Artificial Intelligence

CS 534

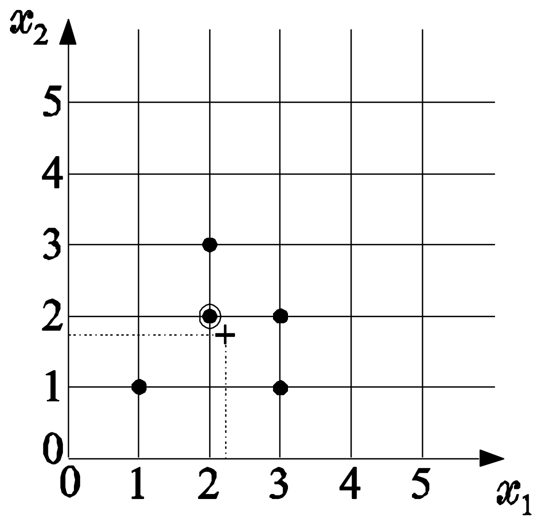
Week 5

Example: K-Means vs k-Medoid

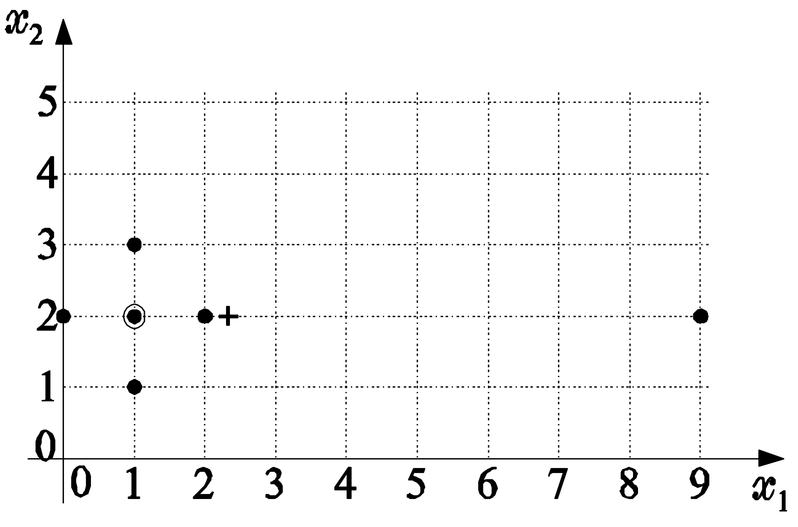
Example: (Illustrates the first two points in the above comparison)

(a) The five-point two-dimensional set stems from the discrete domain $D=\{1,2,3,4,\ldots\}x\{1,2,3,4,\ldots\}$. Its medoid is the circled point and its mean is the "+" point, which does not belong to *D*.

(b) In the six-point two-dimensional set , the point (9,2) can be considered as an outlier. While the outlier affects significantly the mean of the set, it does not affect its medoid.



(a)



(b)

Mean values vs. medoids

Mean Values	Medoids
1. Suited only for continuous domains	1. Suited for either cont. or discrete domains
2. Algorithms using means are sensitive to outliers	2. Algorithms using medoids are less sensitive to outliers
3. The mean possess a clear geometrical and statistical meaning	3. The medoid has not a clear geometrical meaning
4. Algorithms using means are not computationally demanding	4. Algorithms using medoids are more computationally demanding

Popular k-Medoids Algorithms

Algorithms to consider:

- PAM (Partitioning Around Medoids)
- CLARA (Clustering LARge Applications)
- CLARANS (Clustering Large Applications based on RANdomized Search)

Cluster tendency and validity

Cluster validity

- Cluster validity: a task that evaluates quantitatively the results of a clustering algorithm

- A clustering structure ***C*,** resulting from an algorithm may be either
  - A hierarchy of clusterings or
  - A single clustering

Possible approaches

Cluster validity may be approached in three possible directions:

- ***C*** is evaluated in terms of an independently drawn structure, imposed on *X a priori*. The criteria used in this case are called external criteria

- ***C*** is evaluated in terms of quantities that involve the vectors of *X* themselves (e.g., proximity matrix). The criteria used in this case are called internal criteria

- ***C*** is evaluated by comparing it with other clustering structures, resulting from the application of the same clustering algorithm but with different parameter values, or other clustering algorithms, on *X*. Criteria of this kind are called relative criteria

Cluster validity for the cases of external and internal criteria

- Hypothesis testing is employed
- The null hypothesis H_0, which is a statement of randomness concerning the structure of X, is defined
- The generation of a reference data population under the random hypothesis takes place
- An appropriate statistic, q, whose values are indicative of the structure of a data set, is defined. The value of q that results from our data set X is compared against the values obtained for q when the elements of the reference (random) population are considered

Ways for generating reference populations under the null hypothesis (each one used in different situations):

- Random position hypothesis: all arrangements of vectors in a specific region are equally likely
- Random graph hypothesis: adopted with only internal information
- Random label hypothesis: all possible label mappings are equally likely

Statistics suitable for external criteria

- For the comparison of ***C*** with an independently drawn partition ***P*** of X
  - Rand statistics
  - Jaccard statistics
  - Fowlkes-Mallows index
  - Hubert's Γ statistics
  - Normalized Γ statistics

- For assessing the agreement between ***P*** and the proximity matrix P
  - Γ statistics

Statistics suitable for internal criteria

- Validation of hierarchy of clusterings
  - Cophenetic correlation coefficient (*CPCC*)
  - γ statistics
  - Kudall's τ statistics

- Validation of individual clusterings
  - $\mathit{\Gamma}$ statistics
  - Normalized $\mathit{\Gamma}$ statistics

Cluster validity for the cases of relative criteria

Let ***A*** denote the set of parameters of a clustering algorithm.

Statement of the problem:

- *"Among the clusterings produced by a specific clustering algorithm, for different values of the parameters in **A**, choose the one that best fits the data set X".*

We consider two cases

(a) ***A** does not contain the number of clusters m.*
The estimation of the best set of parameter values is carried out as follows:

- Run the algorithm for a wide range of values of its parameters.
- Plot the number of clusters, *m*, versus the parameters of ***A.***
- Choose the widest range for which *m* remains constant.
- Adopt the clustering that corresponds to the values of the parameters in ***A*** that lie in the **middle** of this range.

The cases of relative criteria (contd.)

(b) ***A*** *does contain the number of clusters m.*

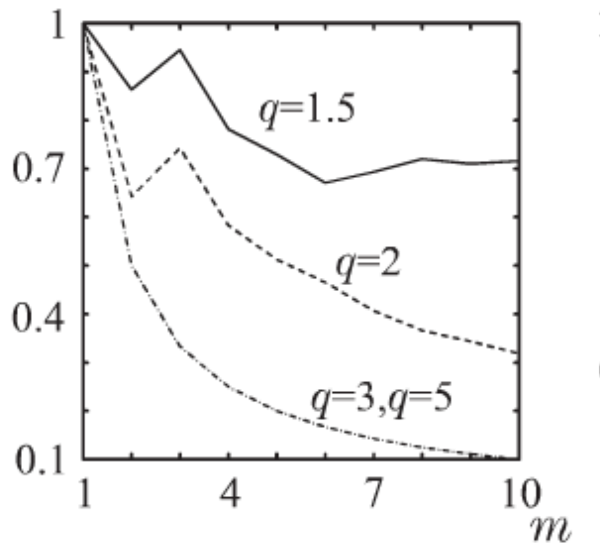
The estimation of the best set of parameter values is carried out as follows:

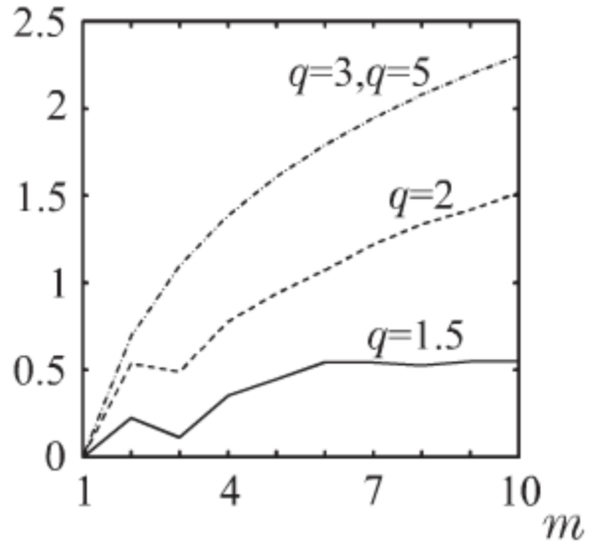
- Select a suitable performance index *q* (the best clustering is identified in terms of *q*)
- For $m=m_{min}$ to m_{max} (chosen *a priori*)
  - Run the algorithm *r* times using different sets of values for the other parameters of ***A*** and each time compute *q*
  - Choose the clustering that corresponds to the best *q*

  End for
- Plot the best values of *q* (for each *m)* versus *m*
- Seek the maximum/minimum of the plot , according to whether large/ small values of *q* indicate good clustering
- The procedure works well if *q* exhibits no trend with growth of *m*

The cases of relative criteria (contd.)

- Often *q*, however, exhibits and increasing/ decreasing trend as *m* grows
- Thus, we no longer can adopt the previous strategy
- Instead, we seek for for values of *m* at which a significant local change in value *q* occurs
- The presence of a significant knee indicates the number of clusters underlying *X*. Adopt the clustering that corresponds to that knee
- The absence of such a knee indicates that *X* possesses no clustering structure.





Supervised learning: Vocabulary, notations and basic concepts

A learning problem

Definition of well-posed learning problem (Tom Mitchell, 1998):
A computer program is said to learn from experience *E* w.r.t some task *T* and some performance measure *P* if its performance on *T*, measured by *P*, is improved with experience *E.*

A learning problem

Definition of well-posed learning problem (Tom Mitchell, 1998):
A computer program is said to learn from experience *E* w.r.t some task *T* and some performance measure *P* if its performance on *T*, measured by *P*, is improved with experience *E.*

Regression Problem: Continuous value output

A learning problem

Definition of well-posed learning problem (Tom Mitchell, 1998):
A computer program is said to learn from experience *E* w.r.t some task *T* and some performance measure *P* if its performance on *T*, measured by *P*, is improved with experience *E.*

Regression Problem: Continuous value output

Classification Problem: Discrete value output

A learning problem

Definition of well-posed learning problem (Tom Mitchell, 1998):
A computer program is said to learn from experience *E* w.r.t some task *T* and some performance measure *P* if its performance on *T*, measured by *P*, is improved with experience *E.*

Regression Problem: Continuous value output

Classification Problem: Discrete value output

Example 1: Predicting whether a tumor is benign or cancerous

A learning problem

Definition of well-posed learning problem (Tom Mitchell, 1998):
A computer program is said to learn from experience *E* w.r.t some task *T* and some performance measure *P* if its performance on *T*, measured by *P*, is improved with experience *E.*

Regression Problem: Continuous value output

Classification Problem: Discrete value output

Example 1: Predicting whether a tumor is benign or cancerous

Example 2: Predicting he amount of snow (inches) in Worcester, MA based on the average summer temperature

A learning problem

Definition of well-posed learning problem (Tom Mitchell, 1998):
A computer program is said to learn from experience *E* w.r.t some task *T* and some performance measure *P* if its performance on *T*, measured by *P*, is improved with experience *E.*

