MACHINE LEARNING
UNDER HUMAN GUIDANCE

Eric Nalisnick
Assistant Professor of Machine Learning
University of Amsterdam
Fall 2023

Contents

1 Background

We will consider both supervised learning and reinforcement learning and hence give brief introductions below.

1.1 Supervised Learning

The goal of supervised learning is to map feature vectors, the independent variable, to labels or responses, the dependent variable. Let $\mathcal{X}$ denote the feature space, and let $\mathcal{Y}$ denote the label / response space. $\mathbf{x}_n \in \mathcal{X}$ denotes a feature vector, and $\mathrm{y}_n \in \mathcal{Y}$ denotes the associated response / label defined by $\mathcal{Y}$. The N-element training sample is then $\mathcal{D} = \{\boldsymbol{x}_n, y_n\}_{n=1}^N$. We will almost always work with a probabilistic formulation, using the negative log-likelihood (NLL) as the training objective. Given a paramerization $f(\mathbf{x}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are the model parameters, we can follow the framework of generalized linear models, letting $\mathbb{E}\mathrm{y}|\mathbf{x} = f(\mathbf{x}; \boldsymbol{\theta})$. The NLL is then:

$$\begin{aligned} \ell(\boldsymbol{\theta}; \mathcal{D}) &= -\log \left\{ \prod_{n=1}^{N} p(\mathrm{y}_n | \mathbf{x}_n, \boldsymbol{\theta}) \right\} \\ &= \sum_{n=1}^{N} -\log p(\mathrm{y}_n | \mathbf{x}_n, \boldsymbol{\theta}). \end{aligned} \tag{1}$$

When $\mathcal{Y} = \mathbb{R}$, real numbers, then the supervised learning problem is usually called *regression*, and when $\mathcal{Y}$ denotes discrete variables, then the task is *classification*. It is an easy calculation to show that, for the case of regression, plugging in the normal distribution to the NLL above yields the squared-error loss function. Similarly, for classification, plugging in the categorical distribution results in the cross-entropy loss function.

1.2 Reinforcement Learning

Reinforcement learning (RL) considers sequential decision-making problems in which models take actions so that their reward is maximized in an environment. Let $s \in \mathcal{S}$ be the space of states, and let $a \in \mathcal{A}$ be the space of actions. RL usually consider the underlying generative model to be a *Markov Decision Process* defined by

$$\mathrm{s}_{t+1} \sim \mathbb{P}(\mathrm{s}_{t+1} | \mathrm{s}_t, \mathrm{a}_t), \qquad r_t = \mathcal{R}(\mathrm{s}_t; \mathrm{a}_{t-1}, , \mathrm{s}_{t-1})$$

where $\mathbb{P}(\mathrm{s}_{t+1}|\mathrm{s}_t, \mathrm{a}_t)$ is the *transition probability* of moving from state s_t to s_{t+1} by taking action a_t. Moreover, $\mathcal{R}(\mathrm{s}_t; \mathrm{a}_{t-1}, , \mathrm{s}_{t-1})$ is a *reward function* that returns the reward of transitioning to s_t from s_{t-1} using action a_{t-1}. The reward not need be a function of the previous state and action, e.g. $r_t = \mathcal{R}(\mathrm{s}_t)$. The goal of RL is to obtain a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that determines the appropriate action to take (such that long-term reward is maximized) when in a given state: $\pi(\mathrm{a}|\mathrm{s}) = \Pr(\mathrm{a}_t = a | \mathrm{s}_t = s_t)$. The policy is determined by parameters $\boldsymbol{\theta}$, so we write $\pi_{\boldsymbol{\theta}}(\mathrm{a}|\mathrm{s})$. Collecting a T-length (with T possibly being infinite) series of state-action pairs

$$\{(s_t, a_t)\}_{t=1}^{T} \quad \text{where} \quad a_t \sim \pi_{\boldsymbol{\theta}}(\mathrm{a}|s_t), \quad s_{t+1} \sim \mathbb{P}(\mathrm{s}_{t+1}|s_t, a_t)$$

is called *rolling out a policy*, or a *rollout*.

Optimization Objective The goal is to learn a policy that maximizes *return*, which is a function of a state s_t:

$$G\left(\mathrm{s}_t; \{\mathrm{a}_t, \mathrm{a}_{t+1}, \ldots\}\right) \;=\; \sum_{t'=0}^{\infty} \gamma^{t'} \cdot r_{t+t'}$$

where $\gamma \in [0, 1)$ is a *discount factor* that emphasizes near-term rewards. The return compute the future rewards $r_{t+t'}$ are obtained by following actions $\{\mathrm{a}_t, \mathrm{a}_{t+1}, \ldots\}$ from an initial state s_t. Given the definition of return, we can then quantify the utility of a given state via the *value function*. It quantifies the expected return from a given state:

$$\begin{aligned} V^{\pi}(s) \;&=\; \mathbb{E}_{\pi}\left[\; G\left(\mathrm{s}_t; \{\mathrm{a}_t, \mathrm{a}_{t+1}, \ldots\}\right) \;\middle|\; \mathrm{s}_t = s \;\right] \\ &=\; \sum_{a \in \mathcal{A}} \pi(a|s) \cdot \mathbb{E}_{\pi}\left[\; G\left(\mathrm{s}_t; \{\mathrm{a}_t, \mathrm{a}_{t+1}, \ldots\}\right) \;\middle|\; \mathrm{s}_t = s, \mathrm{a}_t = a \;\right] \end{aligned}$$

where the expectation is taken over rollouts of π. The final RL optimization objective, known as the *Bellman equation*, is to find the policy whose stationary distribution over states places high probability on 'valuable' states:

$$\begin{aligned} \mathcal{J}(\boldsymbol{\theta}) \;&=\; \sum_{s \in \mathcal{S}} d^{\pi(\boldsymbol{\theta})}(s) \cdot V^{\pi(\boldsymbol{\theta})}(s) \\ &=\; \sum_{s \in \mathcal{S}} d^{\pi(\boldsymbol{\theta})}(s) \sum_{a \in \mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \cdot \underbrace{\mathbb{E}_{\pi(\boldsymbol{\theta})}\left[\; G\left(\mathrm{s}_t; \{\mathrm{a}_t, \mathrm{a}_{t+1}, \ldots\}\right) \;\middle|\; \mathrm{s}_t = s, \mathrm{a}_t = a \;\right]}_{Q^{\pi(\boldsymbol{\theta})}(s,a)} \\ &= \sum_{s \in \mathcal{S}} d^{\pi(\boldsymbol{\theta})}(s) \sum_{a \in \mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \cdot Q^{\pi(\boldsymbol{\theta})}(s, a) \end{aligned} \tag{2}$$

where $d^{\pi(\boldsymbol{\theta})}$ is the *stationary distribution* under policy $\pi_{\boldsymbol{\theta}}$, meaning it is the marginal distribution over state visitations. The expectation within the value function has its own name, the *Q-function*, that quantifies the value of a state-action pair. I write $\pi(\boldsymbol{\theta})$ in the superscripts to emphasize that these quantities are dependent upon the parameters that are being optimized. Ideally, we want to find a policy such that

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta} \in \Theta} \mathcal{J}(\boldsymbol{\theta}).$$

There are many approaches to this optimization problem, of course, with crucial decisions needed about how to calculate various statistics of the states visited during finite-sample rollouts. We will mostly ignore these details, referring the reader to any standard text on RL if needed.

