Algorithms for NLP

Classification I

Sachin Kumar - CMU

Slides: Dan Klein – UC Berkeley, Taylor Berg-Kirkpatrick, Yulia Tsvetkov – CMU
Classification

Image → Digit

![Image with handwritten digits]
Classification

Document → Category
Classification

Query + Web Pages $\rightarrow$ Best Match

“Apple Computers”
The screen was a sea of red
Economic growth has slowed down in recent years.

Das Wirtschaftswachstum hat sich in den letzten Jahren verlangsamt.
Classification

- Three main ideas
  - Representation as feature vectors
  - Scoring by linear functions
  - Learning (the scoring functions) by optimization
Some Definitions

INPUTS

\[ X_i \]

CLOSE THE ____

CANDIDATE SET

\[ \mathcal{Y}(x) \]

\{table, door, … \}

CANDIDATE

\[ y \]

table

TRUE OUTPUT

\[ y^*_i \]

door

FEATURE VECTORS

\[ f(x, y) \]

\[ \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \]

\[ x_{-1} = “the” \land y = “door” \]

\[ “close” \text{ in } x \land y = “door” \]

\[ x_{-1} = “the” \land y = “table” \]

\[ y \text{ occurs in } x \]
Features
Feature Vectors

- Example: web page ranking (not actually classification)

\[ x_i = \text{“Apple Computers”} \]

\[ f_i(\text{Apple}) = [0.3 \ 5 \ 0 \ 0 \ 0 \ \ldots] \]

\[ f_i(\text{Apple Inc.}) = [0.8 \ 4 \ 2 \ 1 \ \ldots] \]
Sometimes, we think of the input as having features, which are multiplied by outputs to form the candidates

\[ \mathbf{x} \quad \cdots \text{win the election} \cdots \]

\[ \text{“f(x)”} \quad \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix} \]

\[ \text{“win”} \quad \text{“election”} \]

\[ \text{... win the election ...} \]
\[ f(\text{SPORTS}) = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \]

\[ \text{... win the election ...} \]
\[ f(\text{POLITICS}) = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \]

\[ \text{... win the election ...} \]
\[ f(\text{OTHER}) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \]
Non-Block Feature Vectors

- Sometimes the features of candidates cannot be decomposed in this regular way.
- Example: a parse tree’s features may be the productions present in the tree.

\[ f(\text{NP } \text{VP}) = [1 \ 0 \ 1 \ 0 \ 1] \]

\[ f(\text{VP } \text{VP}) = [1 \ 1 \ 0 \ 1 \ 0] \]

- Different candidates will thus often share features.
- We’ll return to the non-block case later.
Linear Models
Linear Models: Scoring

- In a linear model, each feature gets a weight $w$

$$f(\text{POLITICS}) = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$f(\text{SPORTS}) = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$w = \begin{bmatrix} 1 & 1 & -1 & -2 & 1 & -1 & 1 & -2 & -2 & -1 & -1 & 1 \end{bmatrix}$$

- We score hypotheses by multiplying features and weights:

$$score(y, w) = w^\top f(y)$$

$$f(\text{POLITICS}) = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$w = \begin{bmatrix} 1 & 1 & -1 & -2 & 1 & -1 & 1 & -2 & -2 & -1 & -1 & 1 \end{bmatrix}$$

$$score(\text{POLITICS}, w) = 1 \times 1 + 1 \times 1 = 2$$
The linear decision rule:

\[
prediction(\text{... win the election ...}, \mathbf{w}) = \arg \max_{y \in \mathcal{Y}(x)} \mathbf{w}^\top f(y)
\]

\[
score(\text{SPORTS}, \mathbf{w}) = 1 \times 1 + (-1) \times 1 = 0
\]

\[
score(\text{POLITICS}, \mathbf{w}) = 1 \times 1 + 1 \times 1 = 2
\]

\[
score(\text{OTHER}, \mathbf{w}) = (-2) \times 1 + (-1) \times 1 = -3
\]

\[
prediction(\text{... win the election ...}, \mathbf{w}) = \text{POLITICS}
\]

