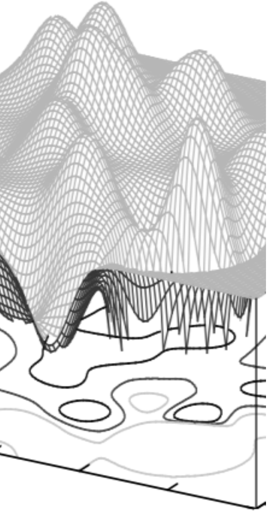


Machine Learning

Classification & Structured Output

Structured output, structured input, discriminative function, joint input-output features, Likelihood Maximization, Logistic regression, binary & multi-class case, conditional random fields

Marc Toussaint
University of Stuttgart
Summer 2019

Structured Output & Structured Input

- regression:

$$\mathbb{R}^d \to \mathbb{R}$$

- structured output:

$$\mathbb{R}^d \to \text{binary class label } \{0, 1\}$$
$$\mathbb{R}^d \to \text{integer class label } \{1, 2, .., M\}$$
$$\mathbb{R}^d \to \text{sequence labelling } y_{1:T}$$
$$\mathbb{R}^d \to \text{image labelling } y_{1:W,1:H}$$
$$\mathbb{R}^d \to \text{graph labelling } y_{1:N}$$

- structured input:

$$\text{relational database} \to \mathbb{R}$$
$$\text{labelled graph/sequence} \to \mathbb{R}$$

The discriminative function

Discriminative Function

- Represent a discrete-valued function $F:\ \mathbb{R}^d \to Y$ via a **discriminative function**

  $$f:\ \mathbb{R}^d \times Y \to \mathbb{R}$$

  such that

  $$\boxed{F:\ x \mapsto \operatorname{argmax}_y f(x,y)}$$

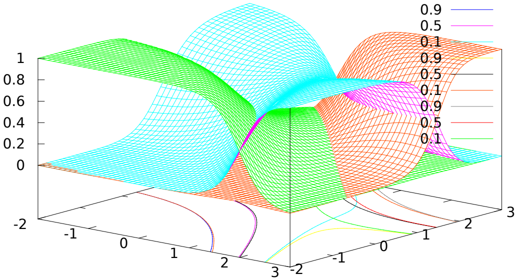
  That is, a discriminative function $f(x,y)$ maps an input x to an output

  $$\hat{y}(x) = \operatorname*{argmax}_y f(x,y)$$

- A discriminative function $f(x,y)$ has high value if y is a correct answer to x; and low value if y is a false answer
- In that way a discriminative function discriminates correct labelling from wrong ones

Example Discriminative Function

- Input: $x \in \mathbb{R}^2$; output $y \in \{1, 2, 3\}$
  displayed are $p(y=1|x)$, $p(y=2|x)$, $p(y=3|x)$



(here already “scaled” to the interval [0,1]... explained later)

- You can think of $f(x, y)$ as M separate functions, one for each class $y \in \{1, .., M\}$. The highest one determines the class prediction $\hat{y}$
- More examples: `plot[-3:3] -x-2,0,x-2` `splot[-3:3][-3:3] -x-y-2,0,x+y-2`

How could we parameterize a discriminative function?

- Linear in features!
  - Same features, different parameters for each output: $f(x,y) = \phi(x)^\top \beta_y$
  - More general input-output features: $f(x,y) = \phi(x,y)^\top \beta$

- Example for joint features: Let $x \in \mathbb{R}$ and $y \in \{1,2,3\}$, might be

$$\phi(x,y) = \begin{pmatrix} 1 & [y=1] \\ x & [y=1] \\ x^2 & [y=1] \\ 1 & [y=2] \\ x & [y=2] \\ x^2 & [y=2] \\ 1 & [y=3] \\ x & [y=3] \\ x^2 & [y=3] \end{pmatrix}, \quad \text{which is equivalent to } f(x,y) = \begin{pmatrix} 1 \\ x \\ x^2 \end{pmatrix}^\top \beta_y$$

- Example when both $x, y \in \{0,1\}$ are discrete:

$$\phi(x,y) = \begin{pmatrix} 1 \\ [x=0][y=0] \\ [x=0][y=1] \\ [x=1][y=0] \\ [x=1][y=1] \end{pmatrix}$$

Notes on features

- Features “connect” input and output. Each $\phi_j(x, y)$ allows f to capture a certain dependence between x and y
- If both x and y are discrete, a feature $\phi_j(x, y)$ is typically a joint indicator function (logical function), indicating a certain “event”
- Each weight β_j mirrors how important/frequent/infrequent a certain dependence described by $\phi_j(x, y)$ is
- $-f(x, y)$ is also called energy, and the is also called **energy-based modelling**, esp. in neural modelling

Loss functions for classification

What is a good objective to train a classifier?

