EECS 445: Machine Learning

Lecture 4. Classification

Honglak Lee

1/24/2020

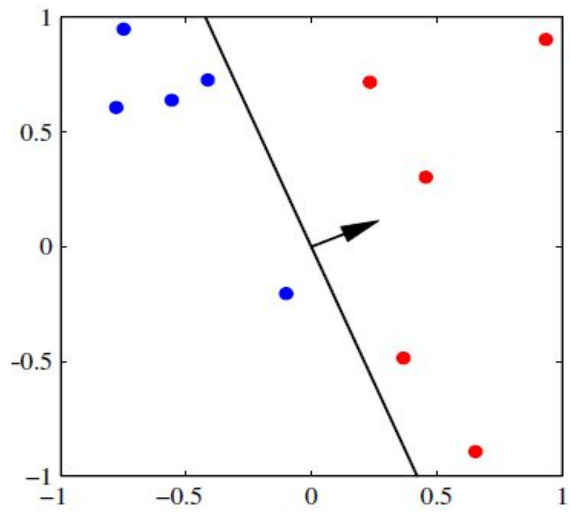
Outline

- Logistic regression
- Newton’s method
- K-Nearest Neighbors (KNN)

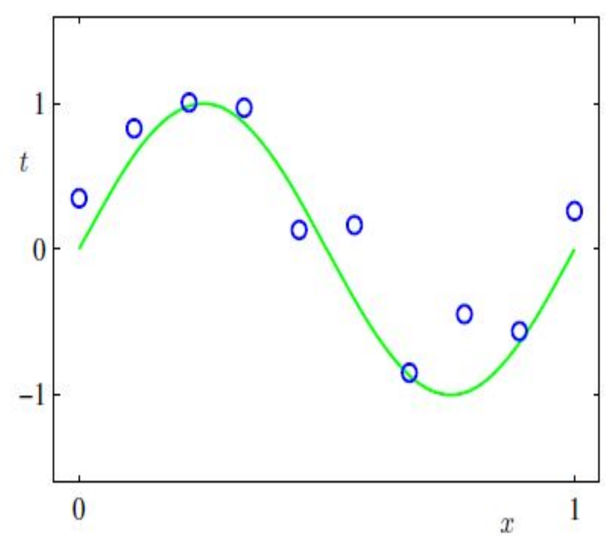
Supervised Learning: Classification

Supervised Learning

- Goal:
  - Given data X in feature space and the labels Y
  - Learn to predict Y from X
- Labels could be discrete or continuous
  - Discrete-valued labels: classification
  - Continuous-valued labels: regression (today's topic)



classification



regression

Classification problem

- The task of classification:
  - Given an input vector $\mathbf{x}$, assign it to one of K distinct classes C_k where $k = 1, \dots K$
- Representing the assignment:
  - For K=2
    - y=1 means that $\mathbf{x}$ is in C_1.
    - y=0 means that $\mathbf{x}$ is in C_2.
      - (Sometimes, y=-1 can be used depending on algorithms.)
  - For K>2,
    - Use 1-of-K coding
    - e.g., $\mathbf{y} = (0, 1, 0, 0, 0)^T$ means that $\mathbf{x}$ is in C_2.
      - (This would also work for K=2, of course.)

Classification problem

- Training: train a classifier h(x) from training data
  - Training data

$$\left\{\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \ldots, \left(x^{(N)}, y^{(N)}\right)\right\}$$

- Testing (evaluation):
  - testing data: $\left\{\left(x_{\text{test}}^{(1)}, y_{\text{test}}^{(1)}\right), \left(x_{\text{test}}^{(2)}, y_{\text{test}}^{(2)}\right), \ldots, \left(x_{\text{test}}^{(N)}, y_{\text{test}}^{(N)}\right)\right\}$
  - The learning algorithm produces predictions

$$h\left(x_{\text{test}}^{(1)}\right), h\left(x_{\text{test}}^{(2)}\right), \ldots, h\left(x_{\text{test}}^{(N)}\right)$$

  - 0-1 loss:

$$\text{classification error} = \frac{1}{m}\sum_{j=1}^{m} 1\left[h\left(x_{\text{test}}^{(j)}\right) \neq y_{\text{test}}^{(j)}\right]$$

Strategies to classification problems

- Nearest neighbor classification
  - Given query data **x**, find the closest training points and do majority vote.
- Discriminant functions
  - Learn a function $h(\mathbf{x})$ that maps **x** onto some C_j.
- Learn the distributions $p(C_k|\mathbf{x})$.

(a) Discriminative models: Directly model $p(C_k|\mathbf{x})$ and learn parameters from the training set.

