Tree Distributions

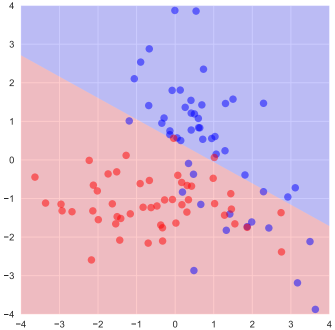
Nick Landolfi and Sanjay Lall
Stanford University

Outline

Overview

Motivation: classification

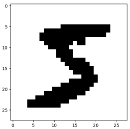
- we have a *data set* of *records* $u^1, \ldots, u^n \in \mathcal{U}$ and $v^1, \ldots, v^n \in \mathcal{V}$ with $\mathcal{V}$ a finite set of *classes*
- we want to build a *classifier* $G : \mathcal{U} \to \mathcal{V}$ and use it to classify a new *independent variable* u as $G(u)$
- for example, $\mathcal{U} = \mathbf{R}^2$ and $\mathcal{V} = \{0, 1\}$



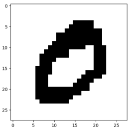
  - the point u^k is colored red if $v^k = 0$ and blue if $v^k = 1$
  - the region $\left\{u \in \mathbf{R}^2 \;\middle|\; G(u) = 0\right\}$ is shaded red and $\left\{u \in \mathbf{R}^2 \;\middle|\; G(u) = 1\right\}$ is shaded blue

Our setting

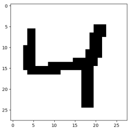
- we consider independent variables in a *large discrete set*
  - $\mathcal{U} = \mathcal{S}^d$ where $\mathcal{S}$ is a finite set; $d > 100$, so $\mathcal{S}^d$ is large
  - in particular, $\mathcal{U}$ is not $\mathbf{R}^2$ as on the previous slide
- for example, $\mathcal{U} = \{0, 1\}^{784}$ and $\mathcal{V} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
  - u represents a hand-written digit ($784 = 28 \times 28$ pixels); v is the Arabic digit depicted by u



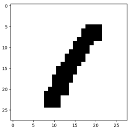
(a) $v^1 = 5$



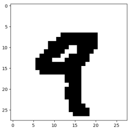
(b) $v^2 = 0$



(c) $v^3 = 4$



(d) $v^4 = 1$



(e) $v^5 = 9$

- one approach is to produce a distribution over $\mathcal{U}$ for each class; called *generative* modeling
  - for a new u, we define $G(u)$ to be a class with *maximum likelihood*

Overview

- so we want to *estimate and store a distribution* over a large discrete space $\mathcal{S}^d$
  - for example, $\mathcal{S} = \{0, 1\}$ and $d = 784$ with $\mathcal{S}^d$ representing 28×28 binary images
- but estimating and storing a distribution over so many outcomes is *infeasible*
  - for a distribution on $\{0, 1\}^{784}$ we need $2^{784} - 1$ parameters to represent the distribution
- so we do not look at all distributions, because that space is too large, we *consider a subset*
  - roughly, only consider distributions which are a product of so-called *second-order* distributions
- we will see an *efficient algorithm* for estimating such distributions
  - original work by Chow and Liu in 1968

Notation

Notation: probability

- $p : \mathcal{U} \to \mathbf{R}$ is a *distribution* on $\mathcal{U} = \mathcal{S}^d$ with $\mathcal{S}$ finite; as usual $p \geq 0$ and $\sum_{u \in \mathcal{U}} p(u) = 1$
- p_i is a distribution on $\mathcal{S}$, called the *ith marginal* distribution, for $i = 1, \ldots, d$
  - defined by $p_i(a) = \displaystyle\sum_{u_i = a} p(u)$
- $p_{i|j}$ is a conditional distribution, called the *i, jth conditional* distribution, for $i, j = 1 \ldots, d$ and $i \neq j$
  - first, we define the *second-order i, jth marginal* distribution p_{ij} on $\mathcal{S}^2$

$$p_{ij}(a, b) = \sum_{u_i = a, u_j = b} p(u)$$

  - then we define $p_{i|j}$ by $p_{i|j}(a, b) p_j(b) = p_{ij}(a, b)$ for all $a, b \in \mathcal{S}$
  - often we will drop the arguments and write $p_{ij} = p_{i|j} p_j$
  - we will use similar notation for conditioning on multiple variables: for example, $p_{i|jkl}$
- roughly speaking, we will approximate a distribution p using terms like p_i and $p_{i|j}$

