Machine Learning

Perceptron and Extensions

Dan Goldwasser

dgoldwas@purdue.edu

Kernels and Dual Perceptron

One of the main limitation of linear models is, well.. ***linearity***!
This means that certain concepts cannot be represented.

One way to deal with this problem is to manually construct new features, or alternatively, systematically project the features to a new, higher dimensional space.

This presents two problems – ***computationally***, it is costly, and ***overfitting*** is more likely to happen (more data is needed).
We will introduce ***Kernels*** *as a way to efficiently learn in a high dimensional space (i.e., the first problem).*

Feature Extraction

$$\text{Linear models: } w \cdot x \geq \theta$$

- Let's build a *food recommendation* classifier
  - Features*: words*
  - ***"John likes cake and hates carrots"***
  - ***"John likes carrots and hates cake"***
- What can you say about the feature vectors of the two sentences?

Feature Representation

- We cannot learn anything when different labels are represented using similar feature vectors
- ***How can we fix it?***
  - Move to a more meaningful representation!
  - *Encode the structure of the input object*
- **How about word conjunctions?**
  - ***(likes_cake, …, hates_carrots)***
  - ***(likes_carrots,.., hates_cake)***
- ***What can we say about the new feature space?***

Simple example: *Bigram features*

- **Let's build a Shakespeare classifier!**

- We will define a classification task over all the works of Shakespeare.
- Corpus size: N=884,647, V = 29,066
- So far, so good..
- But when moving to Bigram features -
- V^2= 844,000,000
- **Why is this a problem?**

Linear Separability

- **Problem**: *The data might not be linearly separable in the original feature space*
- **Solution**: add new features making the data linearly separable in the new space
  - Transform the original space and capture interactions between features
- **Caution**: this may blow up the feature space!
- **Why is this a problem?**
  - ***Efficiency***
  - ***Generalization***

Dual Representation

Examples : x ∈ {0,1}^{n}

Hypothesis : w ∈ R^n

$$f(x) = Th_\theta\left(\sum_{i\in I} w_i x_i(x)\right)$$

Primal

- Let w be an initial weight vector for perceptron.
- Let (x^1,+), (x^2,+), (x^3,-), (x^4,-) be examples
  - assume mistakes are made on x^1, x^2, x^4
- What is the resulting weight vector? **w = w + x^1 + x^2 - x^4**
- ➔ *The weight vector is a linear combination of examples*

$$w = \sum_{1,..,m} r\alpha_i y_i x_i$$

- α_i is the #mistakes of x_i , r is the learning rate (assume r=1)

Dual Representation

Examples : $x \in \{0,1\}^n$

Hypoth

- Let w be a
  (x^3,-), (x^4,-
- What is th
- The weigh
- α_i is the n

Prediction: *compute the dot-product between the weights and a new example*

$$f(x) = w \cdot x =$$

$$(\sum_{1,..,m} r\alpha_i y_i x_i) \cdot x =$$

Dual

$$\sum_{1,..,m} r\alpha_i y_i (x_i \cdot x)$$

Linear sum of dot products between x (new example) and previous ``mistake" examples

Kernel Based Methods

We want to learn a function in a blown up space

- Let I be the set t_1, t_2, t_3 … of conjunctions over the feature space x_1, x_2… x_n.
- Then we can write a linear function over this new feature space.

$$f(x) = Th_\theta\left(\sum_{i\in I} w_i x_i(x)\right) \quad \Rightarrow \quad f(x) = Th_\theta\left(\sum_{i\in I} w_i t_i(x)\right)$$

Example: $\mathbf{x_1 \wedge x_2 \wedge x_4(11010) = 1}$

$\mathbf{x_3 \wedge x_4(11010) = 0}$

For brevity we'll denote these as $x_1x_2x_4$

The Kernel Trick (1)

$$f(x) = Th_\theta \left(\sum_{i \in I} w_i t_i(x) \right)$$

- *Consider the value of w used in the prediction*
  - Each previous mistake, on example z, makes an additive contribution of +/-1 to w, iff t(z) = 1.
- **The value of w is determined by the number of mistakes on which t() was satisfied.**

The Kernel Trick(2)

- **P** – set of examples on which we Promoted
- **D** – set of examples on which we Demoted
- M = **P** U **D**

$$f(x) = Th_\theta\left(\sum_{i\in I} w_i t_i(x)\right) \Rightarrow f(x) = Th_\theta\left(\sum_{i\in I}\left[\sum_{z\in P, t_i(z)=1} 1 - \sum_{z\in D, t_i(z)=1} 1\right] t_i(x)\right) =$$

Where S(z)=1 if $z \in P$ and S(z)=-1 if $z \in D$

- Reordering:

$$Th_\theta\left(\sum_{z\in M} S(z)\sum_{i\in I} t_i(z)t_i(x)\right)$$

The Kernel Trick(3)

- So far we saw:
  - A *mistake* on z contributes the value +/-1 to *all conjunctions* satisfied by z.
- Given an example *x* ,the contribution of z to the sum is the *number of conjunctions satisfying both x and z.*
- Define a dot product in the t-space:

$$K(x,z) = \sum_{i \in I} t_i(z) t_i(x)$$

- We get the standard notation:

$$Th_\theta \left(\sum_{z \in M} S(z) \sum_{i \in I} t_i(z) t_i(x) \right) \Rightarrow f(x) = Th_\theta \left(\sum_{z \in M} S(z) K(x,z) \right)$$

Kernel Based Methods

- What does this representation give us?

$$f(x) = Th_\theta \left(\sum_{z \in M} S(z) K(x,z) \right)$$

$$K(x,z) = \sum_{i \in I} t_i(z) t_i(x)$$

- *We can view this Kernel as the distance between x,z* ***in the t-space***
  - sum of common active conjunctions in x,z
- But, K(x,z) can be measured in the **original space**, *without explicitly writing the t-representation of x, z*

Kernel Based Methods

$$f(x) = Th_\theta\left(\sum_{z \in Z} S(z)K(x,z)\right) \qquad K(x,z) = \sum_{i \in I} t_i(z)t_i(x)$$

- Consider the space of all 3^n conjunctions (allowing both positive and negative literals). Then,

$$K(x,z) = \sum_{i \in I} t_i(z)t_i(x) = 2^{same(x,z)}$$

- **same(x,z)** : *the number of features that have the same value for both x and z (computed over the original feature space)*
- We get:

$$f(x) = Th_\theta\left(\sum_{z \in M} S(z)2^{same(x,z)}\right)$$

Example (**all** conjunctions over x)

$n = 2 \qquad v = (0, 0), \qquad z = (0, 1)$

What is the new feature space?

