Data Programming

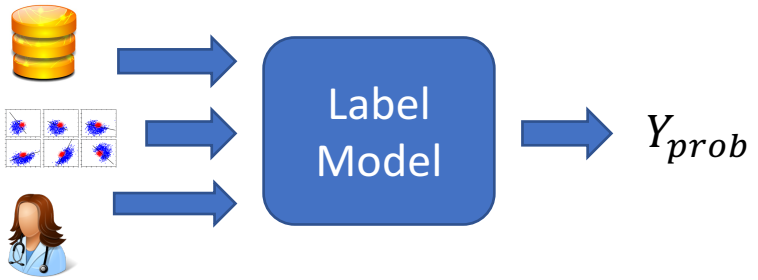
Theory Slides

Technical Challenge: Modeling Weak Supervision

- Input: **noisy, overlapping and conflicting** labels from LFs of ***unknown*** **accuracies**

- Goal: learn the accuracies of the LFs, ***without labeled data***, to reweight and combine their labels

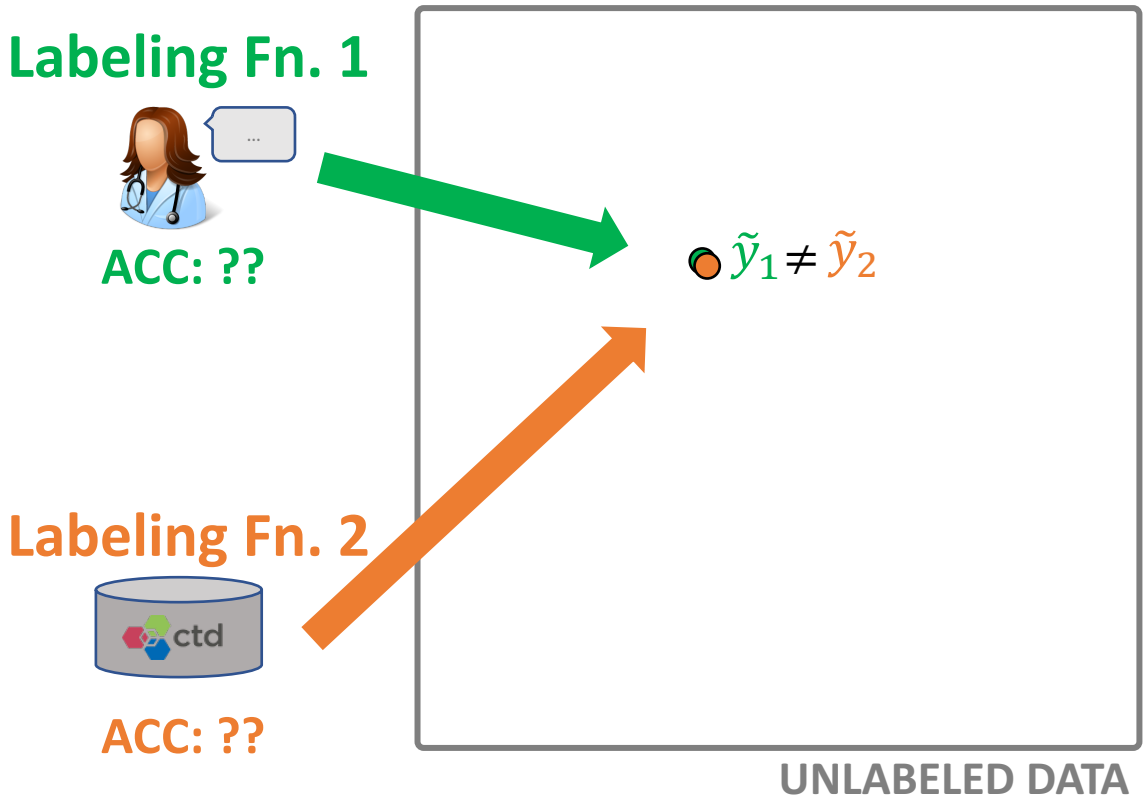
- A classic technical challenge, but with several fundamental twists…
  - Small # of LFs, each with large coverage
  - Correlated LFs
  - *Multi-task LFs*



Let's look at why this is critical…

Challenges of Weak Supervision

- Problem 1: How do we resolve conflicts between weak label sources?
  - How can we estimate their accuracies without ground truth?
- This is a real development burden that our users faced with prior "distant supervision" systems