- **Accuracy, Precision & Recall:**
  For data size n, *false positives* (FP), *true positives* (TP), we define:

  - accuracy = $\frac{\text{TP+TN}}{n}$
  - precision = $\frac{\text{TP}}{\text{TP+FP}}$ (TP+FP = classifier positives)
  - recall (TP-rate) = $\frac{\text{TP}}{\text{TP+FN}}$ (TP+FN = data positives)
  - FP-rate = $\frac{\text{FP}}{\text{FP+TN}}$ (FP+TN = data negatives)

- Such metrics be our actual objective. But they are not differentiable. For the purpose of ML, we need to define a "proxy" objective that is nice to optimize.

- Bad idea: Squared error regression

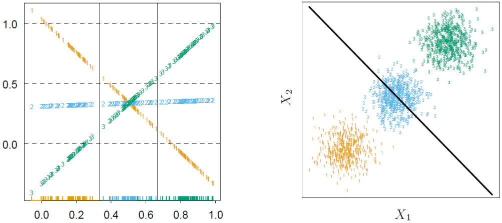
Bad idea: Squared error regression of class indicators

- Train $f(x, y)$ to be the indicator function for class y
  that is, $\forall y$: train $f(x, y)$ on the regression data $D = \{(x_i, I(y{=}y_i))\}_{i=1}^n$:
  train $f(x, 1)$ on value 1 for all x_i with $y_i = 1$ and on 0 otherwise
  train $f(x, 2)$ on value 1 for all x_i with $y_i = 2$ and on 0 otherwise
  train $f(x, 3)$ on value 1 for all x_i with $y_i = 3$ and on 0 otherwise
  ...

Bad idea: Squared error regression of class indicators

- Train $f(x,y)$ to be the indicator function for class y
  that is, $\forall y$: train $f(x,y)$ on the regression data $D = \{(x_i, I(y{=}y_i))\}_{i=1}^n$:
    train $f(x,1)$ on value 1 for all x_i with $y_i = 1$ and on 0 otherwise
    train $f(x,2)$ on value 1 for all x_i with $y_i = 2$ and on 0 otherwise
    train $f(x,3)$ on value 1 for all x_i with $y_i = 3$ and on 0 otherwise
  ...

- This typically fails: (see also Hastie 4.2)



Although the optimal separating boundaries are linear and linear discriminating functions could represent them, the linear functions trained on class indicators fail to discriminate.

$\rightarrow$ *squared error regression on class indicators is the "wrong objective"*

Log-Likelihood

- The discriminative function $f(y, x)$ not only defines the class prediction $F(x)$; we can additionally also define probabilities,

$$p(y \mid x) = \frac{e^{f(x,y)}}{\sum_{y'} e^{f(x,y')}}$$

- Maximizing Log-Likelihood: (minimize neg-log-likelihood, nll)

$$L^{\mathsf{nll}}(\beta) = -\sum_{i=1}^n \log p(y_i \mid x_i)$$

Cross Entropy

- This is the same as log-likelihood for categorical data, just a notational trick, really.
- The categorical data $y_i \in \{1, .., M\}$ are class labels. But assume they are encoded in a **one-hot-vector**

$$\hat{y}_i = \boldsymbol{e}_{y_i} = (0, .., 0, 1, 0, ..., 0) \;, \quad \hat{y}_{iz} = [y_i = z]$$

  Then we can write the neg-log-likelihood as

$$L^{\text{nll}}(\beta) = -\sum_{i=1}^{n} \sum_{z=1}^{M} \hat{y}_{iz} \log p(z \,|\, x_i) = \sum_{i=1}^{n} H(\hat{y}_i, \; p(\cdot, x_i))$$

  where $H(p, q) = -\sum_z p(z) \log q(z)$ is the so-called cross entropy between two normalized multinomial distributions p and q.

