

Machine Learning



Dan Goldwasser

dgoldwas@purdue.edu

SVM Objective Function

General Form of a learning algorithm:

- **Minimize** empirical loss, and **Regularize** (to avoid over fitting)

$$\text{Min } \frac{1}{2} \|w\|^2 + c \sum \max(0, 1 - y_i w x_i)$$

Regularization term

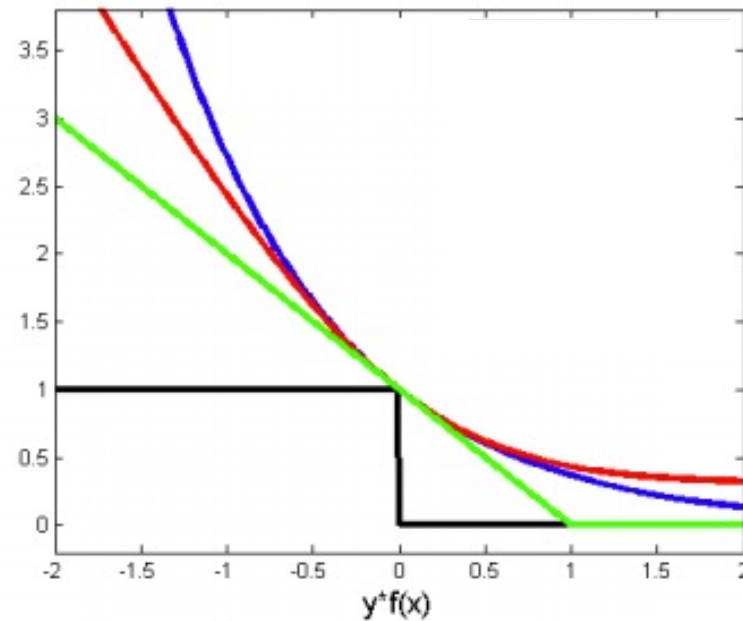
Empirical loss

Can be replaced by other
regularization functions

Can be replaced by
other **loss functions**

Surrogate Loss functions

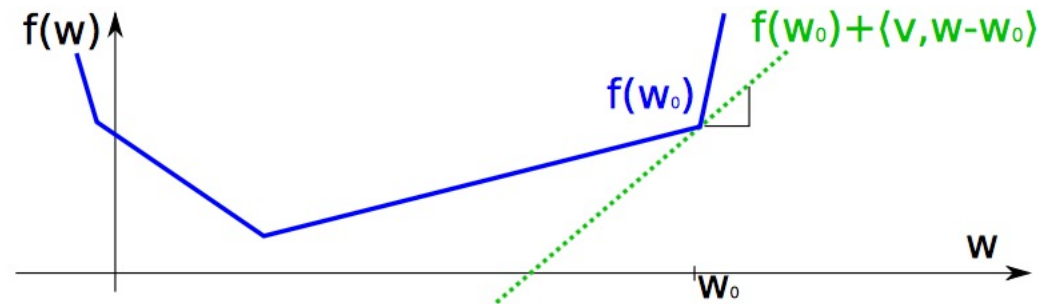
- Surrogate loss function: smooth approximation to the 0-1 loss
 - Upper bound to 0-1 loss



Subgradient descent

Let $f : \mathbb{R}^D \rightarrow \mathbb{R}$ be a convex, not necessarily differentiable, function.
A vector $v \in \mathbb{R}^D$ is called a **subgradient** of f at w_0 , if

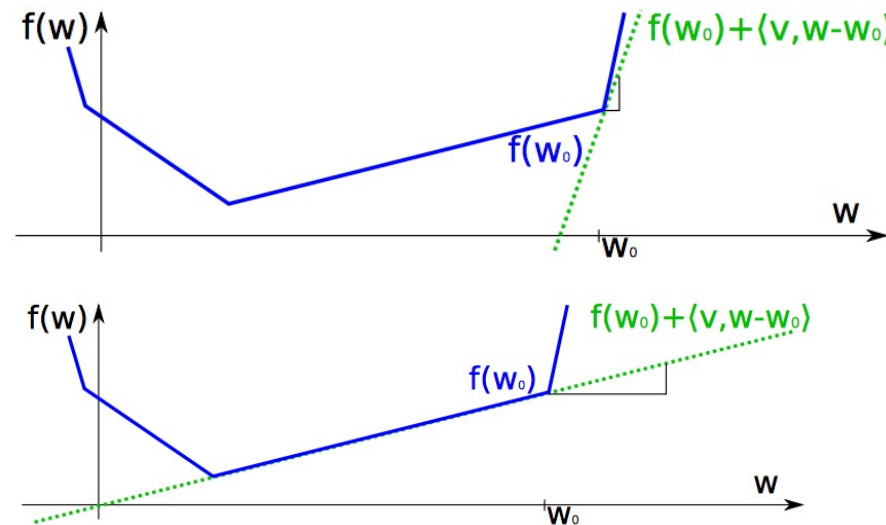
$$f(w) \geq f(w_0) + \langle v, w - w_0 \rangle \quad \text{for all } w.$$



Subgradient descent

Let $f : \mathbb{R}^D \rightarrow \mathbb{R}$ be a convex, not necessarily differentiable, function.
A vector $v \in \mathbb{R}^D$ is called a **subgradient** of f at w_0 , if

$$f(w) \geq f(w_0) + \langle v, w - w_0 \rangle \quad \text{for all } w.$$

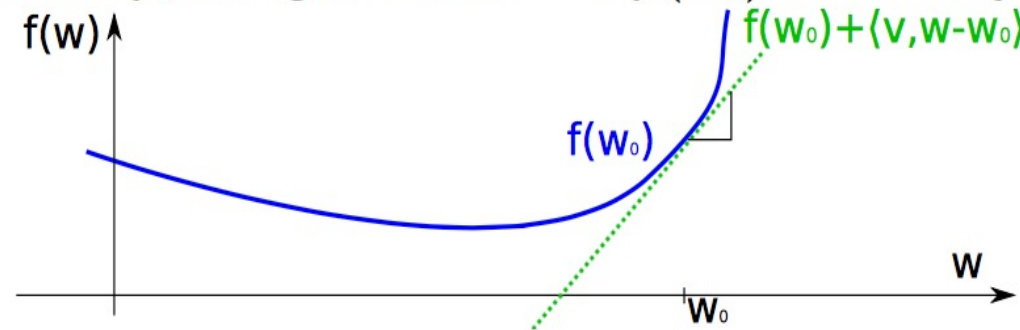


Subgradient descent

Let $f : \mathbb{R}^D \rightarrow \mathbb{R}$ be a convex, not necessarily differentiable, function.
A vector $v \in \mathbb{R}^D$ is called a **subgradient** of f at w_0 , if

$$f(w) \geq f(w_0) + \langle v, w - w_0 \rangle \quad \text{for all } w.$$

For differentiable f , the gradient $v = \nabla f(w_0)$ is the only subgradient.



