Chapter 5

Learning without supervision

So far, we have assumed the following setup:

- a **training set** where you get observations $\boldsymbol{x}$ and labels y;
- a **test set** where you only get observations $\boldsymbol{x}$.

Without labeled data, is it possible to learn anything? This scenario is known as **unsupervised learning**, and we will see that indeed it is possible to learn about the underlying structure of unlabeled observations. This chapter will also explore some related scenarios: **semi-supervised learning**, in which only some instances are labeled, and **domain adaptation**, in which the training data differs from the data on which the trained system will be deployed.

5.1 Unsupervised learning

To motivate unsupervised learning, consider the problem of word sense disambiguation (§ 4.2). The goal is to classify each instance of a word, such as *bank* into a sense,

- `bank#1`: a financial institution
- `bank#2`: the land bordering a river

It is difficult to obtain sufficient training data for word sense disambiguation, because even a large corpus will contain only a few instances of all but the most common words. Is it possible to learn anything about these different senses without labeled data?

Word sense disambiguation is usually performed using feature vectors constructed from the local context of the word to be disambiguated. For example, for the word

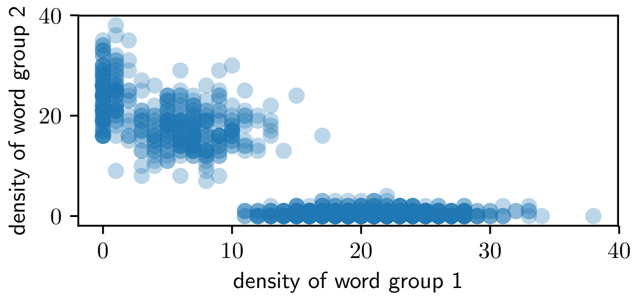


Figure 5.1: Counts of words from two different context groups

bank, the immediate context might typically include words from one of the following two groups:

1. *financial, deposits, credit, lending, capital, markets, regulated, reserve, liquid, assets*
2. *land, water, geography, stream, river, flow, deposits, discharge, channel, ecology*

Now consider a scatterplot, in which each point is a document containing the word *bank*. The location of the document on the x-axis is the count of words in group 1, and the location on the y-axis is the count for group 2. In such a plot, shown in Figure 5.1, two "blobs" might emerge, and these blobs correspond to the different senses of *bank*.

Here's a related scenario, from a different problem. Suppose you download thousands of news articles, and make a scatterplot, where each point corresponds to a document: the x-axis is the frequency of the group of words (*hurricane, winds, storm*); the y-axis is the frequency of the group (*election, voters, vote*). This time, three blobs might emerge: one for documents that are largely about a hurricane, another for documents largely about a election, and a third for documents about neither topic.

These clumps represent the underlying structure of the data. But the two-dimensional scatter plots are based on groupings of context words, and in real scenarios these word lists are unknown. Unsupervised learning applies the same basic idea, but in a high-dimensional space with one dimension for every context word. This space can't be directly visualized, but the goal is the same: try to identify the underlying structure of the observed data, such that there are a few clusters of points, each of which is internally coherent. **Clustering** algorithms are capable of finding such structure automatically.

5.1.1 K-means clustering

Clustering algorithms assign each data point to a discrete cluster, $z_i \in 1, 2, \ldots K$. One of the best known clustering algorithms is K**-means**, an iterative algorithm that maintains

Algorithm 8 K-means clustering algorithm

```
1: procedure $K$-MEANS($\boldsymbol{x}_{1:N}, K$)
2:     for $i \in 1 \dots N$ do                                  ▷ initialize cluster memberships
3:         $z^{(i)} \leftarrow \text{RANDOMINT}(1, K)$
4:     repeat
5:         for $k \in 1 \dots K$ do                              ▷ recompute cluster centers
6:             $\boldsymbol{\nu}_k \leftarrow \frac{1}{\delta(z^{(i)}=k)} \sum_{i=1}^{N} \delta(z^{(i)} = k)\boldsymbol{x}^{(i)}$
7:         for $i \in 1 \dots N$ do                              ▷ reassign instances to nearest clusters
8:             $z^{(i)} \leftarrow \text{argmin}_k \, ||\boldsymbol{x}^{(i)} - \boldsymbol{\nu}_k||^2$
9:     until converged
10:    return $\{z^{(i)}\}$                                      ▷ return cluster assignments
```

a cluster assignment for each instance, and a central ("mean") location for each cluster. K-means iterates between updates to the assignments and the centers:

1. each instance is placed in the cluster with the closest center;

2. each center is recomputed as the average over points in the cluster.

This procedure is formalized in Algorithm 8. The term $||\boldsymbol{x}^{(i)} - \boldsymbol{\nu}||^2$ refers to the squared Euclidean norm, $\sum_{j=1}^{V}(x_j^{(i)} - \nu_j)^2$. An important property of K-means is that the converged solution depends on the initialization, and a better clustering can sometimes be found simply by re-running the algorithm from a different random starting point.

Soft K-means is a particularly relevant variant. Instead of directly assigning each point to a specific cluster, soft K-means assigns to each point a *distribution* over clusters $\boldsymbol{q}^{(i)}$, so that $\sum_{k=1}^{K} q^{(i)}(k) = 1$, and $\forall_k, q^{(i)}(k) \geq 0$. The soft weight $q^{(i)}(k)$ is computed from the distance of $\boldsymbol{x}^{(i)}$ to the cluster center $\boldsymbol{\nu}_k$. In turn, the center of each cluster is computed from a weighted average of the points in the cluster,

$$\boldsymbol{\nu}_k = \frac{1}{\sum_{i=1}^{N} q^{(i)}(k)} \sum_{i=1}^{N} q^{(i)}(k)\boldsymbol{x}^{(i)}. \qquad [5.1]$$

We will now explore a probablistic version of soft K-means clustering, based on **expectation-maximization** (EM). Because EM clustering can be derived as an approximation to maximum-likelihood estimation, it can be extended in a number of useful ways.

5.1.2 Expectation-Maximization (EM)

Expectation-maximization combines the idea of soft K-means with Naïve Bayes classification. To review, Naïve Bayes defines a probability distribution over the data,

$$\log \mathrm{p}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\phi}, \boldsymbol{\mu}) = \sum_{i=1}^{N} \log \left(\mathrm{p}(\boldsymbol{x}^{(i)} \mid y^{(i)}; \boldsymbol{\phi}) \times \mathrm{p}(y^{(i)}; \boldsymbol{\mu}) \right) \qquad [5.2]$$

Now suppose that you never observe the labels. To indicate this, we'll refer to the label of each instance as $z^{(i)}$, rather than $y^{(i)}$, which is usually reserved for observed variables. By marginalizing over the **latent variables** $\boldsymbol{z}$, we obtain the marginal probability of the observed instances $\boldsymbol{x}$:

$$\begin{aligned}
\log \mathrm{p}(\boldsymbol{x}; \boldsymbol{\phi}, \boldsymbol{\mu}) &= \sum_{i=1}^{N} \log \mathrm{p}(\boldsymbol{x}^{(i)}; \boldsymbol{\phi}, \boldsymbol{\mu}) && [5.3] \\
&= \sum_{i=1}^{N} \log \sum_{z=1}^{K} \mathrm{p}(\boldsymbol{x}^{(i)}, z; \boldsymbol{\phi}, \boldsymbol{\mu}) && [5.4] \\
&= \sum_{i=1}^{N} \log \sum_{z=1}^{K} \mathrm{p}(\boldsymbol{x}^{(i)} \mid z; \boldsymbol{\phi}) \times \mathrm{p}(z; \boldsymbol{\mu}). && [5.5]
\end{aligned}$$

The parameters $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$ can be obtained by maximizing the marginal likelihood in Equation 5.5. Why is this the right thing to maximize? Without labels, discriminative learning is impossible — there's nothing to discriminate. So maximum likelihood is all we have.

When the labels are observed, we can estimate the parameters of the Naïve Bayes probability model separately for each label. But marginalizing over the labels couples these parameters, making direct optimization of $\log \mathrm{p}(\boldsymbol{x})$ intractable. We will approximate the log-likelihood by introducing an auxiliary variable $\boldsymbol{q}^{(i)}$, which is a distribution over the label set $\mathcal{Z} = \{1, 2, \ldots, K\}$. The optimization procedure will alternate between updates to $\boldsymbol{q}$ and updates to the parameters $(\boldsymbol{\phi}, \boldsymbol{\mu})$. Thus, $\boldsymbol{q}^{(i)}$ plays here as in soft K-means.