Notation: Kullback-Leibler divergence

- we want a criterion to judge how well a distribution p approximates a given distribution q
- we will use the *Kullback-Leibler divergence*, defined by

$$d_{kl}(q, p) = H(q, p) - H(q)$$

  - where $H(q) = -\sum_a q(a) \log q(a)$ is called the *entropy* of q
  - and $H(q, p) = -\sum_a q(a) \log p(a)$ is called the *cross entropy* of p relative to q
- we interpret d_{kl} as a measure of the difference between two distributions
  - $d_{kl}(q, p) \geq 0$ for all distributions q and p and $d_{kl}(q, q) = 0$
  - if we want to find a distribution p to

$$\text{minimize} \quad d_{kl}(q, p)$$

    then $p = q$ is a solution; later we will constrain p
  - d_{kl} is not symmetric and so not a metric, though we do not mind

Notation: empirical distribution

- the distribution we will approximate is the natural one associated with data
- we are given n *records* $u^1, \dots, u^n$ with $u^k \in \mathcal{U}$ a finite set
- the *empirical distribution* of $u^1, \dots, u^n$ is the distribution q on $\mathcal{U}$ defined by

$$q(u) = \frac{1}{n}\left|\left\{u^k \;\middle|\; u^k = u\right\}\right|$$

  - $q(u)$ is the proportion of records which are u
- the empirical distribution is a useful summary of data, but unwieldy, so we approximate it

Notation: mutual information graph

- a solution to our approximation will be characterized by mutual informations of the empirical distribution
- the *mutual information* of p_{ij} is $d_{kl}(p_{ij}, p_i p_j)$
  - we denote the symmetric *matrix of mutual informations* of p by $I(p)$, and define it by

$$I(p)_{ij} = d_{kl}(p_{ij}, p_i p_j)$$

- the *mutual information graph* of p is a weighted complete undirected graph on $\{1, \ldots, d\}$
  - edge $\{i, j\}$ is weighted by $I(p)_{ij}$
- roughly speaking, good approximations will model interactions between vertices with heavy edges

Trees and Distributions

Rooted trees

- we use trees to discuss factoring a discrete probability distribution
  - we will use such distributions to approximate, since they require fewer parameters
- a *tree* T is an undirected acyclic connected (finite) graph
  - there is a unique path between any two vertices
- we *root* a tree by selecting a vertex and orienting all edges away from it
  - and so obtain a *directed* tree
  - we call the distinguished vertex the *root*
  - each vertex (except the root) has only one parent

Rooted trees: example

- consider tree $T = (\{1,2,3,4,5,6\}, \{\{1,2\},\{2,3\},\{2,4\},\{4,5\},\{4,6\}\})$

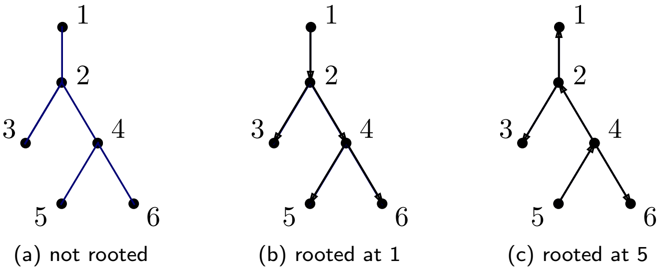
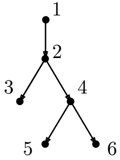


