

Splink and the Fellegi-Sunter Model

October 2026

Goal:

- 1) Explain Splinks use of Fellegi Sunter model, during em blocking and training.
- 2) Go over dob else m-level training issues

Intro

I figured after working with Splink for so long, it would be good to get aligned on how I understand the model/training process.

The goal of Fellegi-Sunter (FS) training is to learn a model of $P(M_i = 1 | \text{what we observe about pair } i)$ where for any given pair of records i we have the unobserved indicator:

$$M_i = \begin{cases} 1 & \text{if the pair is a match} \\ 0 & \text{if the pair is not a match} \end{cases}$$

The “what we observe about pair i ” is collapsed into a space defined by k features where each feature has a set of levels that describes different levels of agreement between the pair on that feature. The features and the level values they take is described by the comparison vector $\gamma_i = (\gamma_{i1}, \dots, \gamma_{iK})$ where γ_{ik} represents the level of feature k for pair i . Each feature has a null level as the top of the level waterfall, then the possible agreement levels are ordered from strongest to weakest. For dob, the levels looks like ($k = 1$ for either null, $k = 2$ for exact match, $k = 3$ for $DL \leq 1$). So if dob is the first feature and pair 2 has exact match then the comparison vector would be $\gamma_2 = (2, \dots)$. Within the model, the comparison vector represents what we observe about each pair.

The goal is to 1) use the comparison vectors to train the model parameters unsupervised and 2) output a prediction probability given the comparison vectors.

Params

The three parameters the models needs to train are

$$\theta = (\lambda = P(M_i = 1), m_{kj} = P(\gamma_{ik} = j | M_i = 1), u_{kj} = P(\gamma_{ik} = j | M_i = 0))$$

for given pair i , feature k , and feature level j .

Weight bearing assumption

FS main assumption is that, given the match status, the resulting levels of each feature are independent of one another so we can write:

$$P(\gamma_i \mid M_i = 1) = \prod_{k=1}^K m_{k\gamma_{ik}}, \quad P(\gamma_i \mid M_i = 0) = \prod_{k=1}^K u_{k\gamma_{ik}}$$

This makes the rest of the model tractable. It is also frequently violated in practice. For example, agreement on a gender feature is near certain given agreement on first name, agreement on last name is near certain given agreement on household, even agreement on first given agreement on last name is more common than implied by the parameters because names cluster. In Splink, this is handled by combining highly conditionally dependent features. Say combining address and phone number, which both measure household evidence, into a singular household feature that uses both variables in its levels. However this cannot solve all the issues. The Splink maintainer acknowledges this and notes that when you suspect that the conditional independence is bad (like when adding address into model, because it is conditionally dependent on last name and other household variables), you should treat the probabilities as simply ordinal scores, and that in his experience those ordinal scores do a good job at ranking matches.

Note that this assumption is also used in the blocking strategy that he implements to make FS training reasonable for large datasets, which I cover later.

Posterior prob

By chain rule we know that:

$$\begin{aligned} P(\gamma_i, M_i = 1) &= P(M_i = 1) P(\gamma_i \mid M_i = 1) = \lambda P(\gamma_i \mid M_i = 1) \\ P(\gamma_i, M_i = 0) &= P(M_i = 0) P(\gamma_i \mid M_i = 0) = (1 - \lambda) P(\gamma_i \mid M_i = 0) \end{aligned}$$

So we can marginalize out M_i such that:

$$P(\gamma_i) = \sum_{m \in \{0,1\}} P(M_i = m) P(\gamma_i \mid M_i = m) = \lambda P(\gamma_i \mid M_i = 1) + (1 - \lambda) P(\gamma_i \mid M_i = 0)$$

Then here invoke the conditional independence assumption above to get:

$$P(\gamma_i \mid \theta) = \lambda \prod_k m_{k,\gamma_{ik}} + (1 - \lambda) \prod_k u_{k,\gamma_{ik}}$$

Finally, we can get posterior with Bayes':

$$p_i(\theta) = P(M_i = 1 \mid \gamma_i, \theta) = \frac{P(M_i = 1 \mid \theta) P(\gamma_i \mid M_i = 1, \theta)}{P(\gamma_i \mid \theta)} = \frac{\lambda \prod_k m_{k,\gamma_{ik}}}{\lambda \prod_k m_{k,\gamma_{ik}} + (1 - \lambda) \prod_k u_{k,\gamma_{ik}}}$$

Turning it into the odds form in logs space, we get Splink's "match weight" given the odds form of

lambda $\pi = \frac{\lambda}{1-\lambda}$ and the bayes factor $K_{kj} = m_{kj}/u_{kj}$:

$$\frac{p_i}{1-p_i} = \pi \prod_k K_{k\gamma_{ik}}, \quad \log \frac{p_i}{1-p_i} = \log \pi + \sum_k \log K_{k\gamma_{ik}}$$

The $\log \pi$ is the prior and each feature contributes the “bits” of from the observed level, where each bit in log space doubles the odds. This constructs how the evidence provides the odds, but it is conditioned on knowing θ which is the subject of the next section.