To derive the updates for this optimization, multiply the right side of Equation 5.5 by the ratio $\frac{q^{(i)}(z)}{q^{(i)}(z)} = 1$,

$$\log \mathrm{p}(\boldsymbol{x}; \boldsymbol{\phi}, \boldsymbol{\mu}) = \sum_{i=1}^{N} \log \sum_{z=1}^{K} \mathrm{p}(\boldsymbol{x}^{(i)} \mid z; \boldsymbol{\phi}) \times \mathrm{p}(z; \boldsymbol{\mu}) \times \frac{q^{(i)}(z)}{q^{(i)}(z)} \quad [5.6]$$

$$= \sum_{i=1}^{N} \log \sum_{z=1}^{K} q^{(i)}(z) \times \mathrm{p}(\boldsymbol{x}^{(i)} \mid z; \boldsymbol{\phi}) \times \mathrm{p}(z; \boldsymbol{\mu}) \times \frac{1}{q^{(i)}(z)} \quad [5.7]$$

$$= \sum_{i=1}^{N} \log E_{\boldsymbol{q}^{(i)}} \left[\frac{\mathrm{p}(\boldsymbol{x}^{(i)} \mid z; \boldsymbol{\phi}) \mathrm{p}(z; \boldsymbol{\mu})}{q^{(i)}(z)} \right], \quad [5.8]$$

where $E_{\boldsymbol{q}^{(i)}}[f(z)] = \sum_{z=1}^{K} q^{(i)}(z) \times f(z)$ refers to the expectation of the function f under the distribution $z \sim \boldsymbol{q}^{(i)}$.

Jensen's inequality says that because $\log$ is a concave function, we can push it inside the expectation, and obtain a lower bound.

$$\log \mathrm{p}(\boldsymbol{x}; \boldsymbol{\phi}, \boldsymbol{\mu}) \geq \sum_{i=1}^{N} E_{\boldsymbol{q}^{(i)}} \left[\log \frac{\mathrm{p}(\boldsymbol{x}^{(i)} \mid z; \boldsymbol{\phi}) \mathrm{p}(z; \boldsymbol{\mu})}{q^{(i)}(z)} \right] \quad [5.9]$$

$$J \triangleq \sum_{i=1}^{N} E_{\boldsymbol{q}^{(i)}} \left[\log \mathrm{p}(\boldsymbol{x}^{(i)} \mid z; \boldsymbol{\phi}) + \log \mathrm{p}(z; \boldsymbol{\mu}) - \log q^{(i)}(z) \right] \quad [5.10]$$

$$= \sum_{i=1}^{N} E_{\boldsymbol{q}^{(i)}} \left[\log \mathrm{p}(\boldsymbol{x}^{(i)}, z; \boldsymbol{\phi}, \boldsymbol{\mu}) \right] + H(\boldsymbol{q}^{(i)}) \quad [5.11]$$

We will focus on Equation 5.10, which is the lower bound on the marginal log-likelihood of the observed data, $\log \mathrm{p}(\boldsymbol{x})$. Equation 5.11 shows the connection to the information theoretic concept of **entropy**, $H(\boldsymbol{q}^{(i)}) = -\sum_{z=1}^{K} q^{(i)}(z) \log q^{(i)}(z)$, which measures the average amount of information produced by a draw from the distribution $q^{(i)}$. The lower bound J is a function of two groups of arguments:

- the distributions $\boldsymbol{q}^{(i)}$ for each instance;
- the parameters $\boldsymbol{\mu}$ and $\boldsymbol{\phi}$.

The expectation-maximization (EM) algorithm maximizes the bound with respect to each of these arguments in turn, while holding the other fixed.

The E-step

The step in which we update $\boldsymbol{q}^{(i)}$ is known as the **E-step**, because it updates the distribution under which the expectation is computed. To derive this update, first write out the expectation in the lower bound as a sum,

$$J = \sum_{i=1}^{N} \sum_{z=1}^{K} q^{(i)}(z) \left[\log \mathrm{p}(\boldsymbol{x}^{(i)} \mid z; \boldsymbol{\phi}) + \log \mathrm{p}(z; \boldsymbol{\mu}) - \log q^{(i)}(z)\right]. \quad [5.12]$$

When optimizing this bound, we must also respect a set of "sum-to-one" constraints, $\sum_{z=1}^{K} q^{(i)}(z) = 1$ for all i. Just as in Naïve Bayes, this constraint can be incorporated into a Lagrangian:

$$J_q = \sum_{i=1}^{N} \sum_{z=1}^{K} q^{(i)}(z) \left(\log \mathrm{p}(\boldsymbol{x}^{(i)} \mid z; \boldsymbol{\phi}) + \log \mathrm{p}(z; \mu) - \log q^{(i)}(z)\right) + \lambda^{(i)}(1 - \sum_{z=1}^{K} q^{(i)}(z)), \quad [5.13]$$

where $\lambda^{(i)}$ is the Lagrange multiplier for instance i.

The Lagrangian is maximized by taking the derivative and solving for $q^{(i)}$:

$$\frac{\partial J_q}{\partial q^{(i)}(z)} = \log \mathrm{p}(\boldsymbol{x}^{(i)} \mid z; \boldsymbol{\phi}) + \log \mathrm{p}(z; \boldsymbol{\theta}) - \log q^{(i)}(z) - 1 - \lambda^{(i)} \quad [5.14]$$

$$\log q^{(i)}(z) = \log \mathrm{p}(\boldsymbol{x}^{(i)} \mid z; \boldsymbol{\phi}) + \log \mathrm{p}(z; \mu) - 1 - \lambda^{(i)} \quad [5.15]$$

$$q^{(i)}(z) \propto \mathrm{p}(\boldsymbol{x}^{(i)} \mid z; \boldsymbol{\phi}) \times \mathrm{p}(z; \mu). \quad [5.16]$$

Applying the sum-to-one constraint gives an exact solution,

$$q^{(i)}(z) = \frac{\mathrm{p}(\boldsymbol{x}^{(i)} \mid z; \boldsymbol{\phi}) \times \mathrm{p}(z; \boldsymbol{\mu})}{\sum_{z'=1}^{K} \mathrm{p}(\boldsymbol{x}^{(i)} \mid z'; \boldsymbol{\phi}) \times \mathrm{p}(z'; \boldsymbol{\mu})} \quad [5.17]$$

$$= \mathrm{p}(z \mid \boldsymbol{x}^{(i)}; \boldsymbol{\phi}, \boldsymbol{\mu}). \quad [5.18]$$

After normalizing, each $\boldsymbol{q}^{(i)}$ — which is the soft distribution over clusters for data $\boldsymbol{x}^{(i)}$ — is set to the posterior probability $\mathrm{p}(z \mid \boldsymbol{x}^{(i)}; \boldsymbol{\phi}, \boldsymbol{\mu})$ under the current parameters. Although the Lagrange multipliers $\lambda^{(i)}$ were introduced as additional parameters, they drop out during normalization.

The M-step

Next, we hold fixed the soft assignments $\boldsymbol{q}^{(i)}$, and **m**aximize with respect to the parameters, $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$. Let's focus on the parameter $\boldsymbol{\phi}$, which parametrizes the likelihood $\mathrm{p}(\boldsymbol{x} \mid z; \boldsymbol{\phi})$, and leave $\boldsymbol{\mu}$ for an exercise. The parameter $\boldsymbol{\phi}$ is a distribution over words for each cluster, so it is optimized under the constraint that $\sum_{j=1}^{V} \phi_{z,j} = 1$. To incorporate this constraint, we introduce a set of Lagrange multiplers $\{\lambda_z\}_{z=1}^{K}$, and from the Lagrangian,

$$J_\phi = \sum_{i=1}^{N}\sum_{z=1}^{K} q^{(i)}(z)\left(\log \mathrm{p}(\boldsymbol{x}^{(i)} \mid z; \boldsymbol{\phi}) + \log \mathrm{p}(z; \mu) - \log q^{(i)}(z)\right) + \sum_{z=1}^{K}\lambda_z(1 - \sum_{j=1}^{V}\phi_{z,j}). \tag{5.19}$$

The term $\log \mathrm{p}(\boldsymbol{x}^{(i)} \mid z; \boldsymbol{\phi})$ is the conditional log-likelihood for the multinomial, which expands to,

$$\log \mathrm{p}(\boldsymbol{x}^{(i)} \mid z, \boldsymbol{\phi}) = C + \sum_{j=1}^{V} x_j \log \phi_{z,j}, \tag{5.20}$$

where C is a constant with respect to $\boldsymbol{\phi}$ — see Equation 2.12 in § 2.2 for more discussion of this probability function.

Setting the derivative of J_ϕ equal to zero,

$$\frac{\partial J_\phi}{\partial \phi_{z,j}} = \sum_{i=1}^{N} q^{(i)}(z) \times \frac{x_j^{(i)}}{\phi_{z,j}} - \lambda_z \tag{5.21}$$

$$\phi_{z,j} \propto \sum_{i=1}^{N} q^{(i)}(z) \times x_j^{(i)}. \tag{5.22}$$

Because $\boldsymbol{\phi}_z$ is constrained to be a probability distribution, the exact solution is computed as,

$$\phi_{z,j} = \frac{\sum_{i=1}^{N} q^{(i)}(z) \times x_j^{(i)}}{\sum_{j'=1}^{V}\sum_{i=1}^{N} q^{(i)}(z) \times x_{j'}^{(i)}} = \frac{E_q\left[\text{count}(z,j)\right]}{\sum_{j'=1}^{V} E_q\left[\text{count}(z,j')\right]}, \tag{5.23}$$

where the counter $j \in \{1, 2, \ldots, V\}$ indexes over base features, such as words.

