Local Learning

Hee-il Hahn

Professor

Department of Information and Communications Engineering

Hankuk University of Foreign Studies

hihahn@hufs.ac.kr

Supervised learning

- Learn a function to predict outputs from (input, output) examples.

  - Classification: $Y \in \{0,1\}$ *or* $Y \in \{-1,+1\}$ - Predicting whether or not a given input belongs to one of two classes, typically in terms of true or false, or positive or negative. (e.g., face *detection*).

  - Regression: $Y \in \mathbb{R}$ - Predicting a real number. (e.g., predicting gas prices).

  - Multi-class classification: $Y \in \{1, \cdots, K\}$: Predicting which of K classes an input belongs to. (e.g., face recognition).

Supervised learning – cont.

- A critical concept in supervised learning is the **training vs. testing data** distinction.
  - **Training:**
    - given a set of training data $(\mathbf{x}_1, y_1)$,…, $(\mathbf{x}_n, y_n)$and allowed to perform computations on the training data to learn an output function $\mathrm{h}(\mathbf{x})$.
  - **Testing:**
    - asked to generate predictions $\mathrm{h}(\mathbf{x}_1)$,…, $\mathrm{h}(\mathbf{x}_m)$ for a set of testing data.
    - then compute an error metric or loss function to determine quantitatively if the learner correctly predicted the true outputs for the test examples.
    - A common metric for classification tasks is the **0–1 loss**, which is just the proportion of incorrect guesses:

$$loss_{01\_error} = \frac{1}{m}\sum_{j=1}^{m} 1\left[\mathrm{h}(\mathbf{x}_j) \neq y_j\right]$$

Supervised learning – cont.

- Important questions about this process:
  - What happens if we use the same data for training and testing?
  - Can we be confident about the algorithm’s accuracy on new datasets?
  - How can we train and test a learner if we only have a finite amount of collected data?
    More generally, how do we collect X and define Y for a given problem?

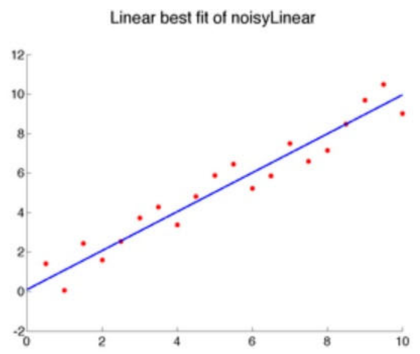
Local learning

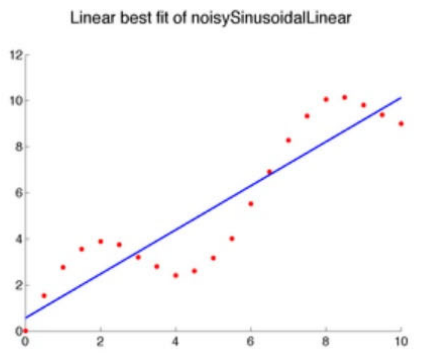
- seems less like learning and more like pure memorization.
- also called memory-based learning, instance-based, case-based, or distance-based.
- given a new example $\mathbf{x}$, find the most similar training example(s) and predict a similar output.

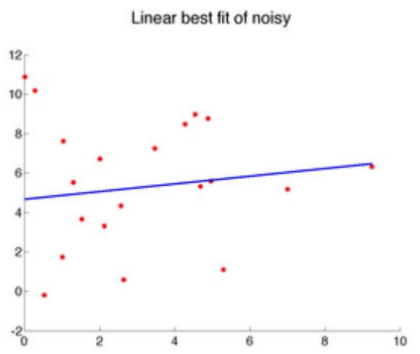
Nearest neighbor methods

- The nearest neighbor idea and local learning, in general, are not limited to classification, and many of the ideas can be more easily illustrated for regression.

- Consider the following one-dimensional regression problems:







- Clearly, linear models do not capture the data well.

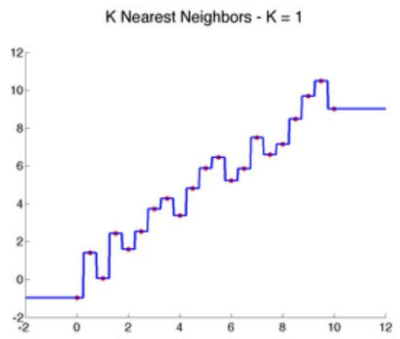
Nearest neighbor methods – cont.