EM theory

To find the parameters the first instinct is to maximixe the likelihood over θ , but the log likelihood is a sum of logs of a sum so there is no closed form:

$$\ell(\theta) = \sum_{i=1}^N \log \left[\lambda \prod_k m_{k,\gamma_{ik}} + (1-\lambda) \prod_k u_{k,\gamma_{ik}} \right]$$

So intead, EM expliots the structure that the likelihood is a mixture and has the hidden variable M_i that, if we knew, would make it seperable. So then we are interested in the joint of the comparison vectors and the known M_i given theta to maximize over θ , compared with the incomplete data version where we were only interested in the distribution of the comparison vectors given theta. Using selector exponents, we can write the joint using the chain rule and the previous definitions:

$$\begin{aligned} P(\gamma_i, M_i \mid \theta) &= P(M_i \mid \theta) P(\gamma_i \mid M_i, \theta) \\ &= \left[\lambda \prod_k m_{k,\gamma_{ik}} \right]^{M_i} \left[(1-\lambda) \prod_k u_{k,\gamma_{ik}} \right]^{1-M_i} \end{aligned}$$

Then taking the log and summing over pairs we get:

$$\begin{aligned} \ell_c(\theta) &= \sum_i \log P(\gamma_i, M_i \mid \theta) \\ &= \sum_i \left[M_i \left(\log \lambda + \sum_k \log m_{k,\gamma_{ik}} \right) + (1-M_i) \left(\log(1-\lambda) + \sum_k \log u_{k,\gamma_{ik}} \right) \right] \end{aligned}$$

But, of course, we don't have complete data. So EM instead uses the expected completed-data log likelihood, replacing M_i with p_i given some current θ' . Since ℓ_c is linear in each M_i , the expectation passes stright through where $\mathbb{E}[M_i] = P(M_i = 1 \mid \gamma_i, \theta') = p_i(\theta')$:

$$\begin{aligned}
Q(\theta \mid \theta') &= \mathbb{E}_{M \sim P(\cdot \mid \gamma, \theta')} [\ell_c(\theta; M)] \\
&= \mathbb{E}_{M \sim P(\cdot \mid \gamma, \theta')} \left[\sum_i \log P(\gamma_i, M_i \mid \theta) \right] \\
&= \sum_i \mathbb{E}_{M_i \sim P(\cdot \mid \gamma_i, \theta')} [\log P(\gamma_i, M_i \mid \theta)] \\
&= \sum_i \sum_{M_i \in \{0,1\}} P(M_i \mid \gamma_i, \theta') \log P(\gamma_i, M_i \mid \theta)
\end{aligned}$$

And expanding given the previous work gives:

$$\begin{aligned}
Q(\theta \mid \theta') &= \sum_i \sum_{M_i \in \{0,1\}} P(M_i \mid \gamma_i, \theta') \log P(\gamma_i, M_i \mid \theta) \\
&= \sum_i \left[P(M_i = 1 \mid \gamma_i, \theta') \log P(\gamma_i, M_i = 1 \mid \theta) + P(M_i = 0 \mid \gamma_i, \theta') \log P(\gamma_i, M_i = 0 \mid \theta) \right] \\
&= \sum_i \left[p_i(\theta') \log \left(\lambda \prod_k m_{k, \gamma_{ik}} \right) + (1 - p_i(\theta')) \log \left((1 - \lambda) \prod_k u_{k, \gamma_{ik}} \right) \right] \\
&= \sum_i \left[p_i(\theta') \left(\log \lambda + \sum_k \log m_{k, \gamma_{ik}} \right) + (1 - p_i(\theta')) \left(\log(1 - \lambda) + \sum_k \log u_{k, \gamma_{ik}} \right) \right]
\end{aligned}$$

Finally, note that we can switch summation order because they are finite sums and our notation with the gammas in the subscript means $\log m_{k, \gamma_{ik}} = \sum_j \mathbf{1}\{\gamma_{ik} = j\} \log m_{kj}$. For brevity, write $p_i = p_i(\theta')$, $N_m = \sum_i p_i$, and $N_u = \sum_i (1 - p_i)$.

$$\begin{aligned}
Q(\theta \mid \theta') &= \sum_i \left[p_i(\theta') \left(\log \lambda + \sum_k \log m_{k, \gamma_{ik}} \right) + (1 - p_i(\theta')) \left(\log(1 - \lambda) + \sum_k \log u_{k, \gamma_{ik}} \right) \right] \\
&= \left(\sum_i p_i \right) \log \lambda + \left(\sum_i (1 - p_i) \right) \log(1 - \lambda) + \sum_k \sum_i p_i \log m_{k, \gamma_{ik}} + \sum_k \sum_i (1 - p_i) \log u_{k, \gamma_{ik}} \\
&= \underbrace{N_m \log \lambda + N_u \log(1 - \lambda)}_{Q_\lambda(\lambda)} + \underbrace{\sum_k \sum_j \left(\sum_i p_i \mathbf{1}\{\gamma_{ik} = j\} \right) \log m_{kj}}_{Q_{m,k}(m_{k\cdot})} \\
&\quad + \underbrace{\sum_k \sum_j \left(\sum_i (1 - p_i) \mathbf{1}\{\gamma_{ik} = j\} \right) \log u_{kj}}_{Q_{u,k}(u_{k\cdot})}
\end{aligned}$$

This form is nice for a couple reasons: 1) p_i are constants because they are computed with θ' , 2) $\log$ turns the products into sums so no term contains two parameters multiplied together, and 3) parameters for different comparisons k never appear in the same sum. This allows Q separability so it's a sum of blocks and each block is a function of one parameter group alone. Noting that a maximizer of a sum of function of disjoint variables is the sum of each maximized function on its own, we can decompose this into $1 + 2K$ independent problems (one for λ and one for each $m_{k\cdot}$,

and u_k , also noting the PMF constraints).

