EMPHASIZING MODULARITY IN MACHINE LEARNING PROGRAMS

Praveen Narayanan
IIS talk series, Feb 17 2016
MACHINE LEARNING

classification

START

get more data

>50 samples

predicting a category

<100K samples

do you have labeled data

<100K samples

SGD Regressor

few features should be important

predicting a category

>50 samples

SGD Classifier

not working

NOT WORKING

SVC Ensemble Classifiers

KNeighbors Classifier

kernel approximation

NOT WORKING

Linear SVC

Text Data

NOT WORKING

Naive Bayes

Spectral Clustering GMM

KMeans

http://peekaboo-vision.blogspot.com/2013/01/machine-learning-cheat-sheet-for-scikit.html
TOPIC MODELS

**Situation**: given a bank of labeled documents, predict the topics (labels) of new documents

**Learning**: use data-bank to learn a model of document-topic correlation

**Inference**: use learned model to predict topic distribution of new documents
TOPIC MODELS: NAIVE BAYES

\[ \eta_w = 1 \quad \text{for each } w \in \text{Words} \]
\[ \beta_z \sim \text{Dirichlet}(\eta) \quad \text{for each } z \in \text{Labels} \]
\[ \alpha_z = 1 \quad \text{for each } z \in \text{Labels} \]
\[ \theta \sim \text{Dirichlet}(\alpha) \]
\[ z_d \sim \text{Categorical}(\theta) \quad \text{for each } d \in \text{Docs} \]
\[ w_{dn} \sim \text{Categorical}(\beta_{zd}) \quad \text{for each } d \in \text{Docs}, \quad n \in \{1, \ldots, |d|\} \]
TOPIC MODELS: NAIVE BAYES

\[
\begin{align*}
\eta &\sim \text{Dirichlet}(\eta) \\
\alpha &\sim \text{Dirichlet}(\alpha) \\
\theta &\sim \text{Categorical}(\theta) \\
\beta &\sim \text{Categorical}(\beta) \\
\end{align*}
\]

- \( \eta_w = 1 \) for each \( w \in \text{Words} \)
- \( \beta_z \sim \text{Dirichlet}(\eta) \) for each \( z \in \text{Labels} \)
- \( \alpha_z = 1 \) for each \( z \in \text{Labels} \)
- \( \theta \sim \text{Dirichlet}(\alpha) \)
- \( z_d \sim \text{Categorical}(\theta) \) for each \( d \in \text{Docs} \)
- \( w_{dn} \sim \text{Categorical}(\beta) \) for each \( d \in \text{Docs}, n \in \{1, \ldots, |d|\} \)

How to take advantage of the dataset (labeled docs)?
How do we perform inference?
I see conjugacies...

We could do Gibbs sampling!
1.2.3 The Gibbs sampling algorithm

Gibbs sampling is applicable in situations where $Z$ has at least two dimensions, i.e. each point $z$ is really $z = \langle z_1, \ldots, z_k \rangle$, with $k > 1$. The basic idea in Gibbs sampling is that, rather than probabilistically picking the next state all at once, you make a separate probabilistic choice for each of the $k$ dimensions, where each choice depends on the other $k - 1$ dimensions. That is, the probabilistic walk through the space proceeds as follows:

1: $z^{(0)} := \langle z_1^{(0)}, \ldots, z_k^{(0)} \rangle$
2: for $t = 1$ to $T$ do
3:   for $i = 1$ to $k$ do
4:       $z_i^{(t+1)} \sim P(Z_i|z_1^{(t+1)}, \ldots, z_{i-1}^{(t+1)}, z_{i+1}^{(t+1)}, z_{i+1}^{(t)}, \ldots, z_k^{(t)})$
5:   end for
6: end for

Note that we can obtain the distribution we are sampling from by using the definition of conditional probability:

$$P(Z_i|z_1^{(t+1)}, \ldots, z_{i-1}^{(t+1)}, z_{i+1}^{(t+1)}, z_{i+1}^{(t)}, \ldots, z_k^{(t)}) = \frac{P(z_1^{(t+1)}, \ldots, z_{i-1}^{(t+1)}, z_i^{(t)}), z_{i+1}^{(t)}, \ldots, z_k^{(t)})}{P(z_1^{(t+1)}, \ldots, z_{i-1}^{(t+1)}, z_{i+1}^{(t)}, z_{i+1}^{(t)}, \ldots, z_k^{(t)})}. \quad (14)$$