Figure 2: A tree and two possible roots

- in a rooted tree, each vertex except the root has *one parent*
  - we write $\mathsf{pa}_j = i$ to mean that the parent of vertex j is vertex i
  - in panel (b), $\mathsf{pa}_2 = 1, \mathsf{pa}_3 = 2, \mathsf{pa}_4 = 2, \mathsf{pa}_5 = 4$, and $\mathsf{pa}_6 = 4$

Tree-structured probability: example

- consider the same tree $T = (\{1,2,3,4,5,6\}, \{\{1,2\},\{2,3\},\{2,4\},\{4,5\},\{4,6\}\})$, rooted at vertex 1



- if p is a distribution on $\mathcal{S}^6$, then by *chain rule* p always satisfies

$$p = p_{6|1,2,3,4,5}\, p_{5|1,2,3,4}\, p_{4|1,2,3}\, p_{3|1,2}\, p_{2|1}\, p_1$$

- we say p *factors* according to the tree T rooted at vertex 1 if p satisfies

$$p = p_{6|4}\, p_{5|4}\, p_{4|2}\, p_{3|2}\, p_{2|1}\, p_1$$

  - so $p_{6|1,2,3,4,5} = p_{6|4}$ (the conditional distribution does not depend on u_1, u_2, u_3 or u_5)
  - and similarly for $p_{5|4}$, $p_{4|2}$ and $p_{3|2}$

Tree-structured probability: rooted definition

- **Definition**: Let T be a tree on $\{1, \ldots, d\}$. A distribution p on $\mathcal{S}^d$ *factors* according to T rooted at vertex i if

$$p = p_i \prod_{j \neq i} p_{j|\mathsf{pa}_j}$$

  - reminder that this statement is for all $u \in \mathcal{U}$ but drops arguments
  - we call p_i and $p_{j|\mathsf{pa}_j}$ for $j \neq i$ the *factors* of p
  - the distribution p is a product of d factors
- this definition says how a distribution factors according to a *rooted* tree

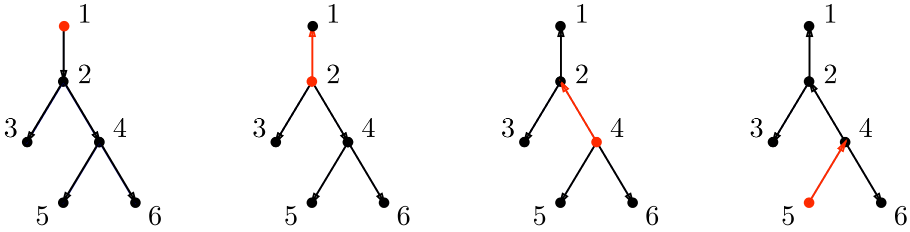
Tree-structured probability: defining theorem

- **Theorem**: Let T be a tree on $\{1, \ldots, d\}$ and let p be a distribution on $\mathcal{S}^d$. If p factors according to T rooted at some vertex, then p factors according to T rooted at any vertex in $\{1, \ldots, d\}$.
  - in other words: if p factors according to one choice of root, it factors according to all choices

- **Definition**: A distribution p on $\mathcal{S}^d$ *factors* according to a tree T on $\{1, \ldots, d\}$ if it factors according to T rooted at any vertex.

Tree-structured probability: defining theorem intuition

- we can successively exchange a root with one of its children to root the tree at the child



(a) rooted at 1 (b) rooted at 2 (c) rooted at 4 (d) rooted at 5

Figure 3: Moving from rooted at 1 to rooted at 5

- the root vertex is red
- in (b), (c) and (d), the only edge differing from (a), (b) and (c), respectively, is red