This is all well and good, but lets not loose track of the final goal which is to maximize the likelihood function. We must show that maximizing Q which is a nice case, also increases ℓ . We can do this with one expectation magic trick and the use of Jensen's.

Maxing Q increases ℓ

To get to use Jensen's, we first have to use the most classic trick of multiplying by a fancy 1. In this case, we choose some arbitrary distribution $q_i(M_i)$ over $M_i \in \{0, 1\}$ with the restriction that $q_i(0) + q_i(1) = 1$ to ensure it is a valid PMF (and $q_i(M_i) > 0$ to ensure division is defined). Then, marginalizing out M_i and multiplying by 1, we have:

$$\begin{aligned} P(\gamma_i | \theta) &= \sum_{M_i} P(\gamma_i, M_i | \theta) \\ &= \sum_{M_i} q_i(M_i) \frac{P(\gamma_i, M_i | \theta)}{q_i(M_i)} \end{aligned}$$

But now we have an expectation! Namley, $\mathbb{E}_{M_i \sim q_i} [\frac{P(\gamma_i, M_i | \theta)}{q_i(M_i)}]$. Then applying Jensen's on the concave function $\log$ gives:

$$\begin{aligned} \log P(\gamma_i | \theta) &= \log \sum_{M_i} q_i(M_i) \frac{P(\gamma_i, M_i | \theta)}{q_i(M_i)} = \log \mathbb{E} \left[\frac{P(\gamma_i, M_i | \theta)}{q_i(M_i)} \right] \\ &\geq \mathbb{E} \left[\log \frac{P(\gamma_i, M_i | \theta)}{q_i(M_i)} \right] = \sum_{M_i} q_i(M_i) \log \frac{P(\gamma_i, M_i | \theta)}{q_i(M_i)} \end{aligned}$$

We then look at the end of the equality to split the ratio of logs into a difference:

$$\begin{aligned} \sum_{M_i} q_i(M_i) \log \frac{P(\gamma_i, M_i | \theta)}{q_i(M_i)} &= \sum_{M_i} q_i(M_i) \log P(\gamma_i, M_i | \theta) - \sum_{M_i} q_i(M_i) \log q_i(M_i) \\ &= \mathbb{E}_{M_i \sim q_i} [\log P(\gamma_i, M_i | \theta)] + H(q_i) \end{aligned}$$

We end up with the expected complete data log likelihood for for i under q_i and $H(q_i)$ which is the entropy (I had to look up exactly what this means, but the important part here is that it does not depend on θ and so as we eventually maximize it will stay a constant). Summing over all pairs we get:

$$\ell(\theta) = \sum_i \log P(\gamma_i | \theta) \geq \sum_i \mathbb{E}_{q_i} [\log P(\gamma_i, M_i | \theta)] + \sum_i H(q_i)$$

The important part to note here is that the above equations describes a family of lower bounds that holds for any choice of q_i . If we choose q_i to be the posterior $P(M_i | \gamma_i, \theta')$, then we get:

$$\ell(\theta) \geq \sum_i \mathbb{E}_{P(M_i|\gamma_i, \theta')} [\log P(\gamma_i, M_i | \theta)] + \sum_i H(q_i) = Q(\theta|\theta') + H(q)$$

With this choice, we can show that the bound is tight at $\theta = \theta'$. To show that, I had to learn the concept of KL divergence. The important point here is that the KL divergence of two distributions is 0 iff the two distributions coincide at the given point. Thus, we know that $KL(P(\theta'|\gamma_i, \theta) || P(\theta'|\gamma_i, \theta)) = 0$. And that by definition of KL divergence $KL(f||g) = \sum_i f(x_i) \log \frac{f(x_i)}{g(x_i)}$. Then, differencing $\ell(\theta) - (Q(\theta|\theta') + H(q))$, and using the fact that q_i sums to 1 we get (and starting with the pre-decomposed form):

$$\begin{aligned} & \log P(\gamma_i | \theta) - \sum_{M_i} q_i(M_i) \log \frac{P(\gamma_i, M_i | \theta)}{q_i(M_i)} \\ &= \log P(\gamma_i | \theta) - \sum_{M_i} q_i(M_i) \log \frac{P(\gamma_i | \theta) P(M_i | \gamma_i, \theta)}{q_i(M_i)} \\ &= \log P(\gamma_i | \theta) - \sum_{M_i} q_i(M_i) \log P(\gamma_i | \theta) - \sum_{M_i} q_i(M_i) \log \frac{P(M_i | \gamma_i, \theta)}{q_i(M_i)} \\ &= \log P(\gamma_i | \theta) - \log P(\gamma_i | \theta) \sum_{M_i} q_i(M_i) - \sum_{M_i} q_i(M_i) \log \frac{P(M_i | \gamma_i, \theta)}{q_i(M_i)} \\ &= \log P(\gamma_i | \theta) - \log P(\gamma_i | \theta) \cdot 1 - \sum_{M_i} q_i(M_i) \log \frac{P(M_i | \gamma_i, \theta)}{q_i(M_i)} \\ &= \sum_{M_i} q_i(M_i) \log \frac{q_i(M_i)}{P(M_i | \gamma_i, \theta)} \\ &= \text{KL}(q_i || P(\cdot | \gamma_i, \theta)) \end{aligned}$$

So when we set $q_i = P(M_i|\gamma_i, \theta')$ and choose $\theta = \theta'$ we get the difference to be $\text{KL}(P(\cdot | \gamma_i, \theta') || P(\cdot | \gamma_i, \theta')) = 0$. Thus, the inequality $\ell(\theta) \geq Q(\theta|\theta') + H(q)$ is tight at $\theta = \theta'$.