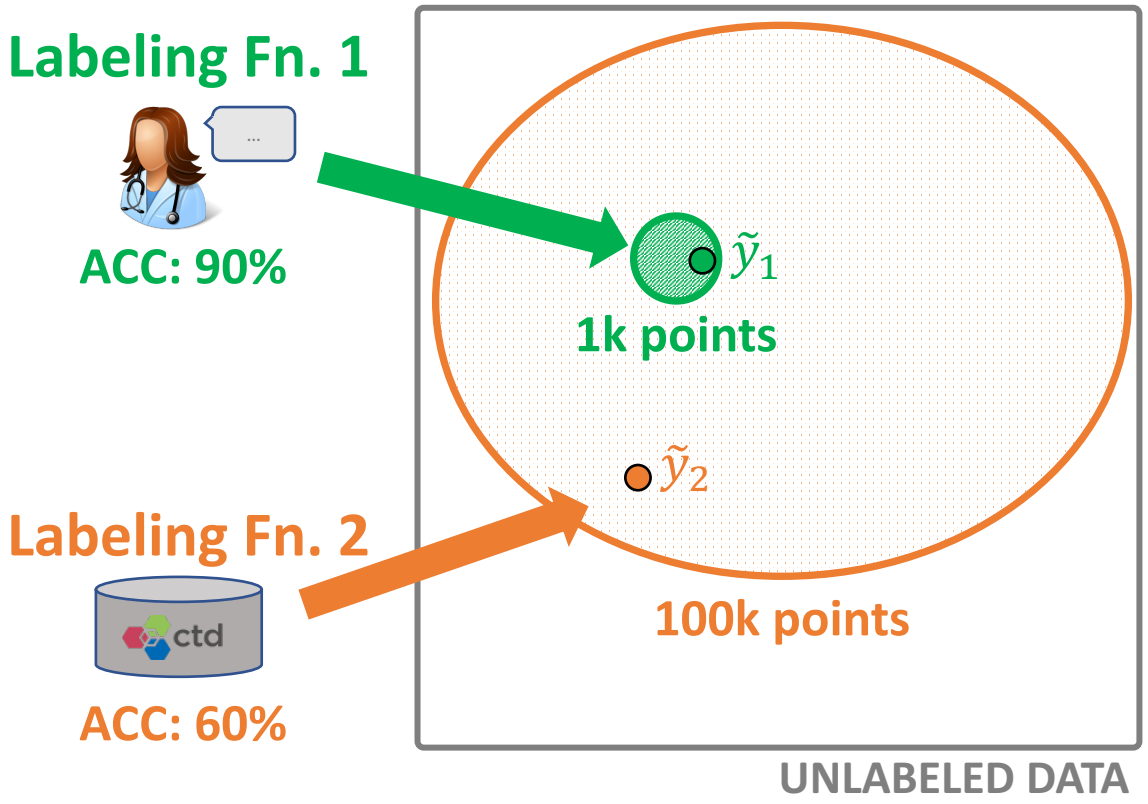


Need to be able to estimate source accuracies

Challenges of Weak Supervision

- Problem 2: Need to communicate training point lineage to model being trained

- Ex:
  - User writes one high-accuracy, low-coverage LF…
  - …and one low-accuracy, high-coverage LF
  - *If we just naively take the union of labels, expected acc. = 60.3%!*



Need to communicate training label *lineage*

Data Programming Pipeline in Snorkel

Input: Labeling Functions, *Unlabeled data*

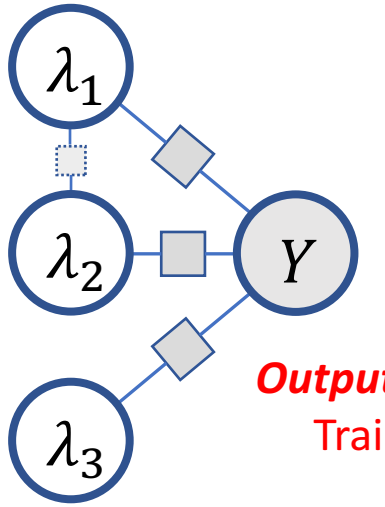
DOMAIN EXPERT

```
def lf1(x):
  cid = (x.chemical_id,
x.disease_id)
  return 1 if cid in KB else 0
```

```
def lf2(x):
  m = re.search(r'.*cause.*',
x.between)
  return 1 if m else 0
```