Regression Problem: Continuous value output

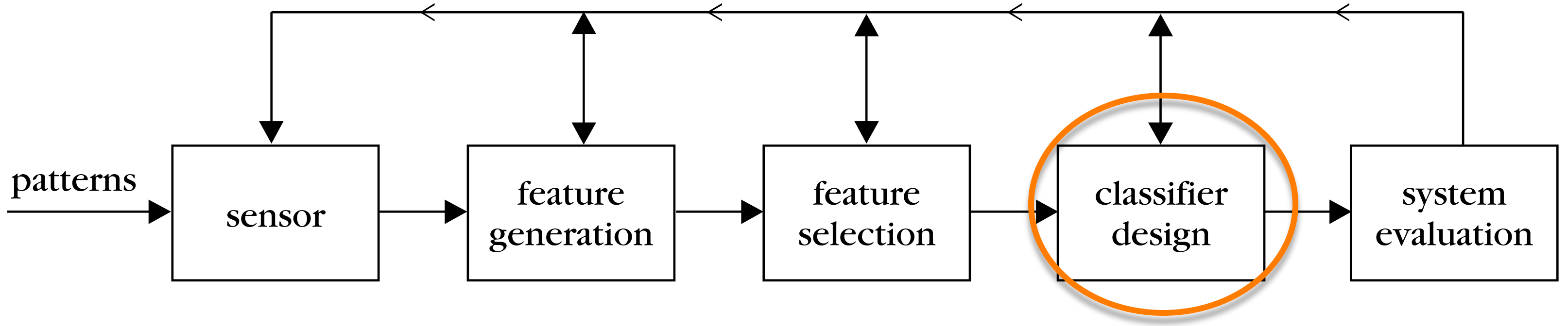
Classification Problem: Discrete value output

Example 1: Predicting whether a tumor is benign or cancerous

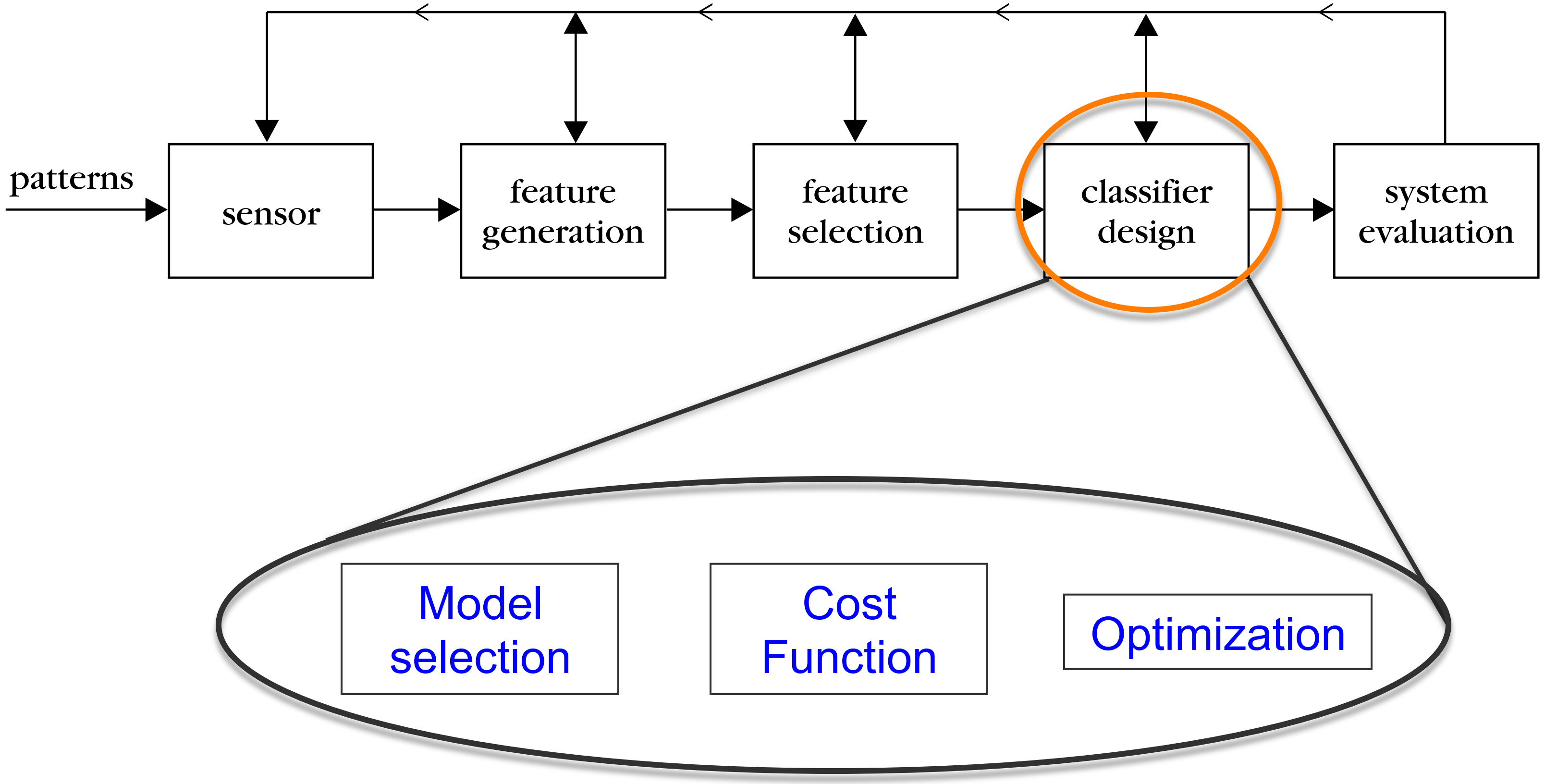
Example 2: Predicting he amount of snow (inches) in Worcester, MA
based on the average summer temperature

Example 3: Classification of geographical locations on a satellite image

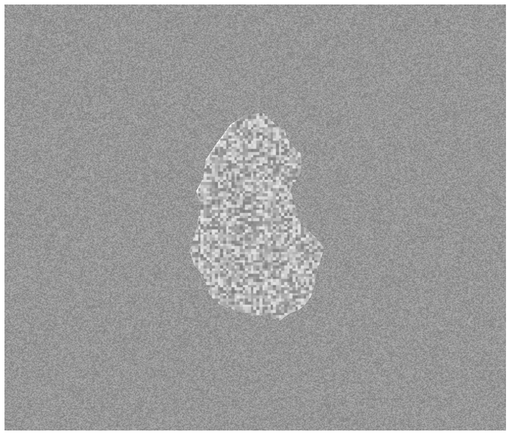
Features, Feature vectors and Classifiers

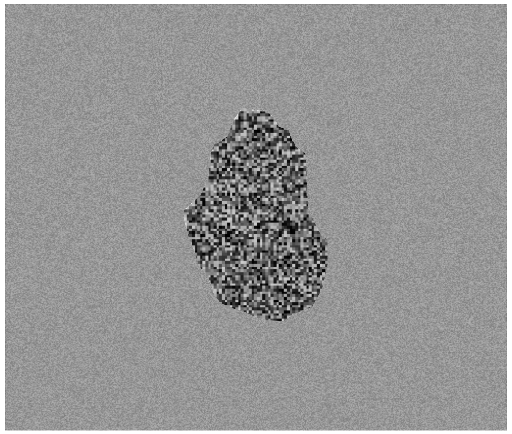


Features, Feature vectors and Classifiers

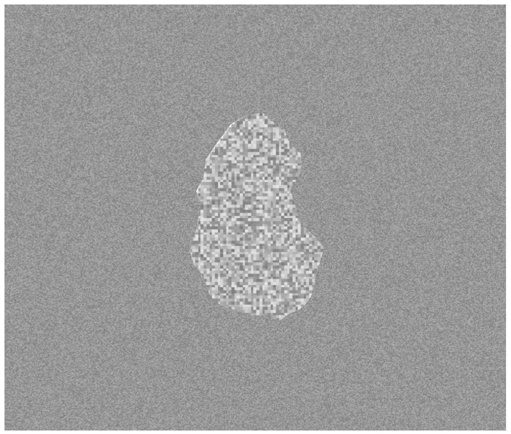


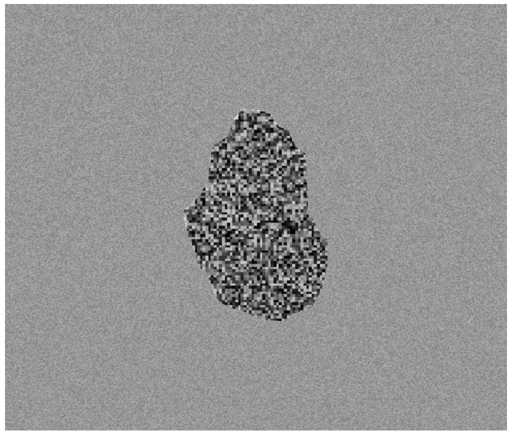
Features and Feature vectors


Class A: Benign Tumor


Class B: Cancer

Features and Feature vectors

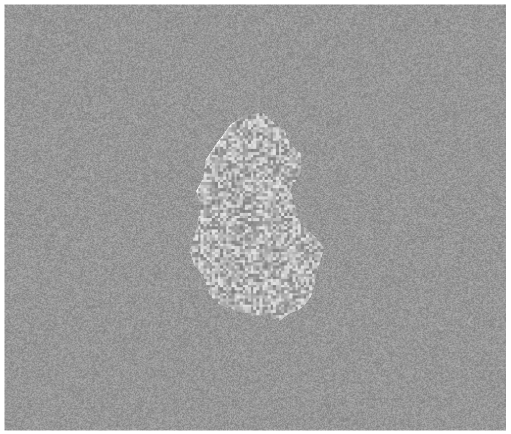

Class A: Benign Tumor

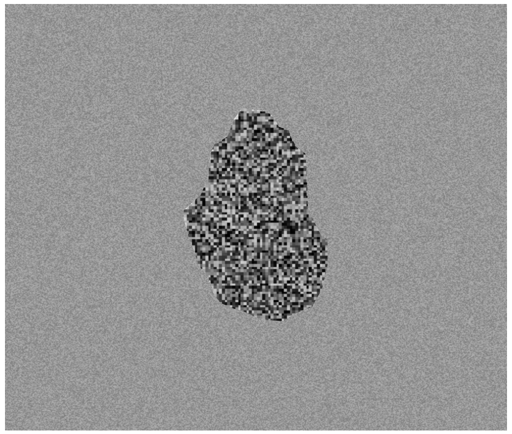

Class B: Cancer

- We need to identify the measurable quantities that make those two images (specifically, the regions) distinct

- Consider the following simple set of two features:
  - x_1: Mean value of intensity in a region of image
  - x_2: Standard deviation of around the mean