This update sets $\boldsymbol{\phi}_z$ equal to the relative frequency estimate of the *expected counts* under the distribution $\boldsymbol{q}$. As in supervised Naïve Bayes, we can smooth these counts by adding a constant α. The update for $\boldsymbol{\mu}$ is similar: $\mu_z \propto \sum_{i=1}^{N} q^{(i)}(z) = E_q\left[\text{count}(z)\right]$, which is the expected frequency of cluster z. These probabilities can also be smoothed. In sum, the M-step is just like Naïve Bayes, but with expected counts rather than observed counts.

The multinomial likelihood $\mathrm{p}(\boldsymbol{x} \mid z)$ can be replaced with other probability distributions: for example, for continuous observations, a Gaussian distribution can be used. In some cases, there is no closed-form update to the parameters of the likelihood. One approach is to run gradient-based optimization at each M-step; another is to simply take a single step along the gradient step and then return to the E-step (Berg-Kirkpatrick et al., 2010).

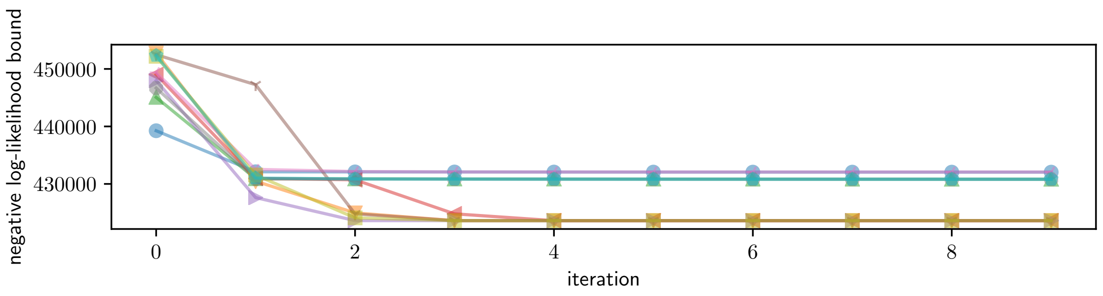


Figure 5.2: Sensitivity of expectation-maximization to initialization. Each line shows the progress of optimization from a different random initialization.

5.1.3 EM as an optimization algorithm

Algorithms that update a global objective by alternating between updates to subsets of the parameters are called **coordinate ascent** algorithms. The objective J (the lower bound on the marginal likelihood of the data) is separately convex in $\boldsymbol{q}$ and $(\boldsymbol{\mu}, \boldsymbol{\phi})$, but it is not jointly convex in all terms; this condition is known as **biconvexity**. Each step of the expectation-maximization algorithm is guaranteed not to decrease the lower bound J, which means that EM will converge towards a solution at which no nearby points yield further improvements. This solution is a **local optimum** — it is as good or better than any of its immediate neighbors, but is *not* guaranteed to be optimal among all possible configurations of $(\boldsymbol{q}, \boldsymbol{\mu}, \boldsymbol{\phi})$.

The fact that there is no guarantee of global optimality means that initialization is important: where you start can determine where you finish. To illustrate this point, Figure 5.2 shows the objective function for EM with ten different random initializations: while the objective function improves monotonically in each run, it converges to several different values.[1] For the convex objectives that we encountered in chapter 2, it was not necessary to worry about initialization, because gradient-based optimization guaranteed to reach the global minimum. But in expectation-maximization — as in the deep neural networks from chapter 3 — initialization matters.

In **hard EM**, each $\boldsymbol{q}^{(i)}$ distribution assigns probability of 1 to a single label $\hat{z}^{(i)}$, and zero probability to all others (Neal and Hinton, 1998). This is similar in spirit to K-means clustering, and can outperform standard EM in some cases (Spitkovsky et al., 2010). Another variant of expectation-maximization incorporates stochastic gradient descent (SGD): after performing a local E-step at each instance $\boldsymbol{x}^{(i)}$, we immediately make a gradient update to the parameters $(\boldsymbol{\mu}, \boldsymbol{\phi})$. This algorithm has been called **incremental expectation maximization** (Neal and Hinton, 1998) and **online expectation maximization** (Sato and Ishii,

[1]The figure shows the upper bound on the *negative* log-likelihood, because optimization is typically framed as minimization rather than maximization.

2000; Cappé and Moulines, 2009), and is especially useful when there is no closed-form optimum for the likelihood $\mathrm{p}(\boldsymbol{x} \mid z)$, and in online settings where new data is constantly streamed in (see Liang and Klein, 2009, for a comparison for online EM variants).

5.1.4 How many clusters?

So far, we have assumed that the number of clusters K is given. In some cases, this assumption is valid. For example, a lexical semantic resource like WORDNET might define the number of senses for a word. In other cases, the number of clusters could be a parameter for the user to tune: some readers want a coarse-grained clustering of news stories into three or four clusters, while others want a fine-grained clustering into twenty or more. But many times there is little extrinsic guidance for how to choose K.

One solution is to choose the number of clusters to maximize a metric of clustering quality. The other parameters $\boldsymbol{\mu}$ and $\boldsymbol{\phi}$ are chosen to maximize the log-likelihood bound J, so this might seem a potential candidate for tuning K. However, J will never decrease with K: if it is possible to obtain a bound of J_K with K clusters, then it is always possible to do at least as well with $K + 1$ clusters, by simply ignoring the additional cluster and setting its probability to zero in $\boldsymbol{q}$ and $\boldsymbol{\mu}$. It is therefore necessary to introduce a penalty for model complexity, so that fewer clusters are preferred. For example, the Akaike Information Crition (AIC; Akaike, 1974) is the linear combination of the number of parameters and the log-likelihood,

$$\text{AIC} = 2M - 2J, \qquad [5.24]$$

where M is the number of parameters. In an expectation-maximization clustering algorithm, $M = K \times V + K$. Since the number of parameters increases with the number of clusters K, the AIC may prefer more parsimonious models, even if they do not fit the data quite as well.

Another choice is to maximize the **predictive likelihood** on heldout data. This data is not used to estimate the model parameters $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$, and so it is not the case that the likelihood on this data is guaranteed to increase with K. Figure 5.3 shows the negative log-likelihood on training and heldout data, as well as the AIC.

***Bayesian nonparametrics** An alternative approach is to treat the number of clusters as another latent variable. This requires statistical inference over a set of models with a variable number of clusters. This is not possible within the framework of expectation-maximization, but there are several alternative inference procedures which can be applied, including **Markov Chain Monte Carlo (MCMC)**, which is briefly discussed in § 5.5 (for more details, see Chapter 25 of Murphy, 2012). Bayesian nonparametrics have been applied to the problem of unsupervised word sense induction, learning not only the word senses but also the number of senses per word (Reisinger and Mooney, 2010).

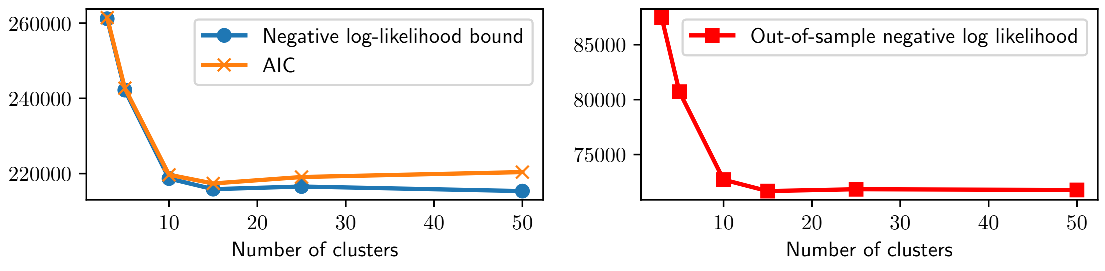


Figure 5.3: The negative log-likelihood and AIC for several runs of expectation-maximization, on synthetic data. Although the data was generated from a model with $K = 10$, the optimal number of clusters is $\hat{K} = 15$, according to AIC and the heldout log-likelihood. The training set log-likelihood continues to improve as K increases.

5.2 Applications of expectation-maximization

EM is not really an "algorithm" like, say, quicksort. Rather, it is a framework for learning with missing data. The recipe for using EM on a problem of interest is:

- Introduce latent variables $\boldsymbol{z}$, such that it is easy to write the probability $P(\boldsymbol{x}, \boldsymbol{z})$. It should also be easy to estimate the associated parameters, given knowledge of $\boldsymbol{z}$.
- Derive the E-step updates for $q(\boldsymbol{z})$, which is typically factored as $q(\boldsymbol{z}) = \prod_{i=1}^{N} q_{z^{(i)}}(z^{(i)})$, where i is an index over instances.
- The M-step updates typically correspond to the soft version of a probabilistic supervised learning algorithm, like Naïve Bayes.

This section discusses a few of the many applications of this general framework.

5.2.1 Word sense induction

The chapter began by considering the problem of word sense disambiguation when the senses are not known in advance. Expectation-maximization can be applied to this problem by treating each cluster as a word sense. Each instance represents the use of an ambiguous word, and $\boldsymbol{x}^{(i)}$ is a vector of counts for the other words that appear nearby: Schütze (1998) uses all words within a 50-word window. The probability $\mathrm{p}(\boldsymbol{x}^{(i)} \mid z)$ can be set to the multinomial distribution, as in Naïve Bayes. The EM algorithm can be applied directly to this data, yielding clusters that (hopefully) correspond to the word senses.