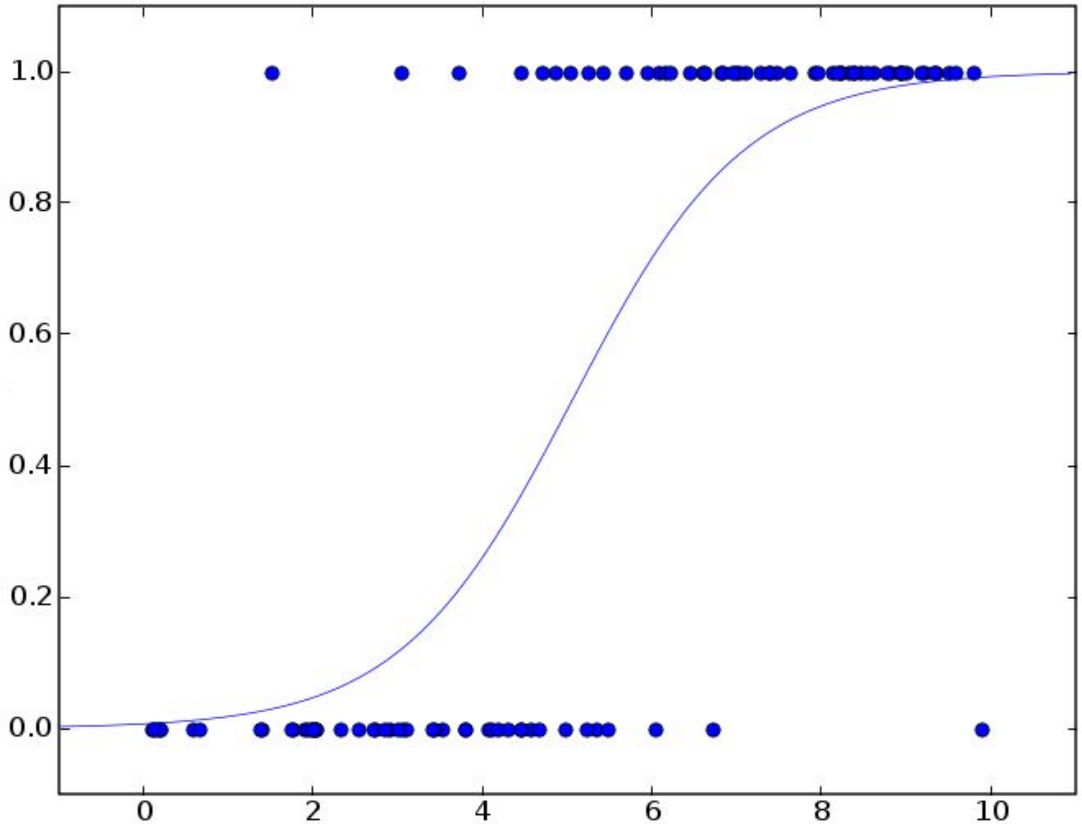
(b) Generative models: Learn class densities $p(x|C_k)$ and priors $p(C_k)$

Logistic Regression

Probabilistic discriminative models

- Model decision boundary as a function of input **x**
  – Learn $P(C_k|\mathbf{x})$ over data (e.g., maximum likelihood)
  – Directly predict class labels from inputs
- Next class: we will cover probabilistic generative models
  – Learn $P(C_k, \mathbf{x})$ over data (maximum likelihood) and then use Bayes' rule to predict $P(C_k|\mathbf{x})$

Example (1-dim. case)



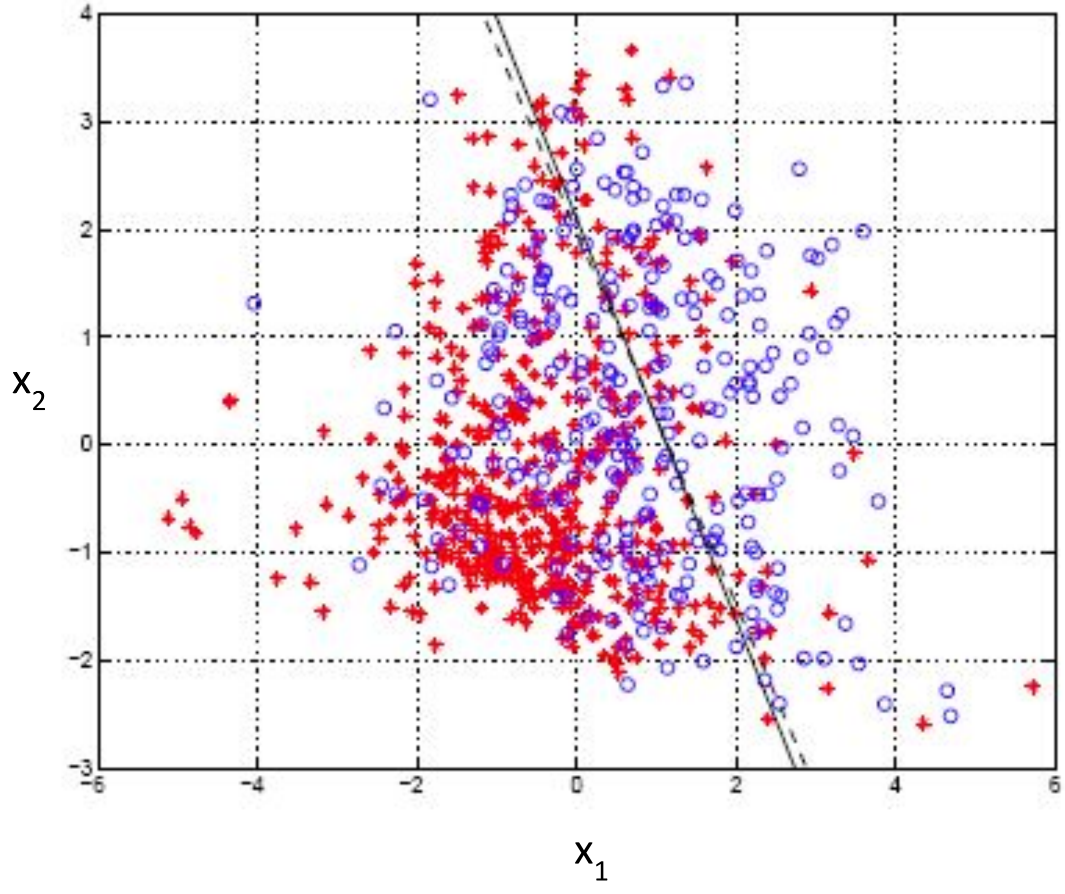
P(y=1|x)

y=1

y=0

x

Example (2-dim. case)