- As a side note, the cross entropy measure would also work if the target $\hat{y}_i$ are probabilities instead of one-hot-vectors.

Hinge loss

- For a data point (x, y^*), the **one-vs-all hinge loss** "wants" that $f(y^*, x)$ is larger than any other $f(y, x)$, $y \neq y^*$, by a margin of 1.
  In other terms, it penalizes when $f(y^*, x) < f(y, x) + 1$, $y \neq y^*$.
- It penalizes linearly, therefore the one-vs-all hinge loss is defined as

$$L^{\text{hinge}}(f) = \sum_{y \neq y^*} [1 - (f(y^*, x) - f(y, x))]_+$$

- This is related to **Support Vector Machines** (only data points inside the margin induce an error and gradient), and also to the **Perceptron Algorithm**

Logistic regression

Logistic regression: Multi-class case

- Data $D = \{(x_i, y_i)\}_{i=1}^n$ with $x_i \in \mathbb{R}^d$ and $y_i \in \{1, .., M\}$
- We choose $f(x, y) = \phi(x)^\top \beta_y$ with separate parameters β_y for each y
- Conditional class probabilties

$$p(y \,|\, x) = \frac{e^{f(x,y)}}{\sum_{y'} e^{f(x,y')}} \qquad \leftrightarrow \qquad f(x, y) - f(x, z) = \log \frac{p(y \,|\, x)}{p(z \,|\, x)}$$

(the discriminative functions model "*log-ratios*")

- Given data $D = \{(x_i, y_i)\}_{i=1}^n$, we minimize the regularized neg-log-likelihood

$$\boxed{L^{\text{logistic}}(\beta) = -\sum_{i=1}^n \log p(y_i \,|\, x_i) + \lambda \|\beta\|^2}$$

Written as cross entropy (with one-hot encoding $\hat{y}_{iz} = [y_i = z]$):

$$L^{\text{logistic}}(\beta) = -\sum_{i=1}^n \sum_{z=1}^M [y_i = z] \log p(z \,|\, x_i) + \lambda \|\beta\|^2$$

Optimal parameters β

- Gradient:

$$\frac{\partial L^{\text{logistic}}(\beta)}{\partial \beta_c}^\top = \sum_{i=1}^n (p_{ic} - y_{ic})\phi(x_i) + 2\lambda I \beta_c = X^\top (p_c - y_c) + 2\lambda I \beta_c$$

  where $p_{ic} = p(y{=}c \,|\, x_i)$

  which is non-linear in $\beta \;\;\Rightarrow\;\; \partial_\beta L = 0$ does not have an analytic solution

- Hessian:

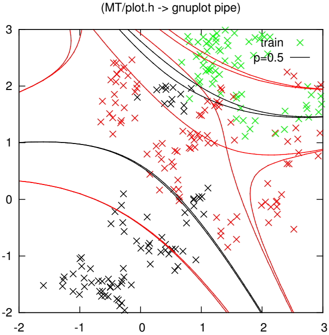
$$H = \frac{\partial^2 L^{\text{logistic}}(\beta)}{\partial \beta_c \partial \beta_d} = X^\top W_{cd} X + 2[c = d]\;\lambda I$$

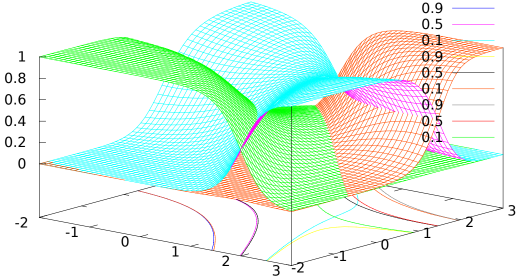
  where W_{cd} is diagonal with $W_{cd,ii} = p_{ic}([c = d] - p_{id})$

- Newton algorithm: iterate

$$\boxed{\beta \leftarrow \beta - H^{\text{-}1}\; \frac{\partial L^{\text{logistic}}(\beta)}{\partial \beta}^\top}$$

polynomial (quadratic) ridge 3-class logistic regression:





`./x.exe -mode 3 -d 2 -n 200 -modelFeatureType 3 -lambda 1e+1`

- Note, if we have M discriminative functions $f(x, y)$, w.l.o.g., we can always choose one of them to be constantly zero. E.g.,

  $$f(x, M) \equiv 0 \text{ or } \beta_M \equiv 0$$

  The other functions then have to be greater/less relative to this baseline.