Entropy-Regularized RL Often MDPs are non-smooth and in need of regularization. Thus entropy-regularized MDPs are often considered, which use the modified reward function:

$$\mathcal{R}_{\mathbb{H}}(\mathrm{s}_t; \mathrm{a}_{t-1}, \mathrm{s}_{t-1}, \lambda) \;=\; \mathcal{R}(\mathrm{s}_t; \mathrm{a}_{t-1}, , \mathrm{s}_{t-1}) \;+\; \lambda \cdot \mathbb{H}\left[\pi(\mathrm{a}_t|\mathrm{s}_t)\right]$$

where $\lambda \in \mathbb{R}^+$ is a weighting constant and $\mathbb{H}[\pi(\mathrm{a}_t|\mathrm{s}_t)]$ is the entropy of the distribution over actions at the current state. Hence the modified reward encourages policies that not only obtain high reward but also lead to states that are not 'dead ends,' meaning that there are relatively many actions with non-negligible probability with which to transition out of the current state. Under entropy-regularization, the optimal policy has a nice (yet self-referential) form:

$$\begin{aligned} \pi^*(\mathrm{a}|s) &= \frac{\exp\left\{\lambda^{-1} \cdot Q^{\pi^*}(s,a)\right\}}{\sum_{a \in \mathcal{A}} \exp\left\{\lambda^{-1} \cdot Q^{\pi^*}(s,a)\right\}} \\ &= \exp\left\{\lambda^{-1}\left(Q^{\pi^*}(s,\mathrm{a}) - V^{\pi^*}(s)\right)\right\}. \end{aligned} \tag{3}$$

We see that the optimal policy is the exponentiated difference between the Q- and value functions.

Advantage Function For a reference policy π, we may at times be curious what could be gained by taking an alternative action, possibly sampled from a difference policy: $\mathrm{a} \sim \pi'$. This can be quantified by the *advantage function*, which is defined as:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s).$$

We see it is the difference between the Q-function, evaluated at the current state and the action under consideration, and the value function defined from the reference policy π. Comparing this equation to Equation 4, we see that the optimal policy under entropy regularization is just the exponentiated advantage function:

$$\begin{aligned} \pi^*(\mathrm{a}|s) &= \exp\left\{\lambda^{-1}\left(Q^{\pi^*}(s,\mathrm{a}) - V^{\pi^*}(s)\right)\right\} \\ &= \exp\left\{\lambda^{-1} \cdot A^{\pi^*}(s,a)\right\} \end{aligned} \tag{4}$$

where λ is again the weight on the entropy term.

Performance Difference Lemma Lastly, the advantage function also has a useful theoretical property, which we show below in the form of the *Performance Difference Lemma* (PDL) (Kakade and Langford, 2002).

Theorem 1.1. Performance Difference Lemma: For two policies π and π', the difference between their value functions is equivalently expressed in terms of the advantage function:

$$V^{\pi'}(s) - V^\pi(s) = \frac{1}{1-\gamma}\mathbb{E}_{\mathrm{s}\sim d(\mathrm{s};\pi')}\mathbb{E}_{\mathrm{a}\sim\pi'}\left[A^\pi(\mathrm{s},\mathrm{a})\right] \tag{5}$$

where γ is the discount, $d(\mathrm{s};\pi')$ is the stationary distribution over states from following π', and $A^\pi(\mathrm{s},\mathrm{a})$ is the advantage function under π.

See pages 6 and 7 of these notes for a proof: `https://nanjiang.cs.illinois.edu/files/cs542f22/note1.pdf`. An alternative, more explicit proof can be found here: `https://wensun.github.io/CS4789_data/PDL.pdf`. The intuition behind the PDL is that the

difference in the value of two policies can be expressed as the advantage of one policy under rollouts (i.e the expectation) of the other. This will be a useful tool when comparing, for example, a policy found through an approximation vs an optimal one.

2 Supervised Learning from Human-Generated Labels

When conducting supervised learning, we often don't have access to some objective mechanism for obtaining the labels. Rather, we need to ask usually several human annotators to provide one. This process is also known as *crowdsourcing* and whole platforms such as *Amazon Mechanical Turk* exist in order to make it easy to query human annotators at scale. Specifically, assume we have N feature vectors $\{\boldsymbol{x}_n\}_{n=1}^N$, pass them to L annotators, and they each produce a response $\mathbf{y}_{n,l} \in \{0,1\}^K$, where K is the total number of classes (i.e. a multi-class classification task) and $\mathbf{y}_{n,l,k} = 1$ denotes that the annotator has produced a label for the kth class. All other dimensions are zero.

We can, of course, train a supervised model on this data directly by assuming each response is an independent observation:

$$p\left(\{\boldsymbol{y}_{n,1},\ldots,\boldsymbol{y}_{n,L}\}_{n=1}^N|\boldsymbol{x}_n,\ldots,\boldsymbol{x}_N\right) = \prod_{n=1}^N\prod_{l=1}^L p\left(\boldsymbol{y}_{n,l}|\boldsymbol{x}_n\right),$$

but this model is in jeopardy of being mislead by noisy labelers. Alternatively, one could use all data, but give a weight to each annotator that reflects their quality:

$$\prod_{n=1}^N\prod_{l=1}^L \left[\left(p\left(\boldsymbol{y}_{n,l}|\boldsymbol{x}_n\right)\right]^{w_l}, \text{ where } w_l \geq 0.\right.$$

This is a better option, but the weights w_l are hard to determine without access to a ground-truth label to separate the experts from the incompetent labelers.

Given these challenges of using all provided labels, methods that infer some ground-truth label have received much attention. The simplest and most common of these methods—already alluded to above—is majority voting:

$$\hat{t}_{n,k} \;=\; 1 \quad \text{if} \;\; k \;=\; \underset{k\in[1,K]}{\arg\max} \sum_{l=1}^L y_{n,l,k}.$$

All methods for inferring ground-truth work by checking for some form of consensus across the annotators. This has the drawback that there may be one expert annotator and four incompetent ones, but if the incompetent ones all agree, then the expert will be seen as an outlier and likely in the wrong. But, of course, this problem is nearly impossible to prevent without an external signal of quality.

Below we will consider more sophisticated models that estimate a confusion matrix to understand how the crowd is generating their labels. A crucial modeling in this setting is whether to model the ability of the human labelers. Doing so, of course, incurs a significant modeling and computational overhead, which can lead one astray if the number of labels observed per annotator is small. Below we present two common probabilistic models for determining consensus labels in crowdsourcing applications—pooled and unpooled. Here,

'pooling' refers to whether we pool all annotators together or try to model their individual abilities.

2.1 Pooled: Multinomial Model

The simplest model of this annotation scheme is a *pooled multinomial* model—*pooled* because we don't model individual annotators. Rather, we assume we have $K \times K$ parameters $\pi_{k,j} \in [0, 1]$, which represents the probability that the labelers will return class j when the true class is k. This implies that these parameters are normalized across potential mislabelings: $\sum_{j=1}^{K} \pi_{k,j} = 1$. We can think of the full $K \times K$ matrix of parameters, denote it as $\mathfrak{C}$, as a *confusion matrix*, again, representing the crowd's mislabeling tendencies. We will see how to extend this model to assess the quality of individual annotators in the next sub-section.

Observed Truth We first discuss an easier case of the model above where we assume that we know the true annotation, denoted by an indicator vector $\boldsymbol{t}$ such that $t_k = 1$ denotes that the true class is k and all other dimensions are zero. The generative process can then be written as:

$$\mathbf{t}_n \sim \text{Categorical}(\text{p}^*), \quad \mathbf{y}_{n,l} \sim \text{Categorical}\left(\mathbf{y}_{n,l};\ \langle \mathfrak{C}, \mathbf{t}_n \rangle\right), \forall\ n \in [1, N], l \in [1, L] \tag{6}$$

where p^* are the underlying truth (i.e. class) probabilities, which are unknown. The inner product $\langle \mathfrak{C}, \mathbf{t} \rangle = \boldsymbol{\pi}_k$, and thus we can think of $\mathbf{t}$ as an indicator vector, selecting out a particular row of $\mathfrak{C}$ corresponding to the ground-truth class. Assume we have observed a data set $\mathcal{Y} = \left\{\{\boldsymbol{y}_{n,l}\}_{l=1}^{L}\right\}_{n=1}^{N}$ and the corresponding ground-truths $\boldsymbol{T} = \{\boldsymbol{t}_n\}_{n=1}^{N}$. We can write the log-likelihood for the above model as:

$$\begin{aligned}
\ell\left(\{\pi_{k,j}\}_{k,j=1}^{K}; \mathcal{Y}, \boldsymbol{T}\right) &= \log\left\{\prod_{n=1}^{N}\prod_{k=1}^{K}\prod_{l=1}^{L} p\left(\boldsymbol{y}_{n,l} | t_{n,k}\right)\right\} \\
&= \log\left\{\prod_{n=1}^{N}\prod_{k=1}^{K}\left[\prod_{l=1}^{L}\text{Categorical}\left(\boldsymbol{y}_{n,l}; \boldsymbol{\pi}_k\right)\right]^{t_{n,k}}\right\} \\
&= \log\left\{\prod_{n=1}^{N}\prod_{k=1}^{K}\left[\prod_{l=1}^{L}\prod_{j=1}^{K}\pi_{k,j}^{y_{n,l,j}}\right]^{t_{n,k}}\right\} \\
&= \log\left\{\prod_{n=1}^{N}\prod_{k=1}^{K}\left[\prod_{j=1}^{K}\pi_{k,j}^{\sum_l y_{n,l,j}}\right]^{t_{n,k}}\right\} \\
&= \sum_{n=1}^{N}\sum_{k=1}^{K} t_{n,k} \cdot \sum_{j=1}^{K}\left(\sum_{l=1}^{L} y_{n,l,j}\right) \cdot \log \pi_{k,j}.
\end{aligned} \tag{7}$$

Now to find a maximum likelihood estimator (MLE) for $\pi_{k,j}$, we can take the derivative and set it equal to zero. Yet, since $\sum_{j=1}^{K} \pi_{k,j} = 1$, we also need to incorporate this constraint

as a lagrangian:

$$\begin{aligned}\frac{\partial}{\partial \pi_{k,j}} \ell\left(\{\pi_{k,j}\}_{k,j=1}^{K}, \lambda; \mathcal{Y}, \boldsymbol{T}\right) &= \frac{\partial}{\partial \pi_{k,j}} \ell\left(\{\pi_{k,j}\}_{k,j=1}^{K}; \mathcal{Y}, \boldsymbol{T}\right) + \frac{\partial}{\partial \pi_{k,j}} \lambda \sum_{k=1}^{K}\left(1 - \sum_{j=1}^{K} \pi_{k,j}\right) \\ &= \left[\sum_{n=1}^{N} t_{n,k}\left(\sum_{l=1}^{L} y_{n,l,j}\right) \cdot \frac{1}{\pi_{k,j}}\right] - \lambda\end{aligned} \tag{8}$$

Setting this equation to zero, we have

$$\begin{aligned}0 &= \left[\sum_{n=1}^{N} t_{n,k}\left(\sum_{l=1}^{L} y_{n,l,j}\right) \cdot \frac{1}{\pi_{k,j}}\right] - \lambda \\ \pi_{k,j} &= \frac{1}{\lambda}\left[\sum_{n=1}^{N} t_{n,k} \sum_{l=1}^{L} y_{n,l,j}\right].\end{aligned} \tag{9}$$

Now making sure we obey the sum-to-one constraint, we have

$$\begin{aligned}1 &= \sum_{j=1}^{K} \pi_{k,j} \\ &= \frac{1}{\lambda} \sum_{j=1}^{K} \sum_{n=1}^{N} t_{n,k} \sum_{l=1}^{L} y_{n,l,j} \\ \lambda &= \sum_{j=1}^{K} \sum_{n=1}^{N} t_{n,k} \sum_{l=1}^{L} y_{n,l,j}.\end{aligned} \tag{10}$$

Putting everything together, we then have the final form for the MLE:

$$\hat{\pi}_{k,j} = \frac{\sum_{n=1}^{N} t_{n,k} \sum_{l=1}^{L} y_{n,l,j}}{\sum_{i=1}^{K} \sum_{n=1}^{N} t_{n,k} \sum_{l=1}^{L} y_{n,l,i}}. \tag{11}$$

We can interpret this quantity as the number of times class j is reported by an annotator when the true class is k, divided by the total number of annotations of any result when the true class is k.

Unobserved Truth Previously we assumed we had access to the ground-truth class $\mathbf{t}_n$, but often this is an unrealistic assumption. If we have the ground-truth, then we probably wouldn't need to query the annotators in the first place. Luckily, this model is amenable to a standard missing data treatment via the *expectation-maximization* (EM) algorithm. We can construct a conditionally conjugate posterior distribution over $\mathbf{t}$, compute its expected value, and then perform the maximization step above using these expectations instead of the ground-truth. Firstly, the aforementioned posterior can be obtained via a prior distribution

over $\mathbf{t}$:

$$\begin{aligned} p\left(\mathrm{t}_{n,k}=1 \mid \boldsymbol{y}_{n,1},\ldots,\boldsymbol{y}_{n,L}\right) &\propto \left(\prod_{l=1}^{L} p\left(\boldsymbol{y}_{n,l}|t_{n,k}=1\right)\right) \cdot p\left(\mathrm{t}_{n,k}=1\right) \\ &= p_k \cdot \prod_{l=1}^{L} \text{Categorical}\left(\boldsymbol{y}_{n,l};\boldsymbol{\pi}_k\right) \end{aligned} \tag{12}$$

where $p_k = p\left(\mathrm{t}_{n,k}=1\right)$, as it is assumed to be the same across all n. Normalizing this distribution, we have

$$\begin{aligned} p\left(\mathrm{t}_{n,k}=1 \mid \boldsymbol{y}_{n,1},\ldots,\boldsymbol{y}_{n,L}\right) &= \frac{p_k \cdot \prod_{l=1}^{L} \text{Categorical}\left(\boldsymbol{y}_{n,l};\boldsymbol{\pi}_k\right)}{\sum_{i=1}^{K} p_i \cdot \prod_{l=1}^{L} \text{Categorical}\left(\boldsymbol{y}_{n,l};\boldsymbol{\pi}_i\right)} \\ &= \frac{p_k \cdot \prod_{j=1}^{K} \pi_{k,j}^{\sum_l y_{n,l,j}}}{\sum_{i=1}^{K} p_i \cdot \prod_{j=1}^{K} \pi_{i,j}^{\sum_l y_{n,l,j}}} \\ &= \hat{p}_{n,k}^{t} \end{aligned} \tag{13}$$

where the denominator is simply a sum over all classes. If the prior probabilities are equal ($p_k = p_i,\ \forall i \in [1,K]$), then these terms would cancel out. We denote the above quantity as $\hat{p}_{n,k}^{t}$ for the sake of notational brevity.

Now, formally, the expectation step (i.e. E-step) of the EM procedure computes the expected log-likelihood:

$$\begin{aligned} \mathbb{E}_{\mathbf{T}|\mathcal{Y}}\left[\ell\left(\{\pi_{k,j}\}_{k,j=1}^{K};\mathcal{Y},\boldsymbol{T}\right)\right] &= \mathbb{E}_{\mathbf{T}|\mathcal{Y}}\left[\sum_{n=1}^{N}\sum_{k=1}^{K} t_{n,k} \cdot \sum_{j=1}^{K}\left(\sum_{l=1}^{L} y_{n,l,j}\right) \cdot \log \pi_{k,j}\right] \\ &= \sum_{n=1}^{N}\sum_{k=1}^{K} \mathbb{E}_{\mathrm{t}|\boldsymbol{y}}\left[t_{n,k}\right] \cdot \sum_{j=1}^{K}\left(\sum_{l=1}^{L} y_{n,l,j}\right) \cdot \log \pi_{k,j} \\ &= \sum_{n=1}^{N}\sum_{k=1}^{K} \hat{p}_{n,k}^{t} \cdot \sum_{j=1}^{K}\left(\sum_{l=1}^{L} y_{n,l,j}\right) \cdot \log \pi_{k,j}. \end{aligned} \tag{14}$$

The M-step is to maximize this equation, just as before, which yields the estimator:

$$\hat{\pi}_{k,j} = \frac{\sum_{n=1}^{N} \hat{p}_{n,k}^{t} \sum_{l=1}^{L} y_{n,l,j}}{\sum_{i=1}^{K}\sum_{n=1}^{N} \hat{p}_{n,k}^{t} \sum_{l=1}^{L} y_{n,l,i}} \tag{15}$$

which can be interpreted as a 'soft' version of the MLE presented in Equation 11. The EM procedures is summarized by iteratively computing Equation 13 and Equation 15 until convergence.