Sub-Gradient

Standard 0/1 loss

Penalizes all incorrectly classified examples with the same amount

Hinge loss

Penalizes incorrectly classified examples and correctly classified examples that lie within the margin

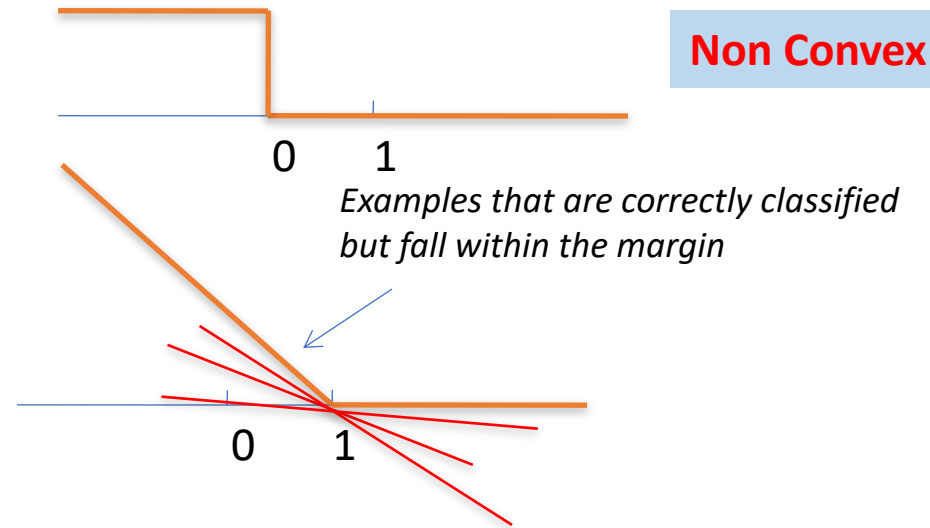
Convex,
but not differentiable at $x=1$

Solution: subgradient

The **sub-gradient** of a function c at x_0 is any vector v such that: $\forall x : c(x) - c(x_0) \geq v \cdot (x - x_0)$.

At **differentiable** points this set only contains the gradient at x_0

Intuition: the set of all tangent lines (lines under c , touching c at x_0)



$$\begin{aligned} & \partial_w \max\{0, 1 - y_n(\mathbf{w} \cdot \mathbf{x}_n + b)\} \\ &= \partial_w \begin{cases} 0 & \text{if } y_n(\mathbf{w} \cdot \mathbf{x}_n + b) > 1 \\ y_n(\mathbf{w} \cdot \mathbf{x}_n + b) & \text{otherwise} \end{cases} \\ &= \begin{cases} \partial_w 0 & \text{if } y_n(\mathbf{w} \cdot \mathbf{x}_n + b) > 1 \\ \partial_w y_n(\mathbf{w} \cdot \mathbf{x}_n + b) & \text{otherwise} \end{cases} \\ &= \begin{cases} 0 & \text{if } y_n(\mathbf{w} \cdot \mathbf{x}_n + b) > 1 \\ y_n \mathbf{x}_n & \text{otherwise} \end{cases} \end{aligned}$$

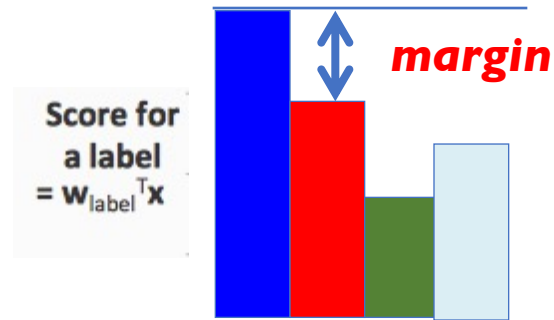
Multiclass SVM

- **Single classifier optimizing a global objective**
 - *Extend the SVM framework to the multiclass settings*
- **Binary SVM:**
 - *Minimize $||W||$ such that the closest points to the hyperplane have a score of ± 1*
- **Multiclass SVM**
 - *Each label has a **different** weight vector*
 - *Maximize **multiclass margin***

Margin in the Multiclass case

Revise the definition for the multiclass case:

- The difference between the score of the correct label and the scores of competing labels



Colors indicate different labels

SVM Objective: Minimize total norm of weights s.t. the *true label is scored at least 1 more than the second best.*

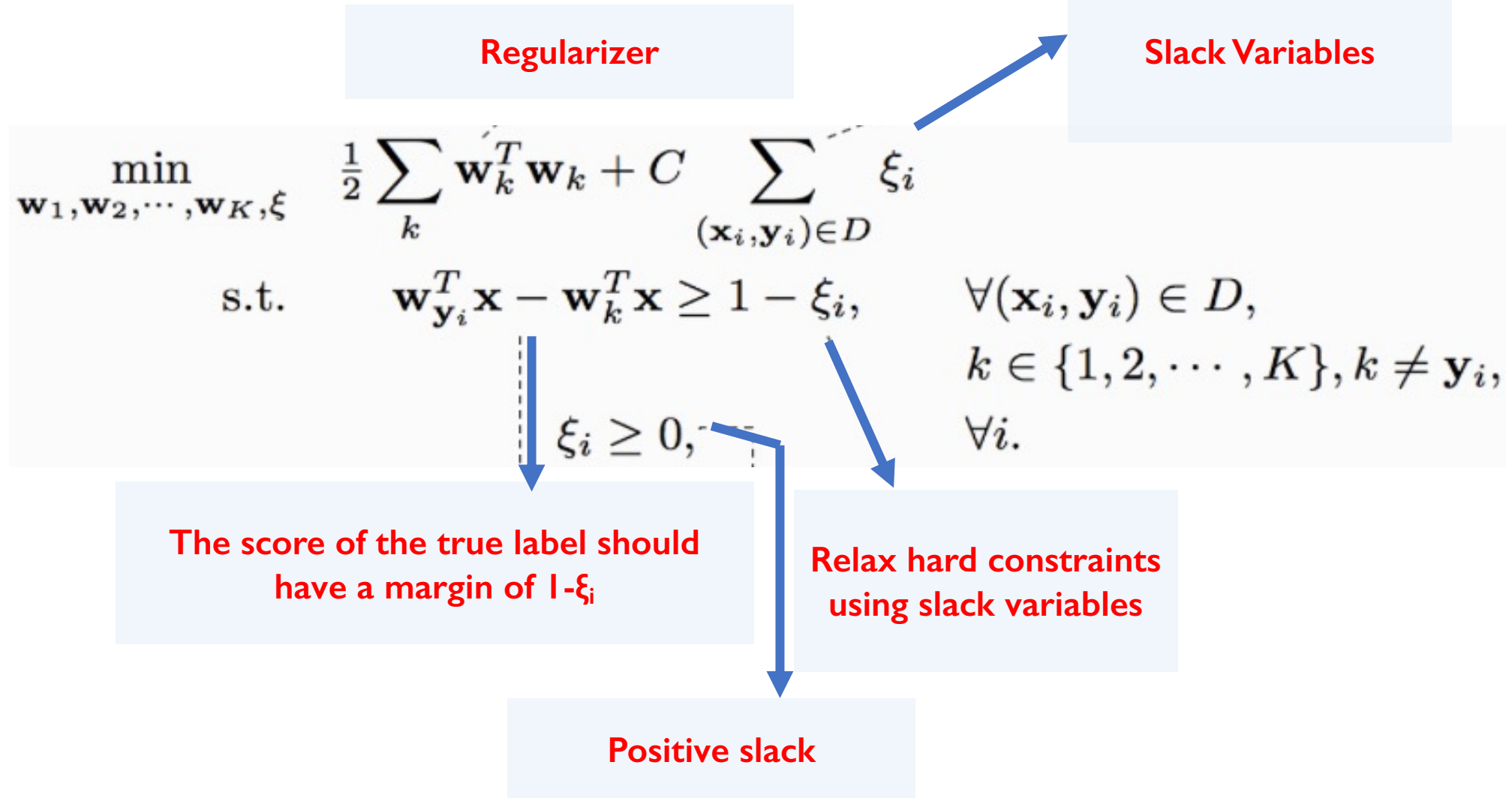
Hard Multiclass SVM

Regularization

$$\begin{aligned} \min_{\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K} \quad & \frac{1}{2} \sum_k \mathbf{w}_k^T \mathbf{w}_k \\ \text{s.t.} \quad & \mathbf{w}_{\mathbf{y}_i}^T \mathbf{x} - \mathbf{w}_k^T \mathbf{x} \geq 1 \quad \forall (\mathbf{x}_i, \mathbf{y}_i) \in D, \\ & k \in \{1, 2, \dots, K\}, k \neq \mathbf{y}_i, \end{aligned}$$