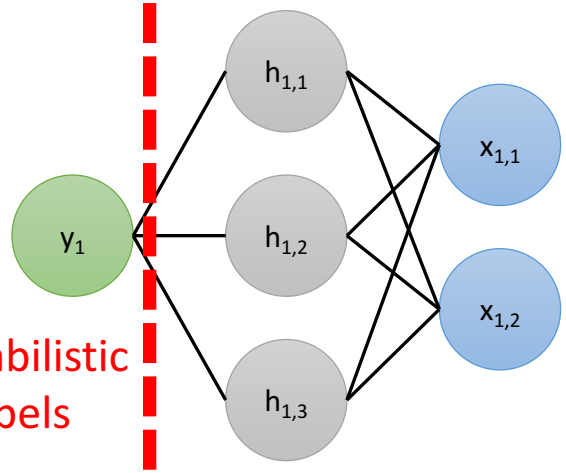
```
def lf3(x):
  m = re.search(r'.*not
cause.*', x.between)
  return 1 if m else 0
```

Label Model

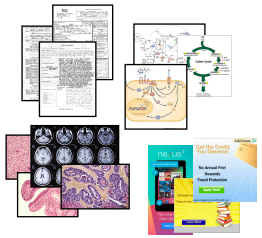


Output: Probabilistic Training Labels

End Model



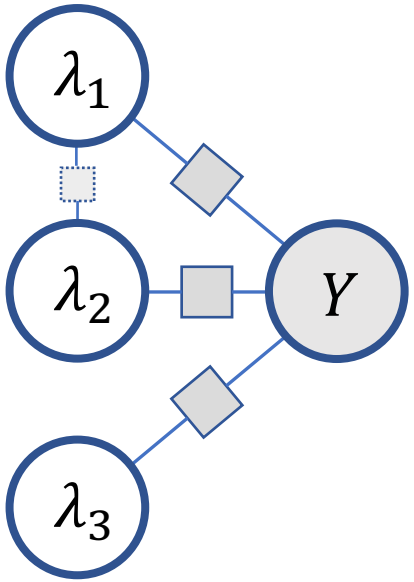
Ex. Application: Knowledge Base Creation (KBC)



1. Users write *labeling functions* to generate noisy labels
2. We model the labeling functions' behavior to de-noise them
3. We use the resulting prob. labels to train a model

Weak Supervision: Core Challenges

- Unified input format
- Modeling
  - Accuracies of sources
  - Correlations between sources
  - Expertise of sources
- Using to train a wide range of models



Modeling LFs as Noisy Voters

- We model each labeling function as having an *accuracy* α and a *labeling propensity* β*:*

$Y \in \{-1,1\}$

$$\lambda_j(x) = \begin{cases} Y & w.p. \ \beta_j \alpha_j \\ -Y & w.p. \ \beta_j(1-\alpha_j) \\ 0 & w.p. \ 1-\beta_j \end{cases}$$

- And can be correlated with other LFs…

Defining basic variables of our problem

Correctness indicator variable

$$\Lambda_i = \mathbb{I}^{\pm}\{\lambda_i(x) = y\} = \lambda_i(x)y$$

Goal: Recover the accuracies

$$\alpha_i = P(\lambda_i(x) = y \,|y, \lambda_i(x) \neq 0)$$

Prob. i^{th} LF is correct

Given it has not abstained

$$\beta_i = P(\lambda_i(x) \neq 0)$$

Prob. i^{th} LF emits label- note this is easily estimated by empirical counts!

Our goal: Estimate the mean of Λ_i

$$\mu_i = \mathrm{E}[\Lambda_i] = \beta_i(2\alpha_i - 1)$$

Problem: We can't observe Λ_i, since it depends on the ground-truth label Y!

How do we learn the accuracies without labeled data?

Approach: Analyzing the covariance matrix

From the definition- three terms:

$$\Sigma = \mathrm{E}[\Lambda\Lambda^T] - \underline{\mathrm{E}[\Lambda]\mathrm{E}[\Lambda]^T}$$

This is just the outer product of the vector of statistics μ

Approach: Analyzing the covariance matrix

From the definition- three terms:

$$\Sigma = \underline{\mathrm{E}[\Lambda\Lambda^T]} - \mu\mu^T$$

This ends up being the matrix of pairwise overlaps between the LFs- ***which is observable!***

$$\mathrm{E}\left[\Lambda_i \Lambda_j\right] = \mathrm{E}\left[(\lambda_i(x)Y)(\lambda_j(x)Y)\right] = \mathrm{E}\left[\lambda_i(x)\lambda_j(x)\right] \coloneqq O_{i,j}$$

Approach: Analyzing the covariance matrix

We now have:

$$\underbrace{\Sigma}_{\text{Covariance matrix}} = \underbrace{O}_{\text{Overlaps matrix (Observable)}} - \underbrace{\mu\mu^T}_{\text{Unknown means (rank-one)}}$$

The form is starting to look familiar… like a rank-one matrix approximation problem where we are fitting to the observed LF-LF agreement rates!