- This is usually not done in the multi-class case, *but almost always in the binary case*.

Logistic regression: Binary case

- In the binary case, we have "two functions" $f(x,0)$ and $f(x,1)$. **W.l.o.g. we may fix $f(x,0)=0$ to zero.** Therefore we choose features

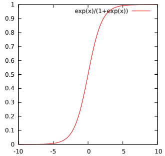
$$\phi(x,y) = \phi(x)\,[y=1]$$

  with arbitrary input features $\phi(x) \in \mathbb{R}^k$

- We have

$$f(x,1) = \phi(x)^\top \beta\ , \quad \hat y = \operatorname*{argmax}_y f(x,y) = \begin{cases} 0 & \text{else} \\ 1 & \text{if } \phi(x)^\top\beta > 0 \end{cases}$$

- and conditional class probabilities



$$p(1\,|\,x) = \frac{e^{f(x,1)}}{e^{f(x,0)} + e^{f(x,1)}} = \sigma(f(x,1))$$

  with the *logistic sigmoid function* $\sigma(z) = \frac{e^z}{1+e^z} = \frac{1}{e^{-z}+1}$.

- Given data $D = \{(x_i, y_i)\}_{i=1}^n$, we minimize the regularized neg-log-likelihood

$$L^{\text{logistic}}(\beta) = -\textstyle\sum_{i=1}^n \log p(y_i\,|\,x_i) + \lambda\|\beta\|^2$$

$$= -\textstyle\sum_{i=1}^n \Big[y_i \log p(1\,|\,x_i) + (1-y_i)\log[1-p(1\,|\,x_i)] \Big] + \lambda\|\beta\|^2$$

Optimal parameters β

- Gradient (see exercises):

$$\frac{\partial L^{\text{logistic}}(\beta)}{\partial \beta}^\top = \sum_{i=1}^n (p_i - y_i)\phi(x_i) + 2\lambda I \beta = X^\top (p - y) + 2\lambda I \beta$$

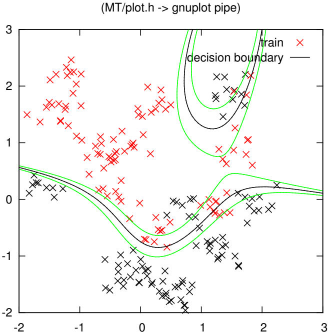
$$\text{where} \quad p_i := p(y{=}1 \,|\, x_i) \;, \quad X = \begin{pmatrix} \phi(x_1)^\top \\ \vdots \\ \phi(x_n)^\top \end{pmatrix} \in \mathbb{R}^{n \times k}$$

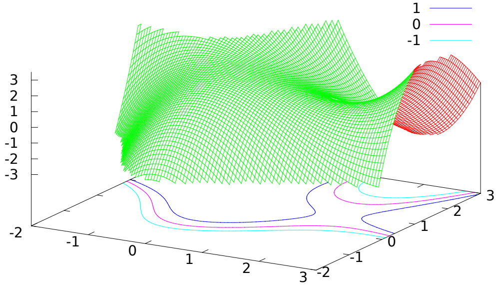
- Hessian $H = \frac{\partial^2 L^{\text{logistic}}(\beta)}{\partial \beta^2} = X^\top W X + 2\lambda I$
  $W = \text{diag}(p \circ (1 - p))$, that is, diagonal with $W_{ii} = p_i(1 - p_i)$

- Newton algorithm: iterate

$$\boxed{\beta \leftarrow \beta - H^{-1} \, \frac{\partial L^{\text{logistic}}(\beta)}{\partial \beta}^\top}$$