We’ve said nothing about where weights come from
Binary Classification

- Important special case: binary classification
  - Classes are $y=+1/-1$

- Decision boundary is a hyperplane
  \[ w^T f(x) = 0 \]
Multiclass Decision Rule

- If more than two classes:
  - Highest score wins
  - Boundaries are more complex
  - Harder to visualize

\[
prediction(x_i, w) = \arg \max_{y \in \mathcal{Y}} \mathbf{w}^T f_i(y)
\]
Learning
Learning Classifier Weights

- Two broad approaches to learning weights

- Generative: work with a probabilistic model of the data, weights are (log) local conditional probabilities
  - Advantages: learning weights is easy, smoothing is well-understood, backed by understanding of modeling

- Discriminative: set weights based on some error-related criterion
  - Advantages: error-driven, often weights which are good for classification aren’t the ones which best describe the data

- We’ll mainly talk about the latter for now
How to pick weights?

- **Goal:** choose “best” vector $w$ given training data
  - For now, we mean “best for classification”

- **The ideal:** the weights which have greatest test set accuracy / F1 / whatever
  - But, don’t have the test set
  - Must compute weights from training set

- **Maybe we want weights which give best training set accuracy?**
Minimize Training Error?

- A loss function declares how costly each mistake is
  \[ \ell_i(y) = \ell(y, y^*_i) \]
  - E.g. 0 loss for correct label, 1 loss for wrong label
  - Can weight mistakes differently (e.g. false positives worse than false negatives or Hamming distance over structured labels)

- We could, in principle, minimize training loss:
  \[ \min_w \sum_i \ell_i \left( \arg \max_y w^\top f_i(y) \right) \]
  - This is a hard, discontinuous optimization problem
The perceptron algorithm
- Iteratively processes the training set, reacting to training errors
- Can be thought of as trying to drive down training error

The (online) perceptron algorithm:
- Start with zero weights \( w \)
- Visit training instances one by one
  - Try to classify
    \[
    \hat{y} = \arg \max_{y \in \mathcal{Y}(x)} w^\top f(y)
    \]
    - If correct, no change!
    - If wrong: adjust weights
  \[
  w \leftarrow w + f(y_i^*)
  \]
  \[
  w \leftarrow w - f(\hat{y})
  \]
Example: “Best” Web Page

\[ w = [1 \ 2 \ 0 \ 0 \ldots] \]

\[ x_i = “\text{Apple Computers}” \]

\[ f_i(\text{Apple}) = [0.3 \ 5 \ 0 \ 0 \ldots] \quad w^\top f = 10.3 \quad \hat{y} \]

\[ f_i(\text{Apple Inc.}) = [0.8 \ 4 \ 2 \ 1 \ldots] \quad w^\top f = 8.8 \quad \hat{y}_i^* \]

\[ w \leftarrow w + f(\hat{y}_i^*) - f(\hat{y}) \]

\[ w = [1.5 \ 1 \ 2 \ 1 \ldots] \]
Examples: Perceptron

- Separable Case
Examples: Perceptron

- Non-Separable Case
Problems with Perceptron

- Perceptron “Goal”: Separate the training data

\[ \forall i, \forall y \neq y^i \quad w^T f_i(y^i) \geq w^T f_i(y) \]

1. This may be an entire feasible space
2. Or it may be impossible
Objective Functions

- What do we want from our weights?
  - So far: minimize (training) errors:
    \[
    \min_w \sum_i \ell_i \left( \arg\max_y w^T f_i(y) \right)
    \]
    or
    \[
    \sum_i \text{step} \left( w^T f_i(y_i^*) - \max_{y \neq y_i^*} w^T f_i(y) \right)
    \]
  - This is the “zero-one loss”
    - Discontinuous, minimizing is NP-complete
  - Maximum entropy and SVMs have other objectives related to zero-one loss
Margin
Linear Separators