Logistic Regression

- Models the class posterior using a sigmoid applied to a linear function of the feature vector:

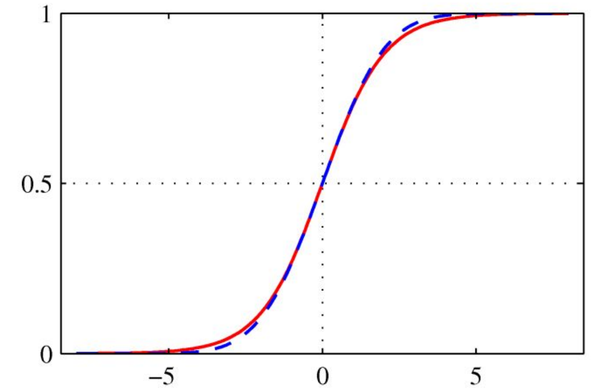
$$p(C_1|\phi) = h(\phi) = \sigma(\mathbf{w}^T \phi(\mathbf{x}))$$

- We can solve the parameter **w** by maximizing the likelihood of the training data.

Sigmoid and Logit functions

- The *logistic sigmoid* function is:

$$\sigma(a) = \frac{1}{1+\exp(-a)}$$



- Its inverse is the *logit* function (aka log odd ratio):

$$a = \ln\left(\frac{\sigma}{1-\sigma}\right)$$

- Generalizes to *normalized exponential*, or *softmax*.

$$p_i = \frac{\exp(q_i)}{\sum_j \exp(q_j)}$$

Likelihood function

- Depending on the label y, the likelihood of x is defined as:

$$P(y=1|\mathbf{x},\mathbf{w}) = \sigma(\mathbf{w}^T\phi(\mathbf{x}))$$
$$P(y=0|\mathbf{x},\mathbf{w}) = 1-\sigma(\mathbf{w}^T\phi(\mathbf{x}))$$

- Therefore:

$$P(y|\mathbf{x},\mathbf{w}) = \sigma(\mathbf{w}^T\phi(\mathbf{x}))^y(1-\sigma(\mathbf{w}^T\phi(\mathbf{x})))^{(1-y)}$$

Logistic Regression

- For a data set $\{(\phi(\mathbf{x}^{(n)}), y^{(n)})\},\ \text{where } y^{(n)} \in \{0,1\}$

the likelihood function is

$$p(\mathbf{y}|\mathbf{w}) = \prod_{n=1}^{N} (h^{(n)})^{y^{(n)}} (1 - h^{(n)})^{1-y^{(n)}}$$

– where

$$h^{(n)} = p(C_1|\phi(\mathbf{x}^{(n)})) = \sigma(\mathbf{w}^T \phi(\mathbf{x}^{(n)}))$$

- Define an loss function $E(\mathbf{w}) = -\log p(\mathbf{y}|\mathbf{w})$
  – (Minimizing *E*(**w**) maximizes likelihood.)

Derivation

- $\log P(\mathbf{y}|\mathbf{w}) = \sum_{n=1}^{N} y^{(n)} \log h^{(n)} + (1 - y^{(n)}) \log(1 - h^{(n)})$
- Gradient (matrix calculus)

$$\nabla_{\mathbf{w}} \log P(\mathbf{y}|\mathbf{x}^{(1)}, ...\mathbf{x}^{(N)}, \mathbf{w})$$

$$= \sum_{n=1}^{N} \nabla_{\mathbf{w}} \left(y^{(n)} \log h(\mathbf{x}^{(n)}, \mathbf{w}) + (1 - y^{(n)}) \log(1 - h(\mathbf{x}^{(n)}, \mathbf{w})) \right)$$

Derivation

- $\log P(\mathbf{y}|\mathbf{w}) = \sum_{n=1}^{N} y^{(n)} \log h^{(n)} + (1 - y^{(n)}) \log(1 - h^{(n)})$
- Gradient (matrix calculus)

$$\sigma^{(n)} \triangleq \sigma\left(\mathbf{w}^T \phi(\mathbf{x}^{(n)})\right) \triangleq h(\mathbf{x}^{(n)}, \mathbf{w})$$

$$\nabla_{\mathbf{w}} \log P(\mathbf{y}|\mathbf{x}^{(1)}, \ldots \mathbf{x}^{(N)}, \mathbf{w})$$

$$= \sum_{n=1}^{N} \nabla_{\mathbf{w}} \left(y^{(n)} \log h(\mathbf{x}^{(n)}, \mathbf{w}) + (1 - y^{(n)}) \log(1 - h(\mathbf{x}^{(n)}, \mathbf{w}))\right)$$