2.2 Unpooled: Dawid-Skene Model

In many applications with crowdsourced data, we want models that can not only understand how the crowd is mislabeling, but we want that information *per annotator*. Having such information allows poor annotators to be excluded from either the current or future requests. That can be done with the multinomial model above but with one change, having a confusion matrix $\mathfrak{C}_l$ for every labeler, $l \in [1,L]$. Doing so expands the number of parameters from $K \times K$

to $L \times K \times K$. Let $\pi_{l,k,j}$ denote the probability of the l-th labeler reporting class j when the true class is k. And again, $\sum_j \pi_{l,k,j} = 1$. This model is known as the *Dawid-Skene* model, named after the authors of the paper in which it was first introduced (Dawid and Skene, 1979).

Observed Truth Again we first discuss the easier case of having access to the true annotation, denoted by an indicator vector $\boldsymbol{t}$ such that $t_k = 1$ denotes that the true class is k and all other dimensions are zero. The generative process can then be written as:

$$\mathbf{t}_n \sim \text{Categorical}(\text{p}^*), \quad \mathbf{y}_{n,l} \sim \text{Categorical}\left(\mathbf{y}_{n,l};\ \langle \mathfrak{C}_l, \mathbf{t}_n \rangle\right), \forall\ n \in [1, N], l \in [1, L] \tag{16}$$

where p^* are the underlying truth (i.e. class) probabilities, which are unknown. Assume we have observed a data set $\mathcal{Y} = \left\{\{\boldsymbol{y}_{n,l}\}_{l=1}^{L}\right\}_{n=1}^{N}$ and the corresponding ground-truths $\boldsymbol{T} = \{\boldsymbol{t}_n\}_{n=1}^{N}$. We can write the log-likelihood for the above model as:

$$\begin{aligned}
\ell\left(\{\mathfrak{C}_l\}_{l=1}^{L}; \mathcal{Y}, \boldsymbol{T}\right) &= \log\left\{\prod_{n=1}^{N}\prod_{k=1}^{K}\prod_{l=1}^{L} p\left(\boldsymbol{y}_{n,l}|t_{n,k}\right)\right\} \\
&= \log\left\{\prod_{n=1}^{N}\prod_{k=1}^{K}\left[\prod_{l=1}^{L}\text{Categorical}\left(\boldsymbol{y}_{n,l}; \boldsymbol{\pi}_{l,k}\right)\right]^{t_{n,k}}\right\} \\
&= \log\left\{\prod_{n=1}^{N}\prod_{k=1}^{K}\left[\prod_{l=1}^{L}\prod_{j=1}^{K}\pi_{l,k,j}^{y_{n,l,j}}\right]^{t_{n,k}}\right\} \\
&= \sum_{n=1}^{N}\sum_{k=1}^{K} t_{n,k} \cdot \sum_{l=1}^{L}\sum_{j=1}^{K} y_{n,l,j} \cdot \log \pi_{l,k,j}.
\end{aligned} \tag{17}$$

Now to find a maximum likelihood estimator (MLE) for $\pi_{l,k,j}$, we can take the derivative and set it equal to zero. Yet, since $\sum_{j=1}^{K} \pi_{l,k,j} = 1$, we also need to incorporate this constraint as a lagrangian:

$$\begin{aligned}
\frac{\partial}{\partial \pi_{l,k,j}} \ell\left(\{\mathfrak{C}_l\}_{l=1}^{L}, \lambda; \mathcal{Y}, \boldsymbol{T}\right) &= \frac{\partial}{\partial \pi_{l,k,j}} \ell\left(\{\mathfrak{C}_l\}_{l=1}^{L}; \mathcal{Y}, \boldsymbol{T}\right) + \frac{\partial}{\partial \pi_{l,k,j}} \lambda \sum_{k=1}^{K}\left(1 - \sum_{j=1}^{K}\pi_{l,k,j}\right) \\
&= \left[\sum_{n=1}^{N} t_{n,k} \cdot y_{n,l,j} \cdot \frac{1}{\pi_{l,k,j}}\right] - \lambda
\end{aligned} \tag{18}$$

Setting this equation to zero, we have

$$\begin{aligned}
0 &= \left[\sum_{n=1}^{N} t_{n,k} \cdot y_{n,l,j} \cdot \frac{1}{\pi_{l,k,j}}\right] - \lambda \\
\pi_{l,k,j} &= \frac{1}{\lambda}\left[\sum_{n=1}^{N} t_{n,k} \cdot y_{n,l,j}\right].
\end{aligned} \tag{19}$$

Making sure we obey the sum-to-one constraint, we have

$$\begin{aligned} 1 &= \sum_{j=1}^{K} \pi_{l,k,j} \\ &= \frac{1}{\lambda} \sum_{j=1}^{K} \sum_{n=1}^{N} t_{n,k} \cdot y_{n,l,j} \\ \lambda &= \sum_{j=1}^{K} \sum_{n=1}^{N} t_{n,k} \cdot y_{n,l,j}. \end{aligned} \tag{20}$$

Putting everything together, we then have the final form for the MLE:

$$\hat{\pi}_{l,k,j} = \frac{\sum_{n=1}^{N} t_{n,k} \cdot y_{n,l,j}}{\sum_{i=1}^{K} \sum_{n=1}^{N} t_{n,k} \cdot y_{n,l,i}}. \tag{21}$$

We can interpret this quantity as the number of times class j is reported by annotator l when the true class is k, divided by the total number of annotations reported by annotator l, of any result, when the true class is k.

Unobserved Truth We again turn to the more interesting case in which the ground-truth class $\mathbf{t}_n$ is not observed. Again we can apply the EM algorithm to fill in the missing truth variables. The first step is the same as above: placing a prior distribution over $\mathbf{t}$:

$$\begin{aligned} p\left(\mathrm{t}_{n,k} \mid \boldsymbol{y}_{n,1}, \ldots, \boldsymbol{y}_{n,L}\right) &\propto \left(\prod_{l=1}^{L} p\left(\boldsymbol{y}_{n,l} | t_{n,k}\right) \right) \cdot p\left(\mathrm{t}_{n,k}\right) \\ &= p_k \cdot \prod_{l=1}^{L} \mathrm{Categorical}\left(\boldsymbol{y}_{n,l}; \boldsymbol{\pi}_{l,k}\right) \end{aligned} \tag{22}$$

where $p_k = p\left(\mathrm{t}_{n,k}\right)$, as it is assumed to be the same across all n. Normalizing this distribution, we have

$$p\left(\mathrm{t}_{n,k} \mid \boldsymbol{y}_{n,1}, \ldots, \boldsymbol{y}_{n,L}\right) = \frac{p_k \cdot \prod_{l=1}^{L} \prod_{j=1}^{K} \pi_{l,k,j}^{y_{n,l,j}}}{\sum_{i=1}^{K} p_i \cdot \prod_{l=1}^{L} \prod_{j=1}^{K} \pi_{l,i,j}^{y_{n,l,j}}} = \hat{p}_{n,k}^{t} \tag{23}$$

where the denominator is simply a sum over all classes. Again the expectation step (i.e. E-step) of the EM procedure computes the expected log-likelihood:

$$\begin{aligned} \mathbb{E}_{\mathbf{T}|\mathcal{Y}}\left[\ell\left(\{\mathfrak{C}_l\}_{l=1}^{L}; \mathcal{Y}, \boldsymbol{T}\right)\right] &= \mathbb{E}_{\mathbf{T}|\mathcal{Y}}\left[\sum_{n=1}^{N} \sum_{k=1}^{K} t_{n,k} \cdot \sum_{l=1}^{L} \sum_{j=1}^{K} y_{n,l,j} \cdot \log \pi_{l,k,j}\right] \\ &= \sum_{n=1}^{N} \sum_{k=1}^{K} \mathbb{E}_{\mathrm{t}|\boldsymbol{y}}\left[t_{n,k}\right] \cdot \sum_{l=1}^{L} \sum_{j=1}^{K} y_{n,l,j} \cdot \log \pi_{l,k,j} \\ &= \sum_{n=1}^{N} \sum_{k=1}^{K} \hat{p}_{n,k}^{t} \cdot \sum_{l=1}^{L} \sum_{j=1}^{K} y_{n,l,j} \cdot \log \pi_{l,k,j}. \end{aligned} \tag{24}$$