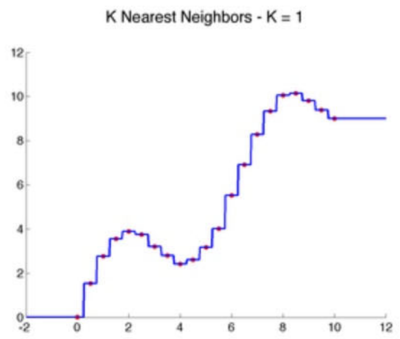
- We can add more features, like higher order polynomial terms, or we can use a local approach, like nearest neighbors:

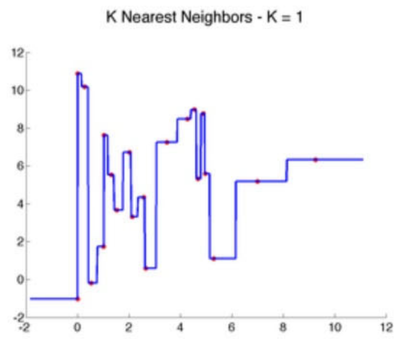
1-Nearest Neighbor Algorithm

1. Given training data $D = \{\mathbf{x}_i, y_i\}$, distance function $d(\cdot,\cdot)$ and input $\mathbf{x}$,
2. Find $j = \arg\min_i d(\mathbf{x}, \mathbf{x}_i)$ and return y_j.

- 1-D Examples with $d(\mathbf{x}, \mathbf{x}_i) = |\mathbf{x} - \mathbf{x}_i|$



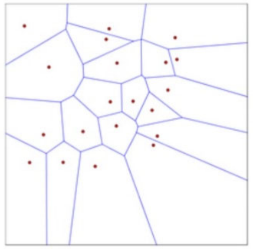




Nearest neighbor methods – cont.

1-Nearest Neighbor Algorithm

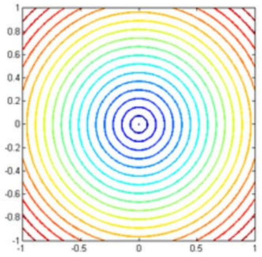
- 2-D Examples with $d(\mathbf{x}, \mathbf{x}_i) = \|\mathbf{x} - \mathbf{x}_i\|_2$



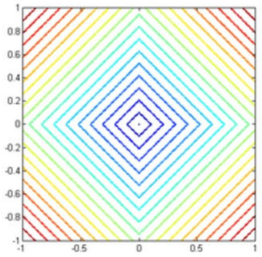
$"Voronoi\ \ diagram"$

- More generally, we can use an arbitrary distance function to define a nearest neighbor.

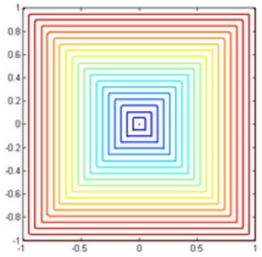
$\sqrt{x^2 + y^2} = c$ $|x| + |y| = c$ $\max(|x|, |y|) = c$



$L_2 norm$



$L_1 norm$

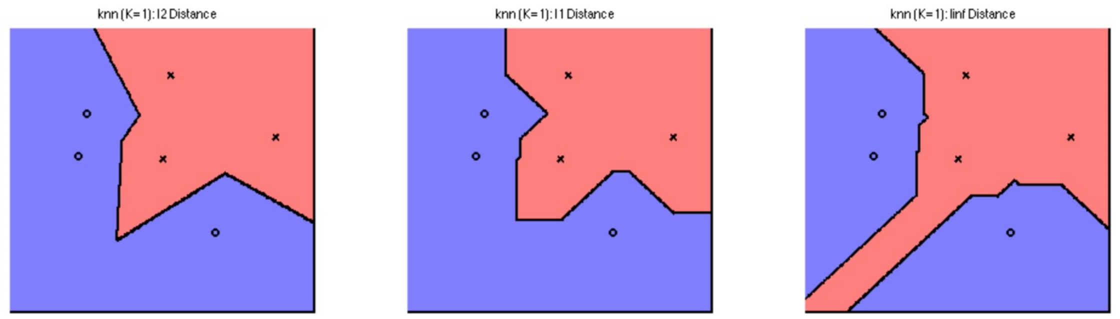


$L_{\inf} norm$

Nearest neighbor methods – cont.