Derivation

- $\log P(\mathbf{y}|\mathbf{w}) = \sum_{n=1}^{N} y^{(n)} \log h^{(n)} + (1 - y^{(n)}) \log(1 - h^{(n)})$
- Gradient (matrix calculus)

$$\sigma^{(n)} \triangleq \sigma\left(\mathbf{w}^T \phi(\mathbf{x}^{(n)})\right) \triangleq h(\mathbf{x}^{(n)}, \mathbf{w})$$

$$\nabla_{\mathbf{w}} \log P(\mathbf{y}|\mathbf{x}^{(1)}, \ldots \mathbf{x}^{(N)}, \mathbf{w})$$

$$= \sum_{n=1}^{N} \nabla_{\mathbf{w}} \left(y^{(n)} \log h(\mathbf{x}^{(n)}, \mathbf{w}) + (1 - y^{(n)}) \log(1 - h(\mathbf{x}^{(n)}, \mathbf{w})) \right)$$

$$= \sum_{n=1}^{N} \left(y^{(n)} \frac{\sigma^{(n)}(1 - \sigma^{(n)})}{\sigma^{(n)}} - (1 - y^{(n)}) \frac{\sigma^{(n)}(1 - \sigma^{(n)})}{1 - \sigma^{(n)}} \right) \nabla_{\mathbf{w}}(\mathbf{w}^T \phi(\mathbf{x}^{(n)}))$$

$$\boxed{\frac{\partial}{\partial s}\sigma(s) = \frac{\partial}{\partial s}\left(\frac{1}{1 + \exp(-s)}\right) = \sigma(s)(1 - \sigma(s))}$$

Derivation

- $\log P(\mathbf{y}|\mathbf{w}) = \sum_{n=1}^{N} y^{(n)} \log h^{(n)} + (1 - y^{(n)}) \log(1 - h^{(n)})$
- Gradient (matrix calculus)

$$\sigma^{(n)} \triangleq \sigma\left(\mathbf{w}^T \phi(\mathbf{x}^{(n)})\right) \triangleq h(\mathbf{x}^{(n)}, \mathbf{w})$$

$$\nabla_{\mathbf{w}} \log P(\mathbf{y}|\mathbf{x}^{(1)}, \ldots \mathbf{x}^{(N)}, \mathbf{w})$$

$$= \sum_{n=1}^{N} \nabla_{\mathbf{w}} \left(y^{(n)} \log h(\mathbf{x}^{(n)}, \mathbf{w}) + (1 - y^{(n)}) \log(1 - h(\mathbf{x}^{(n)}, \mathbf{w})) \right)$$

$$= \sum_{n=1}^{N} \left(y^{(n)} \frac{\sigma^{(n)}(1 - \sigma^{(n)})}{\sigma^{(n)}} - (1 - y^{(n)}) \frac{\sigma^{(n)}(1 - \sigma^{(n)})}{1 - \sigma^{(n)}} \right) \nabla_{\mathbf{w}}(\mathbf{w}^T \phi(\mathbf{x}^{(n)}))$$

$$= \sum_{n=1}^{N} \left(y^{(n)}(1 - \sigma^{(n)}) - (1 - y^{(n)})\sigma^{(n)} \right) \nabla_{\mathbf{w}}(\mathbf{w}^T \phi(\mathbf{x}^{(n)}))$$

Derivation

- $\log P(\mathbf{y}|\mathbf{w}) = \sum_{n=1}^{N} y^{(n)} \log h^{(n)} + (1 - y^{(n)}) \log(1 - h^{(n)})$
- Gradient (matrix calculus)

$$\sigma^{(n)} \triangleq \sigma\left(\mathbf{w}^T \phi(\mathbf{x}^{(n)})\right) \triangleq h(\mathbf{x}^{(n)}, \mathbf{w})$$

$$\begin{aligned}
&\nabla_{\mathbf{w}} \log P(\mathbf{y}|\mathbf{x}^{(1)}, \ldots \mathbf{x}^{(N)}, \mathbf{w}) \\
&= \sum_{n=1}^{N} \nabla_{\mathbf{w}} \left(y^{(n)} \log h(\mathbf{x}^{(n)}, \mathbf{w}) + (1 - y^{(n)}) \log(1 - h(\mathbf{x}^{(n)}, \mathbf{w})) \right) \\
&= \sum_{n=1}^{N} \left(y^{(n)} \frac{\sigma^{(n)}(1 - \sigma^{(n)})}{\sigma^{(n)}} - (1 - y^{(n)}) \frac{\sigma^{(n)}(1 - \sigma^{(n)})}{1 - \sigma^{(n)}} \right) \nabla_{\mathbf{w}}(\mathbf{w}^T \phi(\mathbf{x}^{(n)})) \\
&= \sum_{n=1}^{N} \left(y^{(n)}(1 - \sigma^{(n)}) - (1 - y^{(n)})\sigma^{(n)} \right) \nabla_{\mathbf{w}}(\mathbf{w}^T \phi(\mathbf{x}^{(n)})) \\
&= \sum_{n=1}^{N} \left(y^{(n)} - \sigma^{(n)} \right) \phi(\mathbf{x}^{(n)})
\end{aligned}$$

Logistic Regression: gradient descent

- Taking the gradient of E(w) gives us

$$\nabla \mathbf{E}(\mathbf{w}) = \sum_{n=1}^{N} (h^{(n)} - y^{(n)}) \phi(\mathbf{x}^{(n)})$$

  – Recall

$$h^{(n)} = p(C_1 | \phi(\mathbf{x}^{(n)})) = \sigma(\mathbf{w}^T \phi(\mathbf{x}^{(n)}))$$

