### Classification

Classification: Given input $x$, predict label $y$

### Structured prediction

Structured prediction: Given input $x$, predict label vector $y$

- Naive solution: Predict each component independently
- Better: Assume correlations between some labels.

### Other ML

- Bayesian inference
- Manifold learning
- Kernel methods
- Graphical models
- Structured prediction
Structured prediction

- Structured prediction: Given input $x$, predict label vector $y$
- Naive solution: Predict each component independently
- Better: Assume correlations between some labels.

Example: scene parsing

Linear classification with joint features

- To deal with structured outputs, it is convenient to re-formulate the classification task using a joint feature representation as follows:
  \[
  f(x) = \arg \max_y w^T \phi(x, y)
  \]
  where $\phi(x, y)$ is a feature vector for a class/input combination.
- Learning: minimize $\frac{1}{N} \sum_i \text{loss}(w) + \lambda \|w\|^2$
- We can treat standard classification in this framework, too.
Logistic regression

- Logistic regression minimizes the “log-loss”:
  \[
  \text{loss}_i (\mathbf{w}) = \log \sum_y \exp (\mathbf{w}^T \phi(x_i, y)) - \mathbf{w}^T \phi(x_i, y')
  \]

- This is just the log-probability over labels given inputs, defined as a softmax.

Support vector machines

- SVMs minimize the “hinge-loss”:
  \[
  \text{loss}_i (\mathbf{w}) = \max_y \left( \mathbf{w}^T \phi(x_i, y) + 1 - \delta_{y, y'} \right) - \mathbf{w}^T \phi(x_i, y')
  \]

Kernels

- Linear models can be turned into non-linear models using kernels.
- Recipe:
  1. Reformulate learning equations to use only inner products between data examples
     \[
     \phi(x_i, y_i)^T \phi(x_j, y_j)
     \]
     and inference equations to use inner products between a test case \(\phi(x, y)\) and training examples:
     \[
     \phi(x_i, y_i)^T \phi(x, y)
     \]
  2. Replace each inner product by a positive definite kernel function
     \[
     k(\phi(x_i, y_i), \phi(x_j, y_j))
     \]
One way to define positive definite kernel is to say that any matrix $K$ with entries $K_{ij} = k(\phi(x_i, y_i), \phi(x_j, y_j))$ will be positive definite (for all possible input pairs $\phi(x_i, y_i), \phi(x_j, y_j)$).

It is guaranteed (Mercer’s theorem) that the kernel function corresponds to an inner product in some vector space.

We don’t care what this space is. But if it is non-linearly related to the inputs, we will have done non-linear feature extraction implicitly.

A convex problem can be solved by alternatively solving its dual, which in this case is:

$$\max_{\alpha} -\frac{1}{2\lambda} \alpha^T K \alpha + \frac{1}{\lambda} \alpha^T K \delta + \sum_i H(\alpha^i)$$

s.t. $\alpha^i_y \geq 0 \ \forall i, y; \ \sum_y \alpha^i_y = 1 \ \forall i$

where $K_{ij} = \phi(x_i, y_i)^T \phi(x_j, y_j)$ contains all inner products between training cases.

We may now replace the inner products by a positive definite kernel function $k(\phi(x_i, y_i), \phi(x_j, y_j))$, because it is guaranteed to correspond to an inner product in some space.

The catch: This yields a batch learning method with complexity quadratic in the number of training cases.

Structured prediction

- Instead of scalar labels $y$, consider label vectors $y$.
- Everything else stays the same:
  $$f(x) = \arg \max_y w^T \phi(x, y)$$

  - hinge loss $\rightarrow$ “structured SVM”
  - log loss $\rightarrow$ “conditional random field”
  - perceptron rule $\rightarrow$ “structured perceptron”
  - But learning and inference can be intractable, because the logsumexp (for logreg) or the argmax (for SVM, perceptron) is over exponentially many $y$.
Feature decompositions

▶ To regain tractability, decompose the features as

\[ \phi(x, y) = \sum_s \phi(x, y_s) \]

where \( s \) indexes a clique in some graph defined over \( y \).