Features and Feature vectors

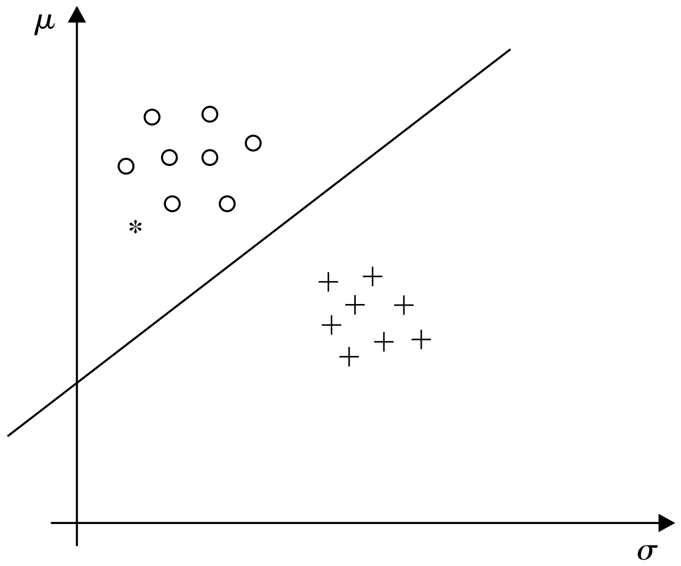




Class A: Benign Tumor   Class B: Cancer

- We need to identify the measurable quantities that make those two images (specifically, the regions) distinct

- Consider the following simple set of two features:
  - x_1: Mean value of intensity in a region of image
  - x_2: Standard deviation of around the mean



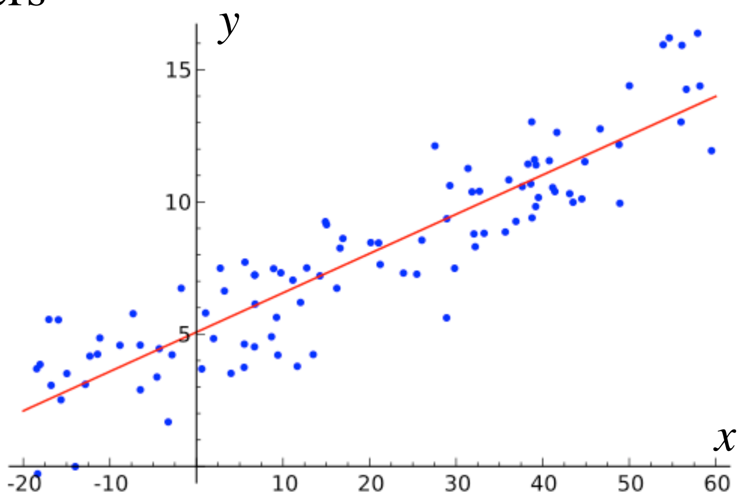
Notation

- m = number if training examples
- n = dimensionality of feature vector space
- x = input feature vector
- y = output variable
- (x, y) = a single training example
- $(x^{(i)}, y^{(i)})$ = i-th training example
- x_j = the j-th feature (coordinate) of x
- h is the hypothesis function (in regression or classification)

Regression basics

- Linear regression: data can be modeled as a linear function
  - $h_w(x) = w_0 + w_1 x$, where w_i are parameters

  Sometimes, different notation is used
  - $h_\theta(x) = \theta_0 + \theta_1 x$



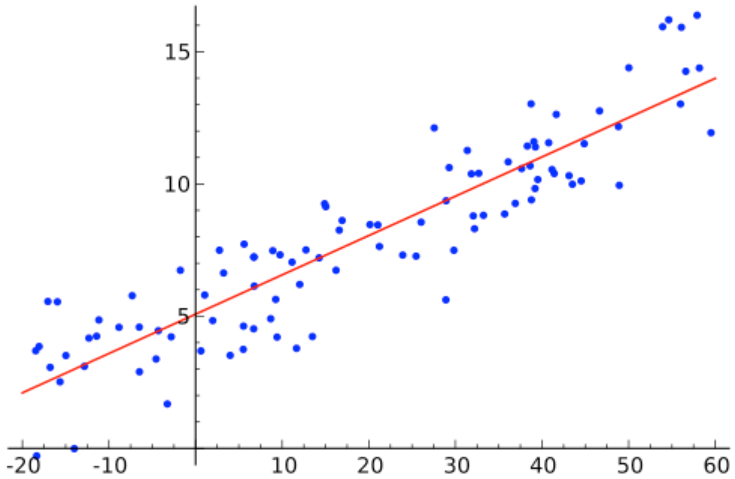
Cost function

Q: How do we know that we perform well for the classification or regression problem?

A: Cost function!

<u>Specifically</u>: Choose parameters w_i such that $h_w(x)$ is close to y for the training set values:

$$\min_{w_0, w_1} \frac{1}{2m} \sum_{i=1}^{m} \left(h_w(x^{(i)}) - y^{(i)} \right)^2$$



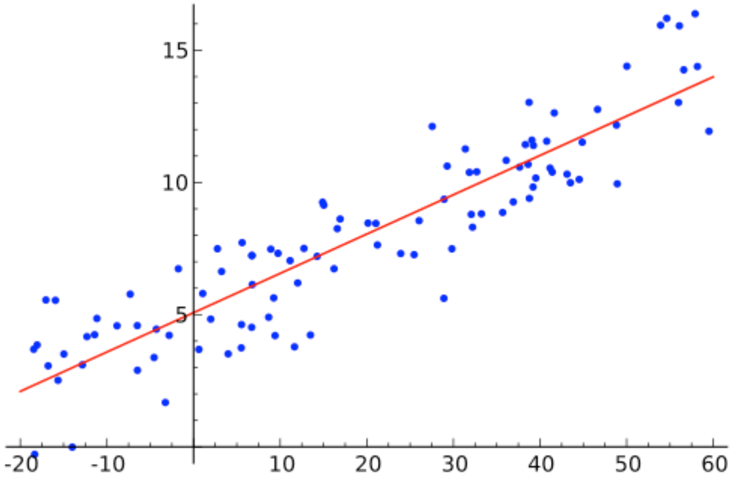
Cost function

Q: How do we know that we perform well for the classification or regression problem?

A: Cost function!

Specifically: Choose parameters w_i such that $h_w(x)$ is close to y for the training set values:

$$\min_{w_0, w_1} \frac{1}{2m} \sum_{i=1}^{m} \left(h_w(x^{(i)}) - y^{(i)} \right)^2$$



Therefore, our cost function is defined as:

$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_w(x^{(i)}) - y^{(i)} \right)^2$$

need to minimize it!

Optimization of cost function

- A popular technique: Gradient descent algorithm
- Given $J(w_0, w_1, \ldots, w_n)$, a general case, minimize it over w_i
- Idea 1:
  - Start with some initial guess for $w_0, w_1, \ldots$
  - Keep changing the values of w_0 trying to reduce J
- Idea 2:
  - Any 2-dimensional surface (works for 3- or even n-dimensions) will have valleys and hills
  - What is the fastest way going down with a small step? Following the gradient vector!

Gradient descent

Repeat until convergence:

$$\{ \quad w_j := w_j - \alpha \frac{\partial}{\partial w_j} J(w_0, w_1, \ldots),\ j = 0,1,\ldots,n \quad \}$$

- α is called a learning rate
- $\frac{\partial}{\partial w_j} J$ is called a derivative term
- How to update w_i? Simultaneously!

Intuition

- Consider a simple example of a single variable cost function $J(w_1)$
- $$w_1 = w_1 - \alpha \frac{\partial}{\partial w_1} J(w_1) =$$
  $$w_1 - \alpha * \text{PosNumber} \quad \text{or} \quad w_1 - \alpha * \text{NegNumber}$$
- If α is too small, then the convergence is too slow
- If α is too big, the the gradient descent can “overshoot”
- If we got exactly the local minimum, next step will be unchanged

More intuition

- Gradient descent can converge to a minimum even when α is fixed
- As we approach a local minimum, we will automatically make smaller steps
- If there are multiple local minima net to each other, we could get the wrong one

Gradient descent for linear regression

Goal: Apply gradient descent to minimize $J(w_0, w_1)$ where

$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_w(x^{(i)}) - y^{(i)} \right)^2$$

- Need to find

$$\frac{\partial}{\partial w_j} J(w_0, w_1)$$

- After taking derivatives:

$$\frac{\partial}{\partial w_0} J(w_0, w_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_w(x^{(i)}) - y^{(i)} \right)$$

$$\frac{\partial}{\partial w_1} J(w_0, w_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_w(x^{(i)}) - y^{(i)} \right) x^{(i)}$$

Algorithm

Repeat until convergence:

{

$$w_0 := w_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_w(x^{(i)}) - y^{(i)} \right)$$

$$w_1 := w_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_w(x^{(i)}) - y^{(i)} \right) x^{(i)}$$

}

Note: For linear regression, the cost function is always a *convex* function – bowl shape

Remarks

- The reviewed algorithm is so-called "Batch" gradient descent
  - Batch: Using all training examples for each step of gradient descent

- Gradient descent is known to scale better than the analytical solutions (such as normal equations)

A case of multi–dimensional feature space

- Our hypothesis is generalized:

$$h_w(x) = w_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n \text{ , where } w_i \text{ are parameters}$$

- Let’s define another coordinate $x_0^{(i)}=1$ for each example i. Our feature vector $x^{(i)}$ is now in a $(\mathrm{n}+1)$-dimensional space!

- Our parameter vector $w^{(i)}$ is also in a $(\mathrm{n}+1)$-dimensional space

- Therefore, to vectorize our hypothesis:

$$h_w(x) = w^{\mathrm{T}}x$$

- Cost function:

$$J(w) = J(w_0, w_1, \ldots, w_n) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_w(x^{(i)}) - y^{(i)}\right)^2$$

Modified algorithm

Repeat until convergence:

$$\{ \quad w_j := w_j - \alpha \frac{\partial}{\partial w_j} J(w)$$

$$\text{(Simultaneously update for every } j = 0,1\ldots,n)$$

$$\}$$

<u>Convergence criterion</u>:

$$\left\| w^{New} - w^{Old} \right\| = \sqrt{\sum_{i=0}^{n} \left(w_i^{New} - w_i^{Old} \right)^2} \leq \varepsilon_0 - \text{a predefined threshold}$$

Modified algorithm: implementation-ready

Repeat until convergence:

$\{$

$$w_j := w_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_w(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

(Simultaneously update for every $j = 0, 1 \ldots, n$)

$\}$

How to improve the method: Feature scaling

- In general, many optimization methods perform better (*e.g.*, faster) if the features have comparable (same order of magnitude) values

- Gradient descent is no exception

- Solution: Scaling/Normalization to [0;1] or [-1;1]

- A simple way--mean normalization:

  $x_i \leftarrow x_i - \mu$, where μ is the mean over the values for the i-th coordinate

- A better way--standardization:

  $x_i \leftarrow \frac{x_i - \mu}{\sigma}$, where μ is the mean and σ is standard deviation

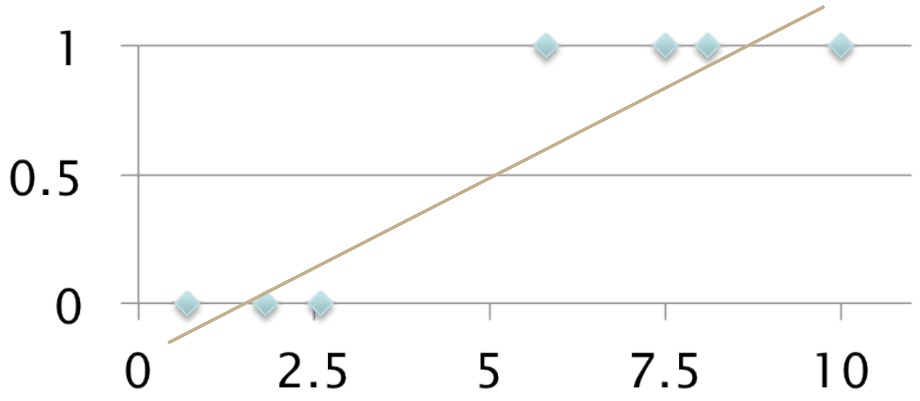
How to improve the method: Learning rate (α)