How do we actually sample from (14) for our specific model? Let’s derive the Gibbs sampler for our model!
2.4.2 Choice of Priors and Simplifying the Joint Probability Expression

So why did we pick the Dirichlet distribution as our prior for $\theta_0$ and $\theta_1$ and the Beta distribution as our prior for $\pi$? Let’s look at what happens in the process of simplifying the joint distribution and see what happens to our estimates of $\theta$ (where this can be either $\theta_0$ or $\theta_1$) and $\pi$ once we observe some evidence (i.e., the words from a single document). Using (19) and (21) from above:

\[
P(\pi|L; \gamma, \gamma_{\theta_0}) = P(\pi)P(\pi|\gamma_{\theta_0}) = \pi^{C_0}(1-\pi)^{C_0} \left[ \pi^{\gamma_{\theta_0}-1}(1-\pi)^{\gamma_{\theta_0}-1} \right]
\]

\[
\propto \pi^{C_1+\gamma_{\theta_0}-1}(1-\pi)^{C_0+\gamma_{\theta_0}-1}
\]

Likewise, for $\theta$ using (24) and (25) from above:

\[
P(\theta_{\pi|W}; \gamma_{\theta_0}) = P(W_{\pi|W})P(\theta_{\pi|W}) = \prod_{i=1}^{V} \theta_{\pi|W}^{W_{\pi|i}} \prod_{i=1}^{V} \theta_{\pi|W}^{\gamma_{\theta_0}-1} = \prod_{i=1}^{V} \theta_{\pi|W}^{W_{\pi|i}+\gamma_{\theta_0}-1}
\]

If we use the words in all of the documents of a given class, then we have:

\[
P(C,L,\pi,\theta_0,\theta_1; \mu) \propto \pi^{C_1+\gamma_{\theta_0}-1}(1-\pi)^{C_0+\gamma_{\theta_0}-1} \prod_{i=1}^{V} \theta_{\pi|W}^{W_{\pi|i}} \theta_{\pi,\theta}^{\gamma_{\theta_0}}
\]

Integrate out hidden variable

\[
P(L,C,\theta_0,\theta_1; \mu) \propto \frac{\Gamma(\gamma_{\theta_1} + C_0) \Gamma(\gamma_{\theta_0})}{\Gamma(\gamma_{\theta_1}) \Gamma(\gamma_{\theta_0})} \prod_{i=1}^{V} \theta_{\pi,\theta}^{\gamma_{\theta_0}}
\]

The conditional distribution needed for the Gibbs sampler ...

\[
P(L_j|L^{(-j)}, C^{(-j)}, \theta_0, \theta_1; \mu) = \frac{P(L_j, W_j, L^{(-j)}, C^{(-j)}, \theta_0, \theta_1; \mu)}{P(L^{(-j)}, C^{(-j)}, \theta_0, \theta_1; \mu)}
\]

... becomes ...

\[
Pr(L_j = x|L^{(-j)}, C^{(-j)}, \theta_0, \theta_1; \mu) = \frac{C_x + \gamma_{\theta_x} - 1}{N + \gamma_{\theta_x} + \gamma_{\theta_0} - 1} \prod_{i=1}^{V} \theta_{x|i}^{W_{x|i}}
\]

And now we are ready to write the inference algorithm for our model!
2.5.4 Putting it all together

Initialization. Define the priors as in Section 2.2 and initialize them as described in Section 2.3.