Warmup: Conditionally independent LFs

- To start, suppose that the Λ_i are ***independent***—i.e. the LFs make uncorrelated errors
- ➔ The covariance matrix is diagonal, meaning that for off-diagonal entries:

$$O_{i,j} = \mu_i \mu_j, \qquad \mathrm{i} \neq j$$

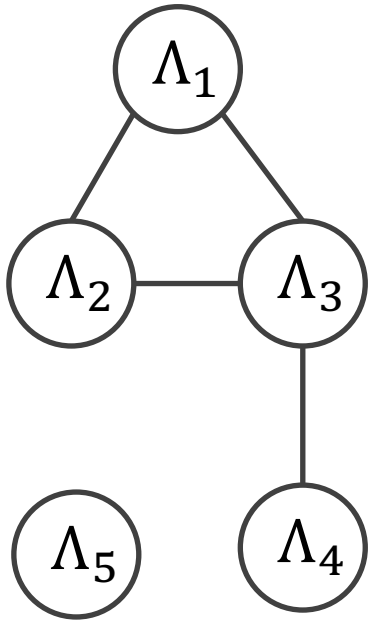
Solving as (masked) rank-one matrix approx.

- Thus, we can use e.g. SGD to solve this!

$$\underset{\mu}{\operatorname{argmin}} \|O - \mu\mu^T\|_{F,i\neq j}$$

- And can break symmetries by assuming ***the LFs are on average better than random*** (more on this later)

Handling correlations between LFs



- Now, suppose we know the correlation structure between the LFs
  - Can represent as a probabilistic graphical model (PGM)
  - Can handle arbitrary dependency structures!*
  - Define the edge set as Ω

- We will use a result from Loh & Wainwright [2013] which states that for our model, ***the inverse covariance matrix is graph structured***

**Technical detail: For this tutorial, to simplify, we'll assume the junction tree of our dependency graph has singleton separator sets*

Using the Inverse Covariance Matrix

$$\Sigma^{-1}$$
$$= (O - \mu\mu^T)^{-1}$$
$$= O^{-1} - \frac{O^{-1}\mu\mu^T O^{-1}}{1 + \mu^T O^{-1}\mu}$$ Sherman-Morrison formula
$$= O^{-1} - zz^T$$ Define $z = \frac{O^{-1}\mu}{\sqrt{c}}$ where $c = 1 + \mu^T O^{-1}\mu$

- Now, since Σ^{-1} is graph-structured (entries not corresponding to edges in the dependency graph are zero), we have:

$$O^{-1}_{i,j} = z_i z_j, \qquad (\mathrm{i}, j) \notin \Omega$$

Once again: Can solve as (masked) rank-one matrix approx. problem

$$\underset{z}{\operatorname{argmin}} \| O^{-1} - zz^T \|_{(\mathrm{i},\mathrm{j}) \notin \Omega}$$

- However, now it's less clear- is this problem identifiable?

Identifiability

- We need a compiler-like check to tell us what dependency structures lead to unique solutions… and what extra information is needed
- By taking log of squares of both sides, we see that:

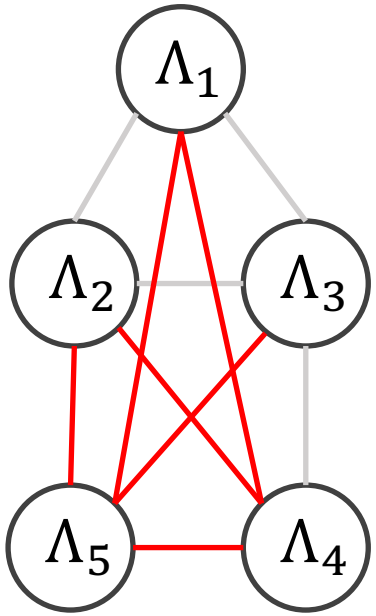
$$\log((O_{i,j}^{-1})^2) = \log(z_i^2) + \log(z_j^2), \qquad (i, j) \notin \Omega$$

- We thus represent our problem as a system of linear equations, ***up to determining the signs of the z_is***:

$$\boxed{M_\Omega l = q_\Omega,} \qquad l_i = \log(z_i^2), \qquad q_{(i,j)} = \log((O_{i,j}^{-1})^2)$$

If M_Ω is invertible, we can uniquely recover the z_i up to sign

Identifiability: Determining the Signs



- For any *connected component in the inverse dependency graph* Ω_{inv}, knowing the sign of one z_i implies all the others…
- ➔We just need to know the ***sign*** of one of the z_is for each component
- If Ω_{inv} is fully-connected (see example to left), we just pick the solution where the LFs on average are non-adversarial!