Taking a step back, this shows that $\ell(\theta') = Q(\theta'|\theta') + H(q)$. So if we maximize to get $\theta'' = \arg \max_{\theta} Q(\theta | \theta')$, which is possible because Q is separable as shown in the EM theory section, then we can write:

$$\ell(\theta'') \geq Q(\theta'' | \theta') + H(q) \geq Q(\theta' | \theta') + H(q) = \ell(\theta')$$

Where 1) first inequality is the original bound at θ'' (noting that $H(q)$ does not change because it doesn't depend on θ), 2) the second inequality is the result of maximization, and 3) the inequality is the tightness at $\theta = \theta'$. In words, this means that each round of maximization which gives a new θ'' given some previous θ' can never decrease the log likelihood function.

So in conclusion, we use the expectation of the complete data log likelihood function because we can make it separable and it also maximizes the log likelihood of the data given the conditional independence assumption. The first reason is due to the result in the EM theory section which shows the separability and thus the easy of maximization. The second reason is due to the result in this section. Namely that we can 1) rewrite the log likelihood function with an expectation trick, 2)

use Jensen's to get a lower bound on the log likelihood and decompose the lower bound family to show a familiar form, 3) note that the one of the choices within the family of lower bounds makes it the expectation of the complete data log likelihood and entropy (constant as theta changes), 3) show that this choice of lower bound makes the inequality tight with KL divergence, and 4) make the final conclusion that maximizing Q over theta will never decrease the desired log likelihood function.

Next, I'll go over exactly how the nice maximization of Q happens within Splink.

EM steps

The high level goal here is for each EM iteration $t \geq 1$, where the first round using seeds is $t = 0$, we use $\theta^{(t-1)}$ to find $p_i(\theta^{(t-1)})$, input those values into Q , then estimate θ until $\max |\theta^{(t)} - \theta^{(t-1)}| < 10^{-4}$ or whatever convergence value is set in the Splink settings, or we reach max iterations (default 25) implying there was no convergence.

EM consists of 1) the E-step (expectation step), and 2) the M-step (maximization step). Recall that $Q(\theta|\theta')$ is (noting these rewritings and simplifications: 1) $\log m_{k,\gamma_{ik}} = \sum_j \mathbf{1}\{\gamma_{ik} = j\} \log m_{kj}$, 2) $p_i = p_i(\theta')$, $N_m = \sum_i p_i$, and $N_u = \sum_i (1 - p_i)$):

$$\begin{aligned} Q(\theta | \theta') = & \underbrace{N_m \log \lambda + N_u \log(1 - \lambda)}_{Q_\lambda(\lambda)} + \underbrace{\sum_k \sum_j \left(\sum_i p_i \mathbf{1}\{\gamma_{ik} = j\} \right) \log m_{kj}}_{Q_{m,k}(m_{k\cdot})} \\ & + \underbrace{\sum_k \sum_j \left(\sum_i (1 - p_i) \mathbf{1}\{\gamma_{ik} = j\} \right) \log u_{kj}}_{Q_{u,k}(u_{k\cdot})} \end{aligned}$$

Next, to make this form more interpretable for the EM steps, note the following interpretations from each seperable part. Given complete data, we should have the counts at level j of comparison k among matches to be $C_{kj}^m = \sum_i M_i \mathbf{1}\{\gamma_{ik} = j\}$. Taking the expectation gives:

$$E[C_{kj}^m | \gamma, \theta'] = \sum_i E[M_i | \gamma_i, \theta'] \mathbf{1}\{\gamma_{ik} = j\} = \sum_i p_i \mathbf{1}\{\gamma_{ik} = j\} = c_{kj}^m.$$

And it follows symmetrically for the u case. Within $Q_{u,k}(u_{k\cdot})$ and $Q_{m,k}(m_{k\cdot})$, we can write $c_{kj}^m = \sum_i p_i \mathbf{1}\{\gamma_{ik} = j\}$ and $c_{kj}^u = \sum_i (1 - p_i) \mathbf{1}\{\gamma_{ik} = j\}$. These represent the expected counts at level j of comparison k among matches and non-matches. Note also that $\sum_j c_{kj}^m = N_m$ and $\sum_j c_{kj}^u = N_u$ for every k , since each pair sits at exactly one level.

Now, starting at the E step, note that all we need to calculate is $p_i(\theta')$, which recall is the posterior:

$$p_i(\theta) = P(M_i = 1 | \gamma_i, \theta) = \frac{P(M_i = 1 | \theta) P(\gamma_i | M_i = 1, \theta)}{P(\gamma_i | \theta)} = \frac{\lambda \prod_k m_{k,\gamma_{ik}}}{\lambda \prod_k m_{k,\gamma_{ik}} + (1 - \lambda) \prod_k u_{k,\gamma_{ik}}}$$

This is all for step E, simply evaluating our posterior at the seed θ for $t = 0$ or evaluating it at $\theta^{(t-1)}$.

The final step is the M step, where we will maximize each seperable component. First, note the simpliest case which is lambda. Q_λ is unconstrained on $(0, 1)$ so simply:

$$\frac{\partial Q_\lambda}{\partial \lambda} = \frac{N_m}{\lambda} - \frac{N_u}{1 - \lambda} = 0 \implies \hat{\lambda} = \frac{N_m}{N_m + N_u} = \frac{1}{n} \sum_i p_i.$$

So, $\hat{\lambda}$ is the expected fration of true matches given the current posteior. As a preview to the next section, in Splink the lambda estimate can also be calculated withouts the model's assumptions and without blocking during EM.