The M-step is to maximize this equation, just as before, which yields the estimator:

$$\hat{\pi}_{k,j} = \frac{\sum_{n=1}^{N} \hat{p}_{n,k}^{t} \cdot y_{n,l,j}}{\sum_{i=1}^{K} \sum_{n=1}^{N} \hat{p}_{n,k}^{t} \cdot y_{n,l,i}} \tag{25}$$

which can be interpreted as a 'soft' version of the MLE presented in Equation 21. The EM procedures is summarized by iteratively computing Equation 23 and Equation 25 until convergence.

2.3 Jointly Learning Ground-Truth and a Predictive Model

In the setting of unobserved truth, the Multinomial and Dawid-Skene models are first applied to infer a ground-truth label, which would then be used to train a traditional classifier. However, if obtaining this downstream classifier is our ultimate, then perhaps it is sub-optimal to have learning be decoupled into a two-stage process. Rather, as proposed by Raykar et al. (2010), it is better to have a joint formulation. A joint objective can be defined as follows. Let $p(\mathbf{t}|\mathbf{x}, \boldsymbol{\theta})$ denote the classifier defined by the conditional probability of the ground-truth variable $\mathbf{t}$, given the features $\mathbf{x}$ and classifier parameters $\boldsymbol{\theta}$. The generative process is then:

$$\mathbf{t}_n \sim p(\mathbf{t}_n|\mathbf{x}_n, \boldsymbol{\theta}), \quad \mathbf{y}_{n,l} \sim \text{Categorical}\left(\mathbf{y}_{n,l};\ \langle \mathfrak{C}_l, \mathbf{t}_n \rangle\right), \forall\ n \in [1, N], l \in [1, L]. \tag{26}$$

Note that this model is essentially the Dawid-Skene model but with now the classifier generating the higher-level distribution over truth.

To learn in this model, the conditional probability of just the annotations can be written by marginalizing out the ground-truth:

$$\begin{aligned}
p\left(\mathcal{Y} \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N, \boldsymbol{\theta}\right) &= \prod_{n=1}^{N} p\left(\boldsymbol{y}_{n,1}, \ldots, \boldsymbol{y}_{n,L} \mid \boldsymbol{x}_n, \boldsymbol{\theta}\right) \\
&= \prod_{n=1}^{N} \sum_{\mathbf{t}_n} p\left(\boldsymbol{y}_{n,1}, \ldots, \boldsymbol{y}_{n,L} \mid \mathbf{t}_n\right) \cdot p\left(\mathbf{t}_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}\right) \\
&= \prod_{n=1}^{N} \sum_{\mathbf{t}_n} \left(\prod_{l=1}^{L} p\left(\boldsymbol{y}_{n,l} \mid \mathbf{t}_n\right) \right) \cdot p\left(\mathbf{t}_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}\right) \\
&= \prod_{n=1}^{N} \sum_{\mathbf{t}_n} \prod_{k=1}^{K} \left[p\left(\mathrm{t}_{n,k} \mid \boldsymbol{x}_n, \boldsymbol{\theta}\right) \cdot \prod_{l=1}^{L} p\left(\boldsymbol{y}_{n,l} \mid \mathrm{t}_{n,k} = 1\right) \right]^{t_{n,k}} \\
&= \prod_{n=1}^{N} \sum_{\mathbf{t}_n} \prod_{k=1}^{K} \left[p\left(\mathrm{t}_{n,k} \mid \boldsymbol{x}_n, \boldsymbol{\theta}\right) \cdot \prod_{l=1}^{L} \text{Categorical}\left(\boldsymbol{y}_{n,l};\ \boldsymbol{\pi}_{l,k}\right) \right]^{t_{n,k}} \\
&= \prod_{n=1}^{N} \sum_{\mathbf{t}_n} \prod_{k=1}^{K} \left[p\left(\mathrm{t}_{n,k} \mid \boldsymbol{x}_n, \boldsymbol{\theta}\right) \cdot \prod_{l=1}^{L} \prod_{j=1}^{K} \pi_{l,k,j}^{y_{n,l,j}} \right]^{t_{n,k}}
\end{aligned} \tag{27}$$

where we see that this joint model is essentially the Dawid-Skene model with the truth variable $\mathbf{t}$ marginalized away via the classifier.

Learning can again be done via the EM algorithm, but here we need to work with a

lower-bound on the marginal likelihood from above:

$$
\begin{aligned}
\ell\left(\{\mathfrak{C}_l\}_{l=1}^L, \boldsymbol{\theta}; \mathcal{Y}\right) &= \log\left\{\prod_{n=1}^N \sum_{\mathbf{t}_n} \prod_{k=1}^K \left[p\left(\mathrm{t}_{n,k} \mid \boldsymbol{x}_n, \boldsymbol{\theta}\right) \cdot \prod_{l=1}^L \prod_{j=1}^K \pi_{l,k,j}^{y_{n,l,j}}\right]^{t_{n,k}}\right\} \\
&= \sum_{n=1}^N \log\left\{\sum_{\mathbf{t}_n} \prod_{k=1}^K \left[p\left(\mathrm{t}_{n,k} \mid \boldsymbol{x}_n, \boldsymbol{\theta}\right) \cdot \prod_{l=1}^L \prod_{j=1}^K \pi_{l,k,j}^{y_{n,l,j}}\right]^{t_{n,k}}\right\} \\
&\geq \sum_{n=1}^N \sum_{\mathbf{t}_n} \sum_{k=1}^K t_{n,k} \cdot p\left(\mathrm{t}_{n,k} \mid \boldsymbol{x}_n, \boldsymbol{\theta}\right) \cdot \sum_{l=1}^L \sum_{j=1}^K y_{n,l,j} \cdot \log \pi_{l,k,j} \\
&= \sum_{n=1}^N \sum_{k=1}^K p\left(\mathrm{t}_{n,k} = 1 \mid \boldsymbol{x}_n, \boldsymbol{\theta}\right) \cdot \sum_{l=1}^L \sum_{j=1}^K y_{n,l,j} \cdot \log \pi_{l,k,j} \\
&= \tilde{\ell}\left(\{\mathfrak{C}_l\}_{l=1}^L, \boldsymbol{\theta}; \mathcal{Y}\right)
\end{aligned}
\tag{28}
$$

where the inequality is obtained via Jensen's inequality for convex functions.