The score of the true label has
to be higher than 1, for any label

Soft Multiclass SVM



Alternative Notation

- For examples with label i we want: $w_i^T \mathbf{x} > w_j^T \mathbf{x}$
- **Alternative notation:** *Stack all weight vectors*

$$\mathbf{w} = \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \vdots \\ \mathbf{w}_K \end{bmatrix}_{nK \times 1} \quad \phi(\mathbf{x}, i) = \begin{bmatrix} \mathbf{0}_n \\ \vdots \\ \mathbf{x} \\ \vdots \\ \mathbf{0}_n \end{bmatrix}_{nK \times 1}$$

\mathbf{x} in the i^{th} block, zeros everywhere else

$$\mathbf{w}^T \phi(\mathbf{x}, i) > \mathbf{w}^T \phi(\mathbf{x}, j) \text{ is equivalent to } w_i^T \mathbf{x} > w_j^T \mathbf{x}$$

Multiclass classification so far

- **Lea**

Solve:
$$\min_{w, \xi} \frac{1}{2} \|w\|^2 + \frac{C}{N} \sum_{n=1}^N \xi^n$$

subject to, for $i = 1, \dots, n$,

$$\langle w, \phi(x^n, y^n) \rangle - \langle w, \phi(x^n, y) \rangle \geq 1 - \xi^n \quad \text{for all } y \in \mathcal{Y} \setminus \{y^n\}.$$

- **Prediction**

$$f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \langle w, \phi(x, y) \rangle$$

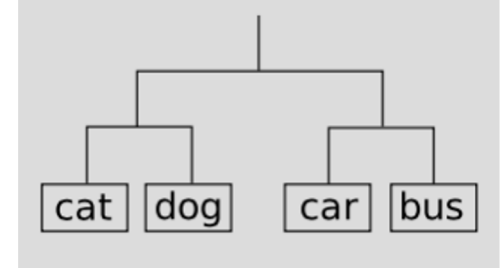
Cost Sensitive Multiclass Classification

- Sometime we are willing to “tolerate” some mistakes more than others



Cost Sensitive Multiclass Classification

- We can think about it as a hierarchy:
- Define a distance metric:
 - $\Delta(y, y') =$ tree distance between y and y'



We would like to incorporate that into our learning model

Solve:
$$\min_{w, \xi} \frac{1}{2} \|w\|^2 + \frac{C}{N} \sum_{n=1}^N \xi^n$$

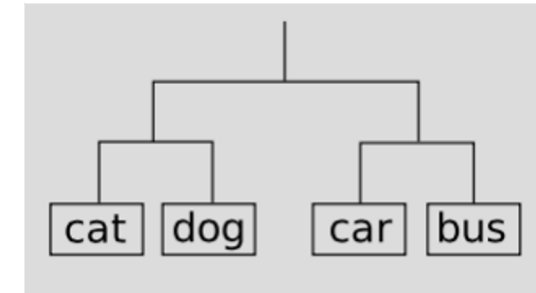
subject to, for $i = 1, \dots, n$,

$$\langle w, \phi(x^n, y^n) \rangle - \langle w, \phi(x^n, y) \rangle \geq 1 - \xi^n \quad \text{for all } y \in \mathcal{Y} \setminus \{y^n\}.$$

what should we change?

Cost Sensitive Multiclass Classification

- We can think about it as a hierarchy:
- Define a distance metric:
 - $\Delta(y, y') =$ tree distance between y and y'



We would like to incorporate that into our learning model

Solve:
$$\min_{w, \xi} \frac{1}{2} \|w\|^2 + \frac{C}{N} \sum_{n=1}^N \xi^n$$

subject to, for $i = 1, \dots, n$,

$$\langle w, \phi(x^n, y^n) \rangle - \langle w, \phi(x^n, y) \rangle \geq 1 - \xi^n \quad \text{for all } y \in \mathcal{Y} \setminus \{y^n\}.$$

$$\Delta(y^n, y)$$

Cost Sensitive Multiclass Classification

Solve:
$$\min_{w, \xi} \frac{1}{2} \|w\|^2 + \frac{C}{N} \sum_{n=1}^N \xi^n$$

subject to, for $i = 1, \dots, n$,

$$\langle w, \phi(x^n, y^n) \rangle - \langle w, \phi(x^n, y) \rangle \geq \Delta(y^n, y) - \xi^n \quad \text{for all } y \in \mathcal{Y} \setminus \{y^n\}.$$

Instead, we can have an unconstrained version -

Solve:
$$\min_{w, \xi} \frac{1}{2} \|w\|^2 + \frac{C}{N} \sum_{n=1}^N l(\mathbf{w}; (\mathbf{x}^n, y^n))$$

Question: What is sub-gradient of this loss function?

$$l(\mathbf{w}; (\mathbf{x}^n, y^n)) = \max_{y' \in Y} \Delta(y, y') - \langle \mathbf{w}, \phi(\mathbf{x}^n, y^n) \rangle + \langle \mathbf{w}, \phi(\mathbf{x}^n, y') \rangle$$

Subgradient for the MC case

Computing a subgradient:

$$\min_w \frac{1}{2} \|w\|^2 + \frac{C}{N} \sum_{n=1}^N \ell^n(w)$$

with $\ell^n(w) = \max_y \ell_y^n(w)$, and

$$\ell_y^n(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y) \rangle - \langle w, \phi(x^n, y^n) \rangle$$

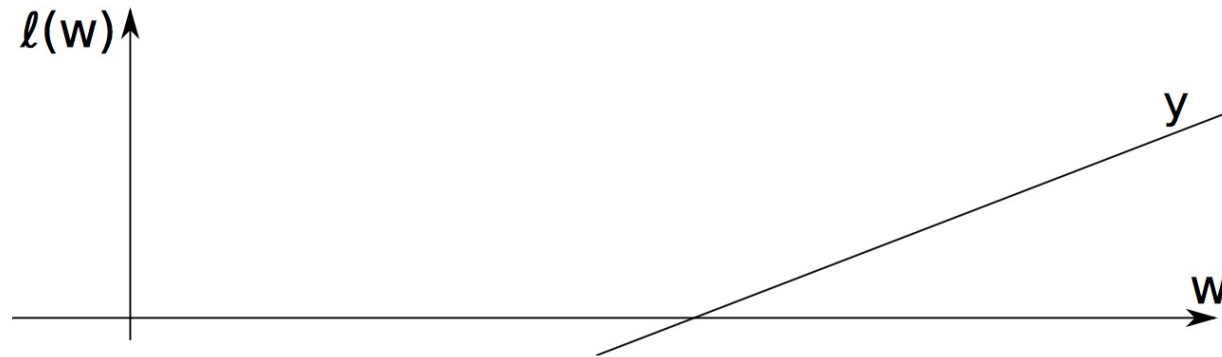
Subgradient for the MC case

Computing a subgradient:

$$\min_w \frac{1}{2} \|w\|^2 + \frac{C}{N} \sum_{n=1}^N \ell^n(w)$$

with $\ell^n(w) = \max_y \ell_y^n(w)$, and

$$\ell_y^n(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y) \rangle - \langle w, \phi(x^n, y^n) \rangle$$



For each $y \in \mathcal{Y}$, $\ell_y(w)$ is a linear function.