Next are the m estimates. For these we need a Lagrangian since we have the PMF constraint:

$$\mathcal{L}_{m,k}(m_{k\cdot}, \nu_k) = \sum_j c_{kj}^m \log m_{kj} - \nu_k \left(\sum_j m_{kj} - 1 \right).$$

Solving the first-order conditions gives:

$$\begin{aligned} \frac{\partial \mathcal{L}_{m,k}}{\partial m_{kj}} = \frac{c_{kj}^m}{m_{kj}} - \nu_k = 0 &\implies m_{kj} = \frac{c_{kj}^m}{\nu_k}, \quad j = 1, \dots, J_k, \\ \frac{\partial \mathcal{L}_{m,k}}{\partial \nu_k} = - \left(\sum_j m_{kj} - 1 \right) = 0 &\implies \sum_j m_{kj} = 1. \end{aligned}$$

Substituting the first set of conditions into the second,

$$\sum_j \frac{c_{kj}^m}{\nu_k} = 1 \implies \nu_k = \sum_j c_{kj}^m = \sum_i p_i \sum_j \mathbf{1}\{\gamma_{ik} = j\} = \sum_i p_i = N_m,$$

using $\sum_j \mathbf{1}\{\gamma_{ik} = j\} = 1$ since each pair occupies exactly one level. Substituting back,

$$\hat{m}_{kj} = \frac{c_{kj}^m}{N_m} = \frac{\sum_i p_i \mathbf{1}\{\gamma_{ik} = j\}}{\sum_i p_i}.$$

So, $\hat{m}_{kj}$ is the expected fractions of matches that have level j for comparison k , given the previous theta estimates. These are the parameters that Splink actually estimates with EM.

Finally, the estimates for the u parameters are symmetrical:

$$\mathcal{L}_{u,k}(u_{k\cdot}, \mu_k) = \sum_j c_{kj}^u \log u_{kj} - \mu_k \left(\sum_j u_{kj} - 1 \right).$$

Solving the first-order conditions gives:

$$\frac{\partial \mathcal{L}_{u,k}}{\partial u_{kj}} = \frac{c_{kj}^u}{u_{kj}} - \mu_k = 0 \implies u_{kj} = \frac{c_{kj}^u}{\mu_k}, \quad j = 1, \dots, J_k,$$

$$\frac{\partial \mathcal{L}_{u,k}}{\partial \mu_k} = -\left(\sum_j u_{kj} - 1\right) = 0 \implies \sum_j u_{kj} = 1.$$

Substituting the first set of conditions into the second,

$$\sum_j \frac{c_{kj}^u}{\mu_k} = 1 \implies \mu_k = \sum_j c_{kj}^u = \sum_i (1 - p_i) \sum_j \mathbf{1}\{\gamma_{ik} = j\} = \sum_i (1 - p_i) = N_u,$$

using $\sum_j \mathbf{1}\{\gamma_{ik} = j\} = 1$ since each pair occupies exactly one level. Substituting back,

$$\hat{u}_{kj} = \frac{c_{kj}^u}{N_u} = \frac{\sum_i (1 - p_i) \mathbf{1}\{\gamma_{ik} = j\}}{\sum_i (1 - p_i)}.$$

So, $\hat{u}_{kj}$ is the expected fraction of non-matched that have the level j for comparison k , given the previous theta estimates.

The EM algorithm in Splink does many iterations of these calculations, using the seed $\theta^{(0)}$ then the previous run's $\theta^{(t-1)}$ as the given priors, until $\max |\theta^{(t)} - \theta^{(t-1)}| < 10^{-4}$ or $t = 25$ which implies EM didn't converge.

With EM covered, I'll now cover how Splink makes a few changes to improve EM stability and make EM feasible for large datasets.

Splink workflow

Splink estimates u by random sampling and λ from deterministic match rules, and then we hold those values fixed during EM. The point of these external estimates is to constrain the mixture. With u and λ free, EM can ascend toward any two-component partition of the pairs that explains the joint frequencies of comparison vectors, and such a partition need not be match/non-match (e.g., same household/different household, clean data/messy data, etc.). Fixing u to the non-match distribution pins one component and fixing λ fixes the other, so the only partition consistent with the data would be match/non-match (given the CI assumption holds), and the stationary point EM reaches is an estimate of m for that partition rather than for some other split.

The way that lambda is estimated depends on 1) a set of deterministic rules which are strict enough such that every pair satisfying the rules is assumed to be a true match and 2) a recall estimate that represents the number of true matches the deterministic rules capture out of all the true matches. The lambda estimate then uses those two values, and the total number of pairs possible, to estimate the total matches over total pairs (which is lambda). Formally, with two datasets A and B of size $|A|$ and $|B|$:

$$\hat{\lambda} = \frac{\# \text{ of pairs satisfying strict rules/recall}}{|A| \cdot |B|}$$

The maintainers of splink mention that recall estimates are robust to misestimation. The first reason is because all pairs get the same lambda so the order is unaffected by a misestimation, only the

threshold is affected. The second reason depends on the effect size. Recall the total match weight for a pair (odds in log space):

$$w_i = \underbrace{\log_2 \frac{\lambda}{1-\lambda}}_{\text{prior weight}} + \sum_k \log_2 \frac{u_{k,\gamma_{ik}}}{m_{k,\gamma_{ik}}}$$

and recall that $\lambda \propto 1/\text{recall}$ as above. So guessing recall wrong by a factor of r makes the estimated $\hat{\lambda}$ wrong by a factor of $1/r$. Then since λ is very small we know $\lambda/(1-\lambda) \approx \lambda$. So then the prior weight above, with the recall misestimate, can be rewritten as $\log \lambda + \log r$. So then if we guess 0.85 and the truth is 0.6 then the prior weight is ~ 0.5 bits too high. If the truth is 0.95, then the prior weight is ~ 0.16 too low. Given that an exact match on DOB could add 10+ bits, a mis-estimate in recall will give a slight miscalibration in match probability.