- Can we choose α?
- If we plot $J(w)$ against the number of iterations, $J(w)$ should decrease after each iteration
- If $J(w)$ increase, we need to choose a different (smaller) learning rate
- If $J(w)$ oscillates, we need to choose a smaller learning rate
- How to choose? A 10-fold increase: 0.001, 0.01, 0.1, …

  Could be a ~3-fold increase: 0.001, 0.003, 0.01, …

Moving to classification problem: Logistic regression

- Consider a problem of classifying a certain type of tumor as benign or cancerous, depending on its size:



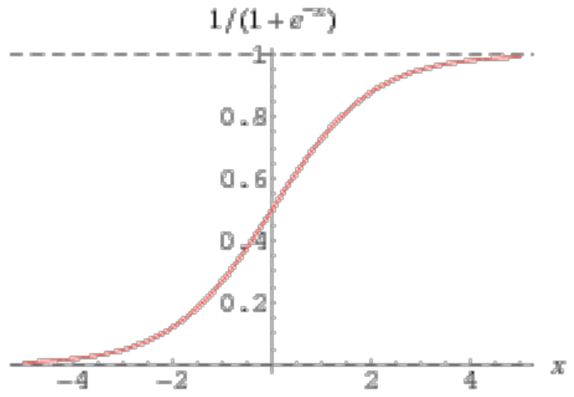
- A possible solution:
  - Apply a linear regression: $h_w(x) = w^T x$
  - Threshold of $h_w(x) =$ 0.5 would be used to classify output as benign (≤0.5) or cancerous (>0.5)

  - Problem: If we add another “cancerous” data point that corresponds to a large size, the regression line will change although it shouldn't't!

Basics of logistic regression

- Intuition: We want $0 \le h_w(x) \le 1$

- Solution: $h_w(x)=g(w^T x)$, where $g(z) = \frac{1}{1+e^{-Z}}$ sigmoid (logistic) function



- We need to fit the parameters from w

- Interpretation: $h_w(x)$= *estimated probability that $y=1$, on input x*

- Formally: $h_w(x)= P(y=1 \mid x;w)$, therefore $P(y=0 \mid x;w) = 1 - P(y=1 \mid x;w)$

Decision boundary

- Prediction follows the same idea:
  - "y=1" when $h_w(x) > 0.5$
  - "y=0" otherwise

- We know that for the exponential function:
  - g$(z) > 0.5$ when $z > 0$

- Therefore:
  - $h_w(x) > 0.5$ when $w^{\mathrm{T}}x > 0$
  - $h_w(x) \leq 0.5$ when $w^{\mathrm{T}}x \leq 0$

Decision boundary: Example

- Consider $h_w(x)=g(w^Tx) = g(w_0x_0+w_1x_1+w_2x_2)=$
  - Let's choose $w_0=-3, w_1=1, w_2=1$

- Therefore:
  - "y=1" when $w^Tx > 0$: $-3+x_1+x_2 > 0$
  - "y=0" when $w^Tx \leq 0$: $-3+x_1+x_2 \leq 0$

- Line $x_1+x_2 = 3$ is called *decision boundary*

- The decision boundary is the property of hypothesis (including our parameter vector w), but NOT the dataset!

- Decision boundaries could be non-linear:
  - "y=1" when: $-1+x_1^2+x_2^2 > 0$
  - "y=0" when $w^Tx \leq 0.5$: $-1+x_1^2+x_2^2 \leq 0$

Cost function

- To choose the parameters, let's go to the linear regression cost function and modify it:

$$J(w) = J(w_0, w_1, \ldots, w_n) = \frac{1}{m}\sum_{i=1}^{m} \underbrace{\frac{1}{2}\left(h_w(x^{(i)}) - y^{(i)}\right)^2}_{\text{Replace by } cost(h_w(x^{(i)}), y)}$$

- Problem: If we leave the same formula (theoretically possible), our cost function becomes non-convex

- Gradient descent will most like get stacked at a local minimum

- Solution: Finding a cost function that is convex

$$cost(h_w(x), y) = \begin{cases} -\log(h_w(x)), \text{ if } y = 1 \\ -\log(1 - h_w(x)), \text{ if } y = 0 \end{cases}$$

Property of the new cost function

- It could be shown that the function is convex

- For $y=1$:
  - if $h_w(x)=1$, $cost = 0$
  - with $h_w(x)$ approaching 0 , $cost$ will approach ∞
  - That means penalizing learning algorithm by a huge number

- For $y=0$:
  - if $h_w(x)=0$, $cost = 0$
  - with $h_w(x)$ approaching 1 , $cost$ will approach ∞

- A more compact way to write the same cost function:

$$cost(h_w(x), y) = -y\log(h_w(x)) - (1-y)\log(1-h_w(x))$$

Finalized cost function

$$J(w) = J(w_0, w_1, \ldots, w_n) = \frac{1}{m}\sum_{i=1}^{m} cost(h_w(x), y)$$

$$= \frac{1}{m}\left[\sum_{i=1}^{m} -y\log(h_w(x)) - (1-y)\log(1-h_w(x))\right]$$

- Fitting parameters $w_0, w_1, \ldots, w_n$: use Gradient Descent (algorithm looks identical to linear regression, but it is different in $h_w(x)$)

- To make a prediction given new x :

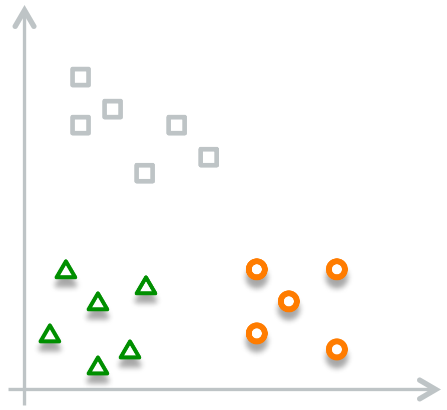
$h_w(x)=g(w^{\mathrm{T}}x)$, where $g(z) = \frac{1}{1+e^{-z}}$

Multiclass classification

- Emails: Spam detection vs. Foldering (GMAIL Folders: Main/Social/Advertisement)

- Disease: Healthy/Sick vs Healthy/Cold/Flu/West Nile

- One-vs-all classification (one-vs-rest)
  Idea: convert multi-class to several binary classifications

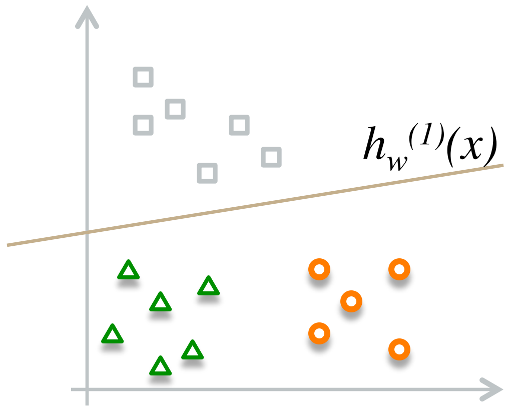


Multiclass classification

- Emails: Spa detection vs. Foldering (GMAIL Folders: Main/ Social/Advertisement)

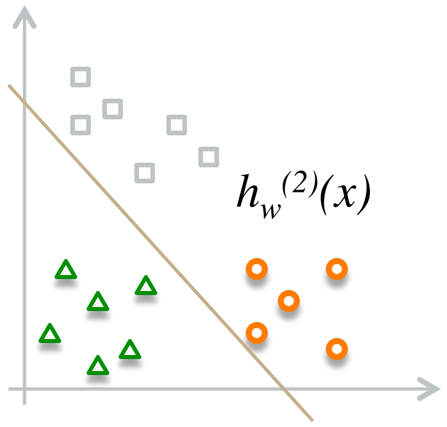
- Disease: Healthy/Sick vs Healthy/Cold/Flu/West Nile

- One-vs-all classification (one-vs-rest)
Idea: convert multi-class to several binary classifications



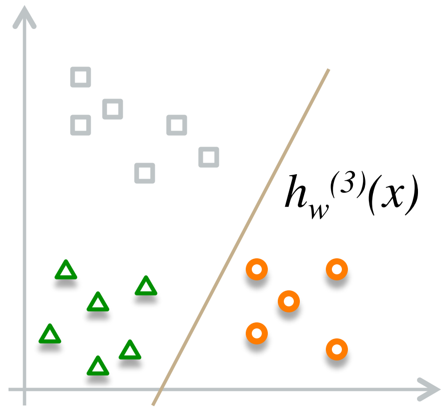
Multiclass classification

- Emails: Spa detection vs. Foldering (GMAIL Folders: Main/ Social/Advertisement)
- Disease: Healthy/Sick vs Healthy/Cold/Flu/West Nile
- One-vs-all classification (one-vs-rest)
  Idea: convert multi-class to several binary classifications



Multiclass classification

- Emails: Spa detection vs. Foldering (GMAIL Folders: Main/ Social/Advertisement)
- Disease: Healthy/Sick vs Healthy/Cold/Flu/West Nile
- One-vs-all classification (one-vs-rest)
  Idea: convert multi-class to several binary classifications



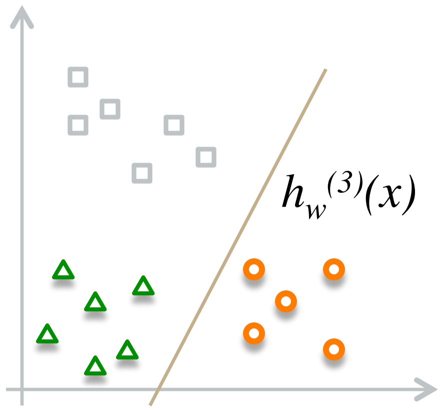
Multiclass classification

- Emails: Spa detection vs. Foldering (GMAIL Folders: Main/ Social/Advertisement)

- Disease: Healthy/Sick vs Healthy/Cold/Flu/West Nile

- One-vs-all classification (one-vs-rest)
  Idea: convert multi-class to several binary classifications

- As a result we have 3 binary classifiers, each one is trained to recognize one class



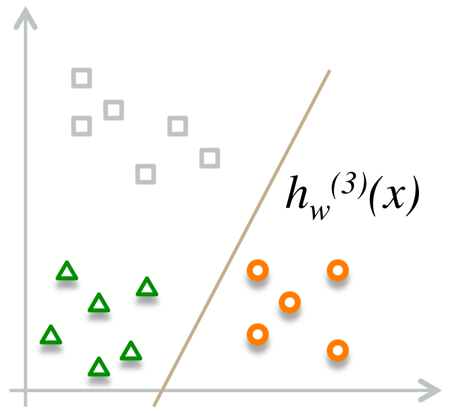
Multiclass classification

- Emails: Spa detection vs. Foldering (GMAIL Folders: Main/Social/Advertisement)

- Disease: Healthy/Sick vs Healthy/Cold/Flu/West Nile

- One-vs-all classification (one-vs-rest)
  Idea: convert multi-class to several binary classifications