1-Nearest Neighbor Algorithm

- Voronoi diagrams are shaped differently according to the norms.



- one of the simplest examples of a **non-parametric** method,

  while linear regression or Naive Bayes are parametric.
- the model structure is determined by the training data.
- Non-parametric models are much more flexible and expressive than parametric ones, and thus **overfitting** is a major concern.

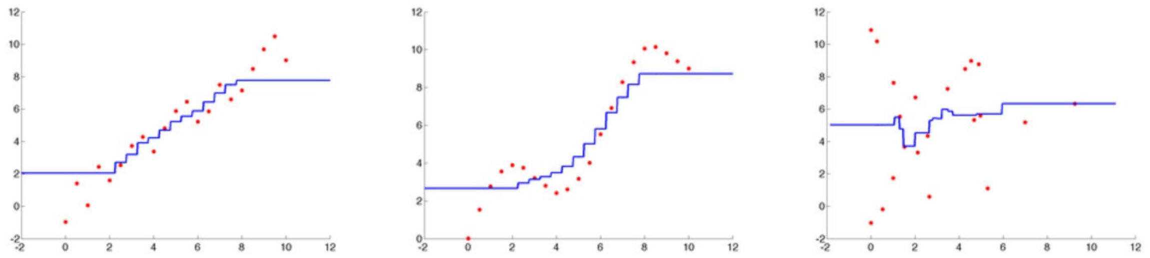
Nearest neighbor methods – cont.

K-Nearest Neighbor Algorithm

- One way to reduce the variance is local averaging: instead of just one neighbor, find K and average their predictions.

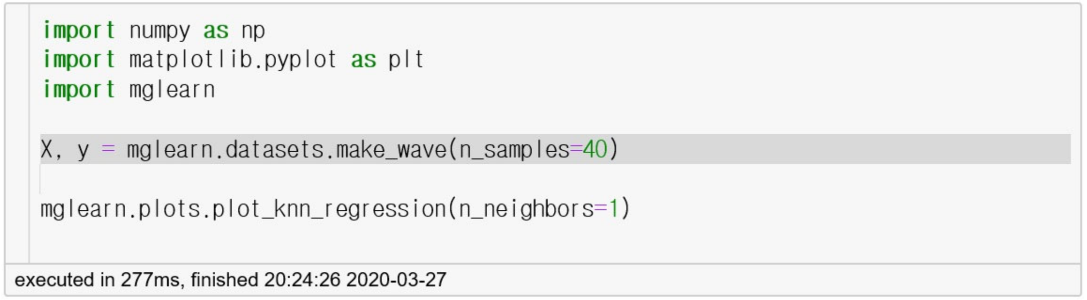
1. Given training data $D = \{\mathbf{x}_i, y_i\}$, distance function $d(\cdot,\cdot)$ and input $\mathbf{x}$,
2. Find $\{j_1, \cdots, j_K\}$ closest examples w.r.t $d(\mathbf{x},\cdot)$
   - (regression) if $y \in \mathbb{R}$, return average: $\frac{1}{K}\sum_{k=1}^{K} y_{j_k}$.
   - (classification) if $y \in \pm 1$, return majority: $sign\left(\sum_{k=1}^{K} y_{j_k}\right)$