1: for $t := 1$ to $T$ do
2:   for $j := 1$ to $N$ do
3:     if $j$ is not a training document then
4:       Subtract $j$'s word counts from the total word counts of whatever class it's currently a member of
5:       Subtract 1 from the count of documents with label $L_j$
6:     Assign a new label $L_j^{(t+1)}$ to document $j$ as described at the end of Section 2.5.1
7:     Add 1 to the count of documents with label $L_j^{(t+1)}$
8:     Add $j$'s word counts to the total word counts for class $L_j^{(t+1)}$
9:   end if
10: end for
11: $t_0 :=$ vector of total word counts from class 0, including pseudocounts
12: $\theta_0 \sim \text{Dirichlet}(t_0)$, as described in Section 2.5.2
13: $t_1 :=$ vector of total word counts from class 1, including pseudocounts
14: $\theta_1 \sim \text{Dirichlet}(t_1)$, as described in Section 2.5.2
15: end for

Combined learner + predictor
SMALL CONCEPTUAL CHANGE: LDA

Mix topics from word to word within the same document

\[ \eta_w = 1 \quad \text{for each } w \in \text{Words} \]
\[ \beta_z \sim \text{Dirichlet}(\eta) \quad \text{for each } z \in \text{Labels} \]
\[ \alpha_z = 1 \quad \text{for each } z \in \text{Labels} \]
\[ \theta_d \sim \text{Dirichlet}(\alpha) \quad \text{for each } d \in \text{Docs} \]
\[ z_{dn} \sim \text{Categorical}(\theta_d) \quad \text{for each } d \in \text{Docs}, \ n \in \{1, \ldots, |d|\} \]
\[ w_{dn} \sim \text{Categorical}(\beta_{z_{dn}}) \quad \text{for each } d \in \text{Docs}, \ n \in \{1, \ldots, |d|\} \]
How to perform inference?

G. Heinrich (2005) shows how we can do Gibbs sampling here.
Figure 8. Gibbs sampling algorithm for latent Dirichlet allocation.
How to perform inference?

Original LDA paper (Blei et al. 2003) describes a variational EM algorithm
(1) initialize $\phi_{ni}^0 := 1/k$ for all $i$ and $n$
(2) initialize $\gamma_i := \alpha_i + N/k$ for all $i$
(3) repeat
(4) for $n = 1$ to $N$
(5) for $i = 1$ to $k$
(6) $\phi_{ni}^{t+1} := \beta_i w_n \exp(\Psi(\gamma_i^t))$
(7) normalize $\phi_{ni}^{t+1}$ to sum to 1.
(8) $\gamma^{t+1} := \alpha + \sum_{n=1}^{N} \phi_{ni}^{t+1}$
(9) until convergence

Figure 6: A variational inference algorithm for LDA.
SMALL CONCEPTUAL CHANGE: LDA WITH LOGISTIC NORMAL

As per model described by Blei and Lafferty in 2007

\[ \eta_w = 1 \quad \text{for each } w \in \text{Words} \]
\[ \beta_z \sim \text{Dirichlet}(\eta) \quad \text{for each } z \in \text{Labels} \]
\[ \xi_d \sim \text{Normal}(\mu, \Sigma) \quad \text{for each } d \in \text{Docs} \]
\[ \theta_{dz} = \frac{\exp(\xi_{dz})}{\sum_z \xi_{dz}} \quad \text{for each } d \in \text{Docs}, z \in \text{Labels} \]
\[ z_{dn} \sim \text{Categorical}(\theta_d) \quad \text{for each } d \in \text{Docs}, n \in \{1, \ldots, |d|\} \]
\[ w_{dn} \sim \text{Categorical}(\beta_{z_{dn}}) \quad \text{for each } d \in \text{Docs}, n \in \{1, \ldots, |d|\} \]
INSUFFICIENT ABSTRACTION

- Small change in concepts, but ...
- ... large change in code
- Model and inference are intertwined
STATUS TODAY

• What does such a workflow involve today?

• Grad students to churn through code?