Tree-structured probability: proof of defining theorem

- roughly, the theorem says
  - p factors according to one possible root if and only if it factors according to every possible root
- proof of the theorem is repeated application of following lemma
- **Lemma**: Let T be a tree on $\{1,\dots,d\}$. Let distribution p on $\mathcal{S}^d$ factor according to T rooted at vertex i. If $j \in \{1,\dots,d\}$ with $\mathrm{pa}_j = i$, then p factors according to T rooted at vertex j.
  - the assumption on p means $p = p_i \prod_{k \neq i} p_{k|\mathrm{pa}_k} = p_i p_{j|i} \prod_{k \neq i,j} p_{k|\mathrm{pa}_k}$
  - since p is a distribution, $p_{j|i} p_i = p_{ij} = p_{j|i} p_j$
  - so we conclude $p = p_j p_{j|i} \prod_{k \neq i,j} p_{k|\mathrm{pa}_k}$
  - which means p factors according to T rooted at j

Tree-structured probability: existence and uniqueness

- a distribution p *need not factor according to a tree*
  - for example, consider a distribution p on $\{0,1\}^3$ with $p(1,1,1) = 4/11$ and $p(u_1,u_2,u_3) = 1/11$ otherwise
  - there does not exist a tree according to which p factors, requires checking cases (symmetry reduces number)
  - compare with: p always factors according to chain rule
- a distribution p *may factor according to multiple trees*
  - for example, consider a distribution p on $\{0,1\}^3$ with $p = p_1 p_2 p_3$
  - then p factors according to every tree on $\{1,2,3\}$
- we conclude that there is not a one-to-one correspondence between trees and distributions
- rather, *trees specify subsets* of distributions

Tree-structured probability: why

- these distributions *can be stored* feasibly in computer memory
  - linear in d rather than exponential in d; $2d$ vs. 2^d for the case $\mathcal{S} = \{0, 1\}$
- we will see, they *can be estimated* efficiently
  - algorithm polynomial in dimension d and size of data set n
- broadly speaking, they are useful *baseline* probabilistic models
- roughly speaking, they are specified by few parameters, which reduces overfitting
- and, also roughly speaking, they may still *capture important dependencies*

Approximation Problem & Solution

Relative entropy approximation

- we have a distribution q on $\mathcal{S}^d$
- we want to find a distribution p on $\mathcal{S}^d$ and tree T on $\{1, \dots, d\}$ to

$$\begin{array}{rl} \text{minimize} & d_{kl}(q, p) \\ \text{subject to} & p \text{ factors according to } T \end{array}$$

  - called the *Chow-Liu problem* to approximate q
- we refer to a solution pair as a *Chow-Liu distribution* and a *Chow-Liu tree* of q
  - a Chow-Liu tree always exists, but need not be unique
- we will solve by finding best parameters for a fixed tree, then finding best tree

Relative entropy approximation: maximum likelihood interpretation

- we have data set $u^1, \ldots, u^n$ with empirical distribution q
- the Chow-Liu problem to approximate q is equivalent to minimizing average negative log likelihood, since

$$
\begin{aligned}
d_{kl}(q,p) &= H(q,p) - H(q) \\
&= -\sum_{u \in \mathcal{U}} \left(q(u) \log p(u)\right) - H(q) \\
&= \underbrace{-\frac{1}{n} \sum_{k=1}^{n} \log p(u^k)}_{\textit{avg. neg. log likelihood}} - H(q)
\end{aligned}
$$

  - and $H(q)$ does not depend on p or T
- in this case, we refer to a Chow-Liu tree T as a *maximum likelihood tree*

Approximation: first theorem

- first, we will see how to select the probability parameters for a given tree
  - later we will see how to select the tree

- **Theorem 1**: Let q be a distribution on $\mathcal{S}^d$. Let T be a tree on $\{1, \ldots, d\}$. Let $\mathrm{pa}_{(\cdot)}$ be defined by T rooted at vertex i. Then the distribution p on $\mathcal{S}^d$ defined by

$$p = q_i \prod_{j \neq i} q_{j \mid \mathbf{pa}_j}$$

  achieves minimum Kullback-Leibler divergence to q among all distributions which factor according to T.