<u>results of using 9 neighbors on our examples</u>



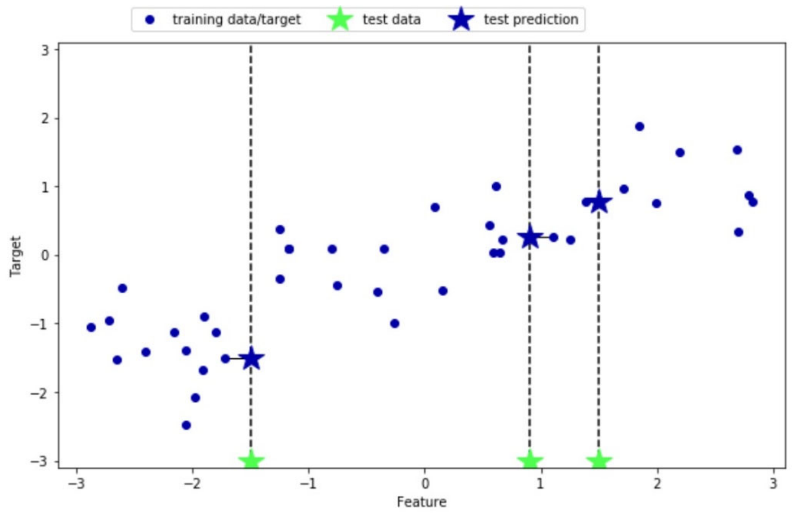
Nearest neighbor methods – cont.

- Simulation of knn_1D



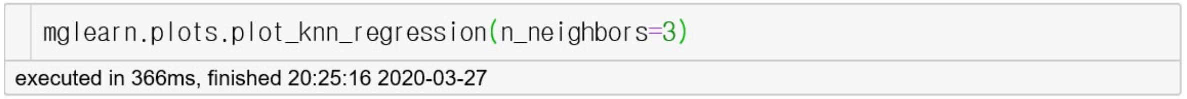
```
import numpy as np
import matplotlib.pyplot as plt
import mglearn

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

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

```
executed in 277ms, finished 20:24:26 2020-03-27
```



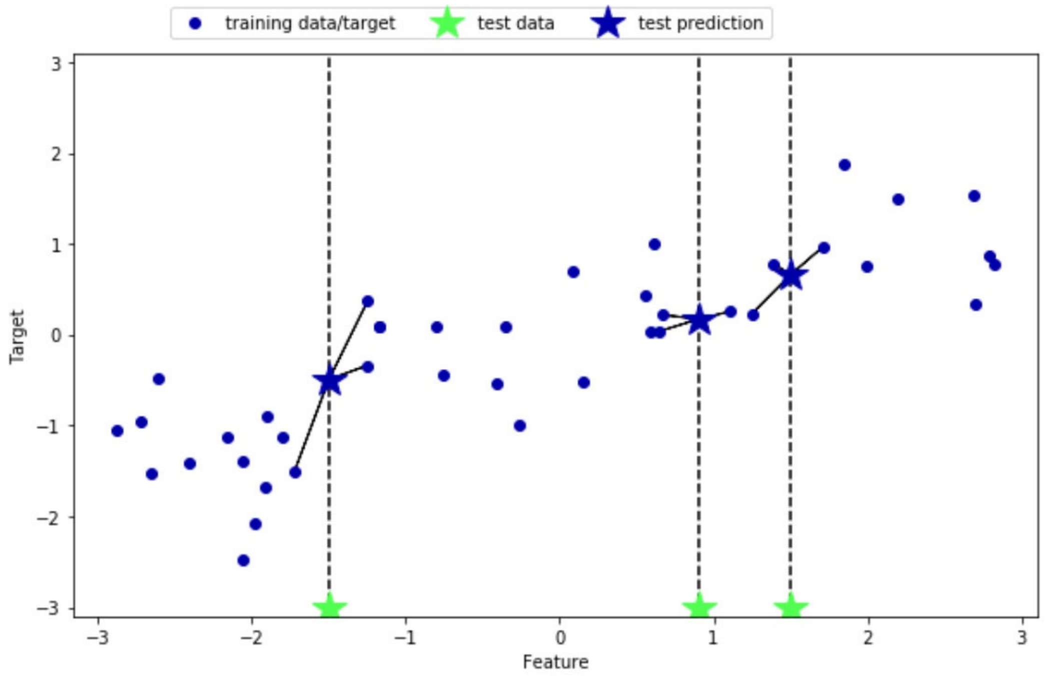
Nearest neighbor methods – cont.

- Simulation of knn_1D



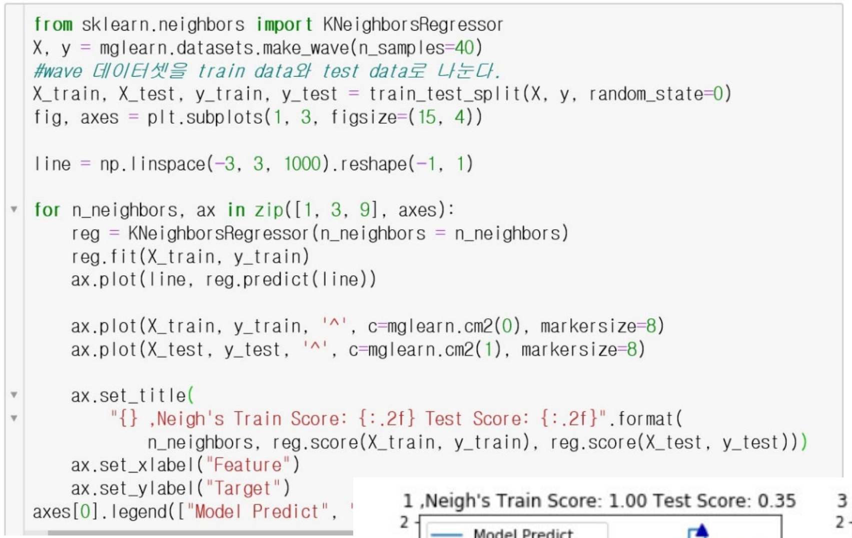
```
mglearn.plots.plot_knn_regression(n_neighbors=3)
```

