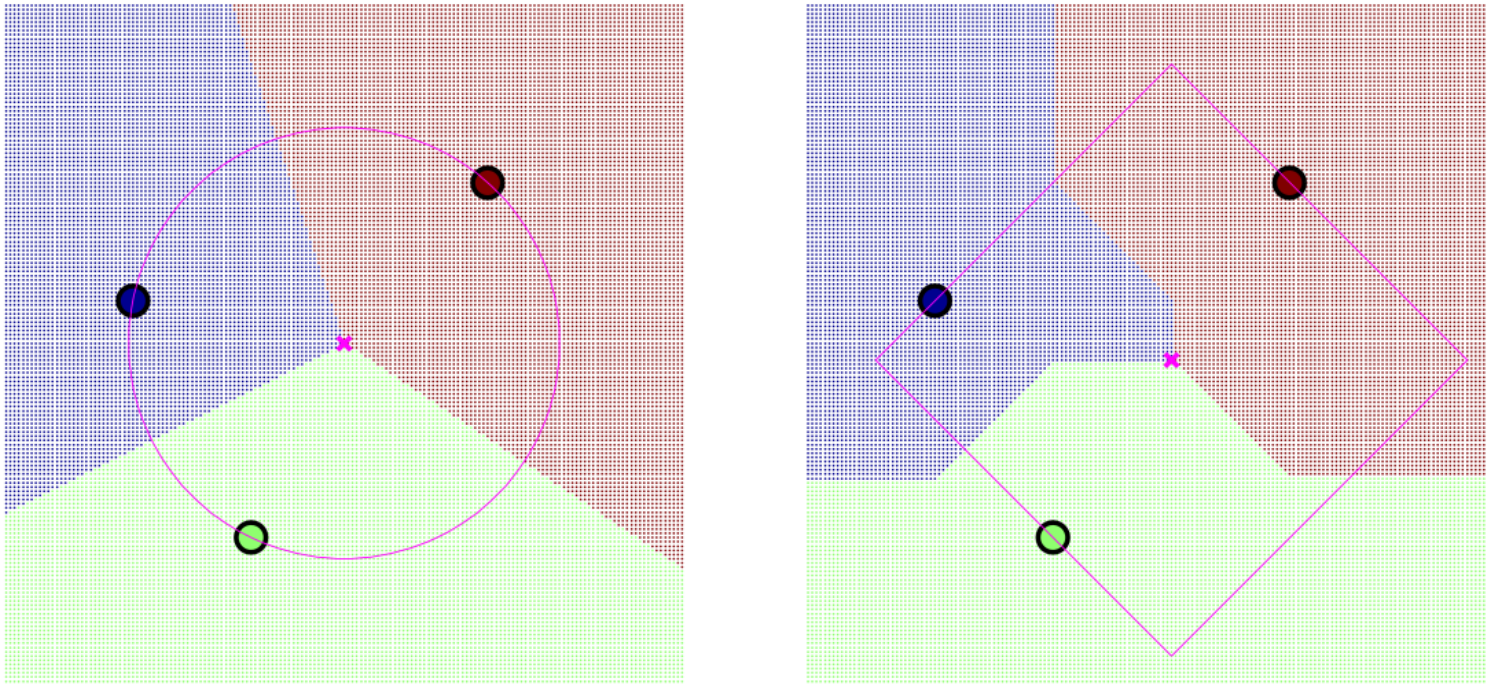
sklearn has a dozen options + ability to define your own

http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.DistanceMetric.html#sklearn.neighbors.DistanceMetric

“Distance” can be tailored to your specific problem

Figure 8.7, p.240

Three-exemplar decision boundaries



(left) Decision regions defined by the 2-norm nearest-exemplar decision rule for three exemplars. **(right)** With Manhattan distance the decision regions become non-convex.

- ☞ kNN uses the training data as exemplars, so training is $O(n)$ (but prediction is also $O(n)$!)
- ☞ 1NN perfectly separates training data, so low bias but high variance
- ☞ By increasing the number of neighbours k we increase bias and decrease variance (what happens when $k = n$?)
- ☞ Easily adapted to real-valued targets, and even to structured objects (nearest-neighbour retrieval). Can also output probabilities when $k > 1$
- ☞ Warning: in high-dimensional spaces everything is far away from everything and so pairwise distances are uninformative (curse of dimensionality)

(k) Nearest Neighbors

- Simplest machine learning algorithm: no real learning required!
- However prediction is much slower O(n)
  - but fancy data structures can get us down to O(log n)
- Only one real parameter to adjust = tradeoff between “bias” and “variance”
- Curse of dimensionality!
- Can prove that error is at most twice the optimal error rate (with infinite training data)

Summary

Pros:

- Simple “white-box” algorithm
- No training: classification is basically a database lookup
- Can define “distance” in many helpful ways
- Only one parameter to tune *(except in distance)*
- Not fooled by outliers if k is big enough

Cons:

- Can be expensive to classify: O(n) or O(log n)
- Curse of dimensionality: less effective with more features

Other References

- http://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html#sphx-glr-auto-examples-neighbors-plot-classification-py
- http://scikit-learn.org/stable/auto_examples/ensemble/plot_voting_decision_regions.html#sphx-glr-auto-examples-ensemble-plot-voting-decision-regions-py