Approximation: proof of Theorem 1

- $i = 1, \ldots, d$ is an arbitrary vertex and p factors according to T rooted at i
- we express the cross entropy of p relative to q

$$
\begin{aligned}
H(q,p) &= -\sum_{u \in \mathcal{U}} q(u) \log p(u) \\
&= -\sum_{u \in \mathcal{U}} q(u) \left(\log p_i(u_i) + \sum_{j \neq i} \log p_{j|\mathsf{pa}_j}(u_j, u_{\mathsf{pa}_j}) \right) \\
&= H(q_i, p_i) + \sum_{j \neq i} \sum_{b \in S} q_{\mathsf{pa}_j}(b) H(q_{j|\mathsf{pa}_j}(\cdot, b), p_{j|\mathsf{pa}_j}(\cdot, b))
\end{aligned}
$$

- this problem *separates across dimension* d
  - one problem to find p_i; solution is $p_i = q_i$
  - $d-1$ problems to find $p_{j|\mathsf{pa}_j}$ for $j \neq i$; solutions are $p_{j|\mathsf{pa}_j} = q_{j|\mathsf{pa}_j}$

Approximation: second theorem

- this theorem will tell us how to select the tree structure
  - recall that the mutual information graph is undirected on $\{1, \ldots, d\}$ and edge $\{i, j\}$ has weight $I(q)_{ij}$

- **Theorem 2**: Let q be a distribution on $\mathcal{S}^d$. A tree T on $\{1, \ldots, d\}$ is a Chow-Liu tree of q if and only if T is a maximum spanning tree of the mutual information graph of q.

Approximation: proof of Theorem 2 (1/2)

- first theorem tells us the optimal choice of p that factors according to a tree T, we write it p_T^*
- recall that $d_{kl}(q,p) = H(q,p) - H(q)$
  - $H(q)$ does not depend on p, so we focus on the cross entropy term
- we will see that we can express the cross entropy

$$H(q, p_T^*) = \sum_{i=1}^{d} H(q_i) - \sum_{\{i,j\}\in T} I(q)_{ij}$$

  - notation $\{i,j\} \in T$ means $\{i,j\}$ is an edge of T
  - for each $i = 1, \ldots, d$, $H(q_i)$ does not depend on T
- the minimize in the Chow-Liu problem results in a maximization over the second sum

Approximation: proof of Theorem 2 (2/2)

- we express the cross entropy of p_T^* relative to q as

$$
\begin{aligned}
H(q, p_T^*) &= H(q_1) - \sum_{j \neq 1} \sum_{u \in \mathcal{U}} q(u) \log q_{j|\mathsf{pa}_j}(u_j, u_{\mathsf{pa}_j}) \\
&= H(q_1) - \sum_{j \neq 1} \sum_{u \in \mathcal{U}} q(u) \left(\log q_{j,\mathsf{pa}_j}(u_j, u_{\mathsf{pa}_j}) - \log q_{\mathsf{pa}_j}(u_{\mathsf{pa}_j})\right) \\
&= H(q_1) - \sum_{j \neq 1} \sum_{u \in \mathcal{U}} q(u) \left(\log q_{j,\mathsf{pa}_j}(u_j, u_{\mathsf{pa}_j}) - \log q_{\mathsf{pa}_j}(u_{\mathsf{pa}_j}) - \log q_j(u_j) + \log q_j(u_j)\right) \\
&= \sum_{i=1}^{d} H(q_i) - \sum_{j \neq 1} I(q)_{j,\mathsf{pa}_j}
\end{aligned}
$$

- this completes the proof, so we want a maximum spanning tree of the mutual information graph

Approximation: algorithm (for data)

- given records $u^1, \ldots, u^n \in \mathcal{S}^d$ with empirical distribution q