```
executed in 366ms, finished 20:25:16 2020-03-27
```

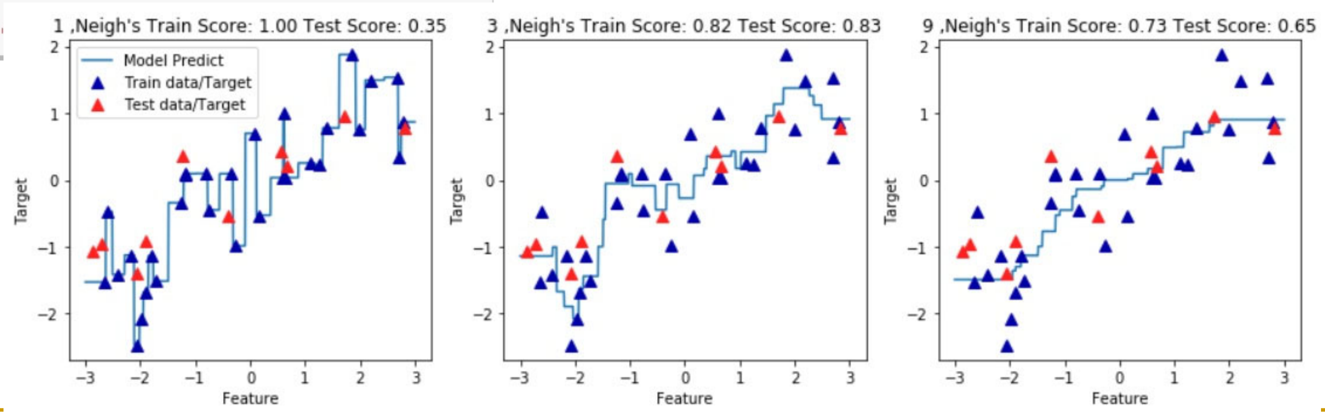


Nearest neighbor methods – cont.

- Simulation of knn_1D



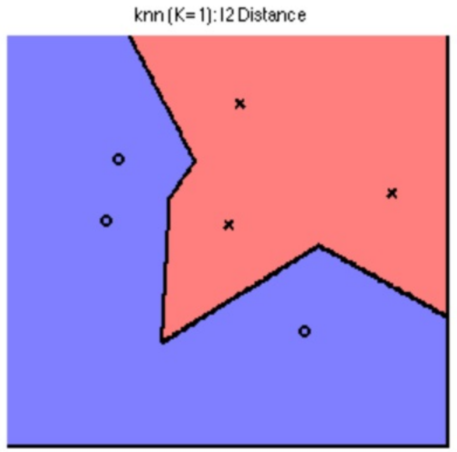
```
from sklearn.neighbors import KNeighborsRegressor
X, y = mglearn.datasets.make_wave(n_samples=40)
#wave 데이터셋을 train data와 test data로 나눈다.
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
fig, axes = plt.subplots(1, 3, figsize=(15, 4))