Better performance can be obtained by first applying **singular value decomposition** (SVD) to the matrix of context-counts $\mathbf{C}_{ij} = \mathrm{count}(i, j)$, where $\mathrm{count}(i, j)$ is the count of word j in the context of instance i. **Truncated** singular value decomposition approximates the matrix $\mathbf{C}$ as a product of three matrices, $\mathbf{U}, \mathbf{S}, \mathbf{V}$, under the constraint that $\mathbf{U}$ and $\mathbf{V}$ are orthonormal, and $\mathbf{S}$ is diagonal:

$$\begin{aligned}
&\min_{\mathbf{U},\mathbf{S},\mathbf{V}} ||\mathbf{C} - \mathbf{U}\mathbf{S}\mathbf{V}^\top||_F \\
&s.t. \mathbf{U} \in \mathbb{R}^{V \times K}, \mathbf{U}\mathbf{U}^\top = \mathbb{I} \\
&\quad \mathbf{S} = \text{Diag}(s_1, s_2, \ldots, s_K) \\
&\quad \mathbf{V}^\top \in \mathbb{R}^{N_p \times K}, \mathbf{V}\mathbf{V}^\top = \mathbb{I},
\end{aligned} \tag{5.25}$$

where $|| \cdot ||_F$ is the **Frobenius norm**, $||X||_F = \sqrt{\sum_{i,j} X_{i,j}^2}$. The matrix $\mathbf{U}$ contains the left singular vectors of $\mathbf{C}$, and the rows of this matrix can be used as low-dimensional representations of the count vectors $\boldsymbol{c}_i$. EM clustering can be made more robust by setting the instance descriptions $\boldsymbol{x}^{(i)}$ equal to these rows, rather than using raw counts (Schütze, 1998). However, because the instances are now dense vectors of continuous numbers, the probability $\text{p}(\boldsymbol{x}^{(i)} \mid z)$ must be defined as a multivariate Gaussian distribution.

In truncated singular value decomposition, the hyperparameter K is the truncation limit: when K is equal to the rank of $\mathbf{C}$, the norm of the difference between the original matrix $\mathbf{C}$ and its reconstruction $\mathbf{U}\mathbf{S}\mathbf{V}^\top$ will be zero. Lower values of K increase the reconstruction error, but yield vector representations that are smaller and easier to learn from. Singular value decomposition is discussed in more detail in chapter 14.

5.2.2 Semi-supervised learning

Expectation-maximization can also be applied to the problem of **semi-supervised learning**: learning from both labeled and unlabeled data in a single model. Semi-supervised learning makes use of annotated examples, ensuring that each label y corresponds to the desired concept. By adding unlabeled examples, it is possible to cover a greater fraction of the features than would appear in labeled data alone. Other methods for semi-supervised learning are discussed in § 5.3, but for now, let's approach the problem within the framework of expectation-maximization (Nigam et al., 2000).

Suppose we have labeled data $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N_\ell}$, and unlabeled data $\{\boldsymbol{x}^{(i)}\}_{i=N_\ell+1}^{N_\ell+N_u}$, where N_ℓ is the number of labeled instances and N_u is the number of unlabeled instances. We can learn from the combined data by maximizing a lower bound on the joint log-likelihood,

$$\mathcal{L} = \sum_{i=1}^{N_\ell} \log \text{p}(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\mu}, \boldsymbol{\phi}) + \sum_{j=N_\ell+1}^{N_\ell+N_u} \log \text{p}(\boldsymbol{x}^{(j)}; \boldsymbol{\mu}, \boldsymbol{\phi}) \tag{5.26}$$

$$= \sum_{i=1}^{N_\ell} \left(\log \text{p}(\boldsymbol{x}^{(i)} \mid y^{(i)}; \boldsymbol{\phi}) + \log \text{p}(y^{(i)}; \boldsymbol{\mu}) \right) + \sum_{j=N_\ell+1}^{N_\ell+N_u} \log \sum_{y=1}^{K} \text{p}(\boldsymbol{x}^{(j)}, y; \boldsymbol{\mu}, \boldsymbol{\phi}). \tag{5.27}$$

Algorithm 9 Generative process for the Naïve Bayes classifier with hidden components

```
for Instance $i \in \{1, 2, \ldots, N\}$ do:
    Draw the label $y^{(i)} \sim \text{Categorical}(\boldsymbol{\mu})$;
    Draw the component $z^{(i)} \sim \text{Categorical}(\boldsymbol{\beta}_{y^{(i)}})$;
    Draw the word counts $\boldsymbol{x}^{(i)} \mid y^{(i)}, z^{(i)} \sim \text{Multinomial}(\boldsymbol{\phi}_{z^{(i)}})$.
```

The left sum is identical to the objective in Naïve Bayes; the right sum is the marginal log-likelihood for expectation-maximization clustering, from Equation 5.5. We can construct a lower bound on this log-likelihood by introducing distributions $\boldsymbol{q}^{(j)}$ for all $j \in \{N_\ell + 1, \ldots, N_\ell + N_u\}$. The E-step updates these distributions; the M-step updates the parameters $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$, using the expected counts from the unlabeled data and the observed counts from the labeled data.

A critical issue in semi-supervised learning is how to balance the impact of the labeled and unlabeled data on the classifier weights, especially when the unlabeled data is much larger than the labeled dataset. The risk is that the unlabeled data will dominate, causing the parameters to drift towards a "natural clustering" of the instances — which may not correspond to a good classifier for the labeled data. One solution is to heuristically reweight the two components of Equation 5.26, tuning the weight of the two components on a heldout development set (Nigam et al., 2000).

5.2.3 Multi-component modeling

As a final application, let's return to fully supervised classification. A classic dataset for text classification is 20 newsgroups, which contains posts to a set of online forums, called newsgroups. One of the newsgroups is `comp.sys.mac.hardware`, which discusses Apple computing hardware. Suppose that within this newsgroup there are two kinds of posts: reviews of new hardware, and question-answer posts about hardware problems. The language in these *components* of the `mac.hardware class` might have little in common; if so, it would be better to model these components separately, rather than treating their union as a single class. However, the component responsible for each instance is not directly observed.

Recall that Naïve Bayes is based on a generative process, which provides a stochastic explanation for the observed data. In Naïve Bayes, each label is drawn from a categorical distribution with parameter $\boldsymbol{\mu}$, and each vector of word counts is drawn from a multinomial distribution with parameter $\boldsymbol{\phi}_y$. For multi-component modeling, we envision a slightly different generative process, incorporating both the observed label $y^{(i)}$ and the latent component $z^{(i)}$. This generative process is shown in Algorithm 9. A new parameter $\boldsymbol{\beta}_{y^{(i)}}$ defines the distribution of components, conditioned on the label $y^{(i)}$. The component, and not the class label, then parametrizes the distribution over words.

(5.1) ☺ Villeneuve a bel et bien **réussi** son pari de changer de perspectives tout en assurant une cohérence à la franchise.[2]

(5.2) ☹ Il est également trop **long** et bancal dans sa narration, tiède dans ses intentions, et tiraillé entre deux personnages et directions qui ne parviennent pas à coexister en harmonie.[3]

(5.3) Denis Villeneuve a **réussi** une suite **parfaitement** maitrisée[4]

(5.4) **Long**, **bavard**, hyper design, à peine agité (le comble de l'action : une bagarre dans la flotte), métaphysique et, surtout, ennuyeux jusqu'à la catalepsie.[5]

(5.5) Une suite d'une écrasante puissance, mêlant **parfaitement** le contemplatif au narratif.[6]

(5.6) Le film impitoyablement **bavard** finit quand même par se taire quand se lève l'espèce de bouquet final où semble se déchaîner, comme en libre parcours de poulets décapités, l'armée des graphistes numériques griffant nerveusement la palette graphique entre agonie et orgasme.[7]

Table 5.1: Labeled and unlabeled reviews of the films *Blade Runner 2049* and *Transformers: The Last Knight*.

The labeled data includes $(\boldsymbol{x}^{(i)}, y^{(i)})$, but not $z^{(i)}$, so this is another case of missing data. Again, we sum over the missing data, applying Jensen's inequality to as to obtain a lower bound on the log-likelihood,

$$\log \mathrm{p}(\boldsymbol{x}^{(i)}, y^{(i)}) = \log \sum_{z=1}^{K_z} \mathrm{p}(\boldsymbol{x}^{(i)}, y^{(i)}, z; \boldsymbol{\mu}, \boldsymbol{\phi}, \boldsymbol{\beta}) \qquad [5.28]$$

$$\geq \log \mathrm{p}(y^{(i)}; \boldsymbol{\mu}) + E_{q^{(i)}_{Z|Y}} [\log \mathrm{p}(\boldsymbol{x}^{(i)} \mid z; \boldsymbol{\phi}) + \log \mathrm{p}(z \mid y^{(i)}; \boldsymbol{\beta}) - \log q^{(i)}(z)]. \qquad [5.29]$$

We are now ready to apply expectation-maximization. As usual, the E-step updates the distribution over the missing data, $q^{(i)}_{Z|Y}$. The M-step updates the parameters,

$$\beta_{y,z} = \frac{E_q[\text{count}(y, z)]}{\sum_{z'=1}^{K_z} E_q[\text{count}(y, z')]} \qquad [5.30]$$

$$\phi_{z,j} = \frac{E_q[\text{count}(z, j)]}{\sum_{j'=1}^{V} E_q[\text{count}(z, j')]}. \qquad [5.31]$$

5.3 Semi-supervised learning

In semi-supervised learning, the learner makes use of both labeled and unlabeled data. To see how this could help, suppose you want to do sentiment analysis in French. In Table 5.1, there are two labeled examples, one positive and one negative. From this data, a learner could conclude that *réussi* is positive and *long* is negative. This isn't much! However, we can propagate this information to the unlabeled data, and potentially learn more.