The next pre-EM estimated parameter is u . The estimation strategy depends on λ being very small, typically $\sim 1 \cdot 10^{-6}$. Given that, if we take a large random sample of x records from the left and right dataset (or one sample from the same dataset if it's a deduplication, not a merge), a very small proportion will be matches. So, we can assume that every pair is a non-match and get a very good estimate. The way it actually works in Splink is that a number of records is drawn by Bernoulli sample with a proportion determined by the `max_pairs` set in the config using `proportion = sqrt(max_pairs / (n_left * n_right))` (to get proportion that results in correct number of max pairs). When it dedups, it inverts `max_pairs = r(r-1)/2` to get r which is the number of records to pair to get `max_pairs` without order or self-pairing. Splink then takes the full cross product between the two samples (or cross product of one sample without self-merge in dedup case). After, splink then converts each pair into its comparison vector (γ_{ik} , the level of comparison k attained by pair i , with $\gamma_{ik} = -1$ when either record is null on the field). Let $\mathcal{S}$ denote the set of sampled pairs, $|\mathcal{S}| \approx n$ (the `max_pairs` target), and let $\mathcal{S}_k = \{i \in \mathcal{S} : \gamma_{ik} \neq -1\}$ be the pairs on which comparison k is non-null. The estimated u values are then simply level shares among non-null sampled pairs:

$$\hat{u}_{kj} = \frac{\sum_{i \in \mathcal{S}_k} \mathbf{1}\{\gamma_{ik} = j\}}{|\mathcal{S}_k|}, \quad j \in \{0, 1, \dots, J_k - 1\},$$

The three things to note about this estimation is that 1) there are still a few matches within the sample (so if `max_pairs` = 1e8, then records sampled is $\sim 14k$ so the expected number of true pairs inside with $\lambda = 10^{-6}$ is 100 vs 99,999,900 true non-matches), 2) u 's are held fixed during EM if they are estimated by random sampling first, 3) After the cross product, pairs that share a record are not independent draws, so the precision of $\hat{u}_{kj}$ scales with the number of sampled records ($\propto \sqrt{\text{max_pairs}}$), not the number of pairs.

Finally, the main star of the show are the m parameter estimations. First, consider the case where we used every record from every dataset to run EM. The cross product results in 10^{14} pairs which need to be built into comparison vectors (scored on JW, DL, etc.) and aggregated into counts. Even optimistically, this is an unrealistic amount of compute. To solve this issue, Splink uses blocking

for EM. Statistically, this is also more efficient since a pair that disagrees on every comparison feature won't be a match anyway. Blocking in this case means creating a set of multiple blocking sets, where each blocking set runs EM once to produce m estimates. The catch is two fold: 1) if you block on a variable (say first name), an unadjusted prior (λ) will be too low since the prior needs to account for a matching first name and 2) we cannot estimate a m parameter for a variable that we are blocking on. Therefore, we need to adjust the prior to be reasonable within a blocked set of pairs and need to run multiple EM runs with different blocking variables to estimate all the m parameters. Also, running multiple EM sessions with different blocks helps to see if the blocks themselves are dramatically changing the m parameter estimation, which I'll cover later.

The main adjustment needed then is to correctly adjust the prior within a EM block to account for the change in distribution from the blocking. However, we have a chicken and the egg situation where we haven't quite learned the parameters needed to adjust the prior. Therefore, Splink will use the seed values or the running estimates of the parameters so far. In addition, Splink will automatically use the exact-level parameters from the category to adjust it. So for example, if the first EM pass is just blocked on exact date of birth, then Splink will use the seed parameters from the dob (or the in-progress learned parameters if it is not the first EM pass) to adjust the prior. However, during each m step of EM, the prior is updated to be consistent with the blocked distribution (the sampled deterministic lambda is actual one used during prediction, these ones are changing to support the m estimation). Therefore the adjusted block prior is only needed to point EM in the best general direction to start. Thus, it is likely a bad idea to fix the prior during EM because the prior must then rely on previous or seed estimates to adjust the total sample prior. This is especially true if some of the feature levels are complex (in that they use multiple variables in non-trivial ways), since blocking on any variable used in that feature will not adjust the prior correctly. As far as I can tell, it will only adjust the prior when blocked on a variable for which there exists a level in the comparisons for that variable with a strict $col_l = col_r$ or an AND of them (and then all the cols in the AND would need to be a part of the blocking specification). In the cases, where the blocking adjustment does take place, then adjusted prior becomes the following, noting that we once again rely on conditional independence for the result and eligible means Splink actually adjusts for the variable being blocked based on the above criteria:

$$\pi_{\text{session}} = \pi \times \prod_{k \in \text{blocked and eligible}} K_{k,\text{exact}} \quad \left(\text{so } \lambda_{\text{session}} = \frac{\pi_{\text{session}}}{1 + \pi_{\text{session}}} \right),$$

or to put it into log odds form:

$$\log \pi_{\text{session}} = \log \pi + \sum_{k \in \text{blocked and eligible}} \log K_{k,\text{exact}}$$

The final nuance to splink's approach is term-frequency (TF) adjustments, which allow for common names to count less on agreement than rarer names. TF adjustments are used in training and prediction, I'll cover training here and prediction below. To understand how TF adjustment plays a part in the training process consider the restatement of the model below.