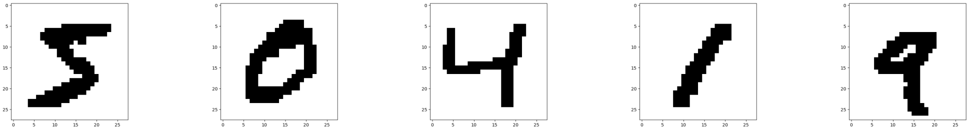
  1. compute the mutual information matrix of the empirical distribution
  2. find a maximum spanning tree of the mutual information graph
     - tree structure represented as element of $\{1, \ldots, d\}^d$
  3. construct distribution $\hat{p} = q_1 \prod_{i \neq 1} q_{i|\mathsf{pa}_i}$ (pa_i is parent function for T rooted at vertex 1)
     - $\hat{p}_1$: prior distribution of u_1 represented as $|\mathcal{S}|$-dimensional vector
     - $\hat{p}_{i|\mathsf{pa}_i}$ for $i \neq 1$: $d-1$ conditional distributions represented as $|\mathcal{S}| \times |\mathcal{S}|$-dimensional matrices

- return distribution $\hat{p}$ on $\mathcal{S}^d$
  - the model is specified by $O(d|\mathcal{S}|^2)$ *parameters*
  - the runtime is $O(nd^2 + d^2 \log d)$, for computing $I(q)$ and then finding T

Example: Binary MNIST

Data set

- train set of 60,000 records, test set of 10,000 records; both constructed by thresholding MNIST
  - originally 28 by 28 gray scale images with pixel values in $\{0, 1, 2, 3, \ldots, 255\}$
  - construct binary images by taking pixels as 1 if original pixel is positive (i.e., not 0)
- so $\mathcal{U} = \{0, 1\}^{784}$ and $\mathcal{V} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- we can visualize as 28 by 28 binary images, some examples:



(a) $v^1 = 5$ (b) $v^2 = 0$ (c) $v^3 = 4$ (d) $v^4 = 1$ (e) $v^5 = 9$

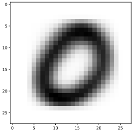
Figure 4: First five images in data set

Classifier

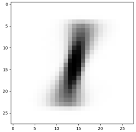
- for each class, we construct a distribution p^v over $\mathcal{U}$ using the train set, for $v = 0, \ldots, 9$
  - we split into ten subsets based on a record's class
  - we approximate the empirical distribution of each class
  - we obtain ten distributions on $\{0,1\}^{784}$
- we define a classifier $G : \mathcal{U} \to \mathcal{V}$ so that $G(u) \in \operatorname{argmax}_v p^v(u)$
  - we classify points according to the class with the maximum likelihood

Distribution sample averages

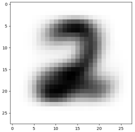
- we can roughly visualize the distributions by drawing 5000 samples and averaging



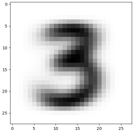
(a) $v = 0$



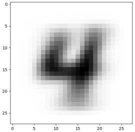
(b) $v = 1$



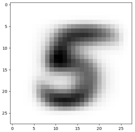
(c) $v = 2$



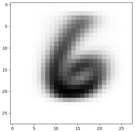
(d) $v = 3$



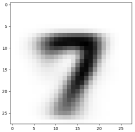
(e) $v = 4$



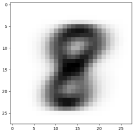
(f) $v = 5$



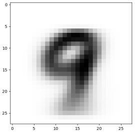
(g) $v = 6$



(h) $v = 7$



(i) $v = 8$



(j) $v = 9$

Confusion matrix

- we want a quantitative way to judge our classifier G
- the *confusion matrix* $C \in \mathbf{R}^{m \times m}$ of G on $u^1, \ldots, u^n$ summarizes performance
  - C_{ij} is the number of records for which $G(u^k) = i$ and $v^k = j$
  - in other words, the number of records we classified as i and the actual class was j
- the *accuracy* of G on $u^1, \ldots, u^n$ is the proportion of records correctly classified
  - can be expressed as $\dfrac{1}{n} \displaystyle\sum_{i=1}^{m} C_{ii}$
- the *error* of G on $u^1 \ldots, u^n$ is the proportion of records misclassified
- we want high accuracy and low error on a test set not used to construct G