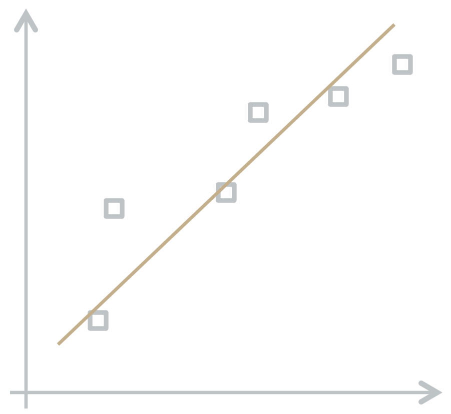
- As a result we have 3 binary classifiers, each one is trained to recognize one class



- For a new element: Apply all and select the most optimistic one: max $(h_w^{(i)}(x))$

Overfitting and underfitting

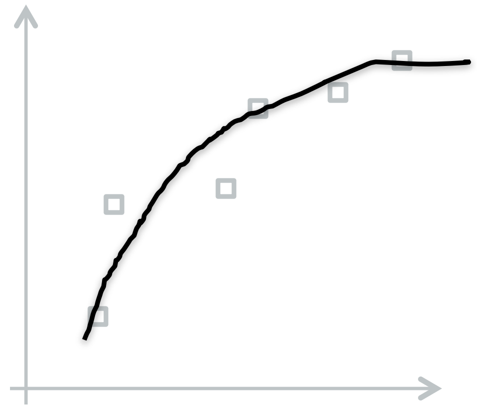
- Consider linear regression



- Fitting a linear function
  - But the data “plateau”
  - Underfitting or “high bias”

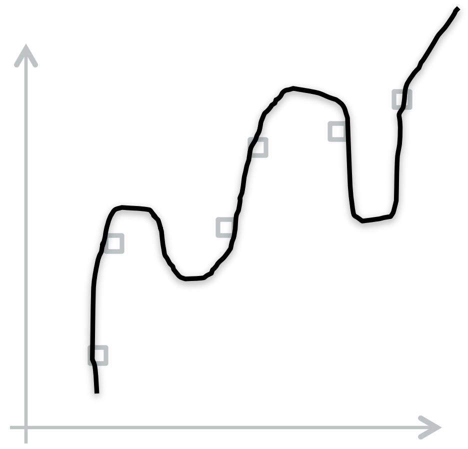
Overfitting and underfitting

- Consider linear regression
- Fitting a linear function
  - But the data “plateau”
  - Underfitting or “high bias”
- Fitting a quadratic function
  - Seems to be a good fit
  - “Just right”



Overfitting and underfitting

- Consider linear regression
- Fitting a linear function
  - But the data “plateau”
  - Underfitting or “high bias”
- Fitting a quadratic function
  - Seems to be a good fit
  - “Just right”
- Fitting a 4-degree polynomial
  - While the fit is perfect , the function does not look as a plausible solution
  - Overfitting or “high variance”: not enough data for the hypothesis



Overfitting and underfitting

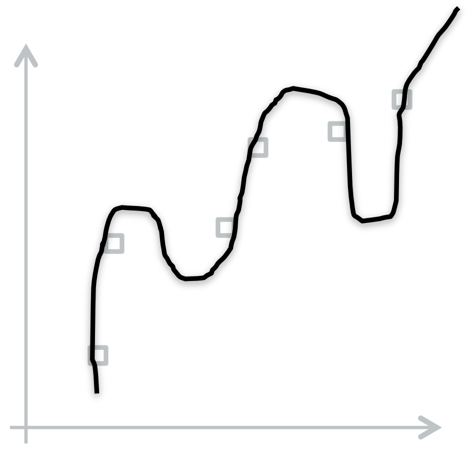
- Consider linear regression

- Fitting a linear function
  - But the data “plateau”
  - Underfitting or “high bias”

- Fitting a quadratic function
  - Seems to be a good fit
  - “Just right”

- Fitting a 4-degree polynomial
  - While the fit is perfect , the function does not look as a plausible solution
  - Overfitting or “high variance”: not enough data for the hypothesis

- Same thing happens with the classification (for instance with linear regression)



Addressing overfitting

- If we have many features, the learned hypothesis may fit the data very well but fail to generalize (that is predict new examples)

- It is very difficult to address overfitting by visualization (especially when we have any features)

Addressing overfitting

- If we have many features, the learned hypothesis may fit the data very well but fail to generalize (that is predict new examples)

- It is very difficult to address overfitting by visualization (especially when we have any features)

- Solutions:
  - Reduce number of features
    - Manually select which features to keep
    - Use a model selection algorithm (will cover later in the course)

Addressing overfitting

- If we have many features, the learned hypothesis may fit the data very well but fail to generalize (that is predict new examples)

- It is very difficult to address overfitting by visualization (especially when we have any features)

- Solutions:
  - Reduce number of features
    - Manually select which features to keep
    - Use a model selection algorithm (will cover later in the course)
  - Regularization
    - Keep all the features but reduce the magnitude/values of parameters w_i
    - Works well when we have many features each of which contributes a little bit to the value of y

Intuition: Cost function

- Consider 2 hypothesis for our linear regression:
  - $w_0+w_1x+w_2x^2$: Just right
  - $w_0+w_1x+w_2x^2+w_3x^3+w_4x^4$: Overfitting

Intuition: Cost function

- Consider 2 hypothesis for our linear regression:
  - $w_0+w_1x+w_2x^2$: Just right
  - $w_0+w_1x+w_2x^2+w_3x^3+w_4x^4$: Overfitting

- Lets penalize and make w_3 and w_4 very small :

$$\min_{w_i} \frac{1}{2m}\sum_{i=1}^{m}\left(h_w(x^{(i)})-y^{(i)}\right)^2+1000w_3+1000w_4$$

- With this cost function we do need to make w_3 and w_4 very small. Thus, our 4-degree polynomial function will behave more like a quadratic one

Intuition: Cost function

- Consider 2 hypothesis for our linear regression:
  - $w_0+w_1x+w_2x^2$: Just right
  - $w_0+w_1x+w_2x^2+w_3x^3+w_4x^4$: Overfitting

- Lets penalize and make w_3 and w_4 very small :

$$\min_{w_i} \frac{1}{2m}\sum_{i=1}^{m}\left(h_w(x^{(i)})-y^{(i)}\right)^2+1000w_3+1000w_4$$

- With this cost function we do need to make w_3 and w_4 very small. Thus, our 4-degree polynomial function will behave more like a quadratic one

- Overall idea: small values for the parameters, which leads to simpler hypotheses, which are less prone to overfitting

Modified cost function

- The updated cost function to minimize:

$$J(w_0, w_1) = \frac{1}{2m}\left[\ \sum_{i=1}^{m}\left(h_w(x^{(i)}) - y^{(i)}\right)^2 + \lambda \sum_{i=1}^{n} w_i{}^2\ \right]$$

- Note that we have no *w_0*
  - This is by convention
  - In practice – little or no difference

- Parameter λ controls the trade-off between 2 terms
  - If λ is very large, then we will end up penalizing our parameters very heavily => $w_i \approx 0$ and $h_w(x) = w_0$, which is underfitting
  - If λ is very small, then no effect on the parameters => overfitting

Logistic regression is similar

$$J(w) = -\frac{1}{m}\left[\sum_{i=1}^{m} y\log(h_w(x)) + (1-y)\log(1-h_w(x))\right] + \frac{\lambda}{2m}\sum_{i=1}^{n} w_i^{\;2}$$

- Parameter lambda regulates how overcomplexified is the decision boundary

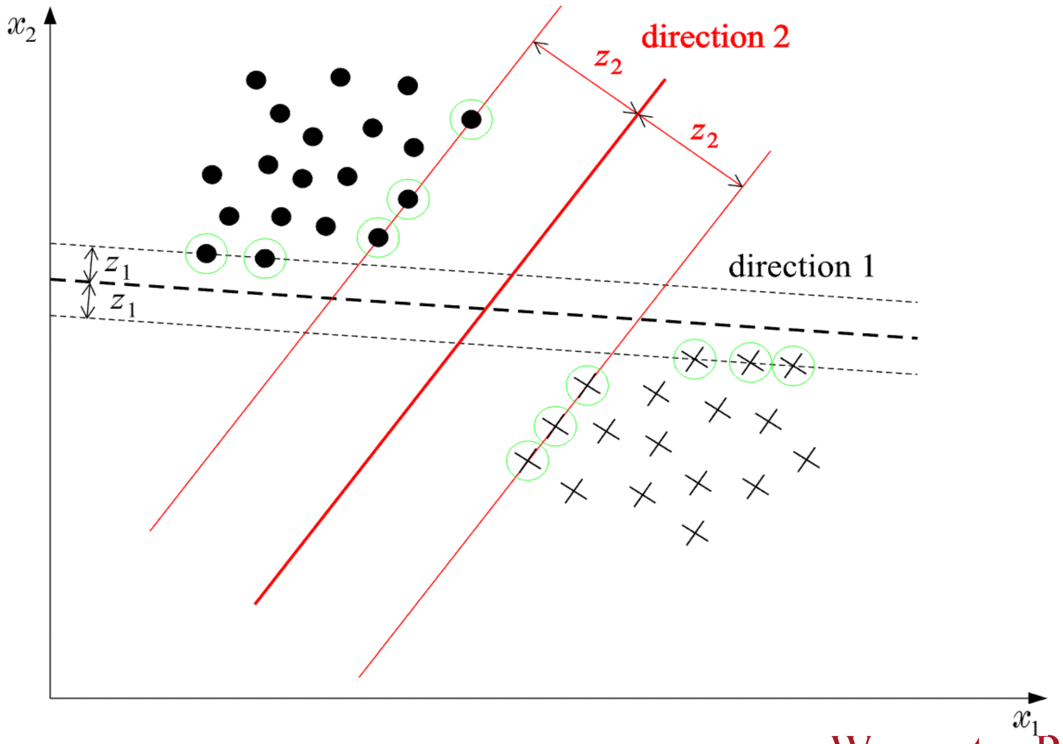
- GD is slightly different

Moving on: Support Vector Machines

Support Vector Machines

– The goal: Given two linearly separable classes, design the classifier

$$g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0 = 0$$

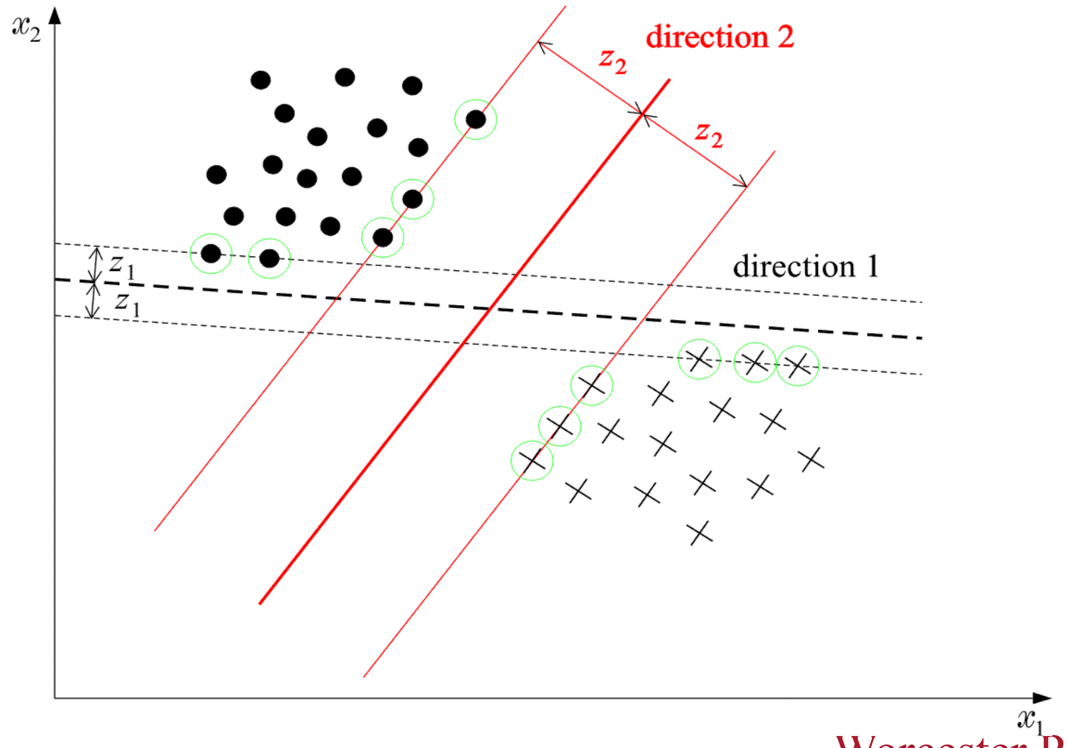


Support Vector Machines

- The goal: Given two linearly separable classes, design the classifier

$$g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0 = 0$$

that leaves the maximum margin from both classes.