- Which of these linear separators is optimal?
• Distance of $x_i$ to separator is its margin, $m_i$
• Examples closest to the hyperplane are support vectors
• Margin $\gamma$ of the separator is the minimum $m$
Classification Margin

- For each example $x_i$ and possible mistaken candidate $y$, we avoid that mistake by a margin $m_i(y)$ (with zero-one loss)
  
  $$m_i(y) = w^\top f_i(y_i^*) - w^\top f_i(y)$$

- Margin $\gamma$ of the entire separator is the minimum $m$

  $$\gamma = \min_i \left( w^\top f_i(y_i^*) - \max_{y \neq y_i^*} w^\top f_i(y) \right)$$

- It is also the largest $\gamma$ for which the following constraints hold

  $$\forall i, \forall y \quad w^\top f_i(y_i^*) \geq w^\top f_i(y) + \gamma \ell_i(y)$$
Maximum Margin

- Separable SVMs: find the max-margin $w$

$$\max_{\|w\| = 1} \gamma$$

$$\ell_i(y) = \begin{cases} 0 & \text{if } y = y_i^* \\ 1 & \text{if } y \neq y_i^* \end{cases}$$

$$\forall i, \forall y \quad w^T f_i(y_i^*) \geq w^T f_i(y) + \gamma \ell_i(y)$$

- Can stick this into Matlab and (slowly) get an SVM
- Won’t work (well) if non-separable
Max Margin / Small Norm

- Reformulation: find the smallest $w$ which separates data

Remember this condition?

\[
\forall i, y \quad w^T f_i(y_i^*) \geq w^T f_i(y) + \gamma \ell_i(y)
\]

- $\gamma$ scales linearly in $w$, so if $\|w\|$ isn’t constrained, we can take any separating $w$ and scale up our margin

\[
\gamma = \min_{i, y \neq y_i^*} \frac{w^T f_i(y_i^*) - w^T f_i(y)}{\ell_i(y)}
\]

- Instead of fixing the scale of $w$, we can fix $\gamma = 1$

\[
\min_w \frac{1}{2} \|w\|^2 \\
\forall i, y \quad w^T f_i(y_i^*) \geq w^T f_i(y) + 1\ell_i(y)
\]
Gamma to w
Soft Margin Classification

- What if the training set is not linearly separable?
- Slack variables $\xi_i$ can be added to allow misclassification of difficult or noisy examples, resulting in a soft margin classifier.
Maximum Margin

- Non-separable SVMs
  - Add slack to the constraints
  - Make objective pay (linearly) for slack:
    \[
    \min_{w,\xi} \frac{1}{2}||w||^2 + C \sum_i \xi_i
    \]
    \[
    \forall i, y, \quad w^\top f_i(y^*_i) + \xi_i \geq w^\top f_i(y) + \ell_i(y)
    \]
  - $C$ is called the *capacity* of the SVM – the smoothing knob

- Learning:
  - Can still stick this into Matlab if you want
  - Constrained optimization is hard; better methods!

Note: exist other choices of how to penalize slacks!
Hinge Loss

- We have a constrained minimization

\[
\min_{w, \xi} \frac{1}{2} \|w\|^2 + C \sum_i \xi_i
\]

\[\forall i, y, \ w^\top f_i(y^*_i) + \xi_i \geq w^\top f_i(y) + \ell_i(y)\]

- ...but we can solve for \(\xi_i\)

\[\forall i, y, \ \xi_i \geq w^\top f_i(y) + \ell_i(y) - w^\top f_i(y^*_i)\]

\[\forall i, \ \xi_i = \max_y \left( w^\top f_i(y) + \ell_i(y) \right) - w^\top f_i(y^*_i)\]

- Giving

\[
\min_w \frac{1}{2} \|w\|^2 + C \sum_i \left( \max_y \left( w^\top f_i(y) + \ell_i(y) \right) - w^\top f_i(y^*_i) \right)
\]
Why Max Margin?

- Why do this? Various arguments:
  - Solution depends only on the boundary cases, or *support vectors*
  - Solution robust to movement of support vectors
  - Sparse solutions (features not in support vectors get zero weight)
  - Generalization bound arguments
  - Works well in practice for many problems
Likelihood
Linear Models: Maximum Entropy

- **Maximum entropy (logistic regression)**
  - Use the scores as probabilities:
    \[
    P(y|x, w) = \frac{\exp(w^T f(y))}{\sum_{y'} \exp(w^T f(y'))}
    \]
    Make positive
    Normalize
  - Maximize the (log) conditional likelihood of training data
    \[
    L(w) = \log \prod_i P(y_i^*|x_i, w) = \sum_i \log \left( \frac{\exp(w^T f_i(y_i^*))}{\sum_y \exp(w^T f_i(y))} \right)
    \]
    \[
    = \sum_i \left( w^T f_i(y_i^*) - \log \sum_y \exp(w^T f_i(y)) \right)
    \]
Maximum Entropy II

- Motivation for maximum entropy:
  - Connection to maximum entropy principle (sort of)
  - Might want to do a good job of being uncertain on noisy cases…
  - … in practice, though, posteriors are pretty peaked

- Regularization (smoothing)

\[
\begin{align*}
\max_w & \quad \sum_i \left( w^\top f_i(y_i^*) - \log \sum_y \exp(w^\top f_i(y)) \right) - k \|w\|^2 \\
\min_w & \quad k \|w\|^2 - \sum_i \left( w^\top f_i(y_i^*) - \log \sum_y \exp(w^\top f_i(y)) \right)
\end{align*}
\]
Maximum Entropy
Loss Comparison
Log-Loss

- If we view maxent as a minimization problem:

\[
\min_w \ k||w||^2 + \sum_i - \left( w^T f_i(y_i^*) - \log \sum_y \exp(w^T f_i(y)) \right)
\]

- This minimizes the “log loss” on each example

\[
- \left( w^T f_i(y_i^*) - \log \sum_y \exp(w^T f_i(y)) \right)
\]

- \(\text{O}(\text{step}) \left( w^T f_i(y_i^*) - \max_{y \neq y_i^*} w^T f_i(y) \right)\)
Consider the per-instance objective:

\[
\min_w \, k\|w\|^2 + \sum_i \left( \max_y \left( w^T f_i(y) + \ell_i(y) \right) - w^T f_i(y_i^*) \right)
\]

This is called the “hinge loss”

- Unlike maxent / log loss, you stop gaining objective once the true label wins by enough
- You can start from here and derive the SVM objective
- Can solve directly with sub-gradient decent (e.g. Pegasos: Shalev-Shwartz et al 07)
Max vs “Soft-Max” Margin

- **SVMs:**

  \[
  \min_w k\|w\|^2 - \sum_i \left( w^T f_i(y_i^*) - \max_y \left( w^T f_i(y) + \ell_i(y) \right) \right)
  \]

  You can make this zero

- **Maxent:**

  \[
  \min_w k\|w\|^2 - \sum_i \left( w^T f_i(y_i^*) - \log \sum_y \exp \left( w^T f_i(y) \right) \right)
  \]

  ...but not this one

- **Ver, better than a function of the other scores**
  - The SVM tries to beat the augmented runner-up
  - The Maxent classifier tries to beat the “soft-max”
Loss Functions: Comparison

- **Zero-One Loss**
  \[
  \sum_i \text{step} \left( w^\top f_i(y_{i}^*) - \max_{y \neq y_i^*} w^\top f_i(y) \right)
  \]

- **Hinge**
  \[
  \sum_i \left( w^\top f_i(y_{i}^*) - \max_y (w^\top f_i(y) + \ell_i(y)) \right)
  \]

- **Log**
  \[
  \sum_i \left( w^\top f_i(y_{i}^*) - \log \sum_y \exp \left( w^\top f_i(y) \right) \right)
  \]
Separators: Comparison
Structure
Handwriting recognition

Sequential structure

[Slides: Taskar and Klein 05]
The screen was a sea of red

Recursive structure
What is the anticipated cost of collecting fees under the new proposal?

En vertu de nouvelles propositions, quel est le coût prévu de perception de les droits?

Combinatorial structure
Definitions

INPUTS

\( X_i \)

CANDIDATE SET

\( \mathcal{Y}(x) \)

CANDIDATES

\( y \)

TRUE OUTPUTS

\( y_i^* \)

FEATURE VECTORS

\( f(x, y) \)
Structured Models

\[ \text{prediction}(x, w) = \arg \max_{y \in \mathcal{Y}(x)} \text{score}(y, w) \]

space of feasible outputs

Assumption:

\[ \text{score}(y, w) = w^\top f(y) = \sum_p w^\top f(y_p) \]

Score is a sum of local “part” scores

Parts = nodes, edges, productions
CFG Parsing

\[ f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^d \]

\[ #(NP \rightarrow \text{DT NN}) \]

\[ \ldots \]

\[ #(PP \rightarrow \text{IN NP}) \]

\[ \ldots \]

\[ #(NN \rightarrow 'sea') \]
What is the anticipated cost of collecting fees under the new proposal?
Efficient Decoding

- Common case: you have a black box which computes

\[
prediction(x) = \arg \max_{y \in \mathcal{Y}(x)} w^\top f(y)
\]

...at least approximately, and you want to learn \( w \)

- Easiest option is the structured perceptron [Collins 01]
  - Structure enters here in that the search for the best \( y \) is typically a combinatorial algorithm (dynamic programming, matchings, ILPs, A* …)
  - Prediction is structured, learning update is not
Structured Margin (Primal)

Remember our primal margin objective?

\[
\min_w \quad \frac{1}{2}\|w\|_2^2 + C \sum_i \left( \max_y \left( w^\top f_i(y) + \ell_i(y) \right) - w^\top f_i(y_i^*) \right)
\]

Still applies with structured output space!
Structured Margin (Primal)

Just need efficient loss-augmented decode:

\[ \bar{y} = \arg \max_y (w^\top f_i(y) + \ell_i(y)) \]

\[
\min_w \quad \frac{1}{2} \|w\|_2^2 + C \sum_i \left( w^\top f_i(\bar{y}) + \ell_i(\bar{y}) - w^\top f_i(y_i^*) \right)
\]

\[ \nabla_w = w + C \sum_i \left( f_i(\bar{y}) - f_i(y_i^*) \right) \]

Still use general subgradient descent methods! (Adagrad)
Structured Margin

- Remember the constrained version of primal:

\[
\begin{align*}
\min_{w, \xi} & \quad \frac{1}{2} ||w||^2 + C \sum_i \xi_i \\
\forall i, y \quad & w^\top f_i(y^*_i) \geq w^\top f_i(y) + \ell_i(y) - \xi_i
\end{align*}
\]
We want:

\[
\arg \max_y \ w^T f(\text{brace}, y) = \text{“brace”}
\]

Equivalently:

\[
w^T f(\text{brace}, \text{“brace”}) > w^T f(\text{brace}, \text{“aaaaa”})
\]

\[
w^T f(\text{brace}, \text{“brace”}) > w^T f(\text{brace}, \text{“aaaab”})
\]

\[
\ldots
\]

\[
w^T f(\text{brace}, \text{“brace”}) > w^T f(\text{brace}, \text{“zzzzz”})
\]
We want:

$$\arg \max_y \ w^T f(\text{"It was red"}, y) = \hat{A}_{C/D}$$

Equivalently:

$$w^T f(\text{"It was red"}, \hat{A}_{C/D}) > w^T f(\text{"It was red"}, \hat{A}_{D/F})$$

$$w^T f(\text{"It was red"}, \hat{A}_{C/D}) > w^T f(\text{"It was red"}, \hat{A}_{D/F})$$

$$\ldots$$

$$w^T f(\text{"It was red"}, \hat{A}_{C/D}) > w^T f(\text{"It was red"}, \hat{F}_{G/H})$$

a lot!
Alignment example

- We want:

\[ \arg \max_y w^\top f(\text{‘What is the’}, y) = \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix} \]

- Equivalently:

\[ w^\top f(\text{‘What is the’}, \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix}) > w^\top f(\text{‘Quel est le’}, \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix}) \]

\[ w^\top f(\text{‘What is the’}, \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix}) > w^\top f(\text{‘Quel est le’}, \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix}) \]

\[ w^\top f(\text{‘What is the’}, \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix}) > w^\top f(\text{‘Quel est le’}, \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix}) \]

\[ \text{...} \]

\[ w^\top f(\text{‘What is the’}, \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix}) > w^\top f(\text{‘Quel est le’}, \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix}) \]

\[ \text{a lot!} \]
A constraint induction method [Joachims et al 09]
- Exploits that the number of constraints you actually need per instance is typically very small
- Requires (loss-augmented) primal-decode only

Repeat:
- Find the most violated constraint for an instance:

\[ \forall y \quad w^T f_i(y^*) \geq w^T f_i(y) + \ell_i(y) \]

\[ \arg \max_y w^T f_i(y) + \ell_i(y) \]

- Add this constraint and resolve the (non-structured) QP (e.g. with SMO or other QP solver)
Some issues:

- Can easily spend too much time solving QPs
- Doesn’t exploit shared constraint structure
- In practice, works pretty well; fast like perceptron/MIRA, more stable, no averaging
Likelihood, Structured

\[ L(w) = -k ||w||^2 + \sum_i \left( w^\top f_i(y_i^*) - \log \sum_y \exp(w^\top f_i(y)) \right) \]

\[ \frac{\partial L(w)}{\partial w} = -2kw + \sum_i \left( f_i(y_i^*) - \sum_y P(y|x_i)f_i(y) \right) \]

- **Structure needed to compute:**
  - Log-normalizer
  - Expected feature counts
    - E.g. if a feature is an indicator of DT-NN then we need to compute posterior marginals \( P(DT-NN|sentence) \) for each position and sum

- **Also works with latent variables (more later)**
Comparison

![Graphs showing comparison of Constituency Parsing and Constituency Parsing, Neural CRF with different algorithms.](image)
Option 0: Reranking

Input

N-Best List
(e.g. n=100)

Output

\[ x = \text{“The screen was a sea of red.”} \]

Baseline Parser

Non-Structured Classification

[e.g. Charniak and Johnson 05]
Reranking

- **Advantages:**
  - Directly reduce to non-structured case

- **Disadvantages:**
  - Stuck with errors of baseline parser
  - Baseline system must produce n-best lists
  - But, feedback is possible [McCloskey, Charniak, Johnson 2006]
Another option: express all constraints in a packed form
- Maximum margin Markov networks [Taskar et al 03]
- Integrates solution structure deeply into the problem structure

Steps
- Express inference over constraints as an LP
- Use duality to transform minimax formulation into min-min
- Constraints factor in the dual along the same structure as the primal; alphas essentially act as a dual “distribution”
- Various optimization possibilities in the dual
Example: Kernels

- Quadratic kernels

\[
K(x, x') = (x \cdot x' + 1)^2
\]
\[
= \sum_{i,j} x_i x_j x'_i x'_j + 2 \sum_i x_i x'_i + 1
\]
\[
K(y, y') = (f(y)^\top f(y') + 1)^2
\]
Non-Linear Separators

- Another view: kernels map an original feature space to some higher-dimensional feature space where the training set is (more) separable
Why Kernels?

- Can’t you just add these features on your own (e.g. add all pairs of features instead of using the quadratic kernel)?
  - Yes, in principle, just compute them
  - No need to modify any algorithms
  - But, number of features can get large (or infinite)
  - Some kernels not as usefully thought of in their expanded representation, e.g. RBF or data-defined kernels [Henderson and Titov 05]

- Kernels let us compute with these features implicitly
  - Example: implicit dot product in quadratic kernel takes much less space and time per dot product
  - Of course, there’s the cost for using the pure dual algorithms…