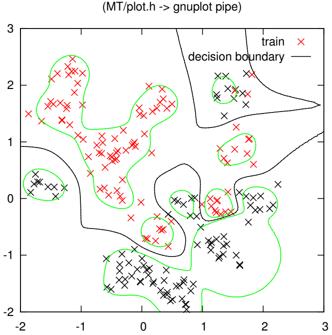
polynomial (cubic) ridge logistic regression:

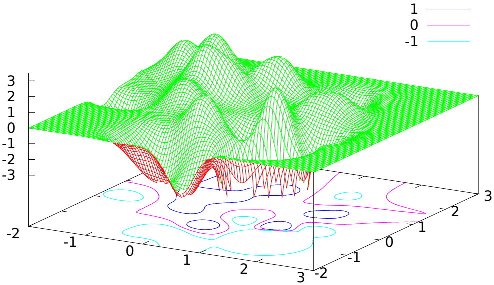




`./x.exe -mode 2 -d 2 -n 200 -modelFeatureType 3 -lambda 1e+0`

RBF ridge logistic regression:






```
./x.exe -mode 2 -d 2 -n 200 -modelFeatureType 4 -lambda 1e+0 -rbfBias 0
-rbfWidth .2
```

Recap

	ridge regression	logistic regression
REPRESENTATION	$f(x) = \phi(x)^\top \beta$	$f(x,y) = \phi(x,y)^\top \beta$
OBJECTIVE	$L^{\text{ls}}(\beta) =$ $\sum_{i=1}^n (y_i - f(x_i))^2 + \lambda \|\beta\|_I^2$	$L^{\text{logistic}}(\beta) =$ $- \sum_{i=1}^n \log p(y_i \,\vert\, x_i) + \lambda \|\beta\|_I^2$ $p(y \,\vert\, x) \propto e^{f(x,y)}$
SOLVER	$\hat\beta^{\text{ridge}} = (X^\top X + \lambda I)^{\text{-1}} X^\top y$	binary case: $\beta \leftarrow \beta - (X^\top W X + 2\lambda I)^{\text{-1}}$ $(X^\top (p - y) + 2\lambda I \beta)$

Conditional Random Fields

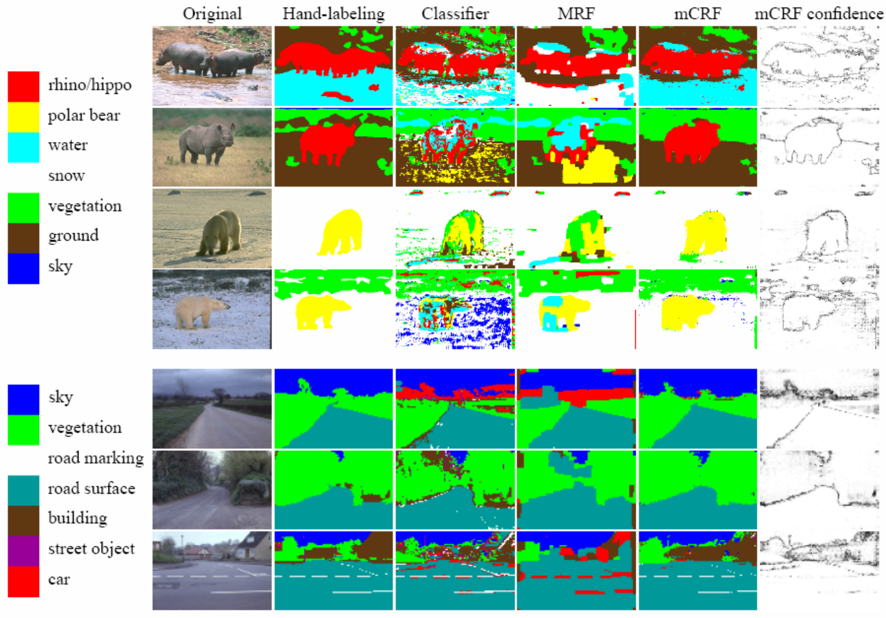
Examples for Structured Output

- Text tagging
  X = sentence
  Y = tagging of each word
  `http://sourceforge.net/projects/crftagger`

- Image segmentation
  X = image
  Y = labelling of each pixel
  `http://scholar.google.com/scholar?cluster=13447702299042713582`

- Depth estimation
  X = single image
  Y = depth map
  `http://make3d.cs.cornell.edu/`