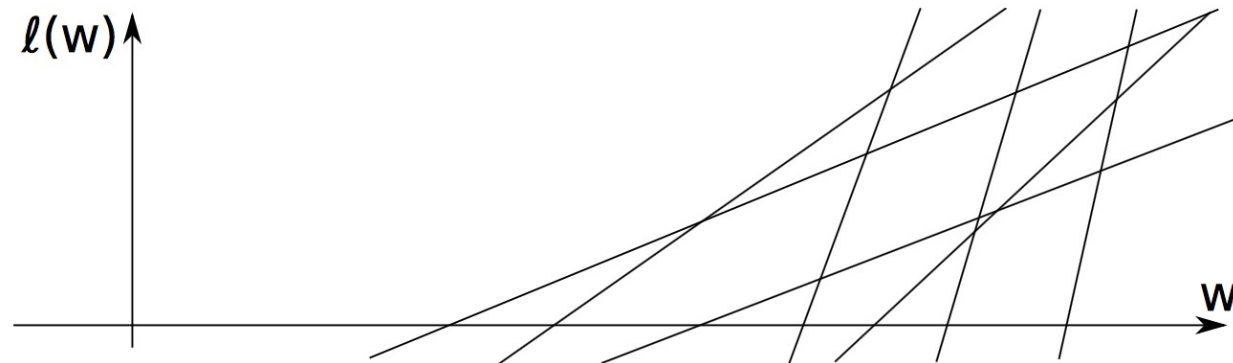
Subgradient for the MC case

Computing a subgradient:

$$\min_w \frac{1}{2} \|w\|^2 + \frac{C}{N} \sum_{n=1}^N \ell^n(w)$$

with $\ell^n(w) = \max_y \ell_y^n(w)$, and

$$\ell_y^n(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y) \rangle - \langle w, \phi(x^n, y^n) \rangle$$



For each $y \in \mathcal{Y}$, $\ell_y(w)$ is a linear function.

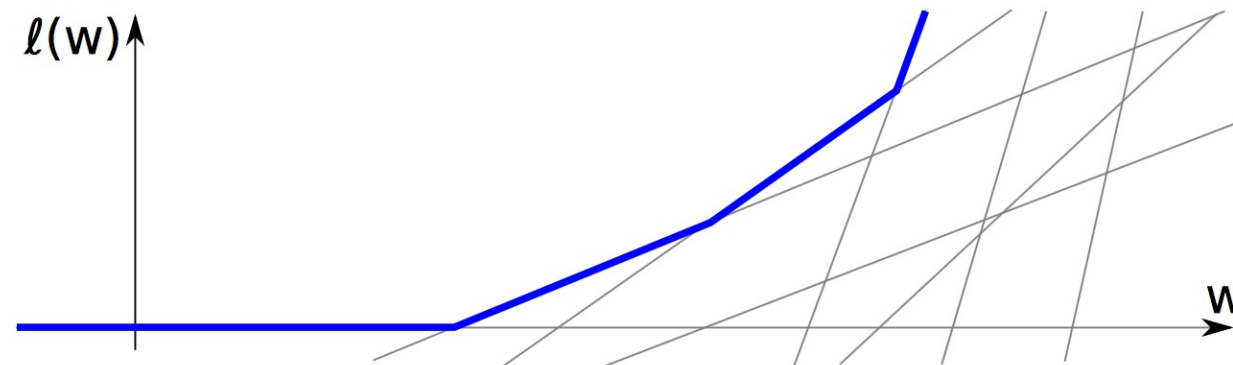
Subgradient for the MC case

Computing a subgradient:

$$\min_w \frac{1}{2} \|w\|^2 + \frac{C}{N} \sum_{n=1}^N \ell^n(w)$$

with $\ell^n(w) = \max_y \ell_y^n(w)$, and

$$\ell_y^n(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y) \rangle - \langle w, \phi(x^n, y^n) \rangle$$



$\ell(w) = \max_y \ell_y(w)$: maximum over all $y \in \mathcal{Y}$.

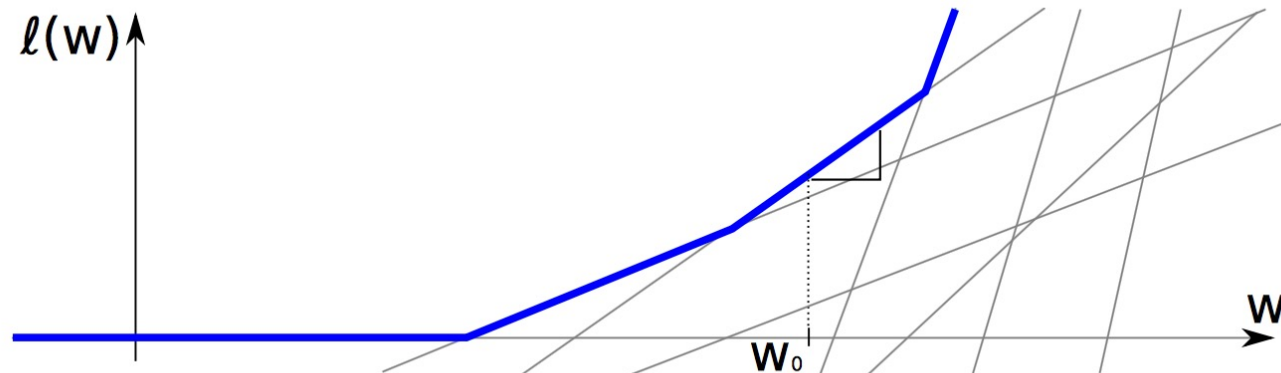
Subgradient for the MC case

Computing a subgradient:

$$\min_w \frac{1}{2} \|w\|^2 + \frac{C}{N} \sum_{n=1}^N \ell^n(w)$$

with $\ell^n(w) = \max_y \ell_y^n(w)$, and

$$\ell_y^n(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y) \rangle - \langle w, \phi(x^n, y^n) \rangle$$



Subgradient of ℓ^n at w_0 : find maximal (active) y , use $v = \nabla \ell_y^n(w_0)$.

Subgradient descent for the MC case

Subgradient Descent S-SVM Training

input training pairs $\{(x^1, y^1), \dots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$,
input feature map $\phi(x, y)$, loss function $\Delta(y, y')$, regularizer C ,
input number of iterations T , stepsizes η_t for $t = 1, \dots, T$

- 1: $w \leftarrow \vec{0}$
- 2: **for** $t=1, \dots, T$ **do**
- 3: **for** $i=1, \dots, n$ **do**
- 4: $\hat{y} \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}} \Delta(y^n, y) + \langle w, \phi(x^n, y) \rangle - \langle w, \phi(x^n, y^n) \rangle$
- 5: $v^n \leftarrow \phi(x^n, \hat{y}) - \phi(x^n, y^n)$
- 6: **end for**
- 7: $w \leftarrow w - \eta_t(w - \frac{C}{N} \sum_n v^n)$
- 8: **end for**

output prediction function $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \langle w, \phi(x, y) \rangle$.