Write down the set of all conjunctions

How are v,z represented in the new space?

Write the values of v,z in the new representation

What is their dot-product?

What is the value of kernel function?

Calculate the value of your kernel function

Example (**all** conjunctions over x)

$$n = 2 \qquad v = (0, 0), \qquad z = (0, 1)$$

$$\text{I}= (\emptyset, x_1, x_2, \neg x_1, \neg x_2, \neg x_1 x_2, x_1 \neg x_2, x_1 x_2, \neg x_1 \neg x_2)$$

$v \in \{0, 1\}^2$ In the new space: $v^i \in \{0, 1\}^9$

$v^i = (1, 0, 0, 1, 1, 0, 0, 0, 1)$

$z^i = (1, 0, 1, 1, 0, 0, 1, 0, 0)$

$v^i \cdot z^i = 2^{same(v,z)}$

$same(v, z) = 1$

Example (**all** conjunctions over x)

$$n = 2 \qquad v = (0, 0), \qquad z = (0, 1)$$

$$\text{I}= (\emptyset, x_1, x_2, \neg x_1, \neg x_2, \neg x_1 x_2, x_1 \neg x_2, x_1 x_2, \neg x_1 \neg x_2)$$

$v \in \{0, 1\}^2$ In the new space: $v^i \in \{0, 1\}^9$

$v^i = (1, 0, 0, 1, 1, 0, 0, 0, 1)$

$z^i = (1, 0, 1, 1, 0, 0, 1, 0, 0)$

$v^i \cdot z^i = 2^{same(v,z)}$

$same(v, z) = 1$

Kernel Based Methods

Proof:

- let k=same(x,z);
- construct a “surviving” conjunctions by
  - (1) choosing to include one of these k literals with the right polarity in the conjunction, or
  - (2) choosing to not include it at all.
- Conjunctions with literals outside this set disappear.

Example II

- Let's look at a different feature mapping
- Let I be the set $t_1, t_2, t_3 \ldots$ of **monotone** conjunctions over the feature space $x_1, x_2 \ldots x_n$.

- How can we calculate the dot product **efficiently**?

$$K(x, z) = \sum_{i \in I} t_i(z) t_i(x)$$

- *How does the monotone requirement change the Boolean kernel we just saw?*

Example II: **Monotone** conjunctions over x

$n = 2 \qquad v = (1, 0), \qquad z = (1, 1)$

What is the new feature space?
Write down the set of all monotone conjunctions

How are v,z represented in the new space?
Write the values of v,z in the new representation
What is their dot-product?

What is the value of kernel function?
Calculate the value of your kernel function

Example II: **Monotone** conjunctions over x

$$n = 2 \qquad v = (1, 0), \qquad z = (1, 1)$$

$$\mathrm{I} = (\emptyset,\ x_1,\ x_2,\ x_1 x_2)$$

$v \in \{0, 1\}^2$ In the new space: $v^i \in \{0, 1\}^4$

$v^i = (1, 1, 0, 0)$

$z^i =$ `(1,1,1,1)`

$v^i \cdot z^i = 2^{samePos(v,z)}$

$same(v, z) = 1$

Kernel Perceptron: Implementation

$$f(x) = Th_\theta\left(\sum_{z\in M} S(z)K(x,z)\right) \qquad K(x,z) = \sum_{i\in I} t_i(z)t_i(x)$$

- Simply run Perceptron in an on-line mode, but keep track of the set M.
- Keeping the set M allows us to keep track of S(z).
- Rather than remembering the weight vector w, remember the set M (P and D) – all those examples on which we made mistakes

Primal and Dual Representations

- **Primal Representation**
  - Works in the transformed space (high dimension)
  - Explicitly compute feature values
  - Maintain a large weight vector
- **Dual Representation**
  - Original space
  - Kernel trick
  - Weight examples (mistakes)

$$f(x) = Th_\theta\left(\sum_{i\in I} w_i t_i(x)\right)$$

$$f(x) = Th_\theta\left(\sum_{z\in M} S(z)K(x,z)\right)$$

$$K(x,z) = \sum_{i\in I} t_i(z)t_i(x)$$

Kernel trick

- Define a feature function $\varphi(x)$ which maps items x into a **high dimensional** space

- The kernel function $K(x_i, x_j)$ computes the inner-product between $\varphi(x_i)$ and $\varphi(x_j)$

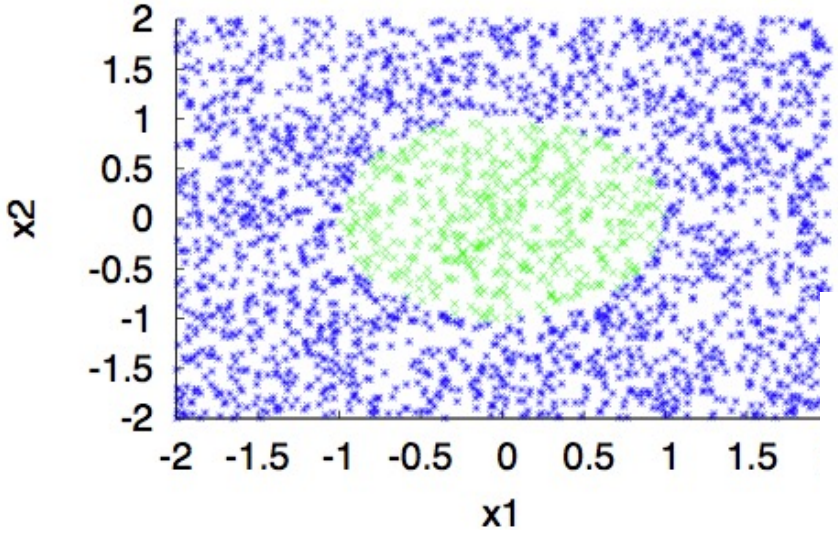
$$K(x_i, x_j) = \varphi(x_i)\varphi(x_j)$$