• Pair programming between domain (eg. linguistics) experts and machine learning experts?
MODULARITY

- Decouple model and inference

- Want to use
  - same inference method on different models
  - different inference methods on same model
DECOUPLE MODEL AND INFERENCE

- Abstraction!
- High level programming languages have helped since 50 years ago
- Software defined networking
ABSTRACTION FOR ML

- plate and tilde
- words like “EM” and “Gibbs”
- a programming language with keywords for ML
- a programming system with inference methods for ML
Decouple model and inference

Synonyms:

- Model and *generative story*
- Inference and *conditioning*
PROBABILISTIC PROGRAMMING SUCCESSES

- LIGO (gravitational waves paper uses Stan)
- Generative 3D modeling (Ritchie et al)
- Mansinghka’s Picture work (using a rendering system as a component of a computer vision system)
• Decouple model and inference

• Synonyms:
  • Model and *generative story*
  • Inference and *conditioning*
First component of the decoupling

Tilde notation models converted to a programming language
def medical():
    lungCancer = bern(0.01)
    cold = bern(0.2)
    cough = cold | lungCancer
    return cough

medical :: measure bool
def medical():
    lungCancer ~ bern(0.01)
    cold ~ bern(0.2)
    cough = cold || lungCancer
    return cough

medical :: measure bool

“The function medical() has no input and a (measure bool) as output”

cold :: prob -> measure bool

What is ~?
...
cold ~ bern(0.2)
cold :: ?

What is =?
...
cold = bern(0.2)
cold :: ?
type document = array(string)

def naiveBayes():
    w = 100
    z = 10
    d = 50
    docLens = [29,43,97,...]

    betas ≈ plate _ of z: dirichlet(array _ of w: 1)
    theta ≈ dirichlet(array _ of z: 1)
    zetas ≈ plate _ of d: categorical(theta)
    docs ≈ plate i of d:
        n = docLens[i]
        beta_z_d = betas[zetas[i]]
        doc ≈ plate _ of n: categorical(beta_z_d)
    return doc

    return pair(docs, zetas)

naiveBayes :: measure(pair(array(document), array(nat))))
type document = array(string)

def naiveBayes():
    w = 100
    z = 10
    d = 50
    docLens = [29,43,97,...]

    betas $\sim$ plate _ of z: dirichlet(array _ of w: 1)
    theta $\sim$ dirichlet(array _ of z: 1)
    zetas $\sim$ plate _ of d: categorical(theta)
    docs $\sim$ plate _ of d:
        n = docLens[i]
        beta_z_d = betas[zetas[i]]
        doc $\sim$ plate _ of n: categorical(beta_z_d)
    return doc

    return pair(docs, zetas)

naiveBayes :: measure(pair(array(document), array(nat))))
def naiveBayes():
...

betas ← plate _ of z: dirichlet(array _ of w: 1)
theta ← dirichlet(array _ of z: 1)
zetas ← plate _ of d: categorical(theta)
...

\[
\begin{align*}
\eta_w &= 1 \\
\beta_z &\sim \text{Dirichlet}(\eta) \\
\alpha_z &= 1 \\
\theta &\sim \text{Dirichlet}(\alpha) \\
z_{dz} &\sim \text{Categorical}(\theta) \\
w_{dn} &\sim \text{Categorical}(\beta_{z_d})
\end{align*}
\]
```python
def naiveBayes():
    ...
    betas ~ plate_z: dirichlet(array_w: 1)
    theta ~ dirichlet(array_z: 1)
    zetas ~ plate_d: categorical(theta)
    ...
```

**Naive bayes**

\[
\begin{align*}
\eta_w &= 1 & \text{for each } w \in \text{Words} \\
\beta_z &\leftarrow \text{Dirichlet}(\eta) & \text{for each } z \in \text{Labels} \\
\alpha_z &= 1 & \text{for each } z \in \text{Labels} \\
\theta &\leftarrow \text{Dirichlet}(\alpha) \quad \forall d \in \text{Docs} \\
\zeta_d &\leftarrow \text{Categorical}(\theta) \quad \forall d \in \text{Docs}, n \in \{1, \ldots, |d|\} \\
w_{dn} &\leftarrow \text{Categorical}(\beta_z) \\
\end{align*}
\]
def naiveBayes():
...

betas ~ plate(z: dirichlet(array(w: 1)))
theta ~ dirichlet(array(z: 1))
zetas ~ plate(d: categorical(theta))
...