Observation: each update of w needs 1 argmax-prediction per example.

Stochastic SubGD for the MC case

Stochastic Subgradient Descent S-SVM Training

input training pairs $\{(x^1, y^1), \dots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$,
input feature map $\phi(x, y)$, loss function $\Delta(y, y')$, regularizer C ,
input number of iterations T , stepsizes η_t for $t = 1, \dots, T$

- 1: $w \leftarrow \vec{0}$
- 2: **for** $t=1, \dots, T$ **do**
- 3: $(x^n, y^n) \leftarrow$ randomly chosen training example pair
- 4: $\hat{y} \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}} \Delta(y^n, y) + \langle w, \phi(x^n, y) \rangle - \langle w, \phi(x^n, y^n) \rangle$
- 5: $w \leftarrow w - \eta_t(w - \frac{C}{N}[\phi(x^n, \hat{y}) - \phi(x^n, y^n)])$
- 6: **end for**

output prediction function $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \langle w, \phi(x, y) \rangle$.

Question: *What is the difference between this algorithm and the perceptron variant for multiclass classification?*

From multiclass to structures

- So far the number of classes was small.
- **That's not always the case...**



Label = DOG



Label = 2 DOGS



Label = 3 DOGS

Can we still think about this scenario as multiclass classification?

Solve:
$$\min_{w, \xi} \frac{1}{2} \|w\|^2 + \frac{C}{N} \sum_{n=1}^N \xi^n$$

subject to, for $i = 1, \dots, n$,

$$\langle w, \phi(x^n, y^n) \rangle - \langle w, \phi(x^n, y) \rangle \geq 1 - \xi^n \quad \text{for all } y \in \mathcal{Y} \setminus \{y^n\}.$$

Summary

- Introduced an optimization framework for learning:
 - Minimization problem
 - Objective: data loss term and regularization cost term
 - Separate learning objective from learning algorithm
 - Many algorithms for minimizing a function
- Can be used for regression and classification
 - Different loss function
 - GD and SGD algorithms
- Classification: use surrogate loss function
 - Smooth approximation to 0-1 loss

And now...

Computational Learning Theory

Computational Learning Theory

- Learning Algorithms:
 - Perceptron, winnow
 - Gradient Descent
- Models
 - Online learning, mistake driven learning
- Learning theory so far: **mistake bound**
 - How **many mistakes** before “good” performance
- *Let's try a more natural measure – **number of examples***

Computational Learning Theory

- *What general laws constrain inductive learning ?*
 - *What learning problems can be solved? (what is learnable?)*
 - *When can we trust the output of a learning algorithm ?*
- **We seek theory to relate**
 - Probability of successful Learning
 - Number of training examples
 - Relation to the *complexity of hypothesis space*
 - *Accuracy* to which target concept is approximated

Recall - *Quantifying Performance*

- We want to be able to say something rigorous about the performance of our learning algorithm
- We will concentrate on discussing the number of examples one needs to see before we can say that our learned hypothesis is good.

Learning Conjunctions

- There is a hidden conjunction the learner is to learn

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

- *How many examples are needed to learn it ? How ?*
 - **Protocol I:** The learner proposes instances as queries to the teacher (“active learner”)
 - **Protocol II:** The teacher (who knows f) provides training examples (“learning with teacher”)
 - **Protocol III:** Some random source (e.g., Nature) provides training examples; the Teacher (Nature) provides the labels ($f(x)$)

Learning Conjunctions

- **Protocol I**: learner proposes instances as queries to the teacher
- Since we **know** we are after a **monotone conjunction**:

- Is \mathbf{x}_{100} in? $\langle (1,1,1,\dots,1,0), ? \rangle$ $f(\mathbf{x})=0$ (conclusion: Yes)
- Is \mathbf{x}_{99} in? $\langle (1,1,\dots,1,0,1), ? \rangle$ $f(\mathbf{x})=1$ (conclusion: No)
- Is \mathbf{x}_1 in? $\langle (0,1,\dots,1,1,1), ? \rangle$ $f(\mathbf{x})=1$ (conclusion: No)

- A straight forward algorithm requires $n=100$ queries, and will produce as a result the hidden conjunction (**exactly**).

$$h = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

Learning Conjunctions

- **Protocol II:** The teacher (who knows f) provides training examples

$\langle (0,1,1,1,1,0,\dots,0,1), 1 \rangle$ (we learned a superset of the good variables)

To show you that all these variables are required...

$\langle (0,0,1,1,1,0,\dots,0,1), 0 \rangle$ **need** x_2

$\langle (0,1,0,1,1,0,\dots,0,1), 0 \rangle$ **need** x_3

.....

$\langle (0,1,1,1,1,0,\dots,0,0), 0 \rangle$ **need** x_{100}

- A straight forward algorithm requires $k = 6$ examples to produce the hidden conjunction (**exactly**).

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

Learning Conjunctions

- **Protocol III:** Some random source (e.g., Nature) provides training examples
- Teacher (Nature) provides the labels ($f(x)$)

```
<(1,1,1,1,1,1,...,1,1), 1>  
<(1,1,1,0,0,0,...,0,0), 0>  
<(1,1,1,1,1,0,...0,1,1), 1>  
<(1,0,1,1,1,0,...0,1,1), 0>  
<(1,1,1,1,1,0,...0,0,1), 1>  
<(1,0,1,0,0,0,...0,1,1), 0>  
<(1,1,1,1,1,1,...,0,1), 1>  
<(0,1,0,1,0,0,...0,1,1), 0>
```

Learning Conjunctions

- **Protocol III:** *Some random source provides examples*

- Teacher (Nature) provides the labels ($f(x)$)

- **Algorithm: *Elimination***

- *Start with the set of all literals as candidates*

- *Eliminate a literal that is not active (0) in a positive example*

- $\langle (1,1,1,1,1,1,\dots,1,1), 1 \rangle$

$$f = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge \dots \wedge x_{100}$$

- $\langle (1,1,1,0,0,0,\dots,0,0), 0 \rangle$ learned nothing

- $\langle (1,1,1,1,1,0,\dots,0,1,1), 1 \rangle$

$$f = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{99} \wedge x_{100}$$

- $\langle (1,0,1,1,0,0,\dots,0,0,1), 0 \rangle$ learned nothing

- $\langle (1,1,1,1,1,0,\dots,0,0,1), 1 \rangle$

$$f = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

- $\langle (1,0,1,0,0,0,\dots,0,1,1), 0 \rangle$ **Final hypothesis:**

- $\langle (1,1,1,1,1,1,\dots,0,1), 1 \rangle$

- $\langle (0,1,0,1,0,0,\dots,0,1,1), 0 \rangle$

$$h = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$



$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

Is that *good* ? **Performance** ? # *examples* ?

Learning Conjunctions

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

$$h = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

Is that *good* ?
Performance ? # of
examples ?

We can determine the **# of mistakes** we'll make before reaching the exact target, but not **# of examples** needed to guarantee good performance.