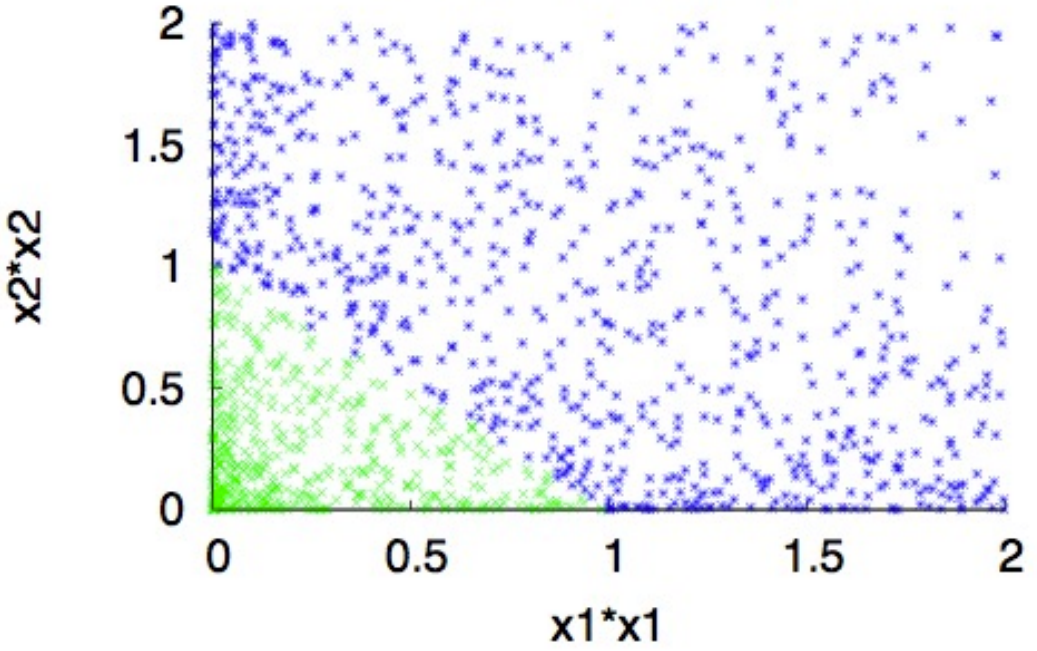
- Kernel function can be computed efficiently
  - No need to work with high dimensional *w*

Polynomial Kernel

A function $K(x,z)$ is a valid **kernel function** if it corresponds to an inner product in some (perhaps infinite dimensional) feature space.

Reminder..


2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
x2
-2 -1.5 -1 -0.5 0 0.5 1 1.5
x1


2
1.5
1
0.5
0
x2*x2
0 0.5 1 1.5 2
x1*x1

Polynomial Kernel

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{xy} + \mathbf{c})^{\mathbf{2}}$$

Example:

$$\mathbf{x} = (\mathbf{x_1}, \mathbf{x_2}) \in \mathbf{R^2} :$$

$$(\mathbf{x_1},\ \mathbf{x_2},\ \mathbf{x_1x_1},\ \mathbf{x_1x_2},\ \mathbf{x_1x_2},\ \mathbf{x_2x_2})$$

Polynomial Kernels

We can extend the quadratic kernel to any (higher) degree polynomial $(xz)^d$

Linear Kernel:

$$k(x, y) = xy$$

Polynomial Kernel:

$$k(x, y) = (xy)^d$$

Polynomial Kernel *up to degree n*:

$$k(x, y) = (xy + c)^d \ , \quad c > 0$$

The Kernel Matrix

The Kernel matrix of D $=\{\mathbf{x_1}...\mathbf{x_n}\}$,

defined by a kernel function $\mathbf{k}(\mathbf{x},\mathbf{z}) = \phi(\mathbf{x})\phi(\mathbf{z})$

is the $\mathbf{n} \times \mathbf{n}$ matrix $\mathbf{K}$, $\mathbf{K_{i,j}} = \mathbf{k}(\mathbf{x_i},\mathbf{x_j})$

The **Gram matrix** of the vectors $\mathbf{x_1}..\ \mathbf{x_n}$

$\mathbf{K}$ is symmetric: $\mathbf{K_{i,j}} = \mathbf{k}(\mathbf{x_i},\mathbf{x_j}) = \phi(\mathbf{x})\phi(\mathbf{z}) = \mathbf{k}(\mathbf{x_j},\mathbf{x_i}) = \mathbf{K_{j,i}}$

$\mathbf{K}$ is positive semi-definite: $(\forall \mathbf{v} : \mathbf{v^T K v} \geq \mathbf{0})$

$$\mathbf{v^T K v} = \sum_{\mathbf{i=0}}^{\mathbf{D}} \sum_{\mathbf{j=0}}^{\mathbf{D}} v_i v_j \mathbf{K_{i,j}} = \sum_{\mathbf{i=0}}^{\mathbf{D}} \sum_{\mathbf{j=0}}^{\mathbf{D}} v_i v_j \langle \phi(\mathbf{x_i})\phi(\mathbf{x_j}) \rangle =$$

$$\sum_{\mathbf{i=0}}^{\mathbf{D}} \sum_{\mathbf{j=0}}^{\mathbf{D}} v_i v_j \sum_{\mathbf{k=1}}^{\mathbf{N}} \phi(\mathbf{x_i})\phi(\mathbf{x_j}) = \sum_{\mathbf{i=0}}^{\mathbf{D}} \sum_{\mathbf{j=0}}^{\mathbf{D}} \sum_{\mathbf{k=1}}^{\mathbf{N}} v_i \phi(\mathbf{x_i})\ v_j \phi(\mathbf{x_j}) =$$

$$\sum_{\mathbf{k=1}}^{\mathbf{N}} \left(\sum_{\mathbf{i=1}}^{\mathbf{D}} v_i \right)^{\mathbf{2}} \geq \mathbf{0}$$

