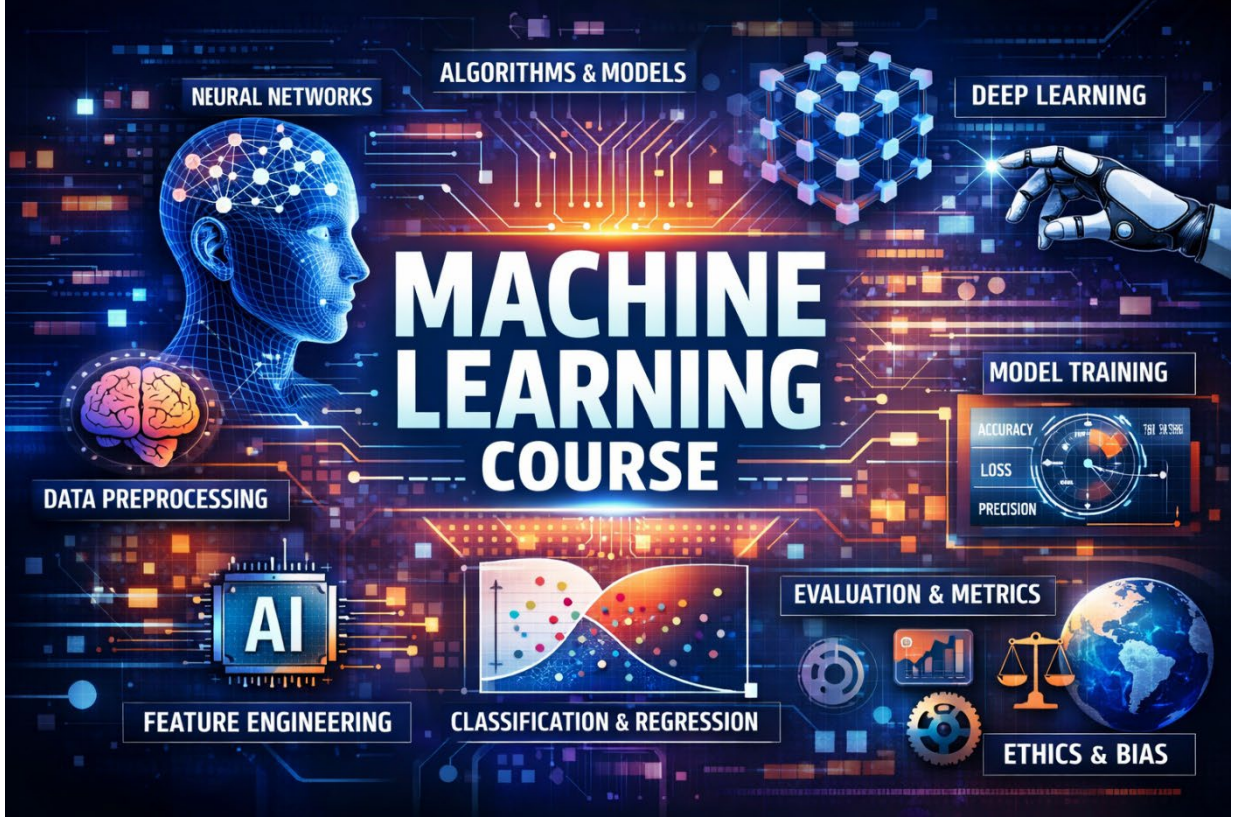



# Nearest Neighbors

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.