To understand this better, let's consider the types of mistakes our learning algorithm will make. Can you characterize the type of mistakes?

Can we make mistakes on negative examples?

Can we make mistakes on positive examples?

When will we make mistakes on a positive example?

We will only make positive mistakes in cases x_1 is not active

What is the probability of a mistake? (aka *error rate of the learned function*)



Two Directions

- **Mistake Driven Learning algorithms**

- Update your hypothesis only when you make mistakes
- **Good**: in terms of how many mistakes you make before you stop, happy with your hypothesis.
- Note: not all on-line algorithms are mistake driven, so performance measure could be different.

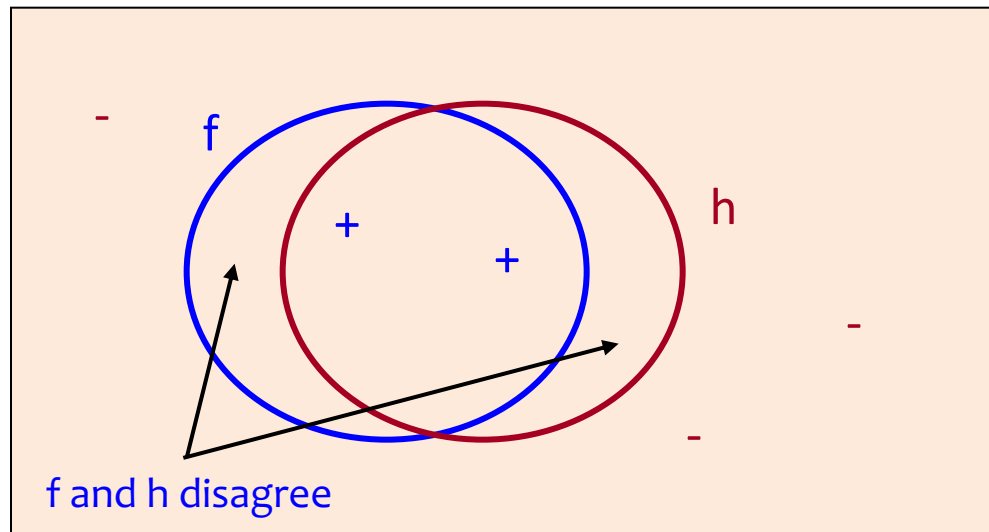
- **Probabilistic Perspective**

- Never saw x_1 in positive examples, maybe we'll never see it?
- And if we will, it will be with small probability, so the concepts we learn may be pretty **good**
- **Good**: in terms of performance on future data
- **PAC framework**

Prototypical Concept Learning

- **Instance Space**: \mathbf{X} (examples)
- **Concept Space**: \mathbf{C}
 - Set of possible target functions: $f \in \mathbf{C}$ is the hidden target function
 - All n -conjunctions; all n -dimensional linear functions.
- **Hypothesis Space**: \mathbf{H} set of possible hypotheses
- **Training instances**: \mathbf{S} positive & negative examples of the target concept $f \in \mathbf{C}$, drawn from a fixed unknown distribution D
- **Learn**: A hypothesis $h \in \mathbf{H}$ such that $h(x) = f(x)$
 - A hypothesis $h \in \mathbf{H}$ such that $h(x) = f(x)$ for all $x \in \mathbf{S}$?
 - A hypothesis $h \in \mathbf{H}$ such that $h(x) = f(x)$ for all $x \in \mathbf{X}$?
- **Probabilistic settings:**
 - $h \in \mathbf{H}$ estimates f , evaluated by performance on subsequent $x \in \mathbf{X}$ drawn from D

PAC Learning – Intuition



The rectangle contains all examples (instance space).

Inside the blue circle: examples labeled as positive by f (**target**) outside: negative examples of f

The red circle represent the *positive* predictions of the **learned** function

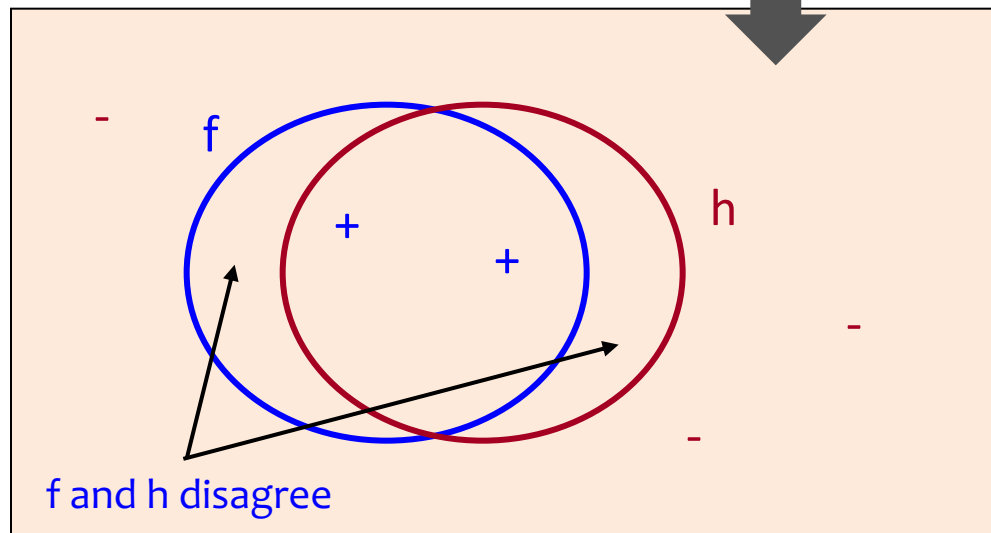
Error: area of disagreement!

$$\text{Error}_D = \Pr_{x \in D} [f(x) \neq h(x)]$$

$$h = \underline{x_1} \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

PAC Learning – Intuition

Given the fact that we are learning Boolean conjunctions using the elimination algorithm, is this diagram accurate?



$$\text{Error}_D = \Pr_{x \in D} [f(x) \neq h(x)]$$

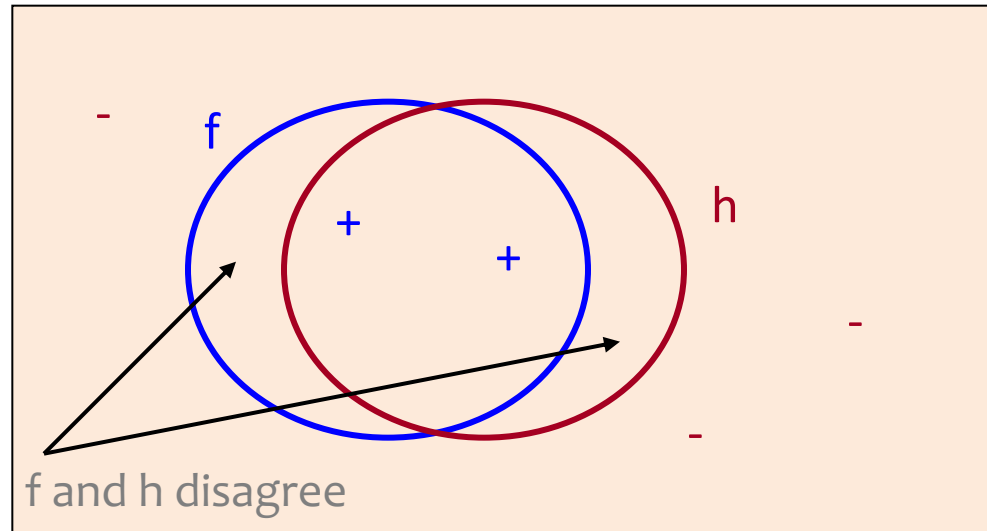
$$h = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