Constructing New Kernels

- You can construct new kernels $k'(\mathbf{x}, \mathbf{x}')$ from existing ones:
  - Multiplying $k(\mathbf{x}, \mathbf{x}')$ by a constant c: $k'(\mathbf{x}, \mathbf{x}') = ck(\mathbf{x}, \mathbf{x}')$
  - Multiplying $k(\mathbf{x}, \mathbf{x}')$ by a function f applied to $\mathbf{x}$ and $\mathbf{x}'$:

    $k'(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})k(\mathbf{x}, \mathbf{x}')f(\mathbf{x}')$
  - Applying a polynomial (w\ non-neg. coeffs.) to $k(\mathbf{x}, \mathbf{x}')$:

    $k'(\mathbf{x}, \mathbf{x}') = P(\ k(\mathbf{x}, \mathbf{x}')\ )$ with $P(z) = \sum_i a_i z^i$ and $a_i \geq \mathbf{0}$
  - Exponentiating $k(\mathbf{x}, \mathbf{x}')$: $k'(\mathbf{x}, \mathbf{x}') = \exp(k(\mathbf{x}, \mathbf{x}'))$

Constructing New Kernels

- You can construct $k'(\mathbf{x}, \mathbf{x'})$ from $k_1(\mathbf{x}, \mathbf{x'})$, $k_2(\mathbf{x}, \mathbf{x'})$:
  - Adding $k_1(\mathbf{x}, \mathbf{x'})$ and $k_2(\mathbf{x}, \mathbf{x'})$: $k'(\mathbf{x}, \mathbf{x'}) = k_1(\mathbf{x}, \mathbf{x'}) + k_2(\mathbf{x}, \mathbf{x'})$
  - Multiplying $k_1(\mathbf{x}, \mathbf{x'})$ and $k_2(\mathbf{x}, \mathbf{x'})$: $k'(\mathbf{x}, \mathbf{x'}) = k_1(\mathbf{x}, \mathbf{x'})k_2(\mathbf{x}, \mathbf{x'})$
  - If $\phi(\mathbf{x}) \in R^m$ and $k_m(\mathbf{z}, \mathbf{z'})$ a valid kernel in R^m, $k(\mathbf{x}, \mathbf{x'}) = k_m(\phi(\mathbf{x}), \phi(\mathbf{x'}))$ is also a valid kernel
  - If $\mathbf{A}$ is a symmetric positive semi-definite matrix, $k(\mathbf{x}, \mathbf{x'}) = \mathbf{x}\mathbf{A}\mathbf{x'}$ is also a valid kernel

Gaussian Kernel (*radial basis function kernel*)

$$k(x, z) = \exp\left(-\frac{||\mathbf{x} - \mathbf{z}||^2}{c}\right)$$

- $||x - z||^2$: squared Euclidean distance between **x** and **z**
- $c = 2\sigma^2$: a free parameter

- Intuition:
  - k(**x**, **z**) ≈ 1 when **x**, **z** close
  - k(**x**, **z**) ≈ 0 when **x**, **z** dissimilar

  - **very small c:** every item is different
  - **very large c**: all items are the same

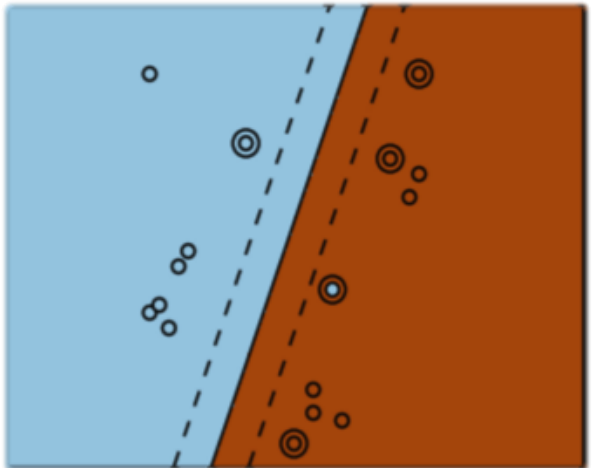
Gaussian Kernel (*radial basis function kernel*)

- k(x, z) = $\exp\left(-\frac{||\mathbf{x}-\mathbf{z}||^2}{c}\right)$ ***Is this a kernel?***

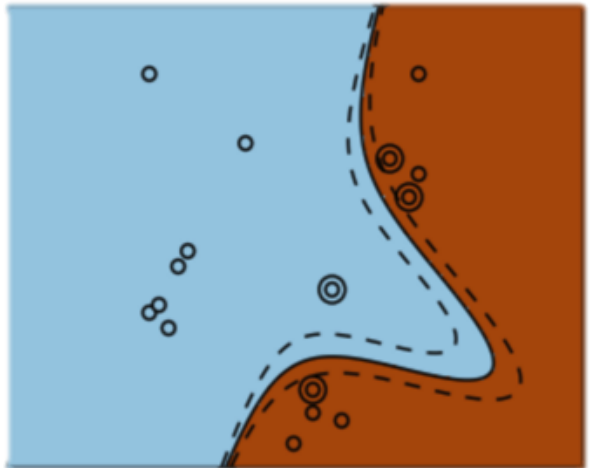
- $k(\mathbf{x}, \mathbf{z}) = \exp(-(\mathbf{x}-\mathbf{z})^2/2\sigma^2)$
  $= \exp(-(\mathbf{xx} + \mathbf{zz} - 2\mathbf{xz})/2\sigma^2)$
  $= \exp(-\mathbf{xx}/2\sigma^2)\ \exp(\mathbf{xz}/\sigma^2)\ \exp(-\mathbf{zz}/2\sigma^2)$
  $= f(\mathbf{x})\ \exp(\mathbf{xz}/\sigma^2)\ f(\mathbf{z})$