Margin intuition

– Margin: Each hyperplane is characterized by:

- Its direction in space, i.e., $\boldsymbol{w}$
- Its position in space, i.e., w_0
- For EACH direction, $\boldsymbol{w}$, choose the hyperplane that leaves the SAME distance from the nearest points from each class. The margin is twice this distance.

Problem formulation

- The distance of a point $\hat{\boldsymbol{x}}$ from a hyperplane is given by:

$$z_{\hat{x}} = \frac{g(\hat{\boldsymbol{x}})}{\|\boldsymbol{w}\|}$$

- Scale $\boldsymbol{w}, \boldsymbol{w}_0$, so that at the nearest points, from each class, the discriminant function is ±1:

$$|g(\boldsymbol{x})| = 1 \quad \{g(\boldsymbol{x}) = +1 \text{ for } \omega_1 \text{ and } g(\boldsymbol{x}) = -1 \text{ for } \omega_2\}$$

- Thus the margin is given by:

$$\frac{1}{\|\boldsymbol{w}\|} + \frac{1}{\|\boldsymbol{w}\|} = \frac{2}{\|\boldsymbol{w}\|}$$

- Also, the following is valid

$$\boldsymbol{w}^T\boldsymbol{x} + w_0 \geq 1 \quad \forall \boldsymbol{x} \in \boldsymbol{w}_1$$

$$\boldsymbol{w}^T\boldsymbol{x} + w_0 \leq -1 \quad \forall \boldsymbol{x} \in \boldsymbol{w}_2$$

SVM linear classifier

- Given

$$g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0$$

- Minimize

$$J(\boldsymbol{w}) = \frac{1}{2}||\boldsymbol{w}||^2$$

- Subject to

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i + w_0) \geq 1, \ \ i = 1,2,\ldots,N$$

$$y_i = 1, \text{ for } \boldsymbol{x}_i \in \omega_i,$$

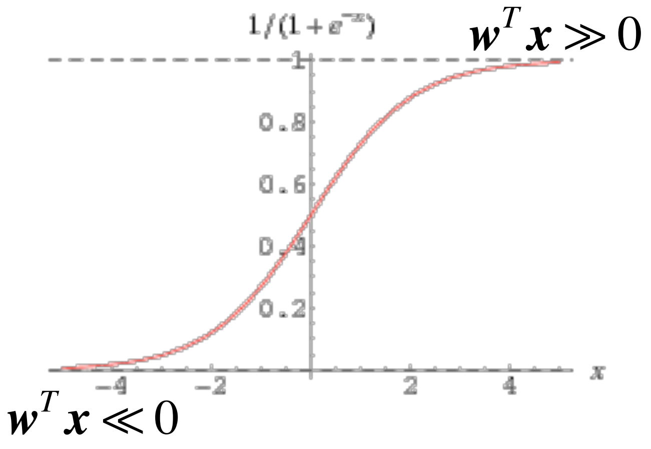
$$y_i = -1, \text{ for } \boldsymbol{x}_i \in \omega_2$$

- The above is justified, since by minimizing $\frac{2}{||\boldsymbol{w}||}$

  the margin $||\boldsymbol{w}||$ is maximized.

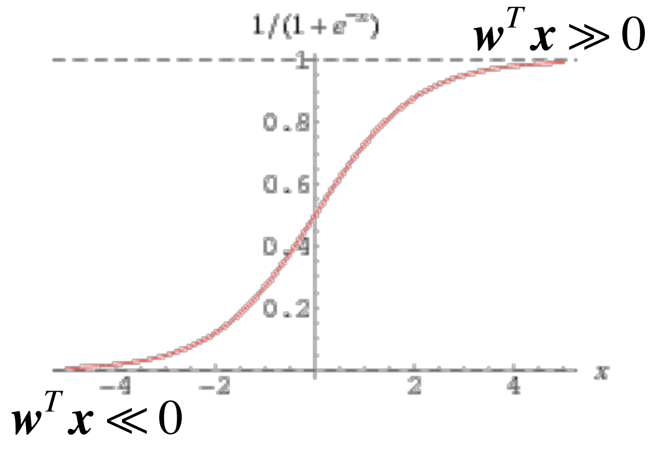
A different look at the same problem

1. Let's define the Optimization Objective
   - Recall logistic regression: $$h_w(x) = \frac{1}{1+e^{-\boldsymbol{w}^T\boldsymbol{x}}}$$
   - If y=1, we want $h_w(x) \sim 1$, that is: $\boldsymbol{w}^T\boldsymbol{x} \gg 0$
   - If y=0, we want $h_w(x) \sim 0$, that is: $\boldsymbol{w}^T\boldsymbol{x} \ll 0$



A different look at the same problem

1. Let's define the Optimization Objective
   - Recall logistic regression: $h_{\mathbf{w}}(x) = \frac{1}{1+e^{-\mathbf{w}^T x}}$
   - If y=1, we want $h_w(x) \sim 1$, that is: $\mathbf{w}^T \mathbf{x} \gg 0$
   - If y=0, we want $h_w(x) \sim 0$, that is: $\mathbf{w}^T \mathbf{x} \ll 0$

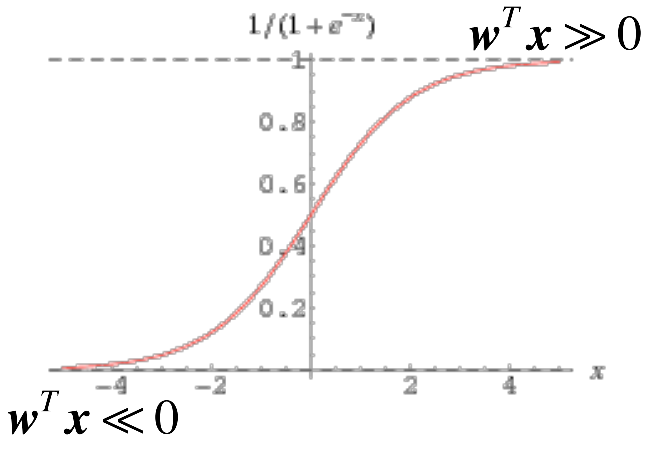


   - Recall cost of example:

$$cost(h_w(x), y) = -y\log(h_w(x)) - (1-y)\log(1-h_w(x))$$

A different look at the same problem

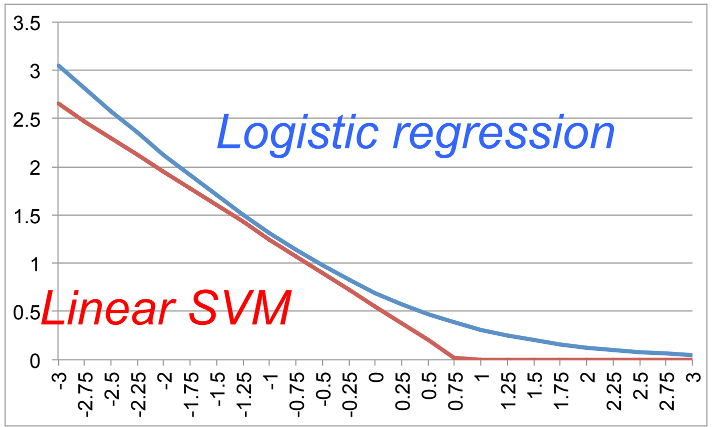
1. Let's define the Optimization Objective
   - Recall logistic regression: $h_w(x) = \dfrac{1}{1+e^{-\boldsymbol{w}^T\boldsymbol{x}}}$
   - If y=1, we want $h_w(x) \sim 1$, that is: $\boldsymbol{w}^T\boldsymbol{x} \gg 0$
   - If y=0, we want $h_w(x) \sim 0$, that is: $\boldsymbol{w}^T\boldsymbol{x} \ll 0$



   - Recall cost of example:

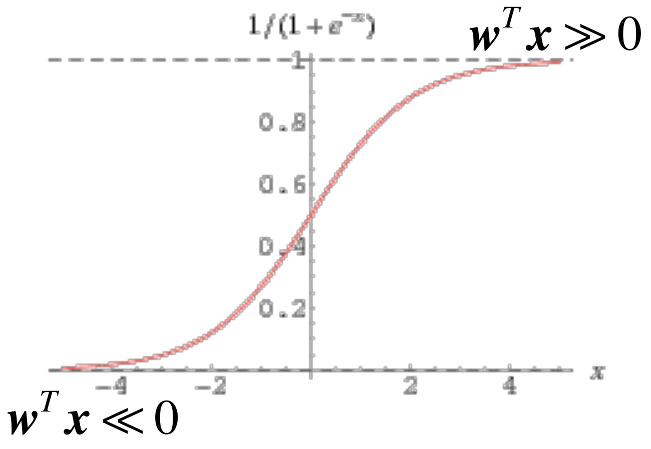
$$cost(h_w(x), y) = -y\log(h_w(x)) - (1-y)\log(1-h_w(x))$$

   - If y =1 $\quad cost = -\log\dfrac{1}{1+e^{-\boldsymbol{w}^T\boldsymbol{x}}}$



A different look at the same problem

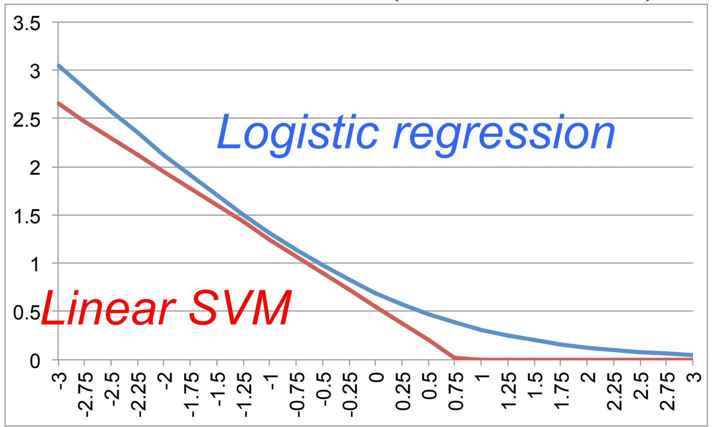
1. Let's define the Optimization Objective
   - Recall logistic regression: $h_w(x) = \frac{1}{1+e^{-w^T x}}$
   - If y=1, we want $h_w(x) \sim 1$, that is: $w^T x \gg 0$
   - If y=0, we want $h_w(x) \sim 0$, that is: $w^T x \ll 0$