- If we are confident that *réussi* is positive, then we might guess that (5.3) is also positive.
- That suggests that *parfaitement* is also positive.
- We can then propagate this information to (5.5), and learn from the words in this example.
- Similarly, we can propagate from the labeled data to (5.4), which we guess to be negative because it shares the word *long*. This suggests that *bavard* is also negative, which we propagate to (5.6).

Instances (5.3) and (5.4) were "similar" to the labeled examples for positivity and negativity, respectively. By using these instances to expand the models for each class, it became possible to correctly label instances (5.5) and (5.6), which didn't share any important features with the original labeled data. This requires a key assumption: that similar instances will have similar labels.

In § 5.2.2, we discussed how expectation-maximization can be applied to semi-supervised learning. Using the labeled data, the initial parameters ϕ would assign a high weight for *réussi* in the positive class, and a high weight for *long* in the negative class. These weights helped to shape the distributions $\boldsymbol{q}$ for instances (5.3) and (5.4) in the E-step. In the next iteration of the M-step, the parameters ϕ are updated with counts from these instances, making it possible to correctly label the instances (5.5) and (5.6).

However, expectation-maximization has an important disadvantage: it requires using a generative classification model, which restricts the features that can be used for classification. In this section, we explore non-probabilistic approaches, which impose fewer restrictions on the classification model.

5.3.1 Multi-view learning

EM semi-supervised learning can be viewed as **self-training**: the labeled data guides the initial estimates of the classification parameters; these parameters are used to compute a label distribution over the unlabeled instances, $\boldsymbol{q}^{(i)}$; the label distributions are used to update the parameters. The risk is that self-training drifts away from the original labeled data. This problem can be ameliorated by **multi-view learning**. Here we take the assumption that the features can be decomposed into multiple "views", each of which is conditionally independent, given the label. For example, consider the problem of classifying a name as a person or location: one view is the name itself; another is the context in which it appears. This situation is illustrated in Table 5.2.

	$\boldsymbol{x}^{(1)}$	$\boldsymbol{x}^{(2)}$	y
1.	Peachtree Street	located on	LOC
2.	Dr. Walker	said	PER
3.	Zanzibar	located in	? $\rightarrow$ LOC
4.	Zanzibar	flew to	? $\rightarrow$ LOC
5.	Dr. Robert	recommended	? $\rightarrow$ PER
6.	Oprah	recommended	? $\rightarrow$ PER

Table 5.2: Example of multiview learning for named entity classification

Co-training is an iterative multi-view learning algorithm, in which there are separate classifiers for each view (Blum and Mitchell, 1998). At each iteration of the algorithm, each classifier predicts labels for a subset of the unlabeled instances, using only the features available in its view. These predictions are then used as ground truth to train the classifiers associated with the other views. In the example shown in Table 5.2, the classifier on $\boldsymbol{x}^{(1)}$ might correctly label instance #5 as a person, because of the feature *Dr*; this instance would then serve as training data for the classifier on $\boldsymbol{x}^{(2)}$, which would then be able to correctly label instance #6, thanks to the feature *recommended*. If the views are truly independent, this procedure is robust to drift. Furthermore, it imposes no restrictions on the classifiers that can be used for each view.

Word-sense disambiguation is particularly suited to multi-view learning, thanks to the heuristic of "one sense per discourse": if a polysemous word is used more than once in a given text or conversation, all usages refer to the same sense (Gale et al., 1992). This motivates a multi-view learning approach, in which one view corresponds to the local context (the surrounding words), and another view corresponds to the global context at the document level (Yarowsky, 1995). The local context view is first trained on a small seed dataset. We then identify its most confident predictions on unlabeled instances. The global context view is then used to extend these confident predictions to other instances within the same documents. These new instances are added to the training data to the local context classifier, which is retrained and then applied to the remaining unlabeled data.

5.3.2 Graph-based algorithms

Another family of approaches to semi-supervised learning begins by constructing a graph, in which pairs of instances are linked with symmetric weights $\omega_{i,j}$, e.g.,

$$\omega_{i,j} = \exp(-\alpha \times ||\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}||^2). \quad [5.32]$$

The goal is to use this weighted graph to propagate labels from a small set of labeled instances to larger set of unlabeled instances.

In **label propagation**, this is done through a series of matrix operations (Zhu et al., 2003). Let $\mathbf{Q}$ be a matrix of size $N \times K$, in which each row $\boldsymbol{q}^{(i)}$ describes the labeling of instance i. When ground truth labels are available, then $\boldsymbol{q}^{(i)}$ is an indicator vector, with $q^{(i)}_{y^{(i)}} = 1$ and $q^{(i)}_{y' \neq y^{(i)}} = 0$. Let us refer to the submatrix of rows containing labeled instances as $\mathbf{Q}_L$, and the remaining rows as $\mathbf{Q}_U$. The rows of $\mathbf{Q}_U$ are initialized to assign equal probabilities to all labels, $q_{i,k} = \frac{1}{K}$.

Now, let $T_{i,j}$ represent the "transition" probability of moving from node j to node i,

$$T_{i,j} \triangleq \Pr(j \to i) = \frac{\omega_{i,j}}{\sum_{k=1}^{N} \omega_{k,j}}. \qquad [5.33]$$

We compute values of $T_{i,j}$ for all instances j and all *unlabeled* instances i, forming a matrix of size $N_U \times N$. If the dataset is large, this matrix may be expensive to store and manipulate; a solution is to sparsify it, by keeping only the κ largest values in each row, and setting all other values to zero. We can then "propagate" the label distributions to the unlabeled instances,

$$\tilde{\mathbf{Q}}_U \leftarrow \mathbf{T}\mathbf{Q} \qquad [5.34]$$

$$\boldsymbol{s} \leftarrow \tilde{\mathbf{Q}}_U \mathbf{1} \qquad [5.35]$$

$$\mathbf{Q}_U \leftarrow \text{Diag}(\boldsymbol{s})^{-1} \tilde{\mathbf{Q}}_U. \qquad [5.36]$$

The expression $\tilde{\mathbf{Q}}_U \mathbf{1}$ indicates multiplication of $\tilde{\mathbf{Q}}_U$ by a column vector of ones, which is equivalent to computing the sum of each row of $\tilde{\mathbf{Q}}_U$. The matrix $\text{Diag}(\boldsymbol{s})$ is a diagonal matrix with the elements of $\boldsymbol{s}$ on the diagonals. The product $\text{Diag}(\boldsymbol{s})^{-1}\tilde{\mathbf{Q}}_U$ has the effect of normalizing the rows of $\tilde{\mathbf{Q}}_U$, so that each row of $\mathbf{Q}_U$ is a probability distribution over labels.

5.4 Domain adaptation

In many practical scenarios, the labeled data differs in some key respect from the data to which the trained model is to be applied. A classic example is in consumer reviews: we may have labeled reviews of movies (the source domain), but we want to predict the reviews of appliances (the target domain). A similar issue arises with genre differences: most linguistically-annotated data is news text, but application domains range from social media to electronic health records. In general, there may be several source and target domains, each with their own properties; however, for simplicity, this discussion will focus mainly on the case of a single source and target domain.

The simplest approach is "direct transfer": train a classifier on the source domain, and apply it directly to the target domain. The accuracy of this approach depends on the extent to which features are shared across domains. In review text, words like *outstanding* and *disappointing* will apply across both movies and appliances; but others, like *terrifying*, may have meanings that are domain-specific. As a result, direct transfer performs poorly: for example, an out-of-domain classifier (trained on book reviews) suffers twice the error rate of an in-domain classifier on reviews of kitchen appliances (Blitzer et al., 2007). **Domain adaptation** algorithms attempt to do better than direct transfer by learning from data in both domains. There are two main families of domain adaptation algorithms, depending on whether any labeled data is available in the target domain.

5.4.1 Supervised domain adaptation

In supervised domain adaptation, there is a small amount of labeled data in the target domain, and a large amount of data in the source domain. The simplest approach would be to ignore domain differences, and simply merge the training data from the source and target domains. There are several other baseline approaches to dealing with this scenario (Daumé III, 2007):

Interpolation. Train a classifier for each domain, and combine their predictions, e.g.,

$$\hat{y} = \underset{y}{\operatorname{argmax}}\, \lambda_s \Psi_s(\boldsymbol{x}, y) + (1 - \lambda_s)\Psi_t(\boldsymbol{x}, y), \qquad [5.37]$$

where Ψ_s and Ψ_t are the scoring functions from the source and target domain classifiers respectively, and λ_s is the interpolation weight.