- **$\exp(\mathbf{xz}/\sigma^2)$ is a valid kernel:**
  - **xz** is the linear kernel;
  - we can multiply kernels by constants ($1/\sigma^2$)
  - we can exponentiate kernels

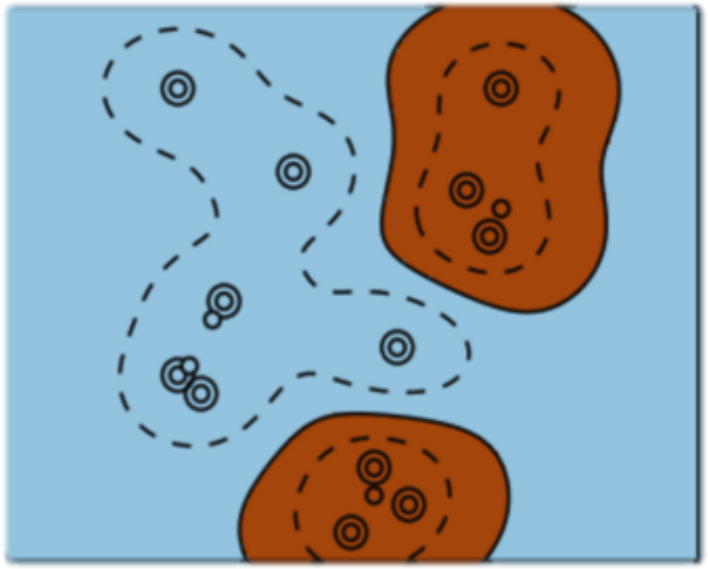
Linear Kernel



Polynomial Kernel



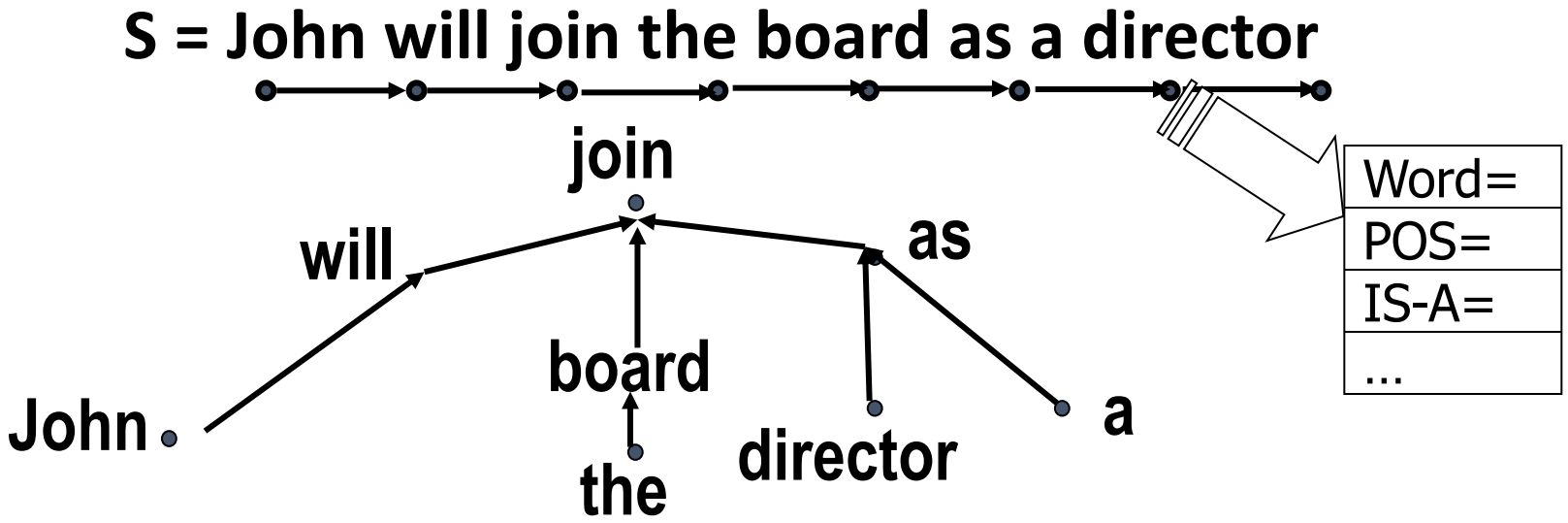
Gaussian (RBF) Kernel



Learning From Structured Input

- Extract features from ***structured domain elements***
  - their internal (hierarchical) structure should be encoded.
- A feature is a mapping from the **instance space** to *{0,1}* or *an integer/real number*
  - It is possible to represent expressive features that constitute an infinite dimensional space
  - ***Learning can be done in the infinite attribute domain.***
- What does it mean to extract features over structured inputs?
  - **Conceptually**: different data instantiations may be abstracted to yield the same representation
  - **Computationally**: graph matching process (graph isomorphism)

Structured Input



We want to unify the representation for all examples:

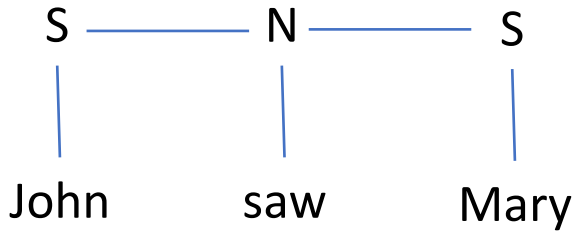
"John will join" , "John joins", "John joined"

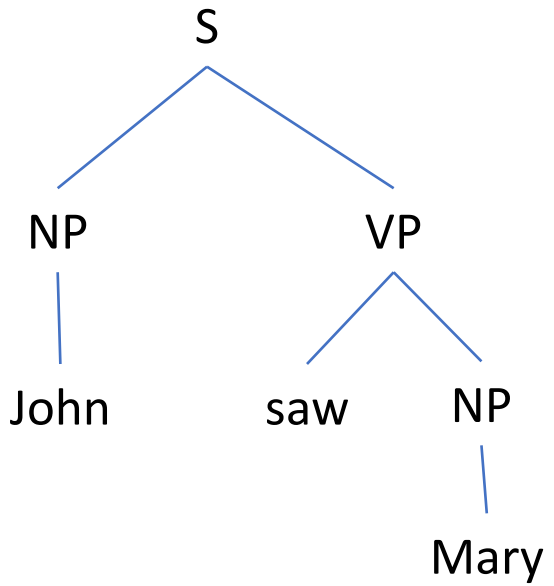
Define patterns (feature functions) that will determine the representation

[NP Which type] [PP of] [NP submarine] [VP was bought]
[ADVPrecently] [PP by] [NP South Korea] (. ?)