If M_Ω is invertible, and we know the sign of one z_i per connected component of Ω_{inv} , then is identifiable

Extension: Learning the Structure of Correlations with Robust PCA

- We can also jointly estimate the LF accuracies and correlation structure → this can be phrased as instance of ***robust PCA***

$$\underbrace{O^{-1}}_{\text{Observed matrix}} = \underbrace{ZZ^T}_{\text{Low-rank matrix}} + \underbrace{\Sigma^{-1}}_{\text{Sparse matrix}}$$

We can learn the correlation structure too!

Data Programming Pipeline in Snorkel

Input: Labeling Functions, *Unlabeled data*

Label Model

End Model

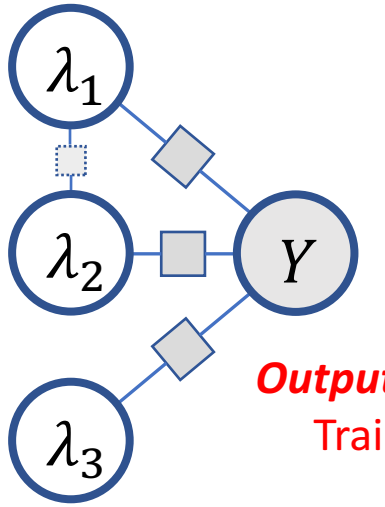
Ex. Application: Knowledge Base Creation (KBC)

DOMAIN EXPERT

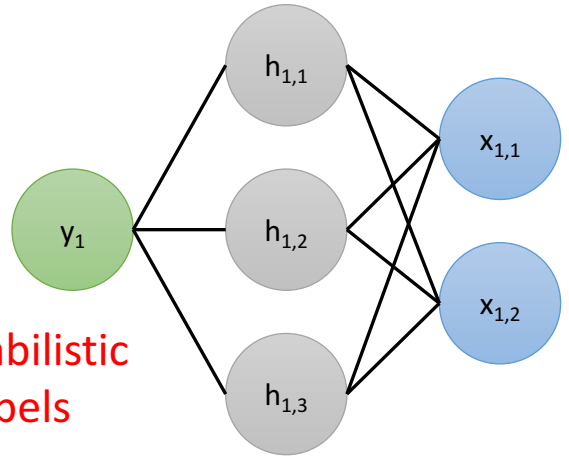
```
def lf1(x):
  cid = (x.chemical_id,
x.disease_id)
  return 1 if cid in KB else 0
```

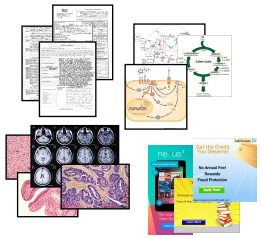
```
def lf2(x):
  m = re.search(r'.*cause.*',
x.between)
  return 1 if m else 0
```

```
def lf3(x):
  m = re.search(r'.*not
cause.*', x.between)
  return 1 if m else 0
```



Output: Probabilistic Training Labels





1. Users write *labeling functions* to generate noisy labels

2. We model the labeling functions' behavior to de-noise them

3. We use the resulting prob. labels to train a model

Training a Noise-Aware Model

In a supervised learning setting, we would learn from ground-truth labels:

$$\hat{w} = \text{argmin}_w \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \sigma_t \, l_t(w, x^{(i)}, y_t{}^{(i)})$$