line = np.linspace(-3, 3, 1000).reshape(-1, 1)

for n_neighbors, ax in zip([1, 3, 9], axes):
    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, '^', c=mglearn.cm2(1), markersize=8)

    ax.set_title(
        "{} ,Neigh's 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 Predict", '
```

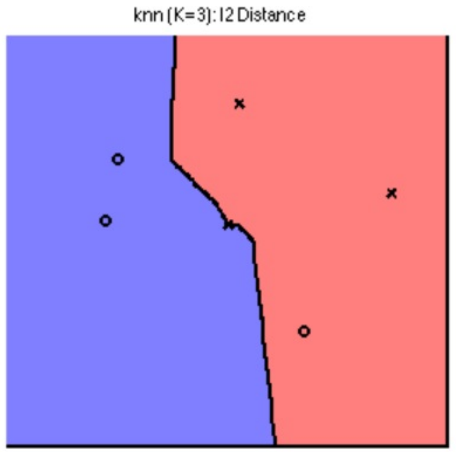


Nearest neighbor methods – cont.

K-Nearest Neighbor Algorithm

2-D results of using ($K = 1 \; or \; 3$) neighbors on our examples

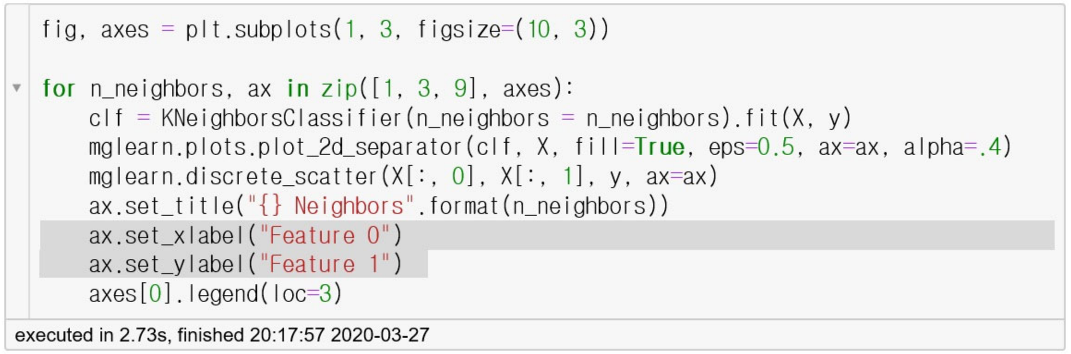




- Achieve a “smoother” decision boundary by increasing K from 1 to 3.

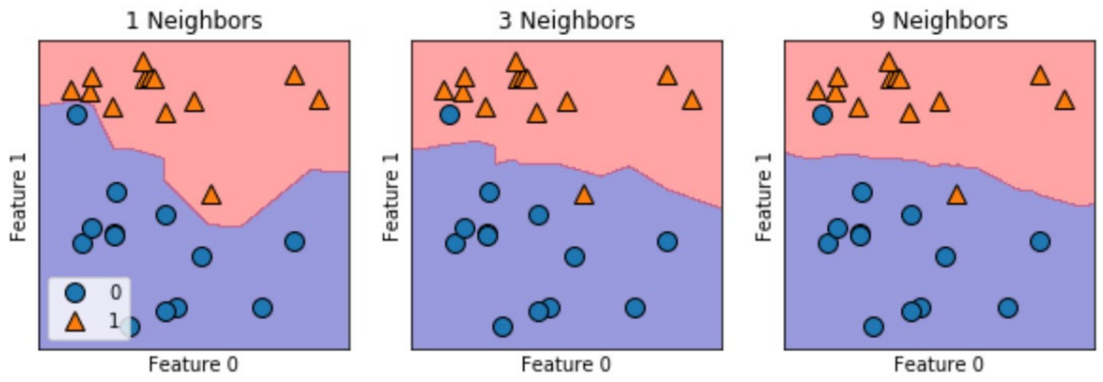
Nearest neighbor methods – cont.

- Simulation of knn_2D