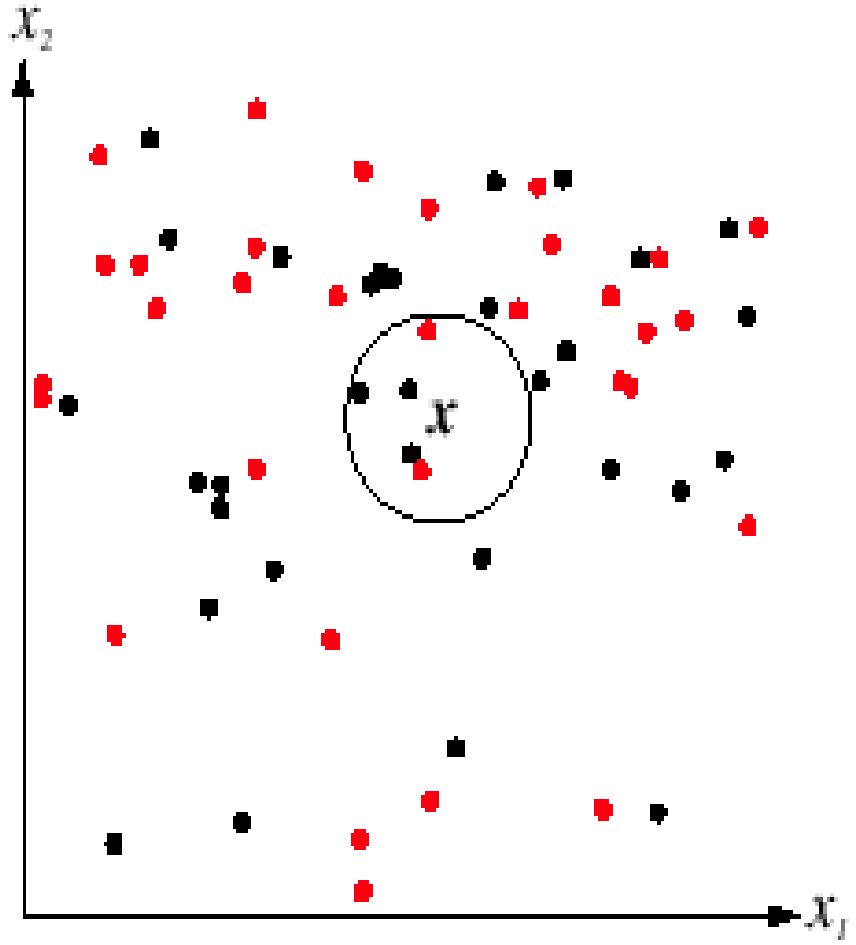
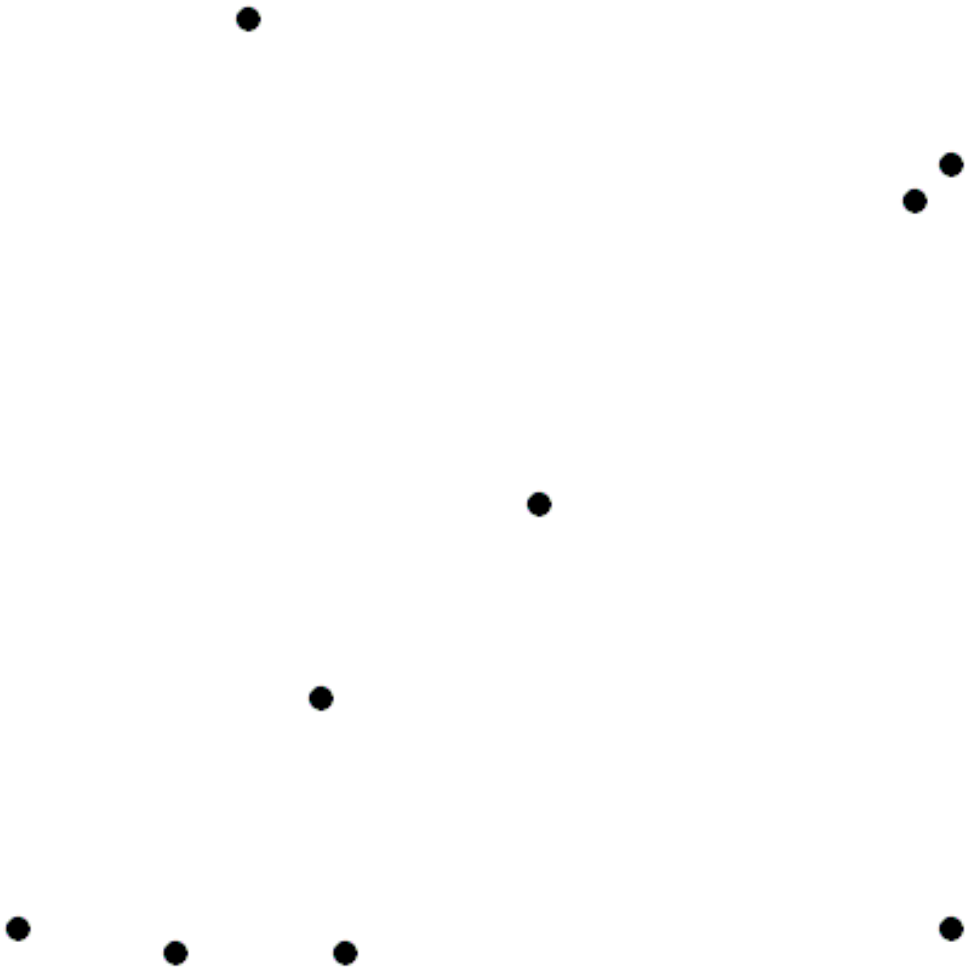