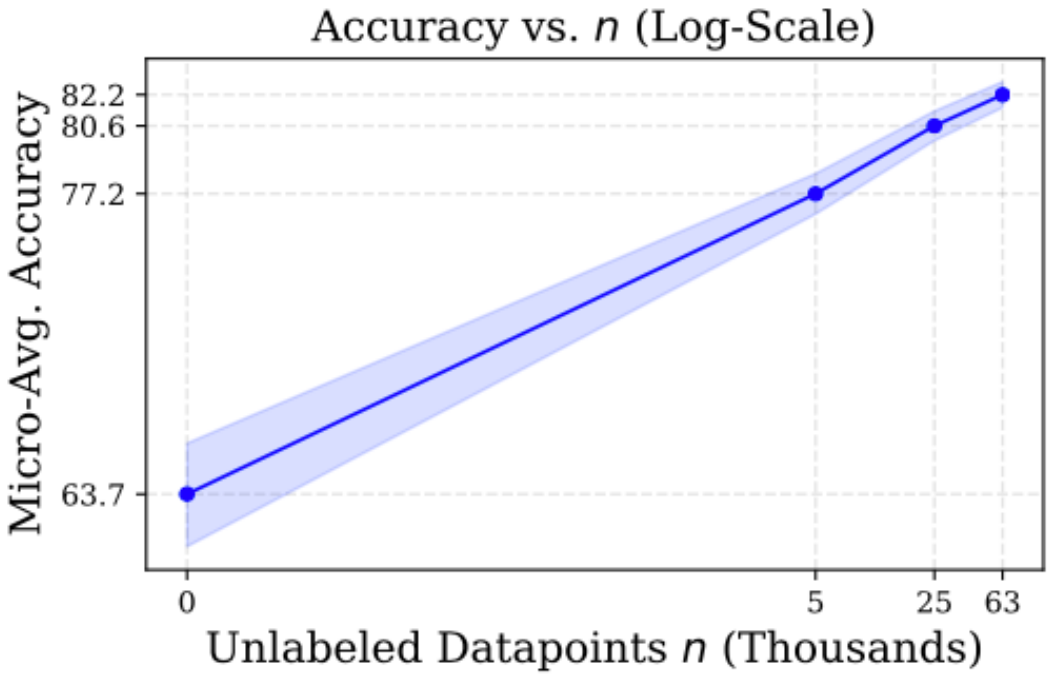
Here, we learn from the *noisy* labels:

$$\hat{w} = \text{argmin}_w \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \sigma_t \, \mathbb{E}_{\tilde{y} \sim \boldsymbol{p_\theta}} [\boldsymbol{l_t}(w, x^{(i)}, \tilde{y})]$$

Only requires simple tweak to loss function works over ***many models*** including Logistic Regression, SVMs and LSTMs.

Theoretical Result I: Scaling with Unlabeled Data

- Theory:
  - Given at least three LFs
  - And dependent on the number of LFs and the structure of their dependencies
  - ➔ end model accuracy should scale with # of *unlabeled* data points n, at same asymptotic rate as in supervised setting!
- Empirically: We indeed see this scaling with unlabeled data!

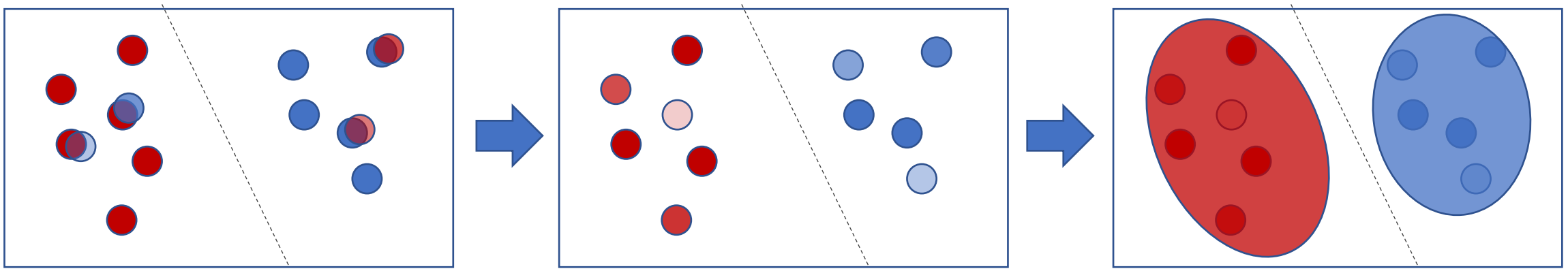


Goal: Training End Model to Generalize

Input: Labeling Functions,
Unlabeled data

Label Model

End Model



Noisy, conflicting labels

Resolve conflicts,
re-weight & combine

Generalize beyond the
labeling functions

Theoretical Results II: End model generalization

- Key question: Why should we expect the end model to generalize beyond the LFs?

- One key idea: **Inductive bias of the end model**
  - Ex: L2 reg. = bias to spread mass across many features (see right)
  - Ex: Semantic generalization via e.g. pre-trained word embeddings
  - Etc.

- Theory results here coming soon!