  – This is essentially the same gradient expression that appeared in linear regression with least-squares.
  – Note the error term between model prediction and target value:
    - Logistic regression: $h^{(n)} - y^{(n)} = \sigma(\mathbf{w}^T \phi(x^{(n)})) - y^{(n)}$
    - Cf. Linear regression: $h^{(n)} - y^{(n)} = \mathbf{w}^T \phi(x^{(n)}) - y^{(n)}$

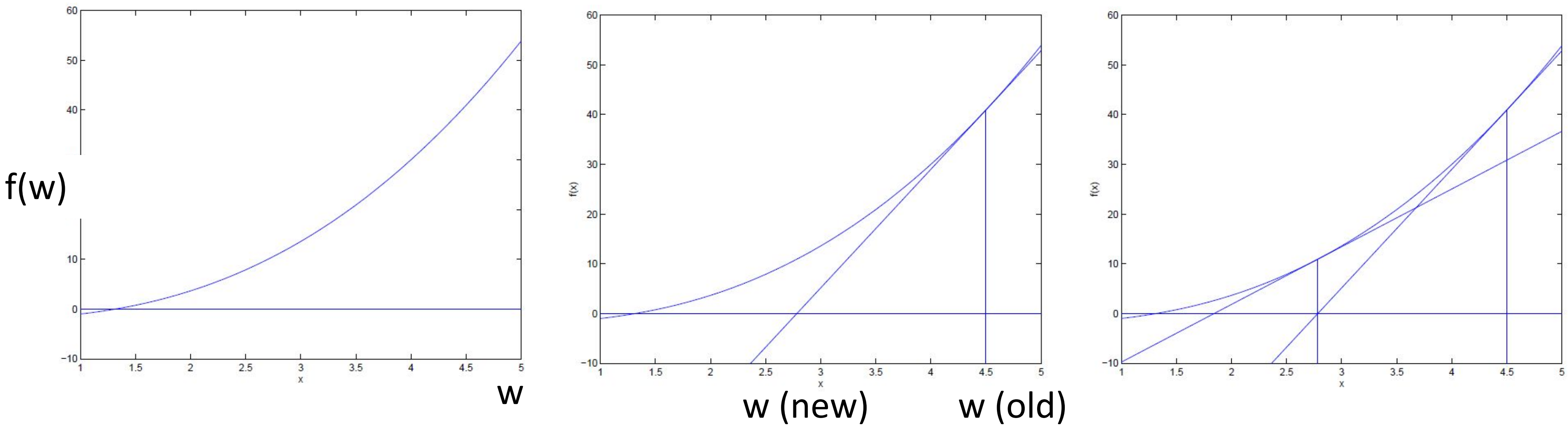
Newton's method

- Goal: Minimizing a general function $E(\mathbf{w})$ (one dimensional case)
  - Approach: solve for $f(\mathbf{w}) = \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = 0$
  - So, how to solve this problem?
- Newton's method (aka Newton-Raphson method)
  - Repeat until convergence:

$$\mathbf{w} := \mathbf{w} - \frac{f(\mathbf{w})}{f'(\mathbf{w})}$$

Newton's method

- Interactively solve until we get f(w) = 0.



- Geometric intuition

$$\mathbf{w} := \mathbf{w} - \frac{f(\mathbf{w})}{f'(\mathbf{w})}$$

Current value

"Slope"

Newton's method

- Now we want to minimize E(w)
  - Convert $E'(\mathbf{w}) = f(\mathbf{w})$
  - Repeat until convergence

$$\mathbf{w} := \mathbf{w} - \frac{E'(\mathbf{w})}{E''(\mathbf{w})}$$

Newton update
when w is a scalar

Newton's method

- Now we want to minimize E(w)
  - Convert $E'(\mathbf{w}) = f(\mathbf{w})$
  - Repeat until convergence

$$\mathbf{w} := \mathbf{w} - \frac{E'(\mathbf{w})}{E''(\mathbf{w})}$$

Newton update when w is a scalar

- This method can be extended for multivariate case:

$$\mathbf{w} := \mathbf{w} - H^{-1} \nabla_{\mathbf{w}} E$$

Newton update when w is a vector

where H is a hessian matrix (evaluated at **w**)

$$H_{ij}(\mathbf{w}) = \frac{\partial^2 E(\mathbf{w})}{\partial \mathbf{w}_i \partial \mathbf{w}_j}$$

Note: For linear regression problem, the Hessian is $\Phi^T \Phi$.

Logistic Regression

- Recall: for linear regression, least-squares has a closed-form solution: $\mathbf{w}_{ML} = (\mathbf{\Phi}^T \mathbf{\Phi})^{-1} \mathbf{\Phi}^T \mathbf{y}$

- This generalizes to weighted-least-squares with an NxN diagonal weight matrix R.

$$\mathbf{w}_{WLS} = (\mathbf{\Phi}^T \mathbf{R} \mathbf{\Phi})^{-1} \mathbf{\Phi}^T \mathbf{R} \mathbf{y}$$

- For logistic regression, however, is non-linear, and there is no closed-form solution. Must iterate (i.e., repeatedly apply Newton steps).