Prediction. Train a classifier on the source domain data, use its prediction as an additional feature in a classifier trained on the target domain data,

$$\hat{y}_s = \underset{y}{\operatorname{argmax}}\, \Psi_s(\boldsymbol{x}, y) \qquad [5.38]$$

$$\hat{y}_t = \underset{y}{\operatorname{argmax}}\, \Psi_t([\boldsymbol{x}; \hat{y}_S], y). \qquad [5.39]$$

Priors. Train a classifier on the source domain data, and use its weights as a prior distribution on the weights of the classifier for the target domain data. This is equivalent to regularizing the target domain weights towards the weights of the source domain classifier (Chelba and Acero, 2006),

$$\ell(\boldsymbol{\theta}_t) = \sum_{i=1}^{N} \ell^{(i)}(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta}_t) + \lambda ||\boldsymbol{\theta}_t - \boldsymbol{\theta}_s||_2^2, \qquad [5.40]$$

where $\ell^{(i)}$ is the prediction loss on instance i, and λ is the regularization weight.

An effective and "frustratingly simple" alternative is EASYADAPT (Daumé III, 2007), which creates copies of each feature: one for each domain and one for the cross-domain setting. For example, a negative review of the film *Wonder Woman* begins, *As boring and flavorless as a three-day-old grilled cheese sandwich....*[8] The resulting bag-of-words feature vector would be,

$$\begin{aligned}\boldsymbol{f}(\boldsymbol{x}, y, d) = \{&(\textit{boring}, ☹, \textsc{movie}) : 1, (\textit{boring}, ☹, *) : 1,\\ &(\textit{flavorless}, ☹, \textsc{movie}) : 1, (\textit{flavorless}, ☹, *) : 1,\\ &(\textit{three-day-old}, ☹, \textsc{movie}) : 1, (\textit{three-day-old}, ☹, *) : 1,\\ &\ldots\},\end{aligned}$$

with $(\textit{boring}, ☹, \textsc{movie})$ indicating the word *boring* appearing in a negative labeled document in the MOVIE domain, and $(\textit{boring}, ☹, *)$ indicating the same word in a negative labeled document in *any* domain. It is up to the learner to allocate weight between the domain-specific and cross-domain features: for words that facilitate prediction in both domains, the learner will use the cross-domain features; for words that are relevant only to a single domain, the domain-specific features will be used. Any discriminative classifier can be used with these augmented features.[9]

5.4.2 Unsupervised domain adaptation

In unsupervised domain adaptation, there is no labeled data in the target domain. Unsupervised domain adaptation algorithms cope with this problem by trying to make the data from the source and target domains as similar as possible. This is typically done by learning a **projection function**, which puts the source and target data in a shared space, in which a learner can generalize across domains. This projection is learned from data in both domains, and is applied to the base features — for example, the bag-of-words in text classification. The projected features can then be used both for training and for prediction.

Linear projection

In linear projection, the cross-domain representation is constructed by a matrix-vector product,

$$\boldsymbol{g}(\boldsymbol{x}^{(i)}) = \mathbf{U}\boldsymbol{x}^{(i)}. \qquad [5.41]$$

The projected vectors $\boldsymbol{g}(\boldsymbol{x}^{(i)})$ can then be used as base features during both training (from the source domain) and prediction (on the target domain).

[8] `http://www.colesmithey.com/capsules/2017/06/wonder-woman.HTML`, accessed October 9. 2017.

[9] EASYADAPT can be explained as a hierarchical Bayesian model, in which the weights for each domain are drawn from a shared prior (Finkel and Manning, 2009).

The projection matrix $\mathbf{U}$ can be learned in a number of different ways, but many approaches focus on compressing and reconstructing the base features (Ando and Zhang, 2005). For example, we can define a set of **pivot features**, which are typically chosen because they appear in both domains: in the case of review documents, pivot features might include evaluative adjectives like *outstanding* and *disappointing* (Blitzer et al., 2007). For each pivot feature j, we define an auxiliary problem of predicting whether the feature is present in each example, using the remaining base features. Let $\boldsymbol{\phi}_j$ denote the weights of this classifier, and us horizontally concatenate the weights for each of the N_p pivot features into a matrix $\boldsymbol{\Phi} = [\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \ldots, \boldsymbol{\phi}_{N_P}]$.

We then perform truncated singular value decomposition on $\boldsymbol{\Phi}$, as described in § 5.2.1, obtaining $\boldsymbol{\Phi} \approx \mathbf{U}\mathbf{S}\mathbf{V}^\top$. The rows of the matrix $\mathbf{U}$ summarize information about each base feature: indeed, the truncated singular value decomposition identifies a low-dimension basis for the weight matrix Φ, which in turn links base features to pivot features. Suppose that a base feature *reliable* occurs only in the target domain of appliance reviews. Nonetheless, it will have a positive weight towards some pivot features (e.g., *outstanding*, *recommended*), and a negative weight towards others (e.g., *worthless*, *unpleasant*). A base feature such as *watchable* might have the same associations with the pivot features, and therefore, $\boldsymbol{u}_{\text{reliable}} \approx \boldsymbol{u}_{\text{watchable}}$. The matrix $\mathbf{U}$ can thus project the base features into a space in which this information is shared.

Non-linear projection

Non-linear transformations of the base features can be accomplished by implementing the transformation function as a deep neural network, which is trained from an auxiliary objective.

Denoising objectives One possibility is to train a projection function to reconstruct a corrupted version of the original input. The original input can be corrupted in various ways: by the addition of random noise (Glorot et al., 2011; Chen et al., 2012), or by the deletion of features (Chen et al., 2012; Yang and Eisenstein, 2015). Denoising objectives share many properties of the linear projection method described above: they enable the projection function to be trained on large amounts of unlabeled data from the target domain, and allow information to be shared across the feature space, thereby reducing sensitivity to rare and domain-specific features.

Adversarial objectives The ultimate goal is for the transformed representations $\boldsymbol{g}(\boldsymbol{x}^{(i)})$ to be domain-general. This can be made an explicit optimization criterion by computing the similarity of transformed instances both within and between domains (Tzeng et al., 2015), or by formulating an auxiliary classification task, in which the domain itself is treated as a label (Ganin et al., 2016). This setting is **adversarial**, because we want

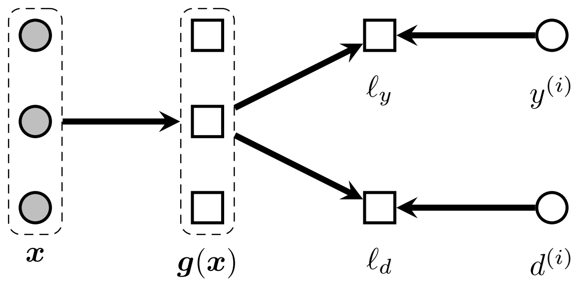


Figure 5.4: A schematic view of adversarial domain adaptation. The loss ℓ_y is computed only for instances from the source domain, where labels $y^{(i)}$ are available.

to learn a representation that makes this classifier perform poorly. At the same time, we want $\boldsymbol{g}(\boldsymbol{x}^{(i)})$ to enable accurate predictions of the labels $y^{(i)}$.

To formalize this idea, let $d^{(i)}$ represent the domain of instance i, and let $\ell_d(\boldsymbol{g}(\boldsymbol{x}^{(i)}), d^{(i)}; \boldsymbol{\theta}_d)$ represent the loss of a classifier (typically a deep neural network) trained to predict $d^{(i)}$ from the transformed representation $\boldsymbol{g}(\boldsymbol{x}^{(i)})$, using parameters $\boldsymbol{\theta}_d$. Analogously, let $\ell_y(\boldsymbol{g}(\boldsymbol{x}^{(i)}), y^{(i)}; \boldsymbol{\theta}_y)$ represent the loss of a classifier trained to predict the label $y^{(i)}$ from $\boldsymbol{g}(\boldsymbol{x}^{(i)})$, using parameters $\boldsymbol{\theta}_y$. The transformation $\boldsymbol{g}$ can then be trained from two criteria: it should yield accurate predictions of the labels $y^{(i)}$, while making *inaccurate* predictions of the domains $d^{(i)}$. This can be formulated as a joint optimization problem,

$$\min_{\boldsymbol{\theta}_g \boldsymbol{\theta}_y, \boldsymbol{\theta}_d} \sum_{i=1}^{N_\ell + N_u} \ell_d(\boldsymbol{g}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}_g), d^{(i)}; \boldsymbol{\theta}_d) - \sum_{i=1}^{N_\ell} \ell_y(\boldsymbol{g}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}_g), y^{(i)}; \boldsymbol{\theta}_y), \qquad [5.42]$$

where N_ℓ is the number of labeled instances and N_u is the number of unlabeled instances, with the labeled instances appearing first in the dataset. This setup is shown in Figure 5.4. The loss can be optimized by stochastic gradient descent, jointly training the parameters of the non-linear transformation $\boldsymbol{\theta}_g$, and the parameters of the prediction models $\boldsymbol{\theta}_d$ and $\boldsymbol{\theta}_y$.