CRFs in image processing



CRFs in image processing

- Google "conditional random field image"
  - Multiscale Conditional Random Fields for Image Labeling (CVPR 2004)
  - Scale-Invariant Contour Completion Using Conditional Random Fields (ICCV 2005)
  - Conditional Random Fields for Object Recognition (NIPS 2004)
  - Image Modeling using Tree Structured Conditional Random Fields (IJCAI 2007)
  - A Conditional Random Field Model for Video Super-resolution (ICPR 2006)

Conditional Random Fields (CRFs)

- CRFs are a generalization of logistic binary and multi-class classification
- The output y may be an arbitrary (usually discrete) thing (e.g., sequence/image/graph-labelling)
- Hopefully we can maximize efficiently

$$\underset{y}{\operatorname{argmax}} f(x, y)$$

  over the output!
  $\to f(x, y)$ should be *structured* in y so this optimization is efficient.

- The name CRF describes that $p(y|x) \propto e^{f(x,y)}$ defines a probability distribution (a.k.a. random field) over the output y conditional to the input x. The word "field" usually means that this distribution is structured (a graphical model; see later part of lecture).

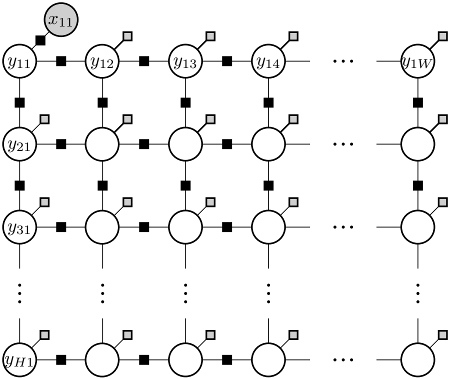
CRFs: the structure is in the features

- Assume $y = (y_1, .., y_l)$ is a tuple of individual (local) discrete labels
- We can assume that $f(x, y)$ is linear in features

$$f(x, y) = \sum_{j=1}^{k} \phi_j(x, y_{\partial j})\beta_j = \phi(x, y)^\top \beta$$

where **each feature** $\phi_j(x, y_{\partial j})$ **depends only on a subset** $y_{\partial j}$ **of labels**. $\phi_j(x, y_{\partial j})$ effectively couples the labels $y_{\partial j}$. Then $e^{f(x,y)}$ is a **factor graph**.

Example: pair-wise coupled pixel labels



- Each black box corresponds to features $\phi_j(y_{\partial j})$ which couple neighboring pixel labels $y_{\partial j}$
- Each gray box corresponds to features $\phi_j(x_j, y_j)$ which couple a local pixel observation x_j with a pixel label y_j

CRFs: Core equations

$$f(x,y) = \phi(x,y)^\top \beta$$

$$p(y|x) = \frac{e^{f(x,y)}}{\sum_{y'} e^{f(x,y')}} = e^{f(x,y)-Z(x,\beta)}$$

$$Z(x,\beta) = \log \sum_{y'} e^{f(x,y')} \quad \text{(log partition function)}$$

$$L(\beta) = -\sum_i \log p(y_i|x_i) = -\sum_i [f(x_i,y_i) - Z(x_i,\beta)]$$

$$\nabla Z(x,\beta) = \sum_y p(y|x)\ \nabla f(x,y)$$

$$\nabla^2 Z(x,\beta) = \sum_y p(y|x)\ \nabla f(x,y)\ \nabla f(x,y)^\top - \nabla Z\ \nabla Z^\top$$

- This gives the neg-log-likelihood $L(\beta)$, its gradient and Hessian

Training CRFs

- Maximize conditional likelihood
  But Hessian is typically too large (Images: $\sim$10 000 pixels, $\sim$50 000 features)
  If $f(x,y)$ has a chain structure over y, the Hessian is usually banded $\rightarrow$ computation time linear in chain length

  Alternative: Efficient gradient method, e.g.:
  Vishwanathan et al.: Accelerated Training of Conditional Random Fields with Stochastic Gradient Methods

- Other loss variants, e.g., hinge loss as with Support Vector Machines
  ("Structured output SVMs")

- Perceptron algorithm: Minimizes hinge loss using a gradient method