Iterative Solution

- Apply Newton-Raphson method to iterate to a solution w for $\nabla E(\mathbf{w}) = 0$
- This involves least-squares with weights R:

$$R_{nn} = y^{(n)}(1 - y^{(n)})$$

- Since R depends on w (and vice versa), we get *iterative reweighted least squares* (IRLS)

  – where $\mathbf{w}^{(new)} = (\mathbf{\Phi}^T \mathbf{R} \mathbf{\Phi})^{-1} \mathbf{\Phi}^T \mathbf{R} \mathbf{z}$

$$\mathbf{z} = \mathbf{\Phi} \mathbf{w}^{(old)} - \mathbf{R}^{-1}(\mathbf{h} - \mathbf{y})$$

K-Nearest Neighbor Classification

k-Nearest Neighbor

- Training method:
  - Save the training examples (no sophisticated learning)
- At prediction (testing) time:
  - Given a test (query) example ***x,*** find the *k* training examples that are closest to the **x**.

$$kNN(x) = \left\{ \left(x^{(1)'}, y^{(1)'}\right), \left(x^{(2)'}, y^{(2)'}\right), \ldots, \left(x^{(k)'}, y^{(k)'}\right) \right\}$$

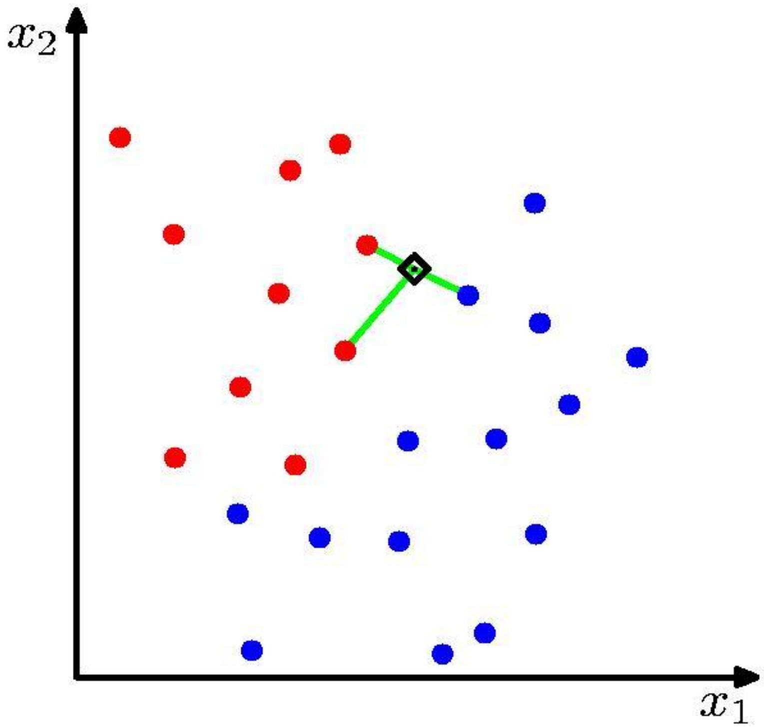
Predict the most frequent class among t_i's from .

$$y = \arg\max_t \sum_{(\mathbf{x}', y') \in kNN(\mathbf{X})} \mathbf{1}[y' = y]$$

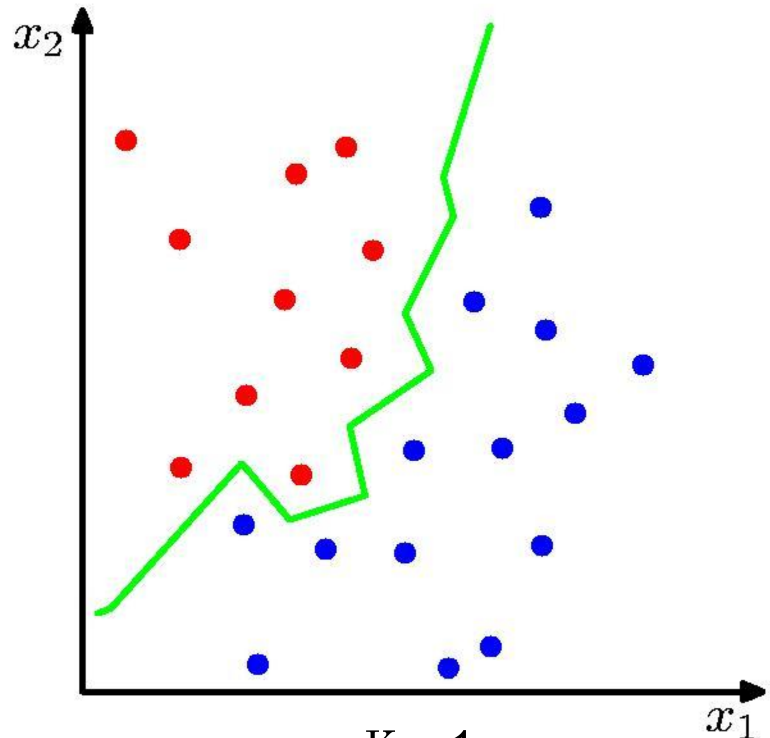
"majority vote"

  - Note: this function can be applied to regression!