Now finding an estimator for $\pi_{l,k,j}$:

$$
\begin{aligned}
\frac{\partial}{\partial \pi_{l,k,j}} \tilde{\ell}\left(\{\mathfrak{C}_l\}_{l=1}^L, \boldsymbol{\theta}, \lambda; \mathcal{Y}\right) &= \frac{\partial}{\partial \pi_{l,k,j}} \tilde{\ell}\left(\{\mathfrak{C}_l\}_{l=1}^L, \boldsymbol{\theta}; \mathcal{Y}\right) \; + \; \lambda \sum_{k=1}^K \left(1 - \sum_{j=1}^K \pi_{l,k,j}\right) \\
&= \sum_{n=1}^N p\left(\mathrm{t}_{n,k} = 1 \mid \boldsymbol{x}_n, \boldsymbol{\theta}\right) \cdot y_{n,l,j} \cdot \frac{1}{\pi_{l,k,j}} - \lambda \\
\pi_{l,k,j} &= \frac{1}{\lambda}\left[\sum_{n=1}^N p\left(\mathrm{t}_{n,k} = 1 \mid \boldsymbol{x}_n, \boldsymbol{\theta}\right) \cdot y_{n,l,j}\right] \\
\lambda &= \sum_{j=1}^K \sum_{n=1}^N p\left(\mathrm{t}_{n,k} = 1 \mid \boldsymbol{x}_n, \boldsymbol{\theta}\right) \cdot y_{n,l,j}.
\end{aligned}
\tag{29}
$$

Putting it all together, we then have:

$$
\hat{\pi}_{l,k,j} \;=\; \frac{\sum_{n=1}^N p\left(\mathrm{t}_{n,k} = 1 \mid \boldsymbol{x}_n, \boldsymbol{\theta}\right) \cdot y_{n,l,j}}{\sum_{i=1}^K \sum_{n=1}^N p\left(\mathrm{t}_{n,k} = 1 \mid \boldsymbol{x}_n, \boldsymbol{\theta}\right) \cdot y_{n,l,i}} \tag{30}
$$

Then given the confusion matrix parameters, we can estimate the classifier as follows. Computing the posterior over truth values via Equation 23, the training objective for the classifier

is:

$$
\begin{aligned}
\mathbb{E}_{\mathbf{T}|\mathcal{Y}}\left[\ell(\boldsymbol{\theta};\mathbf{T},\boldsymbol{X})\right] &= \mathbb{E}_{\mathbf{T}|\mathcal{Y}}\left[\log p(\mathbf{T}|\boldsymbol{X},\boldsymbol{\theta})\right] \\
&= \mathbb{E}_{\mathbf{T}|\mathcal{Y}}\left[\sum_{n=1}^{N}\log p(\mathbf{t}_b|\boldsymbol{x}_n,\boldsymbol{\theta})\right] \\
&= \mathbb{E}_{\mathbf{T}|\mathcal{Y}}\left[\sum_{n=1}^{N}\log \text{Categorical}(\mathbf{t}_b; f(\boldsymbol{x}_n;\boldsymbol{\theta}))\right] \\
&= \mathbb{E}_{\mathbf{T}|\mathcal{Y}}\left[\sum_{n=1}^{N}\sum_{k=1}^{K}\mathrm{t}_{n,k}\log f_k(\boldsymbol{x}_n;\boldsymbol{\theta}))\right] \\
&= \sum_{n=1}^{N}\sum_{k=1}^{K}\mathbb{E}_{\mathbf{T}|\mathcal{Y}}\left[\mathrm{t}_{n,k}\right]\log f_k(\boldsymbol{x}_n;\boldsymbol{\theta})) \\
&= \sum_{n=1}^{N}\sum_{k=1}^{K}\hat{p}^{t}_{n,k}\log f_k(\boldsymbol{x}_n;\boldsymbol{\theta}))
\end{aligned}
\tag{31}
$$

where $\hat{p}^{t}_{n,k}$ is again from Equation 23. We can see this is the usually classifier loss but with 'soft' labels. If the classifier would perfectly recreate these posterior probabilities, then they would be plugged into Equation 30, which in turn would recover Equation 25.

Alternative Formulation Yan et al. (2010) proposed a variant of this model that conditions both the truth and observed label on the features:

$$
\mathbf{t}_n \sim p\left(\mathbf{t}_n|\mathbf{x}_n,\boldsymbol{\theta}_t\right), \quad \mathbf{y}_{n,l} \sim p\left(\mathbf{y}_{n,l}|\mathbf{x}_n,\mathbf{t}_n,\boldsymbol{\theta}_l\right), \quad \forall\ n\in[1,N], l\in[1,L]. \tag{32}
$$

The logic behind this alteration is that, by having $\mathrm{y} \sim p(\mathrm{y}|\mathrm{t})$, the Raykar et al. (2010) model assumes that the observation is a noisy version of the ground-truth and that noise is completely determined by the ground-truth class. However, in many real-world cases, the label error might be due to noise in the input features as well. For example, perhaps the human is annotating an image, and the image is of very poor quality. The low-quality of the input could be the reason why the annotator produces the wrong label—i.e. the noise from the input propagating into the label—and not due to any systematic labeling bias they have for a particular class. This model also has benefits for active learning, as we will consider in the next subsection.

2.4 Active Learning

In the previous subsection, we were mostly concerned with how to filter out noise or come to consensus when given multiple annotations. This subsection focuses on how we obtain labels from humans in the first place and, ideally, through the most efficient means possible. Collecting human annotations is expensive as it costs humans their time. For example, annotating the words in an audio recording takes about ten times longer than the audio recording itself. In turn, the person wanting the labels often must compensate the workers with payment. Hence we'd like methodologies that minimize these costs by minimizing the number of labels collected. This brings us to the sub-field of *active learning*; most of the information from this section is reproduced from the authoritative survey of Settles (2012).

Definition & Types *Active learning* (AL) allows predictive models to choose their own training data, similarly to how a math student might ask their teacher for guidance on a particular problem they find difficult. The high-level idea is that the model will inspect an unlabeled data point and then determine if its label were acquired, then the model's performance would substantially improve. This process is repeated until the model's performance has either reached an acceptable level or plateaus to the level yielded by training on all available data. There are three standard scenarios considered for AL:

- **Membership Query Synthesis** (MQS): This is the most general form of AL. In this setting, the model can request *any* feature vector in the feature space $\mathcal{X}$ as well as its label. This works well when every point in the feature space has a clearly defined label. For example, the task of predicting if a robot arm, in a particular configuration, can hold a glass of water without spilling is a good use case for MQS (assuming $\mathcal{X}$ does not contain any impossible arm configurations). For every feature vector chosen by the model, it is quite easy to test if the arm position is good or not simply by running the experiment and seeing if any water spills from the cup. On the other hand, MQS is not good for image classification since it is very likely feature vectors could be requested for which there is no discernible label. For example, if we are considering the space of binary images representing digits, there are many possible binary images that correspond no recognizable digit.

- **Stream-Based Selective Sampling**: In this setting, feature vectors are assumed to appear sequentially, and the model must decided whether to request its label or discard the vector as 'uninteresting.' This use case is prevalent in data collection on low-resource devices. Imagine there is an autonomous vehicle driving in a new location, and its designers would like the car to save its sensor readings during 'interesting' sections of the road or during novel events. It would be too costly to save all information from the whole driving run and so the system must decide for itself which data points to save for later annotation.

- **Pool-Based Active Learning**: This setting—which is the most commonly studies formulation—assumes that there is an unlabeled data set known as the *pool set*, and the model is allowed to request the label for a particular member of the pool set. Once the label is acquired, that point is removed from the pool, and the model is retrained to include that newly selected point and label. The process repeats until the model reaches a satisfactory performance or the pool set is empty. There is also a batch variant in which the model can request multiple points for labeling at each round.

Acquisition Functions for Pool-Based Active Learning We will discuss *pool-based AL* exclusively from here forward, and thus when writing 'AL', we mean the pool-based formulation. The setting can be defined more formally as follows. We consider a predictive model $p(\mathbf{y}|\boldsymbol{x}, \boldsymbol{\theta})$, where $\mathbf{y}$ is the label, $\mathbf{x}$ are the features, and $\boldsymbol{\theta} \in \Theta$ are the parameters. We assume this model is trained on an initial labeled dataset $\mathcal{D}_0 = \{\boldsymbol{x}_n, \boldsymbol{y}_n\}_{n=1}^N$, and thus we denote this model as $p(\mathbf{y}|\boldsymbol{x}, \boldsymbol{\theta}_{t=0})$ where the subscript on the parameter indexes time. In addition to the initial dataset $\mathcal{D}_0$, we assume access to (i) an unlabeled pool set $\mathcal{X}_p^{t=0} = \{\boldsymbol{x}_m\}_{m=1}^M$, and (ii) an oracle labeling mechanism which can provide labels $\mathcal{Y}_p = \{\boldsymbol{y}_m\}_{m=1}^M$ for the corresponding features in the pool set.