**FIGURE 4.15.** The $k$-nearest-neighbor query starts at the test point $\mathbf{x}$ and grows a spherical region until it encloses $k$ training samples, and it labels the test point by a majority vote of these samples. In this $k = 5$ case, the test point $\mathbf{x}$ would be labeled the category 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



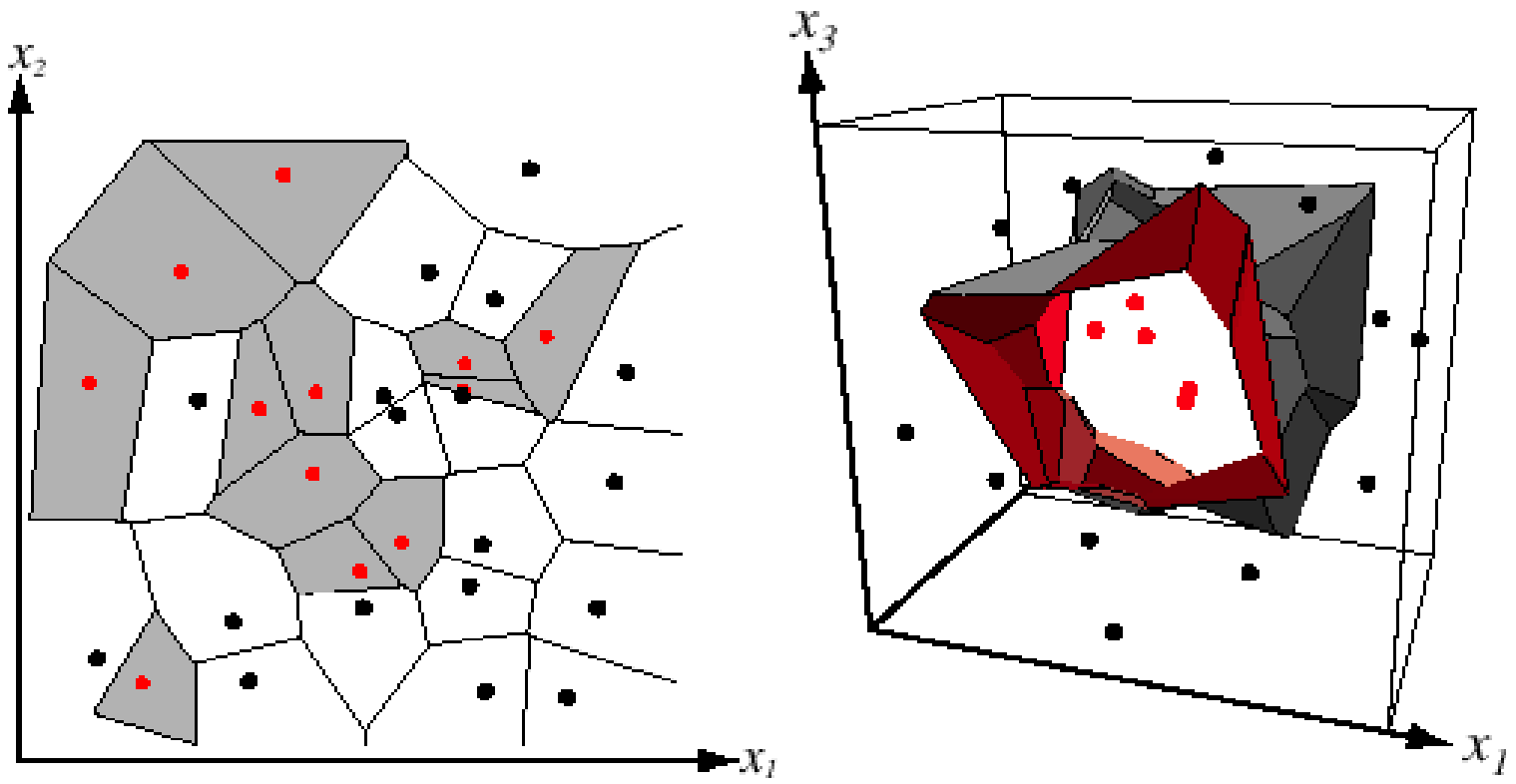
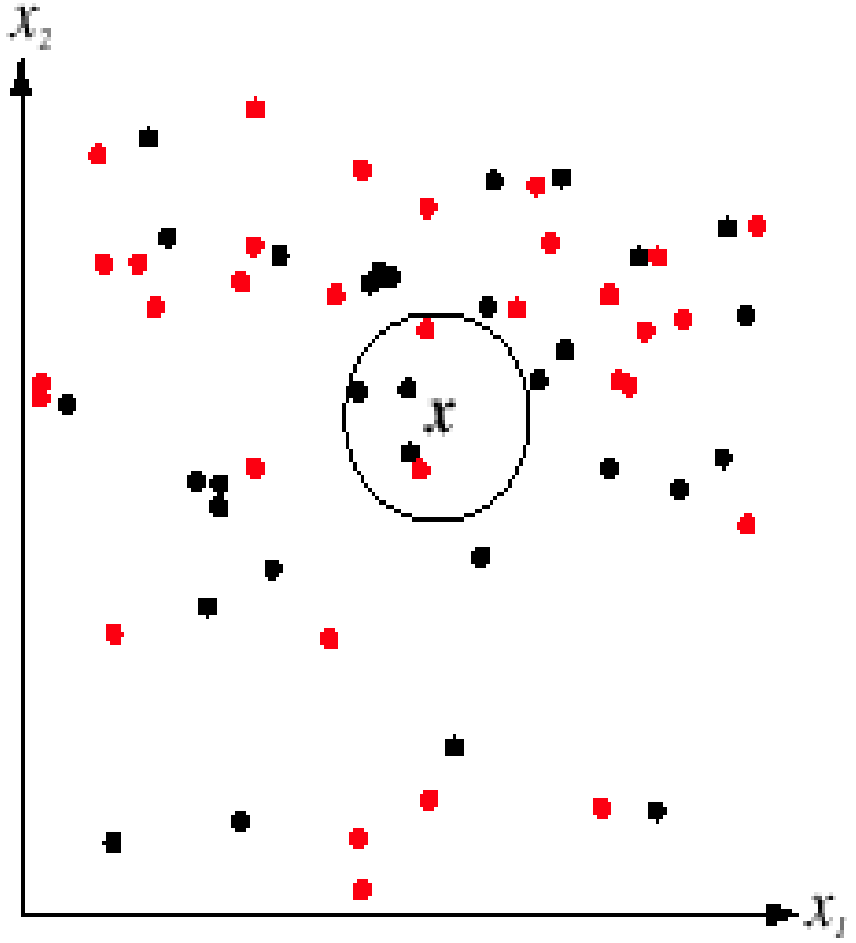