Naive bayes

\begin{align*}
\eta_w &= 1 \\
\beta_z &\sim \text{Dirichlet}(\eta) \\
\alpha_z &= 1 \\
\theta &\sim \text{Dirichlet}(\alpha) \\
z_{dz} &\sim \text{Categorical}(\theta) \\
w_{dn} &\sim \text{Categorical}(\beta_{z_d})
\end{align*}

for each \( w \in \text{Words} \)
for each \( z \in \text{Labels} \)
for each \( z \in \text{Labels} \)
for each \( d \in \text{Docs} \)
for each \( d \in \text{Docs}, \ n \in \{1, \ldots, |d|\} \)
def naiveBayes():
    ...

    betas ◻ plate _ of z: dirichlet(array _ of w: 1)
    theta ◻ dirichlet(array _ of z: 1)
    zetas ◻ plate _ of d: categorical(theta)
    ...

Naive bayes

\[
\begin{align*}
\eta_w &= 1 \\
\beta_z &\sim \text{Dirichlet}(\eta) \\
\alpha_z &= 1 \\
\theta &\sim \text{Dirichlet}(\alpha) \\
z_{d} &\sim \text{Categorical}(\theta) \\
w_{d,n} &\sim \text{Categorical}(\beta_{z_d})
\end{align*}
\]

for each \( w \in \text{Words} \)
for each \( z \in \text{Labels} \)
for each \( z \in \text{Labels} \)
for each \( d \in \text{Docs} \)
for each \( d \in \text{Docs} \), \( n \in \{1, \ldots, |d|\} \)
def naiveBayes():
...

betas \sim \text{plate}_z \text{ of } z \sim \text{dirichlet(array}_w \sim 1)
theta \sim \text{dirichlet(array}_z \sim 1)
zetas \sim \text{plate}_d \text{ of } d \sim \text{categorical}(theta)
...

\text{Naive bayes}

\eta_w = 1
\beta_z \sim \text{Dirichlet}(\eta)
\alpha_z = 1
\theta \sim \text{Dirichlet}(\alpha)
z_{dz} \sim \text{Categorical}(\theta)
w_{dn} \sim \text{Categorical}(\beta_{dz})

for each $w \in \text{Words}$
for each $z \in \text{Labels}$
for each $z \in \text{Labels}$
for each $d \in \text{Docs}$
for each $d \in \text{Docs}$,
$n \in \{1, \ldots, |d|\}$
def naiveBayes():
...

betas ~ plate(z: dirichlet(array(w: 1))
theta ~ dirichlet(array(z: 1))
zetas ~ plate(d: categorical(theta))
...

plate i of n :: measure(a) -> measure(array(a))
def naiveBayes():
    ...
    betas ~ plate z: dirichlet(array w: 1)
    theta ~ dirichlet(array of z: 1)
    zetas ~ plate d: categorical(theta)
    ...

Naive bayes

\[
\begin{align*}
\eta_w &= 1 \\
\beta_z &\sim \text{Dirichlet}(\eta) \\
\alpha_z &= 1 \\
\theta &\sim \text{Dirichlet}(\alpha) \\
z_{zd} &\sim \text{Categorical}(\theta) \\
w_{dn} &\sim \text{Categorical}(\beta_{zd})
\end{align*}
\]

for each \( w \in \text{Words} \)
for each \( z \in \text{Labels} \)
for each \( \theta \in \text{Labels} \)
for each \( d \in \text{Docs} \)
for each \( n \in \{1, \ldots, |d|\} \)

loop index variable (nat)
loop size (nat)
def naiveBayes():
    ...
    betas ← plate _ of z: dirichlet(array _ of w: 1)
    theta ← dirichlet(array _ of z: 1)
    zetas ← plate _ of d: categorical(theta)
    ...
    plate i of n :: measure(a) → measure(array(a))
    loop index variable (nat)
    loop size (nat)
    zetas :: array(nat)

Naive bayes

ηw = 1
βz ← Dirichlet(η)
αz = 1
θ ← Dirichlet(α)
z_{zd} ← Categorical(θ)
w_{dn} ← Categorical(β_{zd})
for each w ∈ Words
for each z ∈ Labels
for each z ∈ Labels
for each d ∈ Docs
for each d ∈ Docs,
    n ∈ {1, ..., |d|}
def naiveBayes():
    ...
    betas <- plate of z: dirichlet(array of w: 1)
    theta <- dirichlet(array of z: 1)
    zetas <- plate of d: categorical(theta)
    ...