In Splink, TF adjustments only happen for a variable when a pairs has an exact match on it. For the sake of explination, consider the first name as the variable that we will TF adjust. Lets first focus on how the u parameter changes in TF adjustment. For two datasets A and B with the first name from in a pair being from A being a and the first name in a pair from B being b , we can write the following:

$$u_{\text{exact}} = P(\text{first name agree} \mid M = 0) = \sum_v u_{\text{exact}}(v), \quad u_{\text{exact}}(v) = P(a = v, b = v \mid M = 0).$$

Then using the independence assumption from the fact that the two records belong to different people so their first names are independent (breaking with siblings or relatives, which is a fine expection since almost all random paris are unrelated):

$$P(a = v, b = v \mid M = 0) = P(a = v \mid M = 0) P(b = v \mid M = 0)$$

But since we know that a random pair is normally a match with probability around 10^{-8} , we can make the assumption that a random record drawn from the file is, to that precision, approximatly:

$$P(a = v \mid M = 0) \approx p_A(v), \quad P(b = v \mid M = 0) \approx p_B(v)$$

So then:

$$u_{\text{exact}}(v) \approx p_A(v)p_B(v)$$

With $p_A(v)$ representing the probability of seeing value v in the first names of dataset A, and parallelly for B. Then, summing over values recovers the ordinary exact level $u_{\text{exact}} = \sum_v u_{\text{exact}}(v) = \sum_v p_A(v) p_B(v)$.

Now, we hit more barriers with TF adjustment for m . The conceptual goal is to split the exact level defined in the first name comparison list into sub-levels such that each sub-level represents “exact and the shared value is some v ”. So starting with:

$$m_{\text{exact}} = P(\text{the two records' values agree} \mid \text{the pair is a match})$$

We can use the sub-level framework to re-write as:

$$m_{\text{exact}} = \sum_v m_{\text{exact}}(v), \quad m_{\text{exact}}(v) = P(a = v, b = v \mid M = 1)$$

Let $q(t) = P(T = t \mid M = 1)$ where T represents a matched person’s first name. We can then decompose the above probability into:

$$\begin{aligned}
m_{\text{exact}}(v) &= P(a = v, b = v \mid M = 1) = \sum_t P(T = t \mid M = 1) P(a = v, b = v \mid T = t, M = 1) \\
&= \sum_t q(t) P(a = v, b = v \mid T = t)
\end{aligned}$$

Then since $t = v$ will dominate the sum, assuming that coincidental identical typos that happen to each side are negligible, we can approximate it as:

$$m_{\text{exact}}(v) \approx q(v) c(v), \quad c(v) = P(\text{both records spell it correctly} \mid \text{true name } v)$$

So then we could turn it back into the pooled parameter with $m_{\text{exact}} = \sum_v q(v) c(v)$, which can be interpreted as the average recording accuracy, weighted by how many matched people have each name. Here we use the assumption that recording accuracy doesn't depend on name, so that $c(v) = c$ for every v . This allows us to make the nice simplification:

$$m_{\text{exact}} = \sum_v q(v) c = c \sum_v q(v) = c$$

Which allows for:

$$m_{\text{exact}}(v) \approx q(v) c(v) = q(v) c = m_{\text{exact}} q(v)$$

The necessity here is that EM can't estimate $c(v)$ separately for each name, it only has one m_{exact} to fit, so it uses the accuracy assumption.

The Bayes factor for agreeing on v is then the ratio of the two “sub-level” probabilities:

$$K_{\text{exact}}(v) = \frac{m_{\text{exact}}(v)}{u_{\text{exact}}(v)} \approx \frac{m_{\text{exact}} q(v)}{p_A(v) p_B(v)}.$$

This still involves three different name distributions: q (names among true matches), p_A and p_B . Splink can't observe q , since it doesn't know which pairs are matches. So we make an assumption that matchability (i.e., having a match in the pairing universe) is independent of the actual name. Then, q is just the name distribution among the A records that have a partner which is equal to the name distribution among the A records, by the assumption. So $q \approx p_A$ and we get:

$$K_{\text{exact}}(v) = \frac{m_{\text{exact}} p_A(v)}{p_A(v) p_B(v)} = \frac{m_{\text{exact}}}{p_B(v)}.$$

One way to interpret this Bayes factor is to fix one record applicant names Maria in dataset A and ask how surprising it is that a birth index record, dataset B, also says Maria. If it's the true partner BI record, then it says it with probability m_{exact} given the recording assumptions. If it's not her true partner, then it will say Maria with $p_B(\text{Maria})$, thus creating the correct odds accounting for name

frequency. Ideally we would have p_B but instead we approximate it by pooling the two datasets together and finding the frequency of each name, notated by $tf(v)$ where $|N_A|$ is the number of non-null first names in A and n_v is the number of records across A and B that have v as thier first name:

$$p_B(v) \approx tf(v) = \frac{n_v}{|N_A| + |N_B|}$$

We pool both datasets here, because we could make an identical arguement, starting with fixing a record in B and then finding the tf in A. So in reality, we need $p_B(v) \approx tf(v) \approx p_A(v) \approx q(v)$, which is reasonable under the matchability independence from name assumption (noting that this is seperate from the recording accuracy assumption about which states that, conditional on true match, both records write the name identically – the matching independence is that the chances of having a partner in the pair universe doesn't depend on the name).