```
fig, axes = plt.subplots(1, 3, figsize=(10, 3))

for n_neighbors, ax in zip([1, 3, 9], axes):
    clf = KNeighborsClassifier(n_neighbors = n_neighbors).fit(X, y)
    mglearn.plots.plot_2d_separator(clf, X, fill=True, eps=0.5, ax=ax, alpha=.4)
    mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
    ax.set_title("{} Neighbors".format(n_neighbors))
    ax.set_xlabel("Feature 0")
    ax.set_ylabel("Feature 1")
    axes[0].legend(loc=3)
```

executed in 2.73s, finished 20:17:57 2020-03-27



Kernel regression

Shortcomings of K-Nearest Neighbor Algorithm

- All neighbors receive equal weight.
- The number of neighbors must be chosen globally.

Kernel regression

- Instead of selected nearest neighbors, all neighbors are used.
- Closer neighbors receive higher weight.
- The weighting function is called a kernel and it measures similarity (as opposed to distance) between examples.
- easy to convert from a distance $d(\cdot,\cdot)$ to a kernel $K(\cdot,\cdot)$.
- One of the most common ways is the Gaussian-type kernel (ignoring the normalization):

$$K(\mathbf{x}, \mathbf{x}_i) = exp\left\{\frac{-d(\mathbf{x}, \mathbf{x}_i)}{\sigma^2}\right\}$$

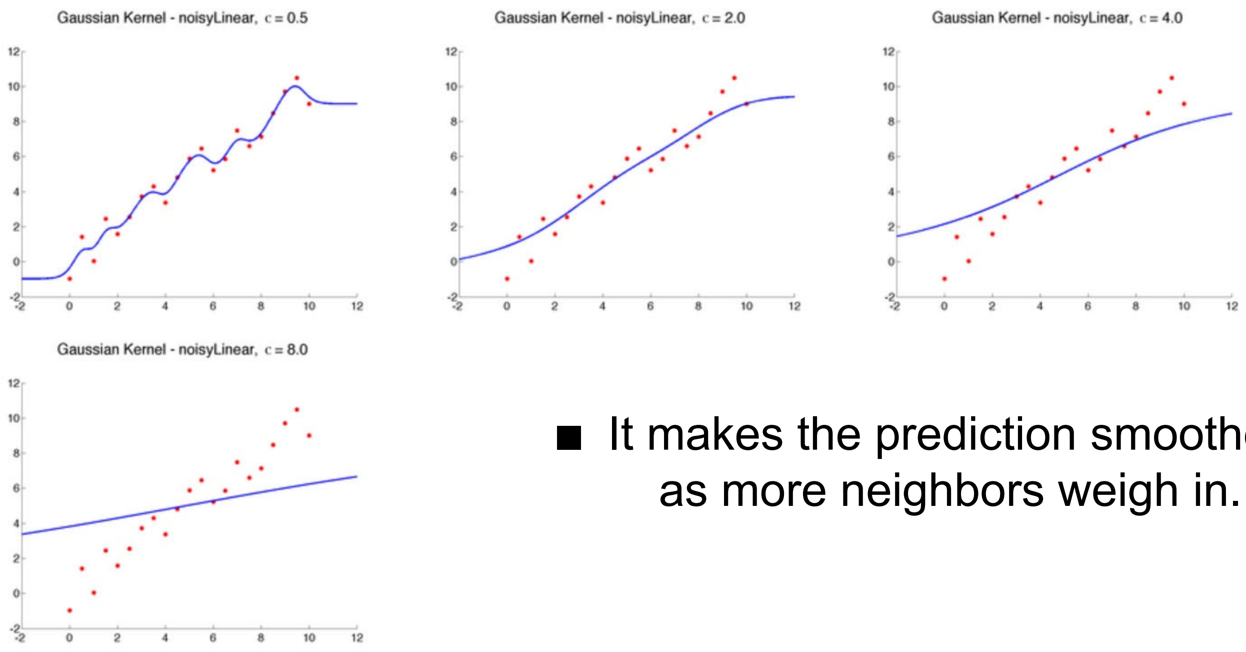
Kernel regression – cont.

Kernel Regression/Classification Algorithm

- Given training data $D = \{\mathbf{x}_i, y_i\}$, kernel function $K(\cdot,\cdot)$ and input $\mathbf{x}$,
  - (regression) if $y \in \mathbb{R}$, return weighted average: $\frac{\sum_{i=1}^{n} K(\mathbf{x},\mathbf{x}_i) y_i}{\sum_{i=1}^{n} K(\mathbf{x},\mathbf{x}_i)}$.
  - (classification) if $y \in \pm 1$, return weighted majority: $sign(\sum_{i=1}^{n} K(\mathbf{x}, \mathbf{x}_i) \mathbf{y}_i)$
- In kernel regression/classification, nearby points contribute much more to the prediction.
- A key parameter in defining the Gaussian kernel is σ , also called the width, which determines how quickly the influence of neighbors falls off with distance.

Kernel regression – cont.

Results of varying σ from 0.5 to 8

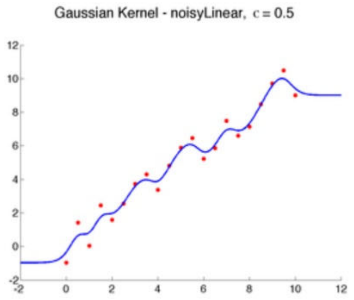


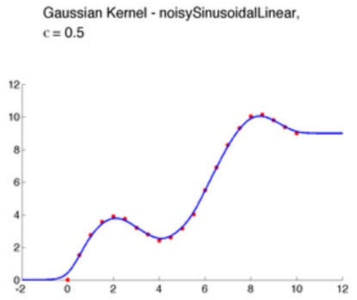
- It makes the prediction smoother, as more neighbors weigh in.

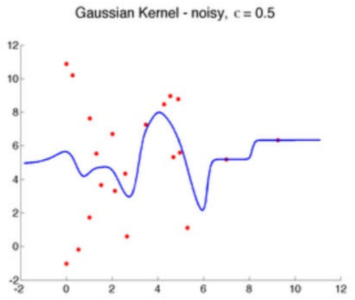
- As $\sigma \rightarrow \infty$, all the neighbors weigh the same and the prediction is the global average or global majority and as $\sigma \rightarrow 0$, the prediction tends to 1-NN.

Kernel regression – cont.

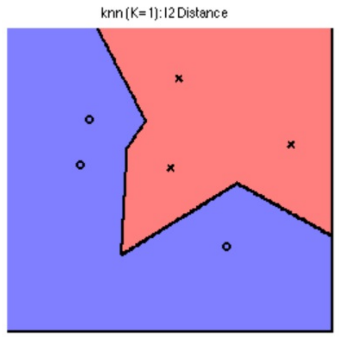
Results of using good kernel width on our examples ($\sigma = 0.5$)

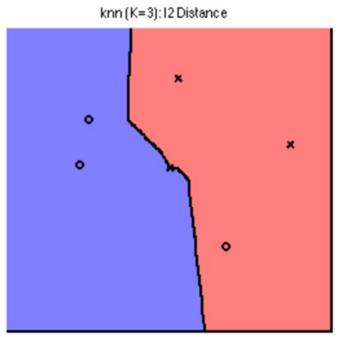


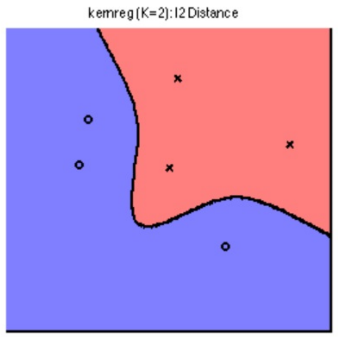




2-D Results of kernel regression







Concluding remarks

- While local learning methods are very flexible and often very effective, they have some disadvantages:
  - Expensive: need to remember (store) and search through all the training data for every prediction.
  - Curse-of-dimensionality: In high dimensions, all points are far.
  - Irrelevant features: If $\mathbf{x}$ has irrelevant, noisy features, distance function becomes useless.

<https://alliance.seas.upenn.edu/~cis520/dynamic/2017/wiki/index.php?n=Lectures.LocalLearning>