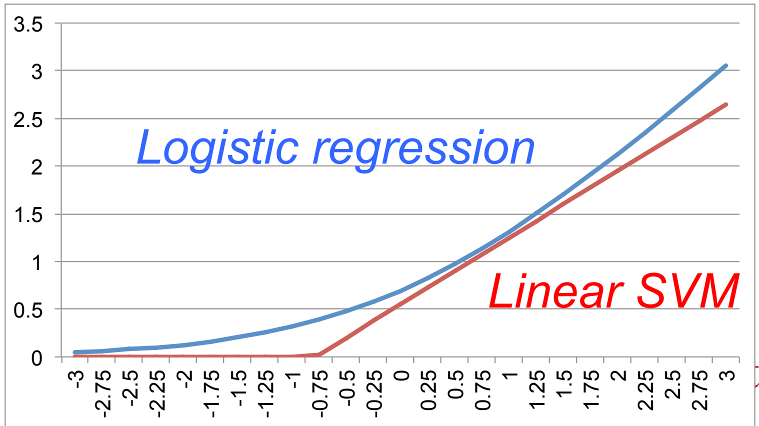


   - Recall cost of example:

$$cost(h_w(x), y) = -y\log(h_w(x)) - (1-y)\log(1-h_w(x))$$

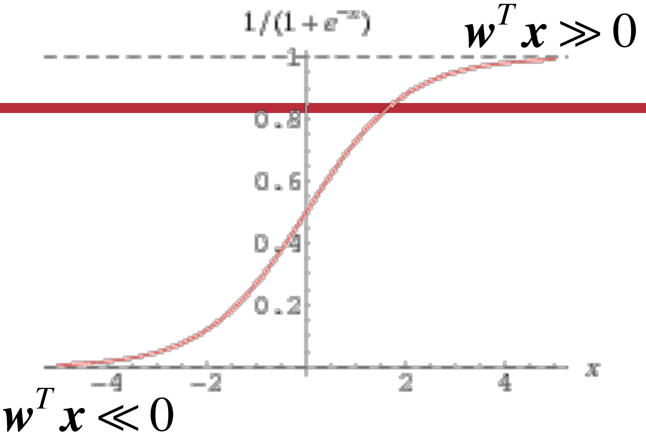
   - If y =1, $cost = -\log\frac{1}{1+e^{-w^T x}}$
   - If y =0, $cost = -\log\left(1-\frac{1}{1+e^{-w^T x}}\right)$





A different look at the same problem

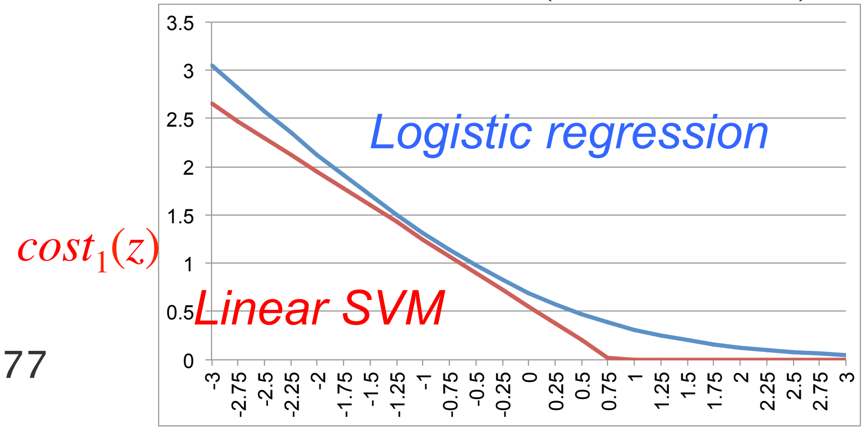
1. Let's define the Optimization Objective
   - Recall logistic regression: $h_w(x) = \frac{1}{1+e^{-w^T x}}$
   - If y=1, we want $h_w(x) \sim 1$, that is: $w^T x \gg 0$
   - If y=0, we want $h_w(x) \sim 0$, that is: $w^T x \ll 0$

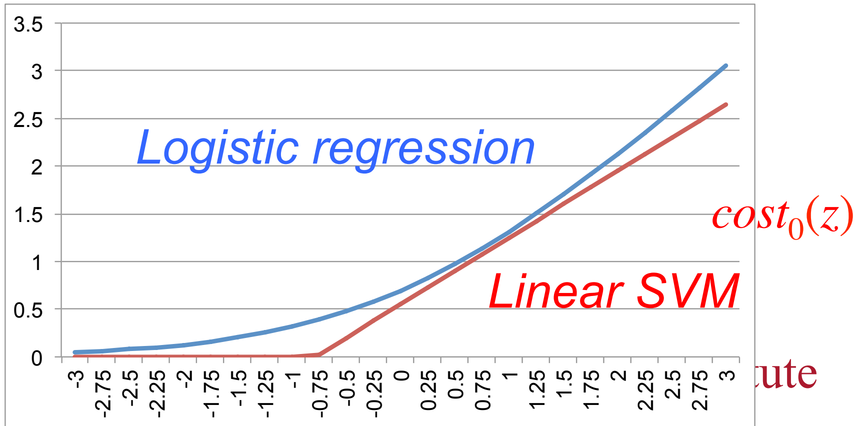


   - Recall cost of example:

$$cost(h_w(x), y) = -y \log(h_w(x)) - (1-y)\log(1-h_w(x))$$

   - If y =1, $cost = -\log \frac{1}{1+e^{-w^T x}}$
   - If y =0, $cost = -\log\left(1 - \frac{1}{1+e^{-w^T x}}\right)$





ute

SVM cost function

Logistic regression:

$$\min_{w} \boxed{\frac{1}{m}} \left[\sum_{i=1}^{m} -y^{(i)} \boxed{\log(h_w(x^{(i)}))} + (1-y^{(i)}) \boxed{\log(1-h_w(x^{(i)}))} \right] + \frac{\boxed{\lambda}}{2m} \sum_{i=1}^{n} w_i^{2}$$

SVM cost function

Logistic regression:

$$\min_{\boldsymbol{w}} \boxed{\frac{1}{m}} \left[\sum_{i=1}^{m} -y^{(i)} \boxed{\log(h_w(x^{(i)}))} + (1 - y^{(i)}) \boxed{\log(1 - h_w(x^{(i)}))} \right] + \boxed{\frac{\lambda}{2m}} \sum_{i=1}^{n} w_i{}^2$$

SVM:

$$\min_{\boldsymbol{w}} C \left[\sum_{i=1}^{m} y^{(i)} cost_1(\boldsymbol{w}^T \boldsymbol{x}^{(i)}) + (1 - y^{(i)}) cost_0(\boldsymbol{w}^T \boldsymbol{x}^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} w_i{}^2$$

SVM cost function

Logistic regression:

$$\min_{w} \boxed{\frac{1}{m}}\left[\sum_{i=1}^{m} -y^{(i)}\boxed{\log(h_w(x^{(i)}))} + (1-y^{(i)})\boxed{\log(1-h_w(x^{(i)}))}\right] + \frac{\boxed{\lambda}}{2m}\sum_{i=1}^{n} w_i^{\,2}$$

SVM:

$$\min_{w} C\left[\sum_{i=1}^{m} y^{(i)} cost_1(\boldsymbol{w}^T \boldsymbol{x}^{(i)}) + (1-y^{(i)}) cost_0(\boldsymbol{w}^T \boldsymbol{x}^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n} w_i^{\,2}$$

<u>Note</u>: Moving from A+λB form to cA+B form

Remarks

- SVMs are often referred to as *Large margin classifiers*
- Based on the cost function:
  - If y=1, we want: $\boldsymbol{w}^T\boldsymbol{x} \geq 1$ (not just $\boldsymbol{w}^T\boldsymbol{x} \geq 0$)
  - If y=0, we want: $\boldsymbol{w}^T\boldsymbol{x} \leq -1$ (not just $\boldsymbol{w}^T\boldsymbol{x} \leq 0$)

Boundary condition

Remarks

- SVMs are often referred to as *Large margin classifiers*
- Based on the cost function*:*
  - If y=1, we want: $\boldsymbol{w}^T\boldsymbol{x} \geq 1$ (not just $\boldsymbol{w}^T\boldsymbol{x} \geq 0$)
  - If y=0, we want: $\boldsymbol{w}^T\boldsymbol{x} \leq -1$ (not just $\boldsymbol{w}^T\boldsymbol{x} \leq 0$)

Boundary condition

- For our cost function*:*

$$\min_{\boldsymbol{w}} \underbrace{C\left[\sum_{i=1}^{m} y^{(i)} cost_1(\boldsymbol{w}^T\boldsymbol{x}^{(i)}) + (1-y^{(i)}) cost_0(\boldsymbol{w}^T\boldsymbol{x}^{(i)})\right]}_{= CA} + \underbrace{\frac{1}{2}\sum_{i=1}^{n} w_i^{\,2}}_{= \mathrm{B}}$$

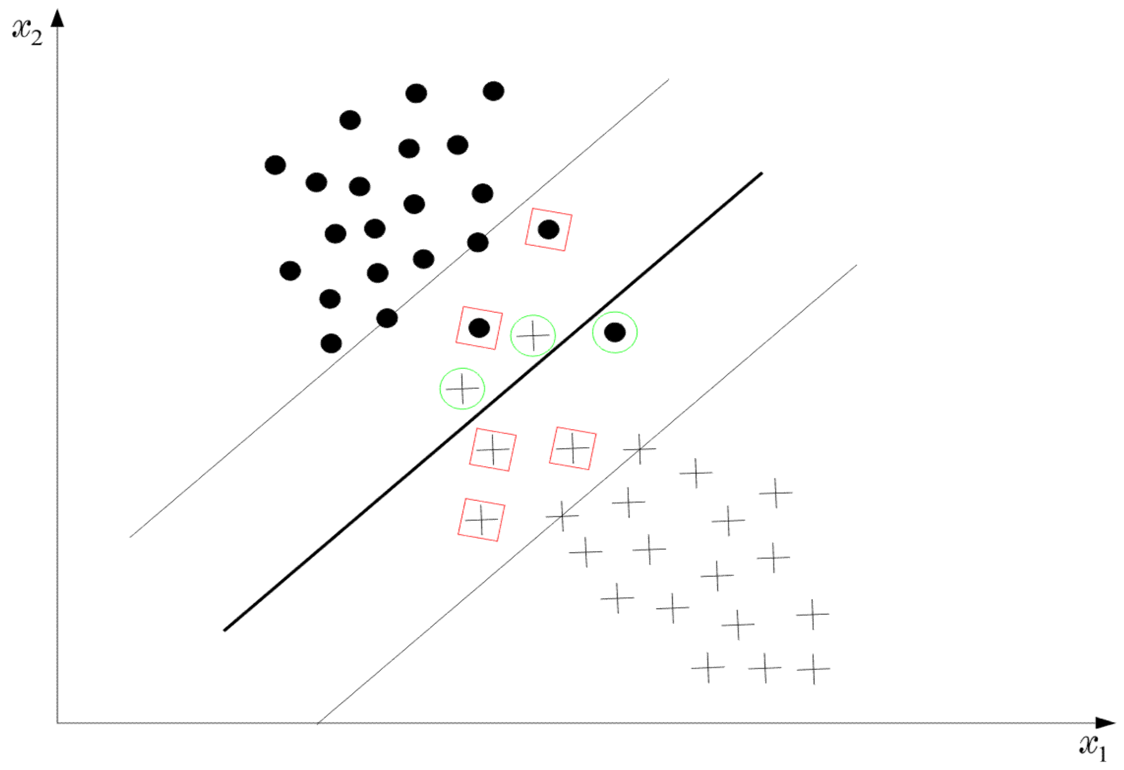
  - If C>>0, we want A=0, which implies: $\min_{\boldsymbol{w}} \frac{1}{2}\sum_{i=1}^{n} w_i^{\,2}$
  - Combined with boundary condition, this optimization will give us the largest minimum distance (margin!) between pos. and neg. examples

Problem with our approach

- If you simply optimize the large margin value (given a large C), the algorithm is sensitive to the outliers!