So we can write the final approximation that Splink uses:

$$K_{\text{exact}}(v) \approx \frac{m_{\text{exact}} tf(v)}{tf(v)^2} = \frac{m_{\text{exact}}}{tf(v)}.$$

The evidence from agreeing on v is the chance that a true match agrees, divided by how common v is.

However, Splink doesn't build a separate level for each name. It keeps the pooled exact level, with its one m_{exact} and one u_{exact} , and multiplies that level's Bayes factor by a correction for each pair:

$$K_{\text{exact}}(v) = \underbrace{\frac{m_{\text{exact}}}{u_{\text{exact}}}}_{\text{pooled level}} \cdot \underbrace{\frac{u_{\text{exact}}}{tf(v)}}_{T(v)} = \frac{m_{\text{exact}}}{tf(v)}.$$

Two things follow. First, u_{exact} cancels, so at the exact level a pair's score depends only on m_{exact} and $tf(v)$. Second, the pooled Bayes factor is the same as the TF adjusted Bayes factor for a name with frequency exactly u_{exact} . Names with $tf(v) < u_{\text{exact}}$ get $T > 1$ and gain weight, and more common names get $T < 1$ and lose weight.

In training during EM, the adjusted Bayes factor enters only in the E-step. Each pair's posterior odds pick up the factor T for every active comparison on which it agrees exactly on a TF column. So the odds and probaility become:

$$O_i^{(t)} = \frac{\lambda}{1 - \lambda} \prod_k \frac{m_{k\gamma_{ik}}^{(t-1)}}{u_{k\gamma_{ik}}} T_{k\gamma_{ik}}(i), \quad p_i^{(t)} = \frac{O_i^{(t)}}{1 + O_i^{(t)}}.$$

Because T is built from a fixed u and a count, it is a constant for each pair and is never updated. Pairs that don't agree exactly on any active TF column have $T = 1$ and get the same posterior as they would without TF. Since two pairs with the same comparison vector can now have different

posteriors, EM can no longer collapse pairs into agreement patterns, and every pair is kept as its own row. Typically, for efficiency, Splink groups the session’s pairs by their comparison vector over the active comparisons, such that pattern g holds n_g pairs. It then calculates an equivalent version of the M-step that we derived above:

$$m_{kj} = \frac{\sum_{g: g_k=j} n_g p_g}{\sum_{g: g_k \neq \text{null}} n_g p_g}$$

But with TF on, we have to go back to the original formula, $m_{kj} = \frac{\sum_{i: \gamma_{ik}=j} p_i}{\sum_{i: \gamma_{ik} \neq \text{null}} p_i}$, which is slower. Note that the lambda updates similarly change.

The M-step formula is unchanged, but it is fed the new posteriors, and the same posteriors weight every comparison. So TF on first name changes the estimates for county, gender and the other names as well as its own m_{exact} (and λ , if it isn’t fixed). What TF does not change is u since it is fixed by random sampling with TF removed, the TF values themselves, the population and blocking-adjusted priors (which use the plain m/u), and the number of parameters, since there is still only one m_{exact} per comparison. A column used in a session’s blocking rule is deactivated for that session, and its TF goes with it.

After training, the parameters EM produced are used for prediction. No further estimation happens here – the m ’s, u ’s and prior are fixed, and each candidate pair is scored once. Generally, prediction is fairly simple. It starts by generating candidate pairs with the prediction blocking rules, which are specified separately from the training rules and are made to be as loose as possible within computational limits, since their job is to capture as many true matches as possible rather than to isolate informative subsets for EM. Splink takes the union across rules, skipping any pair already produced by an earlier rule so that no pair is scored twice. A pair produced by no rule is never compared, so a true match missed by every blocking rule is lost at this stage.

Every candidate pair is then compared on all comparisons defined. Each comparison places the pair at a level given each comparison variable, and the pair’s match weight (log odds, with units of bits) is the prior plus the sum of the level weights, with the TF factor added wherever the pair agrees exactly on a TF column:

$$\text{match_weight}_i = \log_2 \frac{\lambda}{1 - \lambda} + \sum_k \left[\log_2 \frac{m_{k\gamma_{ik}}}{u_{k\gamma_{ik}}} + \log_2 T_{k\gamma_{ik}}(i) \right], \quad p_i = \frac{2^{\text{match_weight}_i}}{1 + 2^{\text{match_weight}_i}}.$$

Here λ is the population prior, not the blocking-adjusted prior of any training session, and $T = 1$ at every level without TF.

One change I made to Splink, because our set of prediction pairs to-be-scored exploded to >10 trillion, is to run prediction in a true parallel fashion – trading memory for speed. When paring dataset A and dataset B, I split the largest dataset – call it A for now – into shards, then I launch a given number of workers. Each worker reads a shard of A and the full dataset B (hence where

the memory speed tradeoff comes from). Then it produces a now smaller set of to-be-scored pairs, which is scored accordingly. Afterwards, the shared scored pairs are merged together, then I proceed with the 1:1 enforcement or the entity resolution. The only thing that I had to change in the splink workflow, was that I calculate the TF for each variable that uses TF adjustments before sharding, then feed each shard the pre-sharded TF table which it then uses for TF adjustments.