Extracting features from Input Structures

- Tree structures are useful representations of hierarchical data
  - *NLP*: Syntactic/semantic aspects can be represented using trees
  - Other examples: social networks, protein structures
- **Issue**: how can we capture structural properties?





Extracting features from Input Structures

- Define a feature function Φ mapping input structures into a feature vector
  - Represent interesting ***structural properties***
  - ***Example:*** ***sub-tree counting feature***

$h_1(x)$: *how many times does* 

appear in x*?*

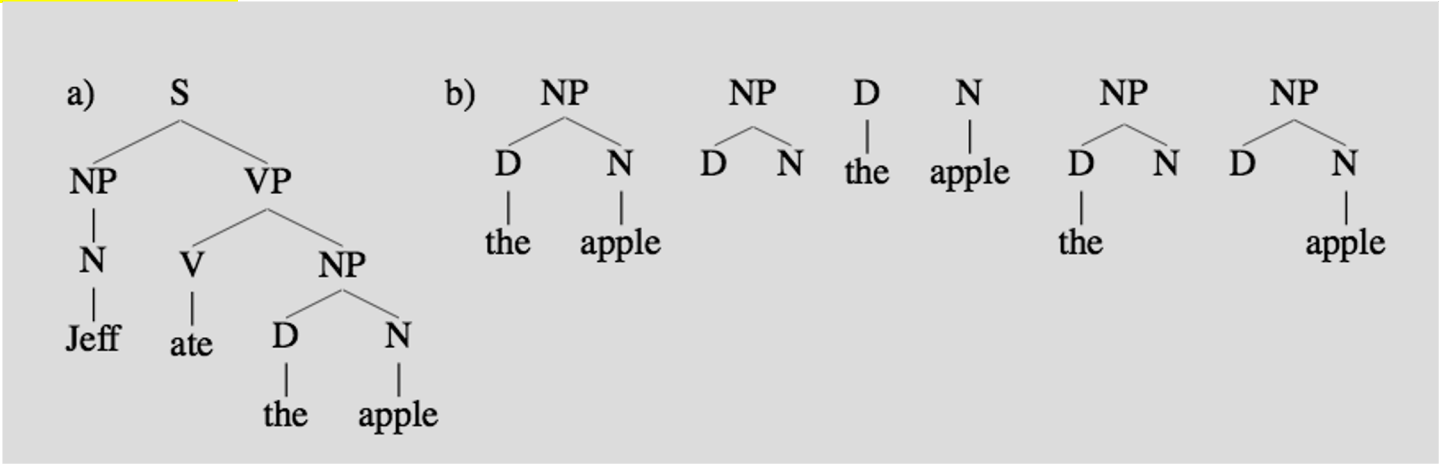
Extracting features from Input Structures

Given a set of Non-Terminal symbols $\{A, B, ..\}$
and Terminal Symbols {a,b,c}

The functions $h_1, ..., h_d$ *define a feature vector*

Each h_i corresponds to a different subtree!

$\Phi(x)=(h_1(x), ..., h_d(x))$



Convolution Kernels for Natural Language. Collins and Duffy. NIPS 2001

Extracting features from Input Structures

$$\Phi(x) = \langle h_1(x), h_2(x), .., h_d(x) \rangle$$

The dot product between two input structures T_1 and T_2

$$\Phi(T_1) \cdot \Phi(T_2) = \sum_{i=1}^{d} h_i(T_1) h_i(T_2)$$

Intuition:

- Two trees are "close" if they share many sub-trees

Dot product can be computed efficiently using dynamic programming

Kernels: Complexity Tradeoff

- It's possible to define kernels for structured data
  - *Equivalent to blowing up the feature space by generating functions of primitive features*
- Is it worth doing in structured domains?
  - *Tree*: not all sub-trees are that important
  - Small matches are more interesting (large trees→ overfits)
  - Increased dimensionality
- **Computationally**: *Dual space* ***vs.*** *Primal Space*
  - Dual space: #examples while the primal: blown up feature space
  - **Consideration**: *the number of examples one needs to use relative to the growth in dimensionality.*

Kernels: Generalization

- ***Do we want to use the most expressive kernels we can?***
  - No; this is equivalent to working in a larger feature space, and will lead to overfitting.
  - You add a lot of irrelevant features
- **In practice, many applications use a linear kernel/work in the primal**
  - Explicitly extend the feature space

Multiclass classification

So far we have focused on Binary prediction problem.
While not an issue for KNN and DT, it did simplify things
when dealing with linear models, we could just view the classifier as a hyperplane dissecting the space into two halfspaces – class 1 and class 2. .. But what happens if you have class 3 and class 4?

We will introduce several approaches for task decomposition
and discuss their advantages

Multi-Categorical Output Tasks

- So far, our discussion was limited to ***binary predictions***
  - Well, ***almost*** (?)
- What happens if our decision is not over binary labels?
  - *Many interesting classification problems are not!*
  - **Credit card**: Approved, Deny, Further investigation needed
  - **Document classification**: sports, finance, politics
  - **OCR**: *0,1,2,3..9,A,..,Z*

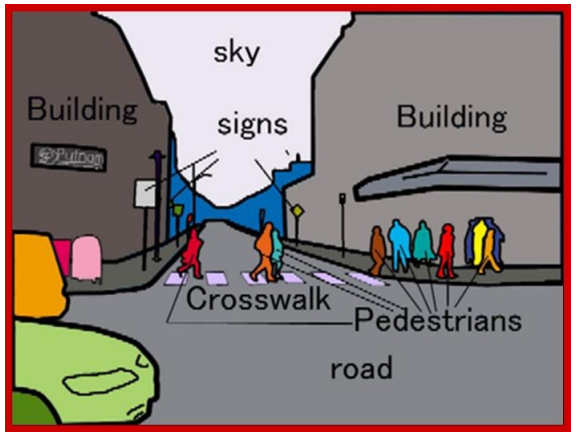
How can we approach these problems?

- *We will look into different reductions to binary classification problems!*

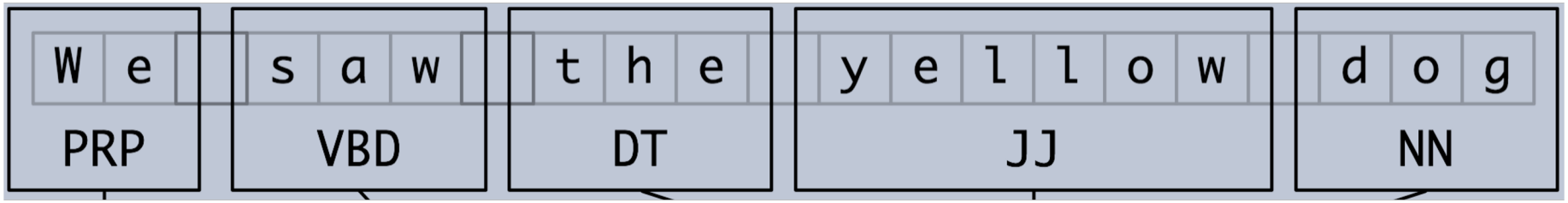
Hint: What is the computer science solution to: *"I can solve problem A, but now I have problem B, so…"*

Beyond Binary Classification: Applications

Image taken from Lior Wolf VOR page



Politics?
Fashions?
Business?



One-Vs-All

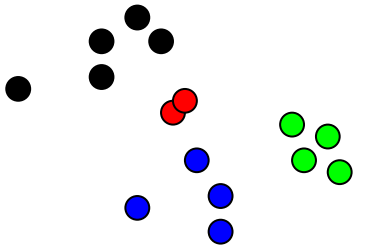
Assumption: *Each class can be separated from the rest using a binary classifier*

- **Learning**: Decomposed to learning k independent binary classifiers, one corresponding to each class
  - An example (x,y) is considered positive for class y and negative to all others.
  - Assume m examples, k class labels (assume m/k in each)
  - Classifier f_i: m/k (positive) and (k-1)m/k (negative)
- **Decision**: Winner Takes All:
  - $f(x) = \text{argmax}_i f_i(x) = \text{argmax}_i (v_i x)$

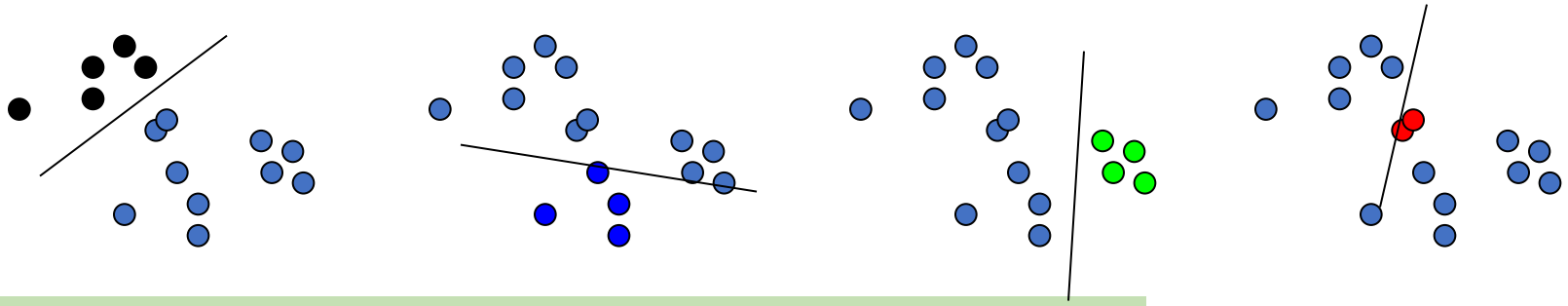
Q: Why do we need the assumption above?

Solving Multi-Class with binary learning

- **Multi-Class classifier**
  - **Function** $f : x \rightarrow \{1,2,3,\ldots,k\}$



- **Decompose into binary problems**



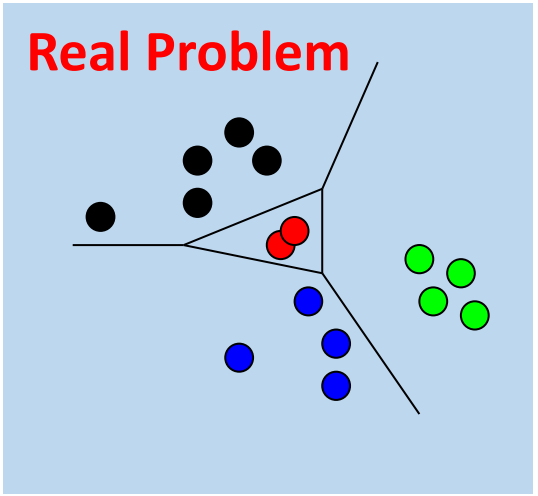
- **Not always possible to learn**
  - *No theoretical justification*

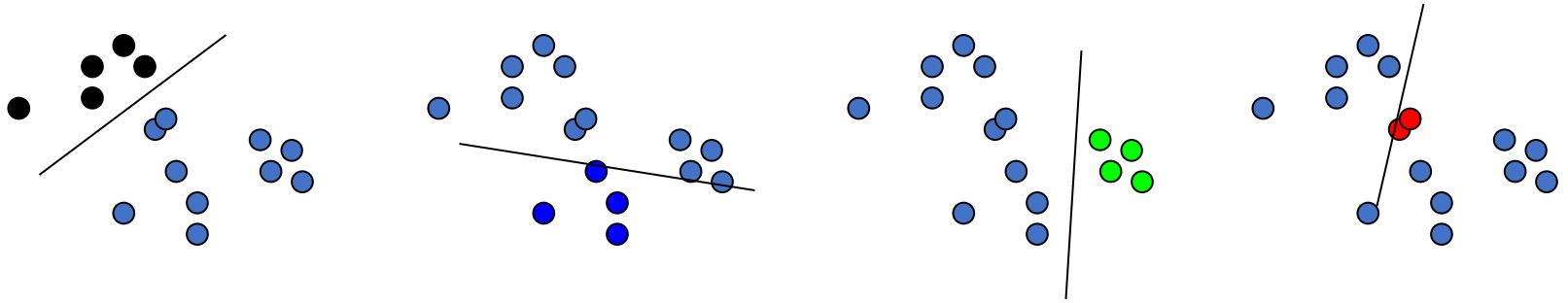
    ..unless the problem is "easy"

Learning via One-Versus-All

- Find $v_r, v_b, v_g, v_y \in \mathbf{R}^n$ such that
  - $v_r \, x > 0$ iff y = red ⊗
  - $v_b \, x > 0$ iff y = blue √
  - $v_g \, x > 0$ iff y = green √
  - $v_y \, x > 0$ iff y = yellow √
- *Classification*: $f(x) = \text{argmax}_i \, (v_i \, x)$

$$\mathbf{H} = \mathbf{R}^{kn}$$





Problems with Decompositions

- **Learning optimizes over *local* metrics**
  - Poor *global* performance
  - What is the metric?
    - We don't care about *local* classifiers performance
  - ***Poor decomposition* $\Rightarrow$ *poor performance***
  - Especially true for Error Correcting Output Codes
- **Difficulty:** *how to ensure that the resulting problems are separable.*
- Can we ensure separability and learn the real objective?
- Real objective: final performance, rather than local metric

Multiclass classification

- The examples given to learner are pairs (x,y), $y \in \{1,\ldots k\}$
- The "black box learner" seems like *a function of only* x but, actually, *we made use of the labels* y

- ***How is y being used?***
  - y "decides" which of the k classifiers should take the example as a positive example (making it a negative to all the others)

Multiclass classification

- *How do we make decisions:*
  - Let: $f_y(x) = w_y^T x$
  - Then, we predict using: $y^* = \text{argmax}_{y=1,\ldots k} \; f_y(x)$

- ***Equivalently, we can say that we predict as follows:***
  - Predict y iff: $\forall y' \in \{1,\ldots k\}, \; y' != y, \; (w_y^T - w_{y'}^T)x > 0$

- *How do we learn the k weight vectors w_y (y = 1,…k) ?*

Linear Separability for Multiclass

- We are learning k n-dimensional weight vectors, so we can concatenate the k weight vectors into $w = (w_1, w_2, \ldots w_k) \in R^{nk}$
- **Key Construction**: (**Kesler Construction**)
- Represent each example (x,y), as an nk-dimensional vector, x_y with x embedded in the y-th part of it (y=1,2,…k) and the other coordinates are 0

E.g., $\mathbf{x}_y = (\mathbf{0},x,\mathbf{0},\mathbf{0}) \in \mathbf{R}^{kn}$ (*here k=4, y=2*)

Example: *predict sentiment of a product review*

Features *– words (unigram)*
Labels*: Good, Bad, Neutral.*

The feature corresponding to "sick" when the label is "good"

(["sick",good], …, ["sick",neutral], …, ["sick",bad],..)

Good label features | Neutral label features | bad label features

Linear Separability for Multiclass

- We are learning k n-dimensional weight vectors, so we can concatenate the k weight vectors into $w = (w_1, w_2, \ldots w_k) \in R^{nk}$
- **Key Construction**: (**Kesler Construction**)
- Represent each example (x,y), as an nk-dimensional vector, x_y with x embedded in the y-th part of it (y=1,2,…k) and the other coordinates are 0

E.g., $\mathbf{x}_y = (\mathbf{0},x,\mathbf{0},\mathbf{0}) \in \mathbf{R}^{kn}$ (*here k=4, y=2*)

- Now we can understand the decision
  - Predict y iff $\forall\ y' \in \{1,\ldots k\}, y!=y \quad (w_y^T - w_{y'}^T)\ x > 0$

… In the nk-dimensional space:

  - Predict y iff $\forall y' \in \{1,\ldots k\}, y'!=y \quad w^T(x_y - x_{y'}) \equiv w^T x_{yy'} > 0$

Conclusion: The set $(x_{yy'}, +) \equiv (x_y - x_{y'}, +)$ is ***linearly separable*** from the set $(x_{yy'}, -)$ using the linear separator $w \in R^{kn}$,

Learning via Kesler Construction

- A **perceptron update** rule applied in the nk-dimensional space due to a mistake in $w^T x_{ij} > 0$
- Implies the following update:

- Given example (x,i) (example $x \in R^n$, labeled i)
  - $\forall j = 1,\ldots k,\ i \neq j$
  - If $(w_i^T - w_j^T)\ x < 0$ (mistaken prediction)
    - $w_i \leftarrow w_i + x$ (*promotion*)
    - $w_j \leftarrow w_j - x$ (*demotion*)

For any given example, you can potentially make K-1 promotion steps for w_i and K-1 demotion steps, for the *different* w_j

Conservative update

- The general scheme suggests that on every mistake:
  - Promote w_i and demote k-1 weight vectors w_j
- A **conservative** update:
  - In case of a mistake: only the weights corresponding to the target node i and that closest node j are updated.
  - Let: $j^* = \text{argmin}_{j=1,\dots k} (w_i^T - w_j^T) x$
  - If $(w_i^T - w_{j^*}^T) x < 0$ (mistaken prediction)
  - $w_i \leftarrow w_i + x$ (promotion) and $w_{j^*} \leftarrow w_{j^*} - x$ (demotion)
  - **Other weight vectors are not updated.**

Margin in the Multiclass case

- How would you change the notion of a margin for multiclass classification case?