5.5 *Other approaches to learning with latent variables

Expectation-maximization provides a general approach to learning with latent variables, but it has limitations. One is the sensitivity to initialization; in practical applications, considerable attention may need to be devoted to finding a good initialization. A second issue is that EM tends to be easiest to apply in cases where the latent variables have a clear decomposition (in the cases we have considered, they decompose across the instances). For these reasons, it is worth briefly considering some alternatives to EM.

5.5.1 Sampling

In EM clustering, there is a distribution $\boldsymbol{q}^{(i)}$ for the missing data related to each instance. The M-step consists of updating the parameters of this distribution. An alternative is to draw samples of the latent variables. If the sampling distribution is designed correctly, this procedure will eventually converge to drawing samples from the true posterior over the missing data, $\mathrm{p}(\boldsymbol{z}^{(1:N_z)} \mid \boldsymbol{x}^{(1:N_x)})$. For example, in the case of clustering, the missing data $\boldsymbol{z}^{(1:N_z)}$ is the set of cluster memberships, $\boldsymbol{y}^{(1:N)}$, so we draw samples from the posterior distribution over clusterings of the data. If a single clustering is required, we can select the one with the highest conditional likelihood, $\hat{\boldsymbol{z}} = \operatorname{argmax}_{\boldsymbol{z}} \mathrm{p}(\boldsymbol{z}^{(1:N_z)} \mid \boldsymbol{x}^{(1:N_x)})$.

This general family of algorithms is called **Markov Chain Monte Carlo (MCMC)**: "Monte Carlo" because it is based on a series of random draws; "Markov Chain" because the sampling procedure must be designed such that each sample depends only on the previous sample, and not on the entire sampling history. **Gibbs sampling** is an MCMC algorithm in which each latent variable is sampled from its posterior distribution,

$$z^{(n)} \mid \boldsymbol{x}, \boldsymbol{z}^{(-n)} \sim \mathrm{p}(z^{(n)} \mid \boldsymbol{x}, \boldsymbol{z}^{(-n)}), \qquad [5.43]$$

where $\boldsymbol{z}^{(-n)}$ indicates $\{\boldsymbol{z} \backslash z^{(n)}\}$, the set of all latent variables except for $z^{(n)}$. Repeatedly drawing samples over all latent variables constructs a Markov chain that is guaranteed to converge to a sequence of samples from $\mathrm{p}(\boldsymbol{z}^{(1:N_z)} \mid \boldsymbol{x}^{(1:N_x)})$. In probabilistic clustering, the sampling distribution has the following form,

$$\begin{aligned} \mathrm{p}(z^{(i)} \mid \boldsymbol{x}, \boldsymbol{z}^{(-i)}) =& \frac{\mathrm{p}(\boldsymbol{x}^{(i)} \mid z^{(i)}; \boldsymbol{\phi}) \times \mathrm{p}(z^{(i)}; \boldsymbol{\mu})}{\sum_{z=1}^{K} \mathrm{p}(\boldsymbol{x}^{(i)} \mid z; \boldsymbol{\phi}) \times \mathrm{p}(z; \boldsymbol{\mu})} & [5.44] \\ \propto& \text{Multinomial}(\boldsymbol{x}^{(i)}; \boldsymbol{\phi}_{z^{(i)}}) \times \boldsymbol{\mu}_{z^{(i)}}. & [5.45] \end{aligned}$$

In this case, the sampling distribution does not depend on the other instances: the posterior distribution over each $z^{(i)}$ can be computed from $\boldsymbol{x}^{(i)}$ and the parameters given the parameters $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$.

In sampling algorithms, there are several choices for how to deal with the parameters. One possibility is to sample them too. To do this, we must add them to the generative story, by introducing a prior distribution. For the multinomial and categorical parameters in the EM clustering model, the **Dirichlet distribution** is a typical choice, since it defines a probability on exactly the set of vectors that can be parameters: vectors that sum to one and include only non-negative numbers.[10]

To incorporate this prior, the generative model must be augmented to indicate that each $\boldsymbol{\phi}_z \sim$ Dirichlet($\boldsymbol{\alpha}_\phi$), and $\boldsymbol{\mu} \sim$ Dirichlet($\boldsymbol{\alpha}_\mu$). The hyperparameters $\boldsymbol{\alpha}$ are typically set to a constant vector $\boldsymbol{\alpha} = [\alpha, \alpha, \ldots, \alpha]$. When α is large, the Dirichlet distribution tends to

[10]If $\sum_i^K \theta_i = 1$ and $\theta_i \geq 0$ for all i, then $\boldsymbol{\theta}$ is said to be on the $K-1$ **simplex**. A Dirichlet distribution with

generate vectors that are nearly uniform; when α is small, it tends to generate vectors that assign most of their probability mass to a few entries. Given prior distributions over $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$, we can now include them in Gibbs sampling, drawing values for these parameters from posterior distributions that are conditioned on the other variables in the model.

Unfortunately, sampling $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$ usually leads to slow "mixing", meaning that adjacent samples tend to be similar, so that a large number of samples is required to explore the space of random variables. The reason is that the sampling distributions for the parameters are tightly constrained by the cluster memberships $\boldsymbol{y}^{(i)}$, which in turn are tightly constrained by the parameters. There are two solutions that are frequently employed:

- **Empirical Bayesian** methods maintain $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$ as parameters rather than latent variables. They still employ sampling in the E-step of the EM algorithm, but they update the parameters using expected counts that are computed from the samples rather than from parametric distributions. This EM-MCMC hybrid is also known as Monte Carlo Expectation Maximization (MCEM; Wei and Tanner, 1990), and is well-suited for cases in which it is difficult to compute $\boldsymbol{q}^{(i)}$ directly.
- In **collapsed Gibbs sampling**, we analytically integrate $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$ out of the model. The cluster memberships $y^{(i)}$ are the only remaining latent variable; we sample them from the compound distribution,

$$\mathrm{p}(y^{(i)} \mid \boldsymbol{x}^{(1:N)}, \boldsymbol{y}^{(-i)}; \alpha_\phi, \alpha_\mu) = \int_{\boldsymbol{\phi},\boldsymbol{\mu}} \mathrm{p}(\boldsymbol{\phi}, \boldsymbol{\mu} \mid \boldsymbol{y}^{(-i)}, \boldsymbol{x}^{(1:N)}; \alpha_\phi, \alpha_\mu) \mathrm{p}(y^{(i)} \mid \boldsymbol{x}^{(1:N)}, \boldsymbol{y}^{(-i)}, \boldsymbol{\phi}, \boldsymbol{\mu}) d\boldsymbol{\phi} d\boldsymbol{\mu}. \quad [5.48]$$

  For multinomial and Dirichlet distributions, this integral can be computed in closed form.

MCMC algorithms are guaranteed to converge to the true posterior distribution over the latent variables, but there is no way to know how long this will take. In practice, the rate of convergence depends on initialization, just as expectation-maximization depends on initialization to avoid local optima. Thus, while Gibbs Sampling and other MCMC algorithms provide a powerful and flexible array of techniques for statistical inference in latent variable models, they are not a panacea for the problems experienced by EM.

parameter $\boldsymbol{\alpha} \in \mathbb{R}_+^K$ has support over the $K-1$ simplex,

$$\mathrm{p}_{\text{Dirichlet}}(\boldsymbol{\theta} \mid \boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^{K} \theta_i^{\alpha_i - 1} \quad [5.46]$$

$$B(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^K \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^K \alpha_i)}, \quad [5.47]$$

with $\Gamma(\cdot)$ indicating the gamma function, a generalization of the factorial function to non-negative reals.

5.5.2 Spectral learning

Another approach to learning with latent variables is based on the **method of moments**, which makes it possible to avoid the problem of non-convex log-likelihood. Write $\overline{\boldsymbol{x}}^{(i)}$ for the normalized vector of word counts in document i, so that $\overline{\boldsymbol{x}}^{(i)} = \boldsymbol{x}^{(i)} / \sum_{j=1}^{V} x_j^{(i)}$. Then we can form a matrix of word-word co-occurrence probabilities,

$$\mathbf{C} = \sum_{i=1}^{N} \overline{\boldsymbol{x}}^{(i)} (\overline{\boldsymbol{x}}^{(i)})^\top. \qquad [5.49]$$

The expected value of this matrix under $\text{p}(\boldsymbol{x} \mid \boldsymbol{\phi}, \mu)$, as

$$\begin{aligned} E[\mathbf{C}] =& \sum_{i=1}^{N} \sum_{k=1}^{K} \Pr(Z^{(i)} = k; \boldsymbol{\mu}) \boldsymbol{\phi}_k \boldsymbol{\phi}_k^\top && [5.50] \\ =& \sum_{k}^{K} N \mu_k \boldsymbol{\phi}_k \boldsymbol{\phi}_k^\top && [5.51] \\ =& \Phi \text{Diag}(N\mu) \Phi^\top, && [5.52] \end{aligned}$$

where Φ is formed by horizontally concatenating $\boldsymbol{\phi}_1 \ldots \boldsymbol{\phi}_K$, and $\text{Diag}(N\mu)$ indicates a diagonal matrix with values $N\mu_k$ at position (k, k). Setting $\mathbf{C}$ equal to its expectation gives,

$$\mathbf{C} = \boldsymbol{\Phi} \text{Diag}(N\mu) \boldsymbol{\Phi}^\top, \qquad [5.53]$$

which is similar to the eigendecomposition $\mathbf{C} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^\top$. This suggests that simply by finding the eigenvectors and eigenvalues of $\mathbf{C}$, we could obtain the parameters $\boldsymbol{\phi}$ and μ, and this is what motivates the name **spectral learning**.