Training confusion matrix

- we train with 60,000 data pairs
- here is the train set confusion matrix

$$C^{\text{train}} = \begin{bmatrix} 5742 & 1 & 27 & 30 & 4 & 20 & 31 & 7 & 12 & 22 \\ 3 & 6586 & 37 & 22 & 18 & 14 & 22 & 24 & 100 & 12 \\ 33 & 80 & 5617 & 133 & 16 & 10 & 10 & 53 & 87 & 6 \\ 9 & 6 & 54 & 5579 & 0 & 105 & 2 & 18 & 212 & 55 \\ 9 & 25 & 51 & 4 & 5538 & 2 & 7 & 74 & 45 & 94 \\ 26 & 4 & 6 & 116 & 7 & 5102 & 98 & 17 & 153 & 43 \\ 40 & 6 & 16 & 2 & 22 & 32 & 5692 & 1 & 18 & 1 \\ 0 & 5 & 33 & 37 & 18 & 4 & 0 & 5606 & 10 & 169 \\ 60 & 18 & 107 & 163 & 24 & 110 & 56 & 44 & 5117 & 81 \\ 1 & 11 & 10 & 45 & 195 & 22 & 0 & 421 & 97 & 5466 \end{bmatrix}$$

  - entry ij is the number of records for which we predicted class i and the actual class was j

Test confusion matrix

- we test with 10,000 data pairs
- here is the test set confusion matrix

$$C^{\text{test}} = \begin{bmatrix} 953 & 0 & 15 & 10 & 3 & 3 & 14 & 2 & 6 & 6 \\ 0 & 1099 & 5 & 0 & 0 & 2 & 4 & 8 & 3 & 6 \\ 1 & 13 & 952 & 14 & 1 & 1 & 0 & 20 & 15 & 7 \\ 1 & 0 & 16 & 917 & 2 & 27 & 1 & 5 & \mathit{38} & 4 \\ 2 & 4 & 9 & 0 & 940 & 0 & 3 & 9 & 5 & 19 \\ 8 & 0 & 0 & 31 & 0 & 830 & 12 & 2 & 25 & 10 \\ 7 & 7 & 4 & 0 & 5 & 8 & 915 & 0 & 4 & 0 \\ 3 & 0 & 7 & 6 & 5 & 1 & 0 & 897 & 10 & 24 \\ 4 & 12 & 23 & 22 & 3 & 15 & 9 & 12 & 849 & 11 \\ 1 & 0 & 1 & 10 & 23 & 5 & 0 & \mathit{73} & 19 & 922 \end{bmatrix}$$

- again, entry ij is the number of records for which we predicted class i and the actual class was j
- we confuse sevens for nines and eights for threes (highlighted in red)

Summary of numerical experiments

- train error (60,000 pairs): 6.59%; test error (10,000 pairs): 7.26%;
  - indicated accuracy is $\approx 93\%$
  - state of the art (neural networks) is $\approx 99\%$
- julia code runs in about 5 minutes to construct distributions
  - nearly all of that time is spent finding second order distributions (counting co-occurrences)
- model specified by 15,680 parameters; compare with $2^{784} - 1$
  - 784 integers for structure of each tree (specifying parent of each node)
  - 784 floating point numbers for log conditional probabilities in each tree
- inference time is trivial

Extensions

- can define relative entropy for two measures
  - if P and Q are probability measures and $P \ll Q$, define the relative entropy
    $$d_{kl}(P, Q) = \int \log\left(\frac{dP}{dQ}\right) dP$$
  - will give probability mass function and probability density function cases
  - less aesthetic patches for cases when $P \not\ll Q$
- can derive Chow-Liu for Gaussian density estimation
  - corresponds to sparsity in the precision matrix