- How to fix it? Have C not very large, so it can ignore several outliers

Non–Separable classes



Non–Separable classes

In this case, there is no hyperplane such that:

$$\boldsymbol{w}^T \boldsymbol{x} + w_0 (><) 1, \quad \forall \boldsymbol{x}$$

- Recall that the margin is defined as twice the distance between the following two hyperplanes:

$$\boldsymbol{w}^T \boldsymbol{x} + w_0 = 1$$

$$\text{and}$$

$$\boldsymbol{w}^T \boldsymbol{x} + w_0 = -1$$

The training vectors belong to one of three possible categories

1) Vectors outside the band which are correctly classified:

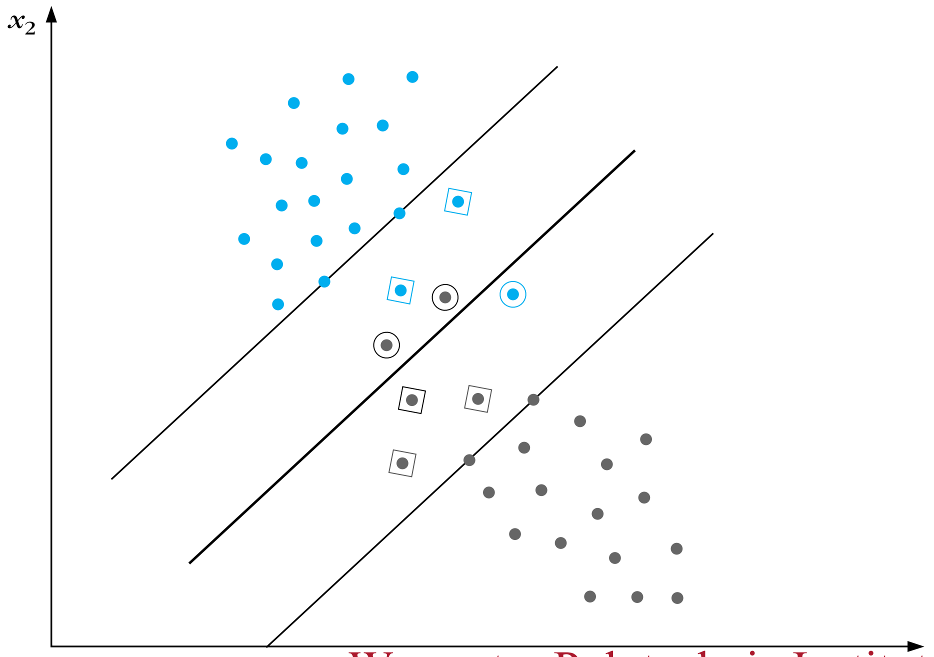
$$y_i(\boldsymbol{w}^T\boldsymbol{x}+w_0)>1$$

2) Vectors inside the band, and correctly classified:

$$0\le y_i(\boldsymbol{w}^T\boldsymbol{x}+w_0)<1$$

3) Vectors misclassified:

$$y_i(\boldsymbol{w}^T\boldsymbol{x}+w_0)<0$$



Slack variables

All three cases above can be represented as:

$$y_i(\boldsymbol{w}^T\boldsymbol{x}+w_0)\geq 1-\xi_i$$

1) $\rightarrow \xi_i = 0$
2) $\rightarrow 0 < \xi_i \leq 1$
3) $\rightarrow 1 < \xi_i$

ξ_i are known as slack variables.

Reformulating optimization goals

The goal of the optimization is now two-fold:

- Maximize margin
- Minimize the number of patterns with $\xi_i > 0$.

One way to achieve this goal is via the cost

$$J(\boldsymbol{w}, w_0, \boldsymbol{\xi}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N} I(\xi_i)$$

where *C* is a constant, and

$$I(\xi_i) = \begin{Bmatrix} 1 & \xi_i > 0 \\ 0 & \xi_i = 0 \end{Bmatrix}$$

Note: *I*(.) is not differentiable. In practice, we use an approximation. A popular choice is:

$$J(\boldsymbol{w}, w_0, \boldsymbol{\xi}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N} \xi_i$$

- Following a similar procedure as before, we obtain KKT conditions

Multi–class generalization

- Although theoretical generalizations exist, the most popular in practice is to look at the problem as M two-class problems (one-against-all)

- Similar to the logistic regression case, for a new element x, we will select the most optimistic classifier

- Problem 1: Asymmetric case of positive/negative classes

- Problem 2: Uneven number of positive/negative examples, especially when the number of classes is large

Steps to try: Revisitting

1. Get more training data: Fixes high variance
2. Try smaller set of features: Fixes high variance
3. Try getting additional features: Fixes high bias
4. Try adding polynomial features: Fixes high bias
5. Try decreasing λ: Fixes high bias
6. Try increasing λ: Fixes high variance

Logistic regression vs. SVM: What to select and when?

Reminder: n – number of features, m - number of training examples

1. n is large (relative to m): *e.g.*, $n = 10{,}000 - 100{,}000$, $m = 10 - 1000$
   Use: Logistic regression or SVM without kernels

2.

3.

Logistic regression vs. SVM: What to select and when?

Reminder: n – number of features, m - number of training examples

1. n is large (relative to m): *e.g.*, $n = 10{,}000 - 100{,}000$, $m = 10 - 1000$
   Use: Logistic regression or SVM without kernels

2. n is small, m is intermediate: *e.g.*, $n = 1 - 1{,}000$, $m = 10 - 10000$
   Use: Use SVM with Gaussian kernel

3.

Logistic regression vs. SVM: What to select and when?

Reminder: n – number of features, m - number of training examples

1. n is large (relative to m): *e.g.*, $n = 10{,}000 — 100{,}000$, $m = 10 — 1000$
   Use: Logistic regression or SVM without kernels

2. n is small, m is intermediate: *e.g.*, $n = 1 — 1{,}000$, $m = 10 — 10000$
   Use: Use SVM with Gaussian kernel

3. n is small, m is large: *e.g.*, $n = 1 — 1{,}000$, $m = 50{,}000 — 1{,}000{,}000$
   Use: Create/add more features, then use logistic regression or SVM without kernels
   Reason: SVM with Gaussian kernel would struggle to run

Combining Classifiers

- The basic philosophy behind the combination of different classifiers lies in the fact that even the “best” classifier fails in some patterns that other classifiers may classify correctly
- Combining classifiers aims at exploiting this complementary information residing in the various classifiers

–

Combining Classifiers

- The basic philosophy behind the combination of different classifiers lies in the fact that even the "best" classifier fails in some patterns that other classifiers may classify correctly
- Combining classifiers aims at exploiting this complementary information residing in the various classifiers

Idea: one designs different optimal classifiers and then combines the results with a specific rule

– Assume that each of the, say, L designed classifiers provides at its output the posterior probabilities:

$$P(\omega_i \mid \boldsymbol{x}),\ i = 1, 2, ..., M$$

Basic Rules

- Product Rule: Assign $\boldsymbol{x}$ to the class ω_i:

$$i = \arg\max_k \prod_{j=1}^{L} P_j\left(\omega_k \mid \boldsymbol{x}\right)$$

where $P_j\left(\omega_k \mid \boldsymbol{x}\right)$ is the respective posterior probability of the *j*th classifier.

Basic Rules

- Product Rule: Assign $\boldsymbol{x}$ to the class ω_i:

$$i = \arg\max_k \prod_{j=1}^{L} P_j\left(\omega_k \mid \boldsymbol{x}\right)$$

where $P_j\left(\omega_k \mid \boldsymbol{x}\right)$ is the respective posterior probability of the *j^th^* classifier

- Sum Rule: Assign $\boldsymbol{x}$ to the class ω_i:

$$i = \arg\max_k \sum_{j=1}^{L} P_j\left(\omega_k \mid \boldsymbol{x}\right)$$

Majority Voting Rule

Majority Voting Rule: Assign $\boldsymbol{x}$ to the class for which there is a consensus or when at least ℓ_c of the classifiers agree on the class label of $\boldsymbol{x}$ where:

$$\ell_c = \begin{cases} \frac{L}{2}+1, & L \text{ is even} \\ \frac{L+1}{2}, & L \text{ is odd} \end{cases}$$

otherwise the decision is rejection, that is no decision is taken.

Majority Voting Rule

Majority Voting Rule: Assign $\boldsymbol{x}$ to the class for which there is a consensus or when at least ℓ_c of the classifiers agree on the class label of $\boldsymbol{x}$ where:

$$\ell_c = \begin{cases} \frac{L}{2}+1, & L \text{ is even} \\ \frac{L+1}{2}, & L \text{ is odd} \end{cases}$$

otherwise the decision is rejection, that is no decision is taken.

Thus, correct decision is made if the majority of the classifiers agree on the correct label, and wrong decision if the majority agrees in the wrong label

Dependent or not Dependent classifiers?

- Although there are not general theoretical results, experimental evidence has shown that the more independent in their decision the classifiers are, the higher the expectation should be for obtaining improved results after combination

- However, there is no guarantee that combining classifiers results in better performance compared to the "best" one among the classifiers

Towards Independence: A number of Scenarios

Train the individual classifiers using different training data points

To this end, we can choose among a number of possibilities:

- Bootstrapping: This is a popular technique to combine unstable classifiers such as decision trees (Bagging belongs to this category of combination)
- Stacking: Train the combiner with data points that have been excluded from the set used to train the individual classifiers
- Use different subspaces to train individual classifiers: According to the method, each individual classifier operates in a different feature subspace. That is, use different features for each classifier

Remarks

- The majority voting and the summation schemes rank among the most popular combination schemes
- Training individual classifiers in different subspaces seems to lead to substantially better improvements compared to classifiers operating in the same subspace
- Besides the above three rules, other alternatives are also possible, such as to use the median value of the outputs of individual classifiers

The Boosting Approach

The origins: Is it possible a weak learning algorithm (one that performs slightly better than a random guessing) to be boosted into a strong algorithm? (Villiant 1984)

— The procedure to achieve it:

- Adopt a weak classifier known as the base classifier.
- Employing the *base* classifier, design a series of classifiers, in a hierarchical fashion, each time employing a different weighting of the training samples. Emphasis in the weighting is given on the hardest samples, *i.e.*, the ones that keep "failing".
- Combine the hierarchically designed classifiers by a weighted average procedure

Acknowledgements

- Lecture materials are based on:
  - The textbook
  - Lecture materials by Stuart Russell (the co-author of the textbook) in Berkeley