While moment-matching and eigendecomposition are similar in form, they impose different constraints on the solutions: eigendecomposition requires orthonormality, so that $\mathbf{Q}\mathbf{Q}^\top = \mathbb{I}$; in estimating the parameters of a text clustering model, we require that $\boldsymbol{\mu}$ and the columns of Φ are probability vectors. Spectral learning algorithms must therefore include a procedure for converting the solution into vectors that are non-negative and sum to one. One approach is to replace eigendecomposition (or the related singular value decomposition) with non-negative matrix factorization (Xu et al., 2003), which guarantees that the solutions are non-negative (Arora et al., 2013).

After obtaining the parameters $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$, the distribution over clusters can be computed from Bayes' rule:

$$\text{p}(z^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\phi}, \boldsymbol{\mu}) \propto \text{p}(\boldsymbol{x}^{(i)} \mid z^{(i)}; \boldsymbol{\phi}) \times \text{p}(z^{(i)}; \mu). \qquad [5.54]$$

Spectral learning yields provably good solutions without regard to initialization, and can be quite fast in practice. However, it is more difficult to apply to a broad family of generative models than EM and Gibbs Sampling. For more on applying spectral learning across a range of latent variable models, see Anandkumar et al. (2014).

Additional resources

There are a number of other learning paradigms that deviate from supervised learning.

- **Active learning**: the learner selects unlabeled instances and requests annotations (Settles, 2012).
- **Multiple instance learning**: labels are applied to bags of instances, with a positive label applied if at least one instance in the bag meets the criterion (Dietterich et al., 1997; Maron and Lozano-Pérez, 1998).
- **Constraint-driven learning**: supervision is provided in the form of explicit constraints on the learner (Chang et al., 2007; Ganchev et al., 2010).
- **Distant supervision**: noisy labels are generated from an external resource (Mintz et al., 2009, also see § 17.2.3).
- **Multitask learning**: the learner induces a representation that can be used to solve multiple classification tasks (Collobert et al., 2011).
- **Transfer learning**: the learner must solve a classification task that differs from the labeled data (Pan and Yang, 2010).

Expectation-maximization was introduced by Dempster et al. (1977), and is discussed in more detail by Murphy (2012). Like most machine learning treatments, Murphy focuses on continuous observations and Gaussian likelihoods, rather than the discrete observations typically encountered in natural language processing. Murphy (2012) also includes an excellent chapter on MCMC; for a textbook-length treatment, see Robert and Casella (2013). For still more on Bayesian latent variable models, see Barber (2012), and for applications of Bayesian models to natural language processing, see Cohen (2016). Surveys are available for semi-supervised learning (Zhu and Goldberg, 2009) and domain adaptation (Søgaard, 2013), although both pre-date the current wave of interest in deep learning.

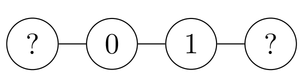
Exercises

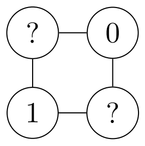
1. Derive the expectation maximization update for the parameter $\boldsymbol{\mu}$ in the EM clustering model.

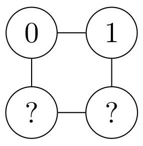
2. Derive the E-step and M-step updates for the following generative model. You may assume that the labels $y^{(i)}$ are observed, but $z_m^{(i)}$ is not.

   - For each instance i,
     - Draw label $y^{(i)} \sim \text{Categorical}(\boldsymbol{\mu})$
     - For each token $m \in \{1, 2, \ldots, M^{(i)}\}$
       * Draw $z_m^{(i)} \sim \text{Categorical}(\pi)$
       * If $z_m^{(i)} = 0$, draw the current token from a label-specific distribution, $w_m^{(i)} \sim \boldsymbol{\phi}_{y^{(i)}}$
       * If $z_m^{(i)} = 1$, draw the current token from a document-specific distribution, $w_m^{(i)} \sim \boldsymbol{\nu}^{(i)}$

3. Using the iterative updates in Equations 5.34-5.36, compute the outcome of the label propagation algorithm for the following examples.







   The value inside the node indicates the label, $y^{(i)} \in \{0, 1\}$, with $y^{(i)} =?$ for unlabeled nodes. The presence of an edge between two nodes indicates $w_{i,j} = 1$, and the absence of an edge indicates $w_{i,j} = 0$. For the third example, you need only compute the first three iterations, and then you can guess at the solution in the limit.

4. Use expectation-maximization clustering to train a word-sense induction system, applied to the word *say*.

   - Import NLTK, run NLTK.DOWNLOAD() and select SEMCOR. Import SEMCOR from NLTK.CORPUS.
   - The command SEMCOR.TAGGED_SENTENCES(TAG='SENSE') returns an iterator over sense-tagged sentences in the corpus. Each sentence can be viewed as an iterator over TREE objects. For TREE objects that are sense-annotated words, you can access the annotation as TREE.LABEL(), and the word itself with TREE.LEAVES(). So SEMCOR.TAGGED_SENTENCES(TAG='SENSE')[0][2].LABEL() would return the sense annotation of the third word in the first sentence.
   - Extract all sentences containing the senses SAY.V.01 and SAY.V.02.
   - Build bag-of-words vectors $\boldsymbol{x}^{(i)}$, containing the counts of other words in those sentences, including all words that occur in at least two sentences.
   - Implement and run expectation-maximization clustering on the merged data.

- Compute the frequency with which each cluster includes instances of SAY.V.01 and SAY.V.02.

In the remaining exercises, you will try out some approaches for semisupervised learning and domain adaptation. You will need datasets in multiple domains. You can obtain product reviews in multiple domains here: `https://www.cs.jhu.edu/~mdredze/datasets/sentiment/processed_acl.tar.gz`. Choose a source and target domain, e.g. dvds and books, and divide the data for the target domain into training and test sets of equal size.

5. First, quantify the cost of cross-domain transfer.

   - Train a logistic regression classifier on the source domain training set, and evaluate it on the target domain test set.
   - Train a logistic regression classifier on the target domain training set, and evaluate it on the target domain test set. This it the "direct transfer" baseline.

   Compute the difference in accuracy, which is a measure of the transfer loss across domains.

6. Next, apply the **label propagation** algorithm from § 5.3.2.

   As a baseline, using only 5% of the target domain training set, train a classifier, and compute its accuracy on the target domain test set.

   Next, apply label propagation:

   - Compute the label matrix $\mathbf{Q}_L$ for the labeled data (5% of the target domain training set), with each row equal to an indicator vector for the label (positive or negative).
   - Iterate through the target domain instances, including both test and training data. At each instance i, compute all w_{ij}, using Equation 5.32, with $\alpha = 0.01$. Use these values to fill in column i of the transition matrix $\mathbf{T}$, setting all but the ten largest values to zero for each column i. Be sure to normalize the column so that the remaining values sum to one. You may need to use a sparse matrix for this to fit into memory.
   - Apply the iterative updates from Equations 5.34-5.36 to compute the outcome of the label propagation algorithm for the unlabeled examples.

   Select the test set instances from $\mathbf{Q}_U$, and compute the accuracy of this method. Compare with the supervised classifier trained only on the 5% sample of the target domain training set.

7. Using only 5% of the target domain training data (and all of the source domain training data), implement one of the supervised domain adaptation baselines in § 5.4.1. See if this improves on the "direct transfer" baseline from the previous problem

8. Implement EASYADAPT (§ 5.4.1), again using 5% of the target domain training data and all of the source domain data.

9. Now try unsupervised domain adaptation, using the "linear projection" method described in § 5.4.2. Specifically:

   - Identify 500 pivot features as the words with the highest frequency in the (complete) training data for the source and target domains. Specifically, let x_i^d be the count of the word i in domain d: choose the 500 words with the largest values of $\min(x_i^{\text{source}}, x_i^{\text{target}})$.
   - Train a classifier to predict each pivot feature from the remaining words in the document.
   - Arrange the features of these classifiers into a matrix $\mathbf{\Phi}$, and perform truncated singular value decomposition, with $k = 20$
   - Train a classifier from the source domain data, using the combined features $\boldsymbol{x}^{(i)} \oplus \mathbf{U}^\top \boldsymbol{x}^{(i)}$ — these include the original bag-of-words features, plus the projected features.
   - Apply this classifier to the target domain test set, and compute the accuracy.

Part II

Sequences and trees