PAC Learning – Intuition

Can we bound the **Error**

$$\text{Error}_D = \Pr_{x \in D} [f(x) \neq h(x)]$$

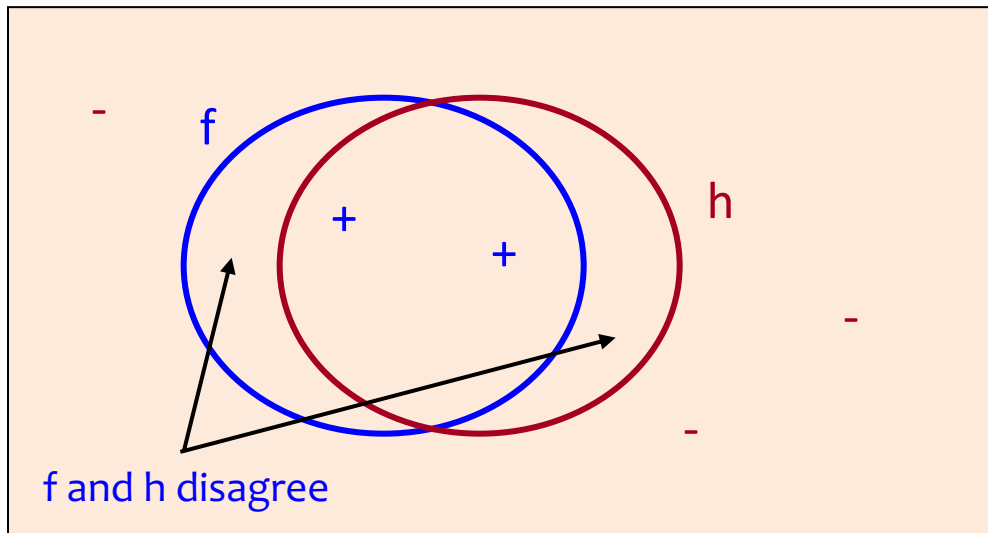
given what we know about the training instances ?



$$h = \underline{x_1} \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

PAC Learning – Intuition

- We have seen many examples (drawn according to D)
 - Since in all the positive examples x_1 was active, it is **very likely** that it will be active in future positive examples
 - **If not**, in any case, x_1 is active only in a small percentage of the examples so the chance of “keeping” x_1 is small.

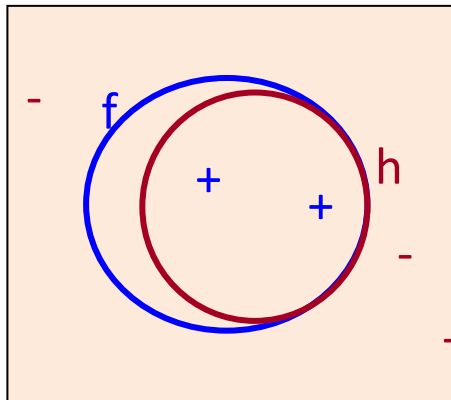


$$\text{Error}_D = \Pr_{x \in D} [f(x) \neq h(x)]$$

$$h = \underline{x_1} \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

Learning Conjunctions– Analysis (1)

- Let z be a *literal*.
- Let $p(z)$ be the probability that z is false in positive example sampled from D .
 - $p(z)$ is also the probability that z is deleted from h in a randomly chosen positive example (during training)
 - If z is in the target concept, then $p(z) = 0$.




$$h = \underline{x_1} \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

Learning Conjunctions– Analysis (1)

- **Claim:** h will make mistakes only on positive examples.
 - A mistake is made only if a literal z , that is in h but not in f , is false in a positive example. In this case, h will say **NEG**, but the example is *positive*.
 - Thus, $p(z)$ is also the probability that z causes h to **make a mistake** on a randomly drawn example from D .
 - There may be overlapping reasons for mistakes, **but the sum clearly bounds it.**

*Can we bound this error?
How many examples do
we need to see?*


$$\text{Error}(h) \leq \sum_{z \in h} P(z)$$

Learning Conjunctions– Analysis (2)

- Call a literal z **bad** if $p(z) > \epsilon/n$.
 - A **bad literal** is a literal that has a significant probability to appear with a positive example but, nevertheless, it has not appeared with one in the training data.
- **Claim**: If there are **no** bad literals, then $\text{error}(h) < \epsilon$.

Flows directly from the definition of a bad variable and the bound the error of learned classifier

$$\text{Error}(h) \leq \sum_{z \in h} P(z)$$

Learning Conjunctions– Analysis (2)

- Call a literal z **bad** if $p(z) > \epsilon/n$.

What if there are bad literals ?

- Let z be a **bad literal**.
- *The probability that it will not be eliminated by a given example:*

$$\begin{aligned} \Pr(z \text{ survives one example}) &= \\ 1 - \Pr(z \text{ is eliminated by one example}) &= 1 - p(z) < 1 - \epsilon/n \end{aligned}$$

The probability that z will not be eliminated by m examples is:

$$\Pr(z \text{ survives } m \text{ independent examples}) = (1 - p(z))^m < (1 - \epsilon/n)^m$$

- There are at most n **bad literals**, so the probability that **some** bad literal survives m examples is bounded by $n(1 - \epsilon/n)^m$

How can we decrease the probability of a bad literal?

Learning Conjunctions– Analysis (3)

- **We want this probability to be small.**
 - We want to choose m large enough such that the probability that **some** z survives m examples is less than δ .
 - *I.e., that z remains in h , and makes it different from the target function*

$$\Pr(z \text{ survives } m \text{ example}) = n(1 - \epsilon/n)^m < \delta$$

- Using $1 - x < e^{-x}$ ($x > 0$) it is sufficient to require that $n e^{-m\epsilon/n} < \delta$
- Therefore, we need m examples to guarantee a probability of failure (error $> \epsilon$) of less than δ , where m is:

$$m > \frac{n}{\epsilon} \{ \ln(n) + \ln(1/\delta) \}$$

Learning Conjunctions– Analysis (3)

- Therefore, we need m examples to guarantee a probability of failure (error $> \epsilon$) of less than δ , where m is $m > \frac{n}{\epsilon} \{\ln(n) + \ln(1/\delta)\}$

Theorem: If m is as above, then:

- With **probability $> 1-\delta$** , there are **no bad literals**;
equivalently, **with probability $> 1-\delta$** , $\text{Err}(h) < \epsilon$

- **Examples:**

- With $\delta=0.1$, $\epsilon=0.1$, and $n=100$, we need 6907 examples.
- With $\delta=0.1$, $\epsilon=0.1$, and $n=10$, we need only 460 example, only 690 for $\delta=0.01$

The change in δ resulted in a smaller increase in examples. **Why?**

Requirements of Learning

- **Cannot** expect a learner to learn a concept **exactly**, since
 - *Typically multiple concepts will be consistent with available data* (a small fraction of the available instance space).
 - *Unseen examples could potentially have any label*
 - *We “agree” to misclassify uncommon examples that do not show up in the training set. (agree = good approximation of f)*
- **Cannot** always expect to learn a **close approximation** to the target concept since
 - Sometimes (*only in rare learning situations, we hope*) *the training set will not be representative* (will contain uncommon examples).
- The only realistic expectation of a good learner is: *with high probability learn a close approximation to the target concept*

Probably Approximately Correct

- Realistically - successful learning has **high probability** to learn a **close approximation** of the target concept.