At each step in the AL loop, an *acquisition function* (AF) $\mathcal{A}(\boldsymbol{x}; \boldsymbol{\theta}_t)$ is evaluated for every member of the pool set. $\mathcal{A}$ is written as a function of $\boldsymbol{\theta}_t$ because the AF changes along with the state of the predictive model. We want the AF to acquire the most interesting point to the current model and thus will collect the label of the point that maximizes the AF, denoted $\boldsymbol{x}^*$:

$$\boldsymbol{x}^* \;=\; \underset{\boldsymbol{x} \in \mathcal{X}_p^t}{\arg\max}\; \mathcal{A}(\boldsymbol{x}; \boldsymbol{\theta}_t). \tag{33}$$

Once the point has been chosen, the oracle provides it's label, and the new labeled set becomes:

$$\mathcal{D}_{t+1} = \mathcal{D}_t \cup \{\boldsymbol{x}^*, \boldsymbol{y}^*\}.$$

Moreover, the chosen point is removed from the pool set: $X_p^{t+1} = X_p^t \setminus \{\boldsymbol{x}^*\}$. The model is then re-trained on $\mathcal{D}_{t+1}$ to produce a new set of parameters $\boldsymbol{\theta}_{t+1}$, and the process repeats by collecting a new point to form $\mathcal{D}_{t+2}$, which in turn produces $\boldsymbol{\theta}_{t+2}$, and so on until either the pool set is empty or a satisfactory level of performance is reached (which would require evaluating the model on a held-out set after every re-training step). We now go on to describe several ways to implement the AF.

Idealized Setting: Error Reduction Ideally, we want to collect $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ that, once the model is trained on the pair, will reduce the model's error on all future data points the model might see. This formulation as been called *optimal active learning* (Roy and McCallum, 2001), and the corresponding idealized AF is:

$$\boldsymbol{x}^* \;=\; \underset{\boldsymbol{x} \in \mathcal{X}_p^t}{\arg\max} \int_{\mathbf{x}} \left(\mathbb{D}\left[\mathbb{P}(\mathrm{y}|\mathbf{x}) \;||\; p\left(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}_t\right)\right] \;-\; \mathbb{D}\left[\mathbb{P}(\mathrm{y}|\mathbf{x}) \;||\; p\left(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}(\boldsymbol{x}, \boldsymbol{y}^*)\right)\right]\right) \; \mathbb{P}(\mathbf{x}) \; d\mathbf{x} \tag{34}$$

where $\boldsymbol{\theta}(\boldsymbol{x}, \boldsymbol{y}^*)$ denotes the parameters produced by training on $\mathcal{D}_t \cup \{\mathbf{x}, \boldsymbol{y}^*\}$ and $\mathbb{D}$ is some measure of discrepancy or divergence between the true generative process $\mathbb{P}(\mathrm{y}|\mathbf{x})$ and the model—either at the current time step ($\boldsymbol{\theta}_t$) or updated with $\mathbf{x}$ ($\boldsymbol{\theta}(\boldsymbol{x}, \boldsymbol{y}^*)$). If $\mathbb{D}$ is the Kullback–Leibler divergence, then this AF is picking the point that results in the best improvement under maximum likelihood estimation. Thus we can think of this—again idealized—AF as looking ahead to the future, if we were to train the model on a particular point from the pool set, and see if that new model would better minimize the divergence between the model and the true distribution that is generating the data. This construction is called 'idealized' because we never have access to $\mathbb{P}(\mathbf{x})$, $\mathbb{P}(\mathrm{y}|\mathbf{x})$, and even if we did, it requires collecting the labels for all elements of the pool set—the very thing we wish to avoid. If $\left(\mathbb{D}\left[\mathbb{P}(\mathrm{y}|\mathbf{x}) \;||\; p\left(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}_t\right)\right] \;-\; \mathbb{D}\left[\mathbb{P}(\mathrm{y}|\mathbf{x}) \;||\; p\left(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}(\boldsymbol{x}, \boldsymbol{y}^*)\right)\right]\right) < 0$ for every element of the pool set, then AL should be stopped with $\boldsymbol{\theta}_t$ being the final model.

However, Roy and McCallum (2001) gives the procedure that aims to approximate the above idealized AF. Firstly, when $\mathbb{D}$ is taken to be the KL divergence, notice that:

$$\begin{aligned}
&\mathbb{KLD}\left[\mathbb{P}(\mathrm{y}|\mathbf{x}) \;||\; p\left(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}_t\right)\right] \;-\; \mathbb{KLD}\left[\mathbb{P}(\mathrm{y}|\mathbf{x}) \;||\; p\left(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}(\boldsymbol{x}, \boldsymbol{y}^*)\right)\right] \\
&= \mathbb{E}_{\mathbb{P}(\mathrm{y}|\mathbf{x})}\left[\log p\left(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}(\boldsymbol{x}, \boldsymbol{y}^*)\right) - \log p\left(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}_t\right)\right] + \mathbb{H}\left[\mathbb{P}(\mathrm{y}|\mathbf{x})\right] - \mathbb{H}\left[\mathbb{P}(\mathrm{y}|\mathbf{x})\right] \\
&= \mathbb{E}_{\mathbb{P}(\mathrm{y}|\mathbf{x})}\left[\log p\left(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}(\boldsymbol{x}, \boldsymbol{y}^*)\right) - \log p\left(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}_t\right)\right].
\end{aligned} \tag{35}$$

Secondly, they use the pool set to approximate the integral over the underlying feature

distribution: $\mathbb{P}(\mathbf{x}) \approx \frac{1}{M}\sum_{m=1}^{M}\delta\left[|\mathbf{x}-\boldsymbol{x}_n|\right]$. Thirdly, this still leaves the expectation over the unknown distribution $\mathbb{P}(\mathrm{y}|\mathbf{x})$; they recommend to approximate this with the current model: $\mathbb{P}(\mathrm{y}|\mathbf{x}) \approx p\left(\mathbf{y}|\mathbf{x},\boldsymbol{\theta}_t\right)$. Note that under this assumption, the difference of divergences further simplifies to:

$$\begin{aligned}\mathbb{E}_{\mathbb{P}(\mathrm{y}|\mathbf{x})}\left[\log p\left(\mathbf{y}|\mathbf{x},\boldsymbol{\theta}(\boldsymbol{x},\boldsymbol{y}^*)\right) - \log p\left(\mathbf{y}|\mathbf{x},\boldsymbol{\theta}_t\right)\right] \\ \approx \mathbb{E}_{p(\mathbf{y}|\mathbf{x},\boldsymbol{\theta}_t)}\left[\log p\left(\mathbf{y}|\mathbf{x},\boldsymbol{\theta}(\boldsymbol{x},\boldsymbol{y}^*)\right)\right] + \mathbb{H}\left[p\left(\mathbf{y}|\mathbf{x},\boldsymbol{\theta}_t\right)\right],\end{aligned} \tag{36}$$

where the second term is the entropy of the current model, and when integrated over the empirical distribution of the pool set, will be a constant (and thus can be dropped). The only remaining difficulty is that the above expression still depends on the true label $\boldsymbol{y}^*$, which we do not want to assume is know. Roy and McCallum (2001) again leverage the existing model for this, assuming that $\mathbf{y}^* \sim p\left(\mathbf{y}|\boldsymbol{x},\boldsymbol{\theta}_t\right)$. The final realizable AF is then:

$$\begin{aligned}\boldsymbol{x}^* &= \underset{\boldsymbol{x}\in\mathcal{X}_p^t}{\arg\max}\ \frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_{p(\mathbf{y}'|\boldsymbol{x},\boldsymbol{\theta}_t)}\mathbb{E}_{p(\mathbf{y}|\boldsymbol{x}_m,\boldsymbol{\theta}_t)}\left[\log p\left(\mathbf{y}|\boldsymbol{x}_m,\boldsymbol{\theta}(\boldsymbol{x},\mathbf{y}')\right)\right] \\ &\approx \underset{\boldsymbol{x}\in\mathcal{X}_p^t}{\arg\max}\ \frac{1}{M}\cdot\frac{1}{J}\cdot\frac{1}{S}\sum_{m=1}^{M}\sum_{j=1}^{J}\sum_{s=1}^{S}\log p\left(\hat{\boldsymbol{y}}_s|\boldsymbol{x}_m,\boldsymbol{\theta}(\boldsymbol{x},\hat{\boldsymbol{y}'}_j)\right)\end{aligned} \tag{37}$$

where the approximation in the second line uses J samples of the proxy true label, $\hat{\boldsymbol{y}'}_j \sim p\left(\mathbf{y}'|\boldsymbol{x},\boldsymbol{\theta}_t\right)$, and S samples $\hat{\boldsymbol{y}}_s \sim p\left(\mathbf{y}|\boldsymbol{x}_m,\boldsymbol{\theta}_t\right)$ to approximate the two expectations. The intuition behind this AF is that it will look for points from the pool set that will result in models that 'agree' with the current distribution. Or as Roy and McCallum (2001) put it: "An example will be selected if it dramatically reinforces the learner's existing belief over unlabeled examples for which it is currently unsure." While it may seem this could cause a self-reinforcing effect, with the model selecting points that are already probable under the current model, this behavior is prevented by calculating the outer sum (over M) over the pool set, which by definition, are points that have not yet been used for training. Despite the aggressive approximations used above, the above AF is still computationally expensive as it requires the model be re-trained J times for one evaluation of the AF. Roy and McCallum (2001) get around this problem by using naive Bayes classifiers that can be built using one-pass sufficient statistics. Thus, 're-training' simply requires the sufficient statistics be changed by one count.

Uncertainty Sampling Now that we have covered an optimal (but unrealizable) AF, we turn to simpler alternatives. A very popular approach to constructing AFs is known as *uncertainty sampling*, which simply takes the point from the pool set for which the model is most uncertain. One way of quantifying this uncertainty is through the entropy of the current model:

$$\begin{aligned}\boldsymbol{x}^* &= \underset{\boldsymbol{x}\in\mathcal{X}_p^t}{\arg\max}\ \mathcal{A}(\boldsymbol{x};\boldsymbol{\theta}_t) \\ &= \underset{\boldsymbol{x}\in\mathcal{X}_p^t}{\arg\max}\ \mathbb{H}\left[p(\mathrm{y}|\boldsymbol{x},\boldsymbol{\theta}_t)\right] \\ &= \underset{\boldsymbol{x}\in\mathcal{X}_p^t}{\arg\max}\ -\int_{\mathbf{y}} p(\mathbf{y}|\boldsymbol{x},\boldsymbol{\theta}_t)\cdot\log p(\mathbf{y}|\boldsymbol{x},\boldsymbol{\theta}_t)\ d\mathbf{y}.\end{aligned}$$

An often-competitive alternative is to simply look for the predictive distribution with the least-confident mode:

$$\begin{aligned} \boldsymbol{x}^* &= \underset{\boldsymbol{x} \in \mathcal{X}_p^t}{\arg\max}\ \mathcal{A}(\boldsymbol{x}; \boldsymbol{\theta}_t) \\ &= \underset{x \in \mathcal{X}_p^t}{\arg\max}\ 1 - \max_{\boldsymbol{y} \in \mathcal{Y}}\ p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}_t). \end{aligned}$$

For binary models, recall that both their entropy and modal probability are maximized at $p(\mathbf{y}|\boldsymbol{x}, \boldsymbol{\theta}_t) = 0.5$. So for logistic regression, the logistic function equals 0.5 when its input is 0, and thus uncertainty sampling looks for the point in the pool set that is closest to the decision boundary of the current model.

Query-by-Committee Another well known AF construction is Query-by-Committee (QC). This method uses a C-sized ensemble of models to perform AL: $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}_{1,t}), \ldots, p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}_{C,t})$. The standard AF then checks for disagreement across these models, e.g. in a pair-wise fashion:

$$\mathcal{A}(\boldsymbol{x}; \boldsymbol{\theta}_{1,t}, \ldots, \boldsymbol{\theta}_{C,t}) = \sum_{c=1}^{C} \sum_{j \neq c} \text{disagreement}\left[p(\mathbf{y}|\boldsymbol{x}, \boldsymbol{\theta}_{c,t}) \mid\mid p(\mathbf{y}|\boldsymbol{x}, \boldsymbol{\theta}_{j,t})\right]$$

where $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}_{c,t})$ and $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}_{j,t})$ are two ensemble different models. Higher-order comparisons are possible but become expensive. For implementing the disagreement function, one simple procedures is to compare the top-ranked predictions:

$$\mathcal{A}(\boldsymbol{x}; \boldsymbol{\theta}_{1,t}, \ldots, \boldsymbol{\theta}_{C,t}) = \sum_{c=1}^{C} \sum_{j \neq c} \mathbb{I}\left[\hat{\boldsymbol{y}}_c \neq \hat{\boldsymbol{y}}_j\right]$$

where $\hat{\boldsymbol{y}}_c = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}_{c,t})$, the prediction from the cth model and $\mathbb{I}$ is an indicator function equal to one when its argument is true. Another example would be to compare the models via some statistical divergence function. Like uncertainty sampling, QC is also computing a notion of uncertainty but across an ensemble, not just via one model.

Bayesian Active Learning by Disagreement For a Bayesian predictive model, the most popular approach at AL is known as *Bayesian active learning by disagreement* (BALD) (MacKay, 1992; Houlsby et al., 2011). For a model

$$\mathbf{y} \sim p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}), \quad \boldsymbol{\theta} \sim p(\boldsymbol{\theta}),$$

where $p(\boldsymbol{\theta})$ is the prior, the BALD AF is the mutual information between the parameters and labels, i.e.:

$$\begin{aligned} \mathcal{A}(\mathbf{x}; \mathcal{D}_t) &= \mathcal{I}[\mathbf{y}, \boldsymbol{\theta}|\boldsymbol{x}, \mathcal{D}_t] \\ &= \int_{\boldsymbol{\theta}} \sum_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{D}_t) \log \frac{p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{D}_t)}{p(\boldsymbol{y}|\boldsymbol{x}, \mathcal{D}_t)\ p(\boldsymbol{\theta}|\mathcal{D}_t)}\ d\boldsymbol{\theta} \\ &= \int_{\boldsymbol{\theta}} \sum_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{D}_t) \log \frac{p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})}{p(\boldsymbol{y}|\boldsymbol{x}, \mathcal{D}_t)}\ d\boldsymbol{\theta} \\ &= \mathbb{E}_{\boldsymbol{\theta}|\mathcal{D}_t} \mathbb{KLD}\left[p(\mathbf{y}|\boldsymbol{x}, \boldsymbol{\theta}) \mid\mid p(\mathbf{y}|\boldsymbol{x}, \mathcal{D}_t)\right], \end{aligned}$$

which, as seen above, can be written as the KL divergence between the likelihood and predictive distribution, averaged over the posterior distribution. However, Houlsby et al. (2011) recommend working with an equivalent formulation, written in terms of entropy:

$$\begin{aligned}\mathcal{A}(\mathbf{x}; \mathcal{D}_t) &= \mathcal{I}[\mathbf{y}, \boldsymbol{\theta} | \boldsymbol{x}, \mathcal{D}_t] \\ &= \mathbb{H}\left[p(\mathbf{y}|\boldsymbol{x}, \mathcal{D}_t)\right] - \mathbb{E}_{\boldsymbol{\theta}|\mathcal{D}_t}\left[\mathbb{H}\left[p(\mathbf{y}|\boldsymbol{x}, \boldsymbol{\theta})\right]\right]\end{aligned}$$

where the first term is the entropy of the posterior predictive distribution and the second term is the expected entropy of the likelihood, with the expectation taken over the posterior. The intuition behind this AