CPSC 540: Machine Learning

Structured Prediction

Mark Schmidt

University of British Columbia

Winter 2018

Last Time: Kernel Trick

- Alternative approach to L2-regularized least squares with features Z:
  1. Derive non-linear features Z from X.
  2. Compute $K = ZZ^T$ containing all inner products $\langle z^i, z^j \rangle$.
  3. Fit model,
  $$v = (\underbrace{ZZ^T}_{K} + \lambda I)^{-1} y,$$
  4. Use the model to make predictions,
  $$\hat{y} = \underbrace{\tilde{Z}Z^T}_{\tilde{K}} v.$$
- This assumes we can compute Z.

Last Time: Kernel Trick

- Kernel trick for L2-regularized least squares with features Z:
  1. (No need for explicit features Z)
  2. Compute $K = ZZ^T$ containing all inner products $\langle z^i, z^j \rangle = k(x^i, x^j)$.
  3. Fit model,
  $$v = (\underbrace{K}_{n\times n} + \lambda I)^{-1} y,$$
  4. Use the model to make predictions,
  $$\hat{y} = \underbrace{\tilde{K}}_{t\times n} v.$$
- This does not assume we can compute Z.
  - Allows exponential- or infinite-sized features.
  - Instead of features, we could work with "similarity" $k(x^i, x^j)$.

Valid Kernels

- Can we use any function k for our kernel/similarity function $k(x^i, x^j)$?
- We need to have kernel k be an inner product in some space:
  - There exists transformation $z^i = \phi(x^i)$ such that $k(x^i, x^j) = \langle \phi(x^i), \phi(x^j) \rangle$.

We can decompose a (continuous or finite-domain) function k into

$$k(x^i, x^j) = \langle \phi(x^i), \phi(x^j) \rangle,$$

iff it is symmetric and for any finite $\{x^1, x^2, \ldots, x^n\}$ we have $K \succeq 0$.

- For finite domains you can show existence of ϕ using spectral theorem (bonus).
  - The general case is called Mercer's Theorem.

Valid Kernels

- Mercer's Theorem is nice in theory, what do we do in practice?
  - You could show explicitly that $k(x^i, x^j) = \langle\langle \phi(x^i), \phi(x^j) \rangle\rangle$ for some function ϕ.
  - You could that K is positive semi-definite by construction.
  - Or you can show k is constructed from other valid kernels.

(If we use invalid kernel, lose feature-space interpretation but may work fine.)

Constructing Valid Kernels

- If $k_1(x^i, x^j)$ and $k_2(x^i, x^j)$ are valid kernels, then the following are valid kernels:
  - Non-negative scaling: $\alpha k_1(x^i, x^j)$ for $\alpha \geq 0$.
  - Sum: $k_1(x^i, x^j) + k_2(x^i, x^j)$.
  - Product: $k_1(x^i, x^j) k_2(x^i, x^j)$.
    - Special case: $\phi(x^i) k_1(x^i, x^j) \phi(x^j)$ for any function ϕ.
  - Exponentiation: $\exp(k_1(x^i, x^j))$.
  - Recursion: $k_1(\phi(x^i), \phi(x^j))$ for any function ϕ.

- Example: Gaussian-RBF kernel:

$$k(x^i, x^j) = \exp\left(-\frac{\|x^i - x^j\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\|x^i\|^2}{2\sigma^2} + \frac{1}{\sigma^2}(x^i)^T x_j - \frac{1}{2\sigma^2}\|x^i\|^2\right)$$

$$= \underbrace{\exp\left(-\frac{\|x^i\|^2}{2\sigma^2}\right)}_{\phi(x^i)} \underbrace{\exp\left(\underbrace{\frac{1}{\sigma^2}}_{\alpha>0} \underbrace{(x^i)^T x^j}_{\text{valid}}\right)}_{\exp(\text{valid})} \underbrace{\exp\left(-\frac{\|x^j\|^2}{2\sigma^2}\right)}_{\phi(x^j)}.$$

Models allowing Kernel Trick

- Besides L2-regularized least squares, when can we apply the kernel trick?
  - Distance-based methods from CPSC 340:

$$\begin{aligned}\|z^i - z^j\|^2 &= \langle z_i, z_i\rangle - 2\langle z^i, z^j\rangle + \langle z^j, z_j\rangle \\ &= k(x^i, x^i) - 2k(x^i, x^j) + k(x^j, x^j).\end{aligned}$$

    - k-nearest neighbours.
    - Clustering algorithms (k-means, density-based clustering, hierarchical clustering).
    - Distance-based outlier detection (KNN-based, outlier ratio)
    - "Amazon product recommendation".
    - Multi-dimensional scaling (ISOMAP, t-SNE).
    - Label propagation.
  - L2-regularized linear models (today).
  - Eigenvalue methods:
    - Principle component analysis (need trick for centering in high-dimensional space).
    - Canonical correlation analysis.
    - Spectral clustering.

Representer Theorem

- Consider linear model with differentiable losses f_i and L2-regularization,

$$\underset{w \in \mathbb{R}^d}{\text{argmin}} \sum_{i=1}^n f_i(w^T x^i) + \frac{\lambda}{2}\|w\|^2.$$

- Setting the gradient equal to zero we get

$$0 = \sum_{i=1}^n \nabla f_i(w^T x^i) x^i + \lambda w.$$

- So any solution w^* be can written as a linear combination of features x^i,

$$w^* = -\frac{1}{\lambda} \sum_{i=1}^n \underbrace{\nabla f_i((w^*)^T x^i)}_{v_i} x^i = \sum_{i=1}^n v_i x^i = X^T v.$$

Representer Theorem

- Let's use the representation $w = X^T v$ in original problem,

$$\underset{w \in \mathbb{R}^d}{\text{argmin}} \sum_{i=1}^{n} f_i(w^T x^i) + \frac{\lambda}{2}\|w\|^2$$

$$= \underset{v \in \mathbb{R}^n}{\text{argmin}} \sum_{i=1}^{n} f_i(\underbrace{v^T X x^i}_{(x^i)^T X^T v}) + \frac{\lambda}{2}\|X^T v\|^2.$$

- Now defining $f(u) = \sum_{i=1}^{n} f_i(u_i)$ for a vector u we have

$$\equiv \underset{v \in \mathbb{R}^n}{\text{argmin}}\, f(XX^T v) + \frac{\lambda}{2} v^T X X^T v$$

$$\equiv \underset{v \in \mathbb{R}^n}{\text{argmin}}\, f(Kv) + \frac{\lambda}{2} v^T K v.$$

- Which is a kernelized version of the problem.

Representer Theorem

- Using $w = X^T v$, at test time we use

$$\begin{aligned}\hat{y} &= \tilde{X}w\\ &= \tilde{X}X^T v\\ &= \tilde{K}v,\end{aligned}$$

  or that each $\hat{y}^i = \sum_{j=1}^n v_j k(\tilde{x}^i, x^j)$.

- That prediction is a linear combination of kernels is called representer theorem.
  - It holds under more general conditions, including non-smooth f_i like SVMs.

Multiple Kernel Learning

- We can kernelize L2-regularized linear models,

$$\operatorname*{argmin}_{w \in \mathbb{R}^d} f(Xw, y) + \frac{\lambda}{2}\|w\|^2 \Leftrightarrow \operatorname*{argmin}_{v \in \mathbb{R}^n} f(Kv, y) + \frac{\lambda}{2}\|v\|_K^2,$$

  under fairly general conditions.
- What if we have multiple potential kernels and don't know which to use?
  - Obvious approach: cross-validation to choose the best one.
- What if we have multiple potentially-relevant kernels?
  - Multiple kernel learning:

$$\operatorname*{argmin}_{v_1 \in \mathbb{R}^n, v_2 \in \mathbb{R}^n, \dots, v_k \in \mathbb{R}^n} f\left(\sum_{c=1}^{k} K_c v_c, y\right) + \frac{1}{2}\sum_{c=1}^{k} \lambda_c \|v\|_{K_c}.$$

  - Defines a valid kernel and is convex if f is convex (affine function).
  - Group L1-regularization of parameters associated with each kernel.
    - Selects a sparse set of kernels.
  - Hiearchical kernel learning:
    - Use structured sparsity to search through exponential number of kernels.

Large-Scale Kernel Methods

- Obvious drawback of kernel methods: we can't compute/store K for large n.
  - It has $O(n^2)$ elements.

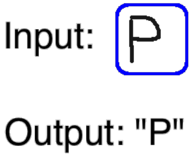
- Standard general approaches:
  1. Kernels with special structure (low bandwidth, low-rank, Toepelitz, Kronecker).
  2. Losses that are sparse in dual (SVMs, support vector regression, 1-class SVM, etc.).
  3. Subsampling methods (Nystrom approximation, subset of regressors).
  4. Explicit feature construction (random kitchen sinks, homogeneous kernel maps).

- If you're interested, I put the slides from last year here:
  `https://www.cs.ubc.ca/~schmidtm/Courses/540-W18/L12.5.pdf`

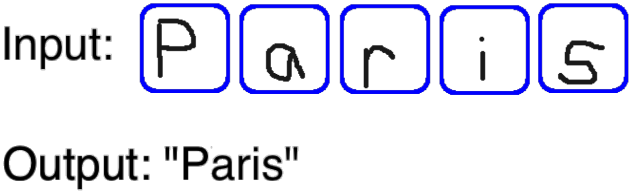
Outline

Motivation: Structured Prediction

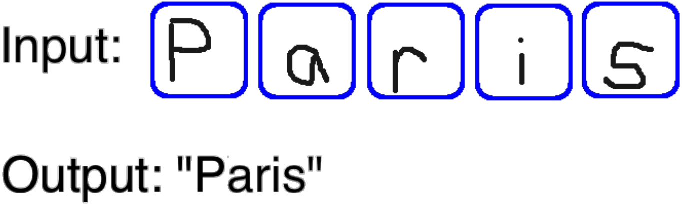
Classical supervised learning focuses on predicting single discrete/continuous label:



Structured prediction allows general objects as labels:



"Classic" ML for Structured Prediction



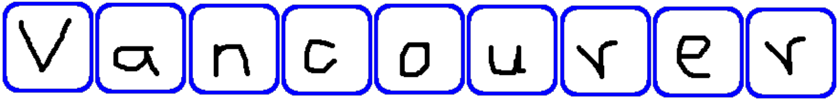
Two ways to formulate as "classic" machine learning:

1. Treat each word as a different class label.
   - Problem: there are too many possible words.
   - You will never recognize new words.
2. Predict each letter individually:
   - Works if you are really good at predicting individual letters.
   - But some tasks don't have a natural decomposition.
   - Ignores dependencies between letters.

Motivation: Structured Prediction

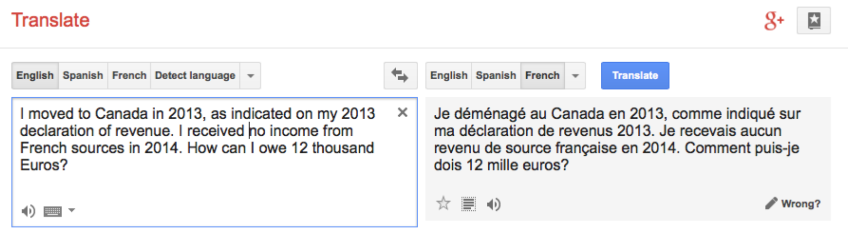
- What letter is this?

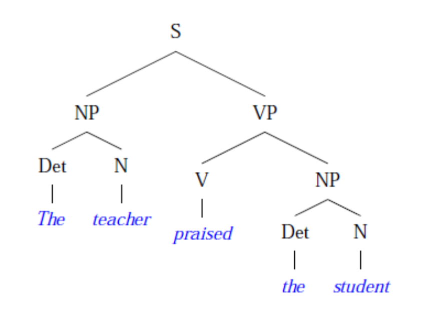
- What are these letters?



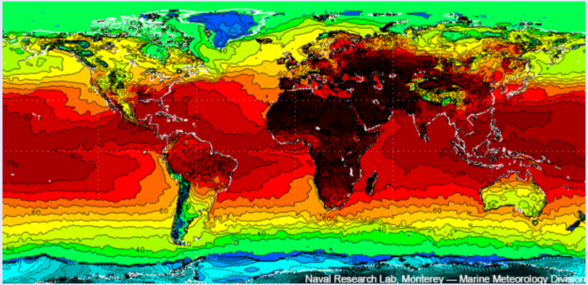
- Predict each letter using "classic" ML and features from neighbouring images?
- This classic appraoch can be good or bad depending on goal:
  - Good if you want to predict individual letters.
  - Bad if goal is to predict entire word.

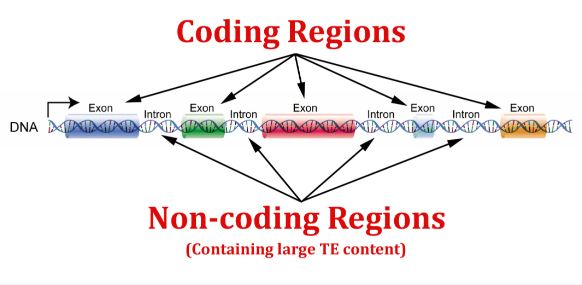
Examples of Structured Prediction



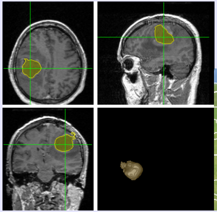


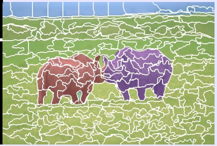
Examples of Structured Prediction

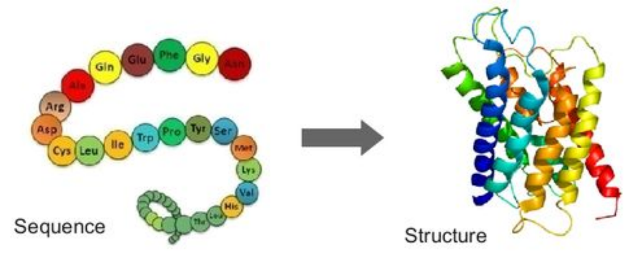


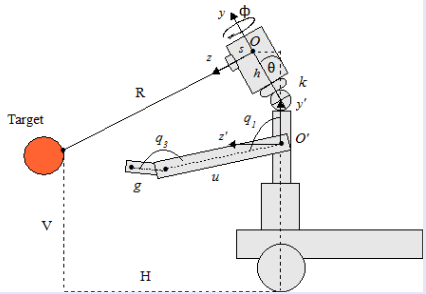


Examples of Structured Prediction



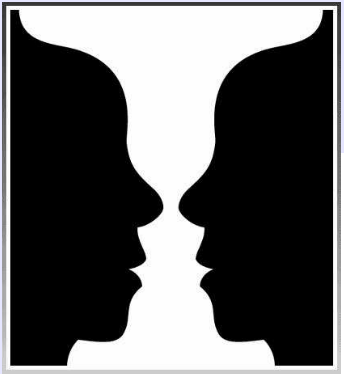


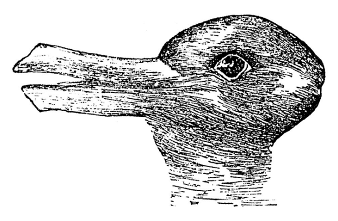




Does the brain do structured prediction?

Gestalt effect: "whole is other than the sum of the parts".





What do you see?
By shifting perspective you might see an old woman or a young woman.

Supervised Learning vs. Structured Prediction

- In 340 we focused a lot on “classic” supervised learning:
  - Model $p(y|x)$ where y is a single discrete/continuous variable.
- In the next few classes we'll focus on density estimation:
  - Model $p(x)$ where x is a vector or general object.
- Structured prediction is the logical combination of these:
  - Model $p(y|x)$ where y is a vector or general object.

3 Classes of Structured Prediction Methods

3 main approaches to structured prediction:

1. Generative models use $p(y|x) \propto p(y,x)$ as in naive Bayes.
   - Turns structured prediction into density estimation.
     - But we'll want to go beyond naive Bayes.
2. Discriminative models directly fit $p(y|x)$ as in logistic regression.
   - View structured prediction as conditional density estimation.
     - Lets you use complicated features x that make the task easier.
3. Discriminant functions just try to map from x to y as in SVMs.
   - Now you don't even need to worry about calibrated probabilities.

Density Estimation

- The next topic we'll focus on is density estimation:

$$X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \quad \tilde{X} = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$

  - What is probability of x^i for a generic feature vector x^i?

- For the training data this is easy:
  - Set $p(x^i)$ to "number of times x^i is in the training data" divided by n.

- We're interested in the probability of test data,
  - What is probability of seeing feature vector $\tilde{x}^i$ for a new example i.

Density Estimation Applications

- Density estimation could be called a "master problem" in machine learning.
  - Solving this problem lets you solve a lot of other problems.
- If you have $p(x^i)$ then:
  - Outliers could be cases where $p(x^i)$ is small.
  - Missing data in x^i can be "filled in" based on $p(x^i)$.
  - Vector quantization can be achieved by assigning shorter code to high $p(x^i)$ values.
  - Association rules can be computed from conditionals $p(x^i_j | x^i_k)$.
- We can also do density estimation on (x^i, y^i) jointly:
  - Supervised learning can be done by conditioning to give $p(y^i | x^i)$.
  - Feature relevance can be analyzed by looking at $p(x^i | y^i)$.

Unsupervised Learning

- Density estimation is an unsupervised learning method.
  - We only have x^i values, but no explicit target labels.
  - You want to do "something" with them.
- Some unsupervised learning tasks from CPSC 340:
  - Clustering: what types of x^i are there?
  - Association rules: which x_j and x_k occur together?
  - Outlier detection: is this a "normal" x^i?
  - Latent-factors: what "parts" are x^i made from?
  - Data visualization: what do the high-dimensional x^i look like?
  - Ranking: which are the most important x^i?
- You can probably address all these if you can do density estimation.

Summary

- Valid kernels are typically constructed from other valid kernels.
- Representer theorem allows kernel trick for L2-regularized linear models.

- Structured prediction is supervised learning with a complicated y^i.
  - 3 flavours are generative models, discriminative models, and discriminant functions.

- Density estimation: unsupervised modelling of probability of feature vectors.

- Next time: everyone's favourite distributions...

Constructing Feature Space (Finite Domain)

- Why is positive semi-definiteness important?
  - With finite domain we can define K over all points.
  - By symmetry of K it has a spectral decomposition

$$K = U^T \Lambda U,$$

    and $K \succeq 0$ means $\lambda_i \geq 0$ and so we have a real diagonal $\Lambda^{\frac{1}{2}}$.
  - Thus we hav $K = U^T \Lambda^{\frac{1}{2}} \Lambda^{\frac{1}{2}} U = (\Lambda^{\frac{1}{2}} U)^T (\Lambda^{\frac{1}{2}} U)$ and we could use

$$Z = \Lambda^{\frac{1}{2}} U, \text{ which means } z_i = \Lambda^{\frac{1}{2}} U_{:,i}.$$

- The above reasoning isn't quite right for continuous domains.
- The more careful generalization is known as "Mercer's theorem".