Naive bayes

<table>
<thead>
<tr>
<th></th>
<th>for each w ∈ Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>β’</td>
<td>Dirichlet(η)</td>
</tr>
<tr>
<td>α’</td>
<td>Dirichlet(α)</td>
</tr>
<tr>
<td>θ’</td>
<td>Dirichlet(α)</td>
</tr>
<tr>
<td>η</td>
<td>Dirichlet(η)</td>
</tr>
<tr>
<td>n</td>
<td>∈ {1, ...,</td>
</tr>
</tbody>
</table>

plate : measure(a) -> measure(array(a))

loop index variable (nat)

loop size (nat)

zetas : array(nat)

betas : array(array(prob))
```python
type document = array(string)

def naiveBayes():
  w = 100
  z = 10
  d = 50
  docLens = [29, 43, 97, ...]

  betas  \sim \text{plate of } z: \text{dirichlet(array of } w: 1)\
  theta  \sim \text{dirichlet(array of } z: 1)\
  zetas  \sim \text{plate of } d: \text{categorical(theta)}\
  docs  \sim \text{plate of } d:
    n    = docLens[i]
    beta_z_d = betas[zetas[i]]
    doc  \sim \text{plate of } n: \text{categorical(beta_z_d)}
  return doc

return pair(docs, zetas)

naiveBayes :: measure(pair(array(document), array(nat)))
```

Naive bayes
type document = array(string)

def naiveBayes():
    w = 100
    z = 10
    d = 50
    docLens = [29, 43, 97, ...]

    betas <- plate _ of z: dirichlet(array _ of w: 1)
    theta <- dirichlet(array _ of z: 1)
    zetas <- plate _ of d: categorical(theta)
    docs <- plate _ of d:
        n = docLens[i]
        beta_z_d = betas[zetas[i]]
        doc <- plate _ of n: categorical(beta_z_d)
        return doc

    return pair(docs, zetas)

naiveBayes :: measure(pair(array(document), array(nat))))
type document = array(string)

def naiveBayes():
    w = 100
    z = 10
    d = 50
    docLens = [29, 43, 97, ...]

    betas ~ plate z: dirichlet(array of w: 1)
    thetas ~ plate d: dirichlet(array of z: 1)
    zetas ~ plate d:
        n = docLens[i]
        zeta ~ plate n: categorical(thetas[i])
        return zeta

docs ~ plate d:
    n = docLens[i]
    beta_z_d = betas[zetas[i]]
    doc ~ plate n: categorical(beta_z_d)
    return doc

    return pair(docs, zetas)

naiveBayes :: measure(pair(array(document), array(nat))))
COMPOSING MODELS

- Naive bayes $\rightarrow$ smoothed LDA
- Smoothed LDA $\rightarrow$ LDA with logistic normal
Imagine a model that needs a document classifier in one part

- Eg: predict flu from Twitter (using topic of the tweet)

- Use Naive bayes or smoothed LDA inside this model (just satisfy the type system)
MORE STORIES

- Medical diagnosis
- HMM
- PCFG
The language provides general tools for conditioning (aka inferring) desired variables inside the generative story (aka model).
We can “always fall back” to rejection-sampling the joint model

\[
\text{naiveBayes} :: \text{measure}((\text{pair}(\text{array}(\text{doc}), \text{array}(\text{nat}))))
\]

Samples will be pairs of doc and labels

But we can do better

Hakaru provides disintegrate and algebraic simplify
COMPOSING INFERENCE

- Compose inference methods from smaller inference methods
- Rob’s talk in 2 weeks for more detail
• Using labeled data to improve Naive bayes model

• One model for learning parameters from labeled documents (learning as inference)

• One model for predicting label

• Use the same disintegrate tool at each step
TWO PIPELINES THAT WORK TODAY

- burglary - alarm
- thermometer (linear dynamic model)
- Naive bayes with Gibbs for 2 labels (beta-bernoulli conjugacy) and a small number of words
THANK YOU

https://github.com/hakaru-dev/hakaru