In **Probably Approximately Correct (PAC)** learning, one requires that given small parameters δ and ϵ , with probability at least $(1 - \delta)$ a learner produces a hypothesis with error at most ϵ

*The reason we can hope for that is the **Consistent Distribution** assumption.*

PAC Learnability

- Consider a *concept class* C , defined over an *instance space* X (containing instances of length n), and a learner L using a *hypothesis space* H .

C is **PAC learnable** by L using H if for all $f \in C$, for all distribution D over X , and fixed $0 < \epsilon, \delta < 1$, Given a collection of m examples sampled independently according to the distribution D , L produces with probability at least $(1 - \delta)$ a hypothesis $h \in H$ with error at most ϵ , where m is polynomial in $1/\epsilon$, $1/\delta$, n and $\text{size}(C)$

C is **efficiently learnable** if L can produce the hypothesis in time polynomial in $1/\epsilon$, $1/\delta$, n and $\text{size}(C)$

PAC Learnability

- Polynomial sample complexity (information theoretic constraint)
 - *Is there enough information in the sample to distinguish a hypothesis h that approximate f ?*
- Polynomial time complexity (computational complexity)
 - *Is there an efficient algorithm that can process the sample and produce a good hypothesis h ?*

To be PAC learnable, there must be a hypothesis $h \in H$ with an *arbitrary* small error **for every $f \in C$** . We generally assume $H \supseteq C$

- *(Properly PAC learnable if $H=C$)*

Worst Case definition: the algorithm must meet its accuracy

- for **every *distribution*** (The distribution free assumption)
- for **every *target function*** f in the class C

Occam's Razor

"plurality should not be posited without necessity"
William of Ockham (ca. 1285-1349)

Claim

The probability that there exists a hypothesis $h \in H$ that

(1) is consistent with m examples and

(2) satisfies $\text{error}(h) > \varepsilon$ ("bad hypothesis")

is less than $|H|(1 - \varepsilon)^m$

$$(\text{Error}_D(h) = \Pr_{x \in D} [f(x) \neq h(x)])$$

Proof

Let h be such a bad hypothesis.

The probability that h is consistent with one example of f is

$$\Pr_{x \in D} [f(x) = h(x)] < 1 - \varepsilon$$

- Since the m examples are drawn independently of each other.
- The prob. that h is consistent with m examples of f is $< (1 - \varepsilon)^m$
- The probability that some hypothesis in H is consistent with m examples is less than $|H| (1 - \varepsilon)^m$

Occam's Razor

- We want the probability of finding a **bad consistent** hypothesis to be bounded by δ

$$|H|(1 - \epsilon)^m < \delta$$

$$\ln |H| + m \ln (1 - \epsilon) < \ln (\delta)$$

Using the inequality:

$$e^{-x} > 1 - x;$$

Note: $\ln (1 - \epsilon) < -\epsilon$; gives a safer δ

$$m > \frac{1}{\epsilon} \{ \ln(|H|) + \ln(1/\delta) \}$$

What do we know now about the Consistent Learner scheme?

We showed that a **m-consistent hypothesis** generalizes well ($\text{err} < \epsilon$) (Appropriate m is a function of $|H|, \epsilon, \delta$)

It is called Occam's razor, because it indicates a **preference** towards small hypothesis spaces

Consistent Learners

- From the definition, we get the following general scheme for learning:

Given a sample D of m examples,

Find some $h \in H$ that is consistent with all m examples

- If m is large enough, a consistent hypothesis must be close enough to f

- Check that m need not be too large:

we showed that the “closeness” guarantee requires that

$$m > 1/\varepsilon (\ln |H| + \ln 1/\delta)$$

- Show that the consistent hypothesis $h \in H$ can be computed efficiently

- In the case of conjunctions

- We used the **Elimination algorithm** to find a hypothesis h that is consistent with the training set (*easy to compute*)
- We showed that if we have sufficiently many examples (polynomial in the parameters), then h is close to the target function.

Examples: Conjunctions

- Conjunction (general): *The size of the hypothesis space is 3^n*

$$m > \frac{1}{\varepsilon} \{ \ln(3^n) + \ln(1/\delta) \} = \frac{1}{\varepsilon} \{ n \ln 3 + \ln(1/\delta) \}$$

(slightly different than previous bound)

- If we want to guarantee a 95% chance of learning a hypothesis of at least 90% accuracy, with $n=10$ Boolean variable,

$$m > (\ln(1/0.05) + 10 \ln(3))/0.1 = 140.$$

- If we go to $n=100$, this goes just to 1130, (linear with n) but changing the confidence to 1% it goes just to 1145 (logarithmic with δ)
- These results hold for any **consistent learner**.

Examples: K-CNF

CNF: Conjunctive Normal Form

- Very expressive class, consists of conjunctions of disjunctions of literals:

$$(l_1 \vee l_2 \vee \dots \vee l) \wedge (l_1 \vee l_2 \vee \dots \vee l_{30})$$

K- CNF: Each disjunction clause consists of k-terms

$$f = \bigwedge_{i=1}^m (l_{i_1} \vee l_{i_2} \vee \dots \vee l_{i_k})$$

K-term CNF: Formula contains k disjunctions

$$f = \bigwedge_{i=1}^k (l_{i_1} \vee l_{i_2} \vee \dots \vee l_{i_m})$$

Examples: K-CNF

$$f = \bigwedge_{i=1}^m (l_{i_1} \vee l_{i_2} \vee \dots \vee l_{i_k})$$

How can we
prove learnability?

- (1) Sample complexity
- (2) Time complexity

Occam Algorithm for $f \in k\text{-CNF}$

- Draw a sample D of size m
- Find a hypothesis h that is consistent with all the examples in D
- Determine sample complexity:

Sample complexity: correlates
with the size of the hypothesis
class. **Count!**

Examples: K-CNF

Occam Algorithm for $f \in k\text{-CNF}$

- Draw a sample D of size m
- Find a hypothesis h that is consistent with all the examples in D
- Determine sample complexity:

*Sample complexity: correlates with the size of the hypothesis class. **Count!***

$$f = C_1 \wedge C_2 \wedge \dots \wedge C_m; \dots; C_i = l_1 \vee l_2 \vee \dots \vee l_k$$

(1) How many disjunctions, over n variables, of size k are there?

Each clause has to “pick” k literals out of $2n$ possibilities: $\binom{2n}{k} \approx 2n^k$

(2) How many possibilities for combinations of disjunctive clauses?

There are n^k possible disjunctions, each can appear or not $2^{(2n)^k}$

This is a HUGE number! Is K-CNF learnable?

Examples: K-CNF

Occam Algorithm for $f \in k\text{-CNF}$

- Draw a sample D of size m
- Find a hypothesis h that is consistent with all the examples in D
- Determine sample complexity:

$$\ln(|k\text{-CNF}|) = O(n^k) \quad \dots\dots\dots 2^{(2n)^k} \quad \dots\dots\dots (2n)^k$$

- Due to the sample complexity result h is guaranteed to be a PAC hypothesis; but we need to learn a consistent hypothesis.

How do we find the consistent hypothesis h ?