**FIGURE 4.13.** In two dimensions, the nearest-neighbor algorithm leads to a partitioning of the input space into Voronoi cells, each labeled by the category of the training point it contains. In three dimensions, the cells are three-dimensional, and the decision boundary resembles the surface of a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

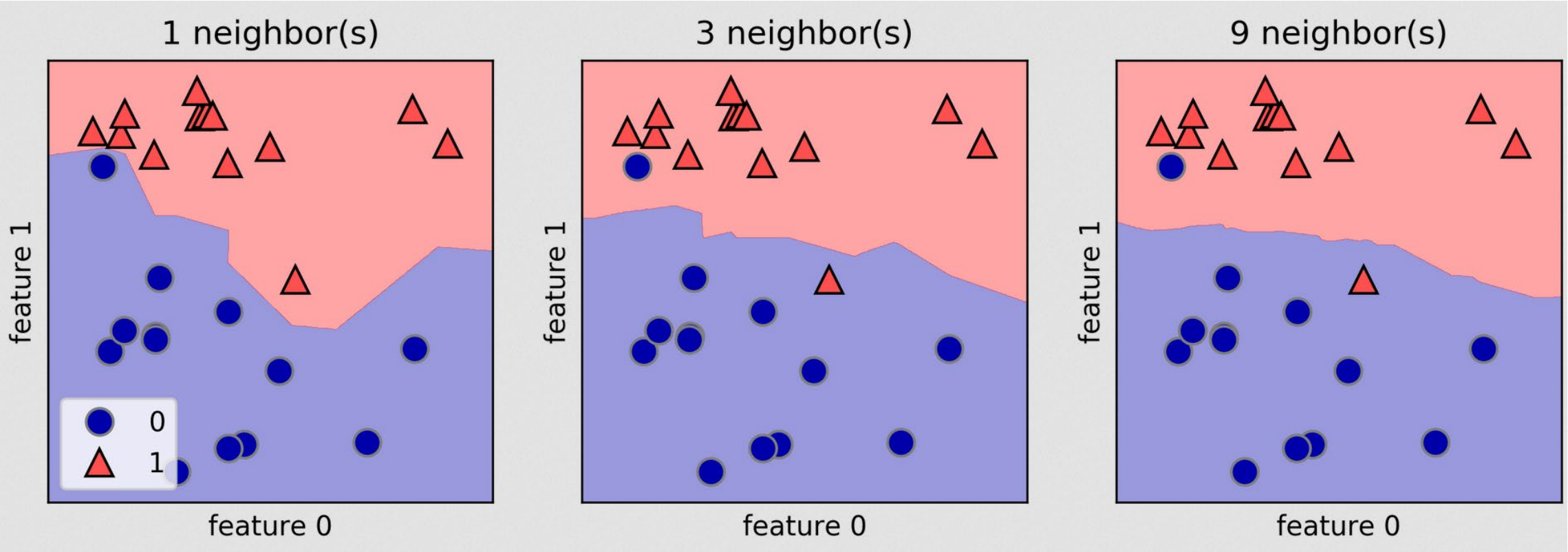
# Probability Density Function

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

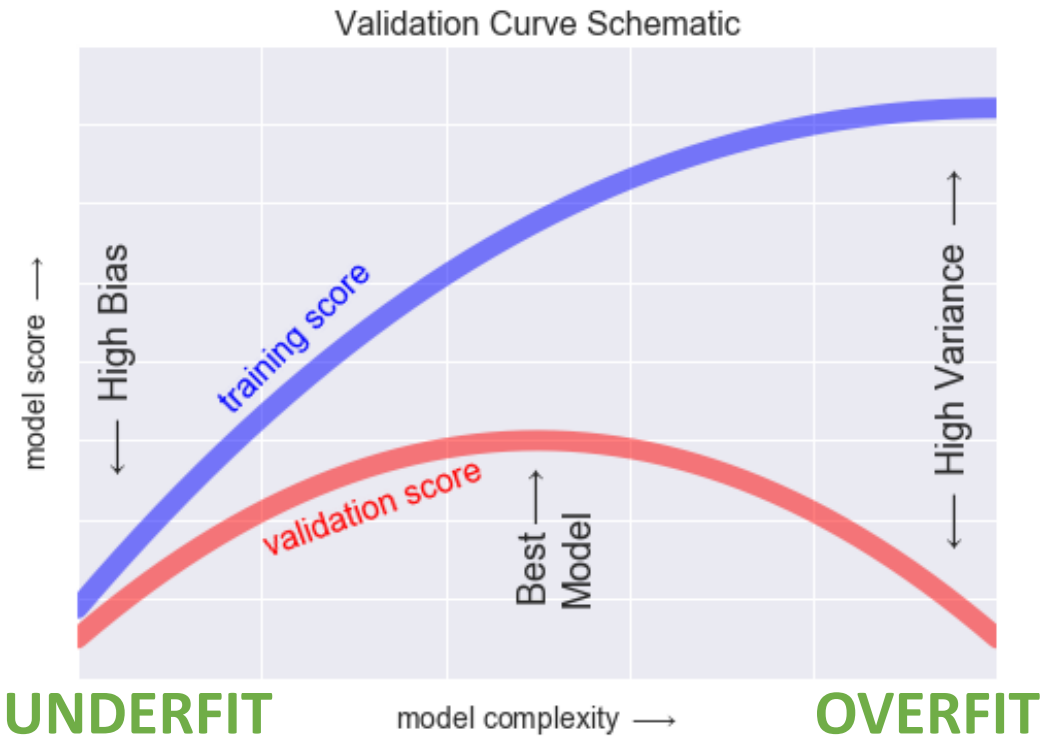



**low bias**
**high variance**

**high bias**
**low variance**

Notice how changing k affects the red and blue outliers

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




•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.