K-Nearest-Neighbours for Classification



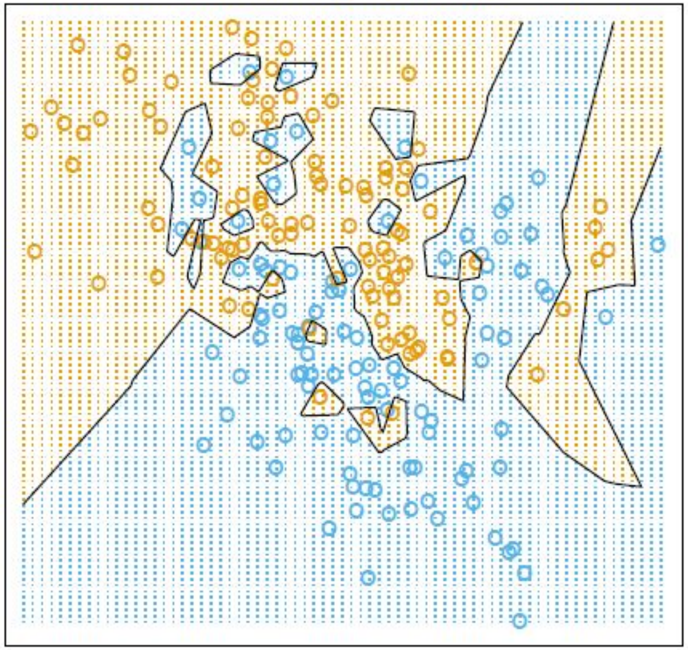
K = 3

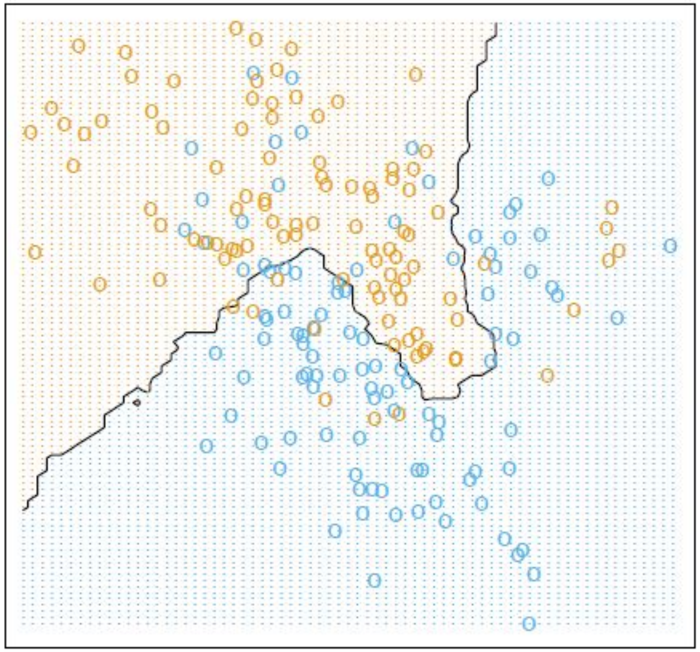


K = 1

Slide credit: Ben Kuipers

K-Nearest-Neighbours for Classification





- K acts as a smother (bias-variance trade-off)
- Classification performance generally improves as N (training set size) increases.
- For $N \to \infty$, the error rate of the 1-nearest-neighbor classifier is never more than twice the optimal error (obtained from the true conditional class distributions). See ESL Ch. 13.3.

Slide credit: Ben Kuipers

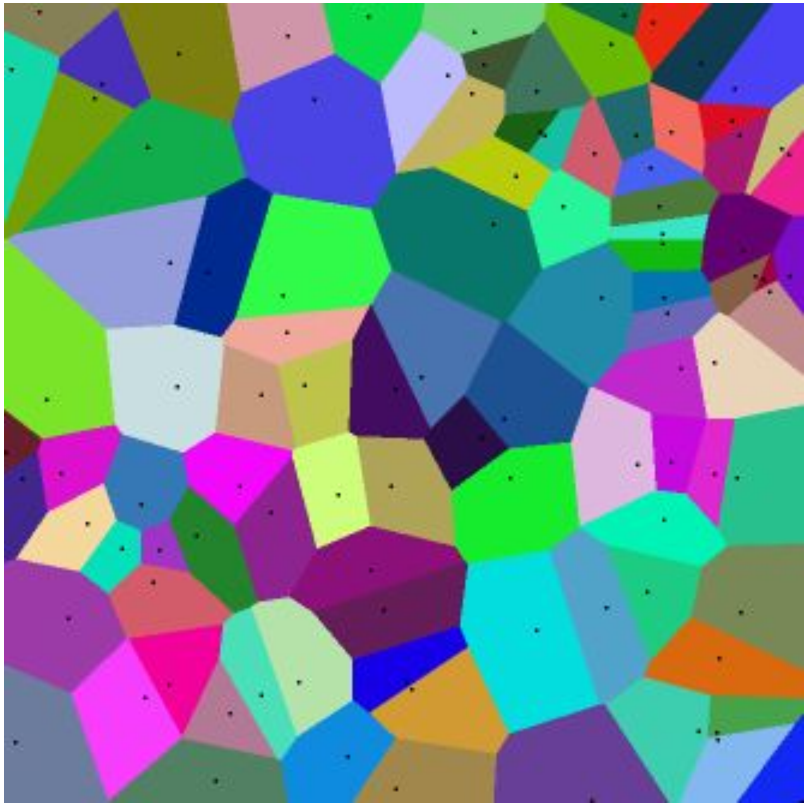
Factors (hyperparameters) affecting kNN

- Distance metric D(x, x’)
  – how to define distance between two examples between x and x’?

- The value of K
  – K determines how much we “smooth out” the prediction

What is the decision boundary?

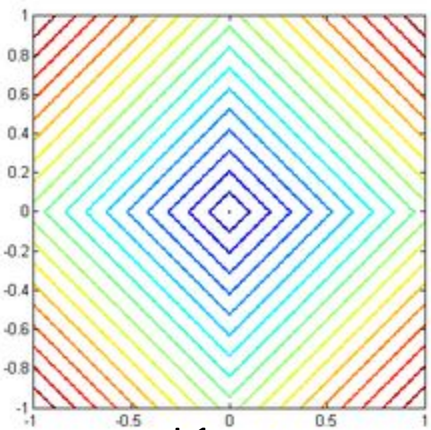
Voronoi diagram: Euclidean (L2) distance



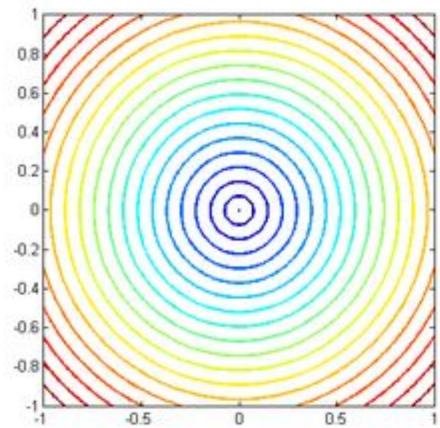
Slide credit: William Cohen

Dependence on distance metric (L^q norm)

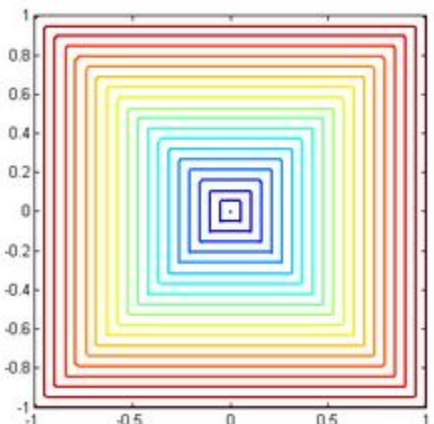
Distance between i-th and j-th example: $\sqrt[q]{\sum_l \left(x_l^{(i)} - x_l^{(j)}\right)^q}$



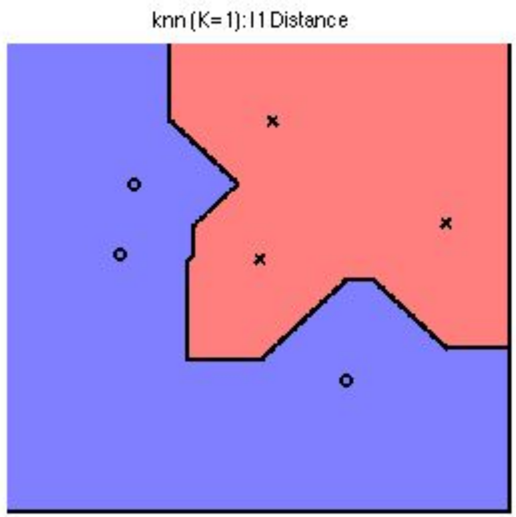
L1 norm

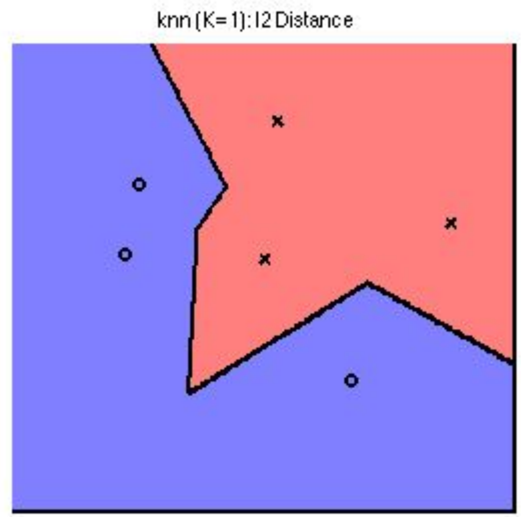


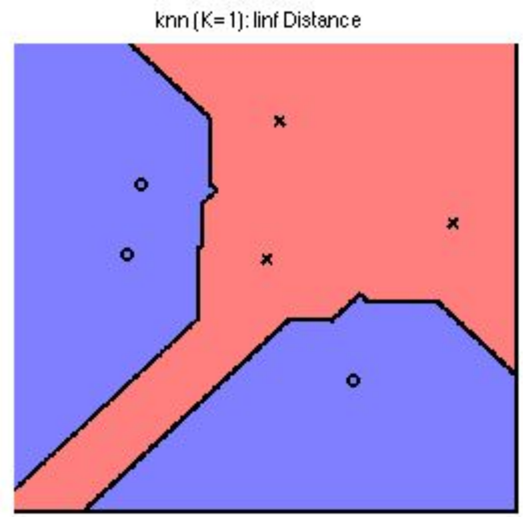
L2 norm



L_∞ norm







Slide credit: Ben Taskar

K-NN: Classification vs Regression

- Note: it is very easy to formulate K-NN into regression/classification
- For classification, where the label y is categorical, we take the “majority vote” over target labels.

$$y = \arg\max_{t} \sum_{(\mathbf{x}',y')\in kNN(\mathbf{x})} \mathbf{1}[y' = y]$$

- For regression, where the label y is real-valued numbers, we take “average” over target labels.

$$y = \frac{1}{k} \sum_{(\mathbf{x}',y')\in kNN(\mathbf{x})} y'$$

Advantage/disadvantages of k-NN methods

- Advantage:
  - very simple and flexible (no assumption on distribution)
  - effective (e.g., for low dimensional inputs)
- 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
  - Not robust to irrelevant features: If x has irrelevant/noisy features, then distance function does not reflect similarity between examples