▶ If the graph is a tree, we can tractably compute

\[ f(x) = \arg \max_y \mathbf{w}^T \sum_s \phi(x, y_s) \]

as well as logsumexp’s, using variations of the distributive law:

\[ \boxed{\text{−−> Do the argmax (or logsumexp) using dynamic programming}} \]

Feature decompositions

▶ The same idea will work not just for chains but for any label vectors structured as tree.
▶ The generalization to trees is called belief propagation. It amounts to interpreting the intermediate computations as “messages” that are sent from node to node.
▶ For non-tree structures (eg. MRF), we can still send around messages (pretending that this makes sense), and hope the values converge. Surprisingly they often do. This is called loopy belief propagation.
▶ Many other techniques for approximate inference have been developed in recent years.

Feature decompositions

▶ Consider, for example, the chain structure

\[ \phi(x, y_s) = \phi(x, y_t, y_{t+1}) \]

\[ f(x) = \arg \max_y \mathbf{w}^T \sum_s \phi(x, y_s) = \arg \max_{y_1, \ldots, y_T} \mathbf{w}^T \sum_t \phi(x, y_t, y_{t+1}) = \arg \max_{y_1} \arg \max_{y_2} \cdots \arg \max_{y_T} \mathbf{w}^T \phi(x, y_1, y_2) + \ldots + \arg \max_{y_T} \phi(x, y_{T-1}, y_T) \]

▶ Example: Conditional Random Fields

\[ p(y|x; \mathbf{w}) = \frac{1}{Z(x; \mathbf{w})} \exp(\mathbf{w}^T \phi(x, y)) \]

▶ The clique-wise potentials \( \phi(x, y_s) \) may be complicated non-linear functions, such as neural networks.
▶ Learning in that case combines belief propagation and back-propagation.
▶ This is possible, because these two don’t interfere:
Example: Conditional Random Fields

\[ p(y|x; w) = \frac{1}{Z(x; w)} \exp(w^T \phi(x, y)) \]

- Derivative wrt. to some parameter \( \theta \):

\[
\frac{\partial \log p(y|x)}{\partial \theta} = w^T \frac{\partial \phi(x, y)}{\partial \theta} - w^T \sum_y p(y|x) \frac{\partial \phi(x, y)}{\partial \theta}
\]

- Now think of each \( \phi(x, y_s) \) as a separate neural network that is indexed by \( y_s \)!
- The networks may share weights (which is trivial to implement)

Cost-augmented inference

- "argmax"-models may have an advantage over conditional random fields, because they allows us to insert more complicated cost functions into the inference:

\[
\text{loss}_i(w) = \max_y (w^T \phi(x, y) + \text{cost}(y, y_i)) - w^T \phi(x, y_i)
\]

where \( \text{cost}(y, y_i) \), like the features, has to decompose in some way to be tractable.

Kernels

- For vector \( y \) this is a problem with exponentially many constraints.
- But there are ways to solve it (eg. "cutting plane method").
- Unfortunately, these methods come with quadratic complexity in the number of training case and quadratic complexity in the number of joint clique instantiations.
- So it seems hopeless to make them work on large datasets.
Output representation learning

- A possible alternative to complicated graphical models on the outputs is to use **feature learning** on the outputs themselves.

Feature learning MRFs

- MRFs defined via feature learning have been getting quite popular in vision, because they perform very well in denoising and inpainting tasks.
- For example, *Fields of Experts* (Roth & Black), which are convolutional RBMs.
- The free energy itself is a sum over the individual clique free energies, and it can be written as a convolution.
- Same for the derivative of the free energy wrt. the input image. (What is it ?!?)
- So it is easy to optimize it to perform inference.
Feature learning MRFs

Figure 4: Denoising results. (a) Original noiseless image. (b) Image with additive Gaussian noise ($\sigma = 25$, PSNR = 35.76dB). (c) Denoised image using a Field of Experts, PSNR = 29.70dB. (d) Denoised image using the approach from [20], PSNR = 26.64dB. (e) Denoised image using standard non-linear diffusion, PSNR = 27.14dB.

Figure 6: Inpainting with a Field of Experts. (a) Original image with random loss. (b) Inpainting result from diffusion using the FOD prior. (c) Close-up comparison between (a) (left), (b) (middle), and the results from [7] (right).

Other stuff

- Saliency, attention
- Other modalities
- Geometry (not much learning, but some starting now with complex cells)
- 3d models, shape
- Models of humans, MOCAP