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

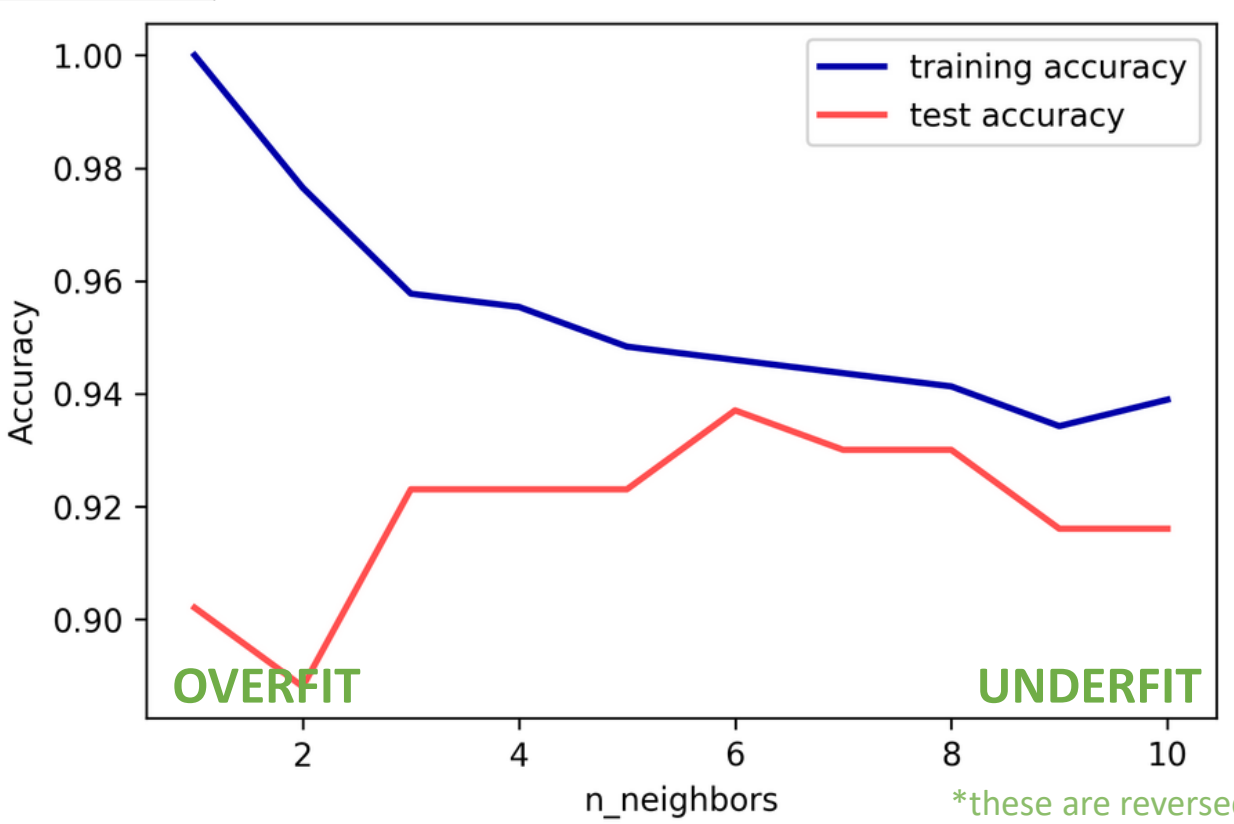
```
from sklearn.datasets import load_breast_cancer

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()
```




*these are reversed from previous plot because low k=high complexity

- $k = 1$ **ALWAYS** has 100% training accuracy
- Increasing k decreases training score but can increase test score
- What happens at $k = N$?

# KNeighborsClassifier

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier

## sklearn.neighbors.KNeighborsClassifier

*class* sklearn.neighbors.**KNeighborsClassifier**(*n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None*) [source]

Classifier implementing the k-nearest neighbors vote.

Read more in the User Guide.

**Parameters:**

**n_neighbors : *int, default=5***

Number of neighbors to use by default for `kneighbors` queries.

**weights : *{'uniform', 'distance'}, callable or None, default='uniform'***

Weight function used in prediction. Possible values:

- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

Refer to the example entitled Nearest Neighbors Classification showing the impact of the `weights` parameter on the decision boundary.

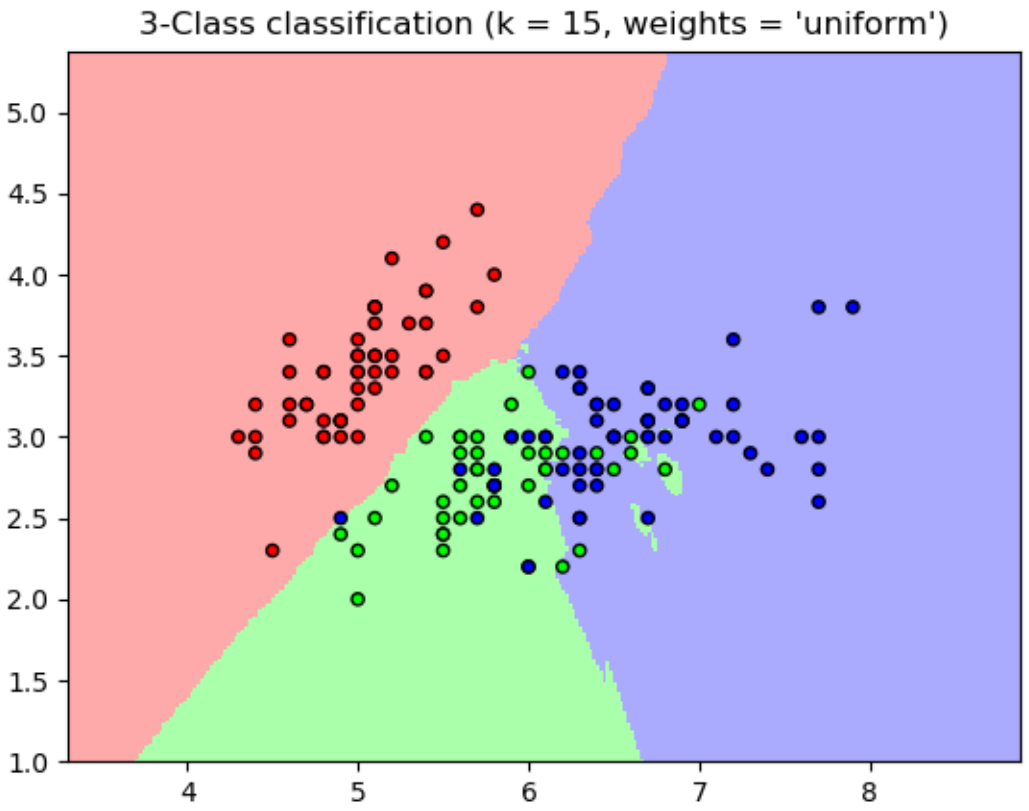
**Two parameters worth changing:**

n_neighbors (i.e. *k*)

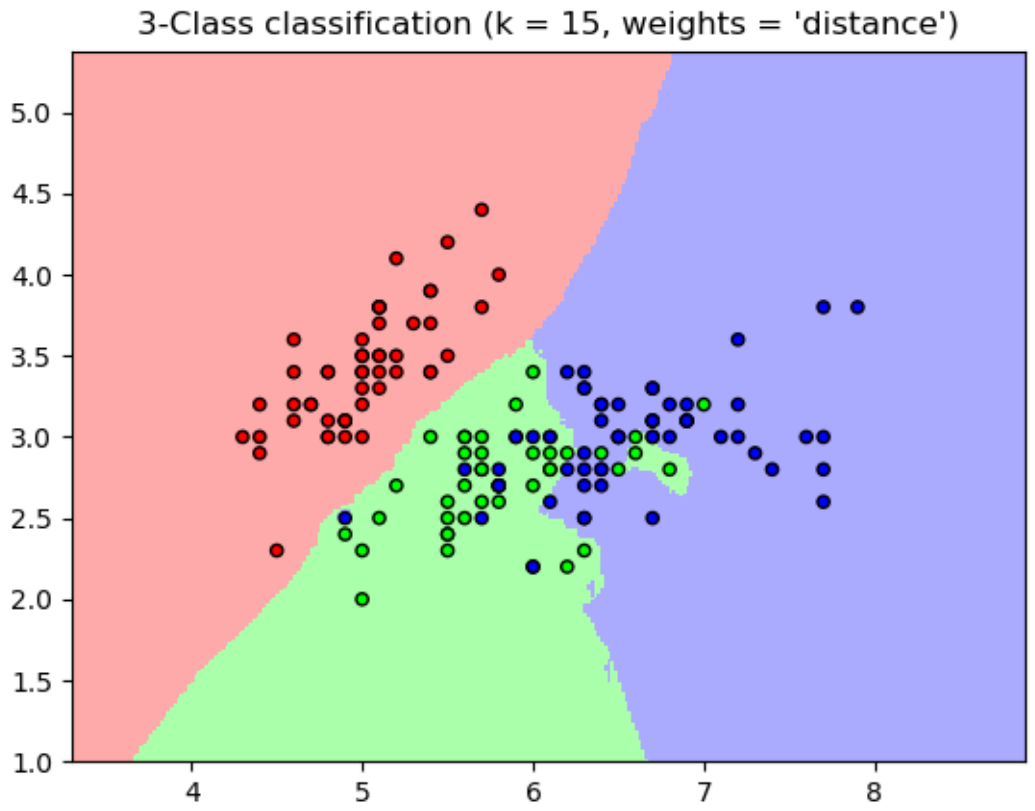
weights

There are others, but you don't need to mess with them

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: $dist = \#\ bits\ to\ flip\ to\ change\ x\ into\ y$

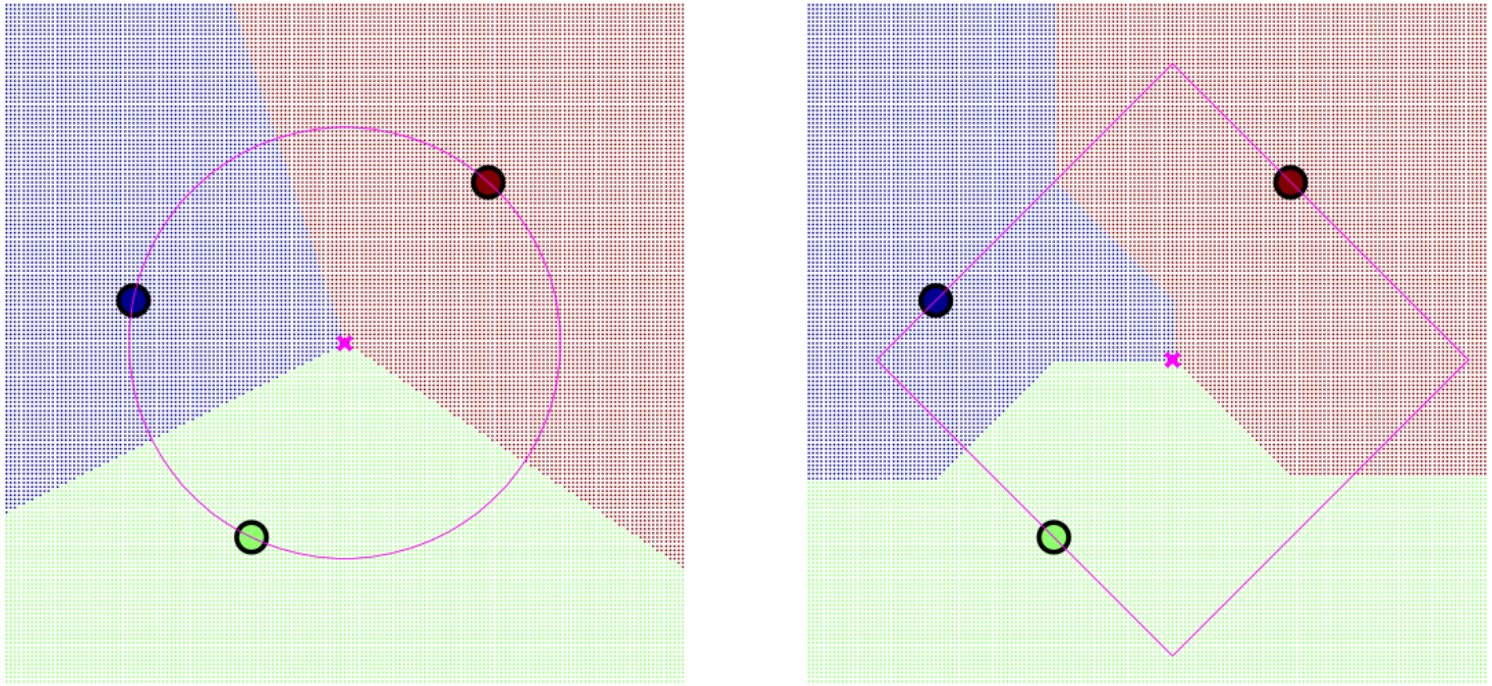
sklearn has a dozen options + ability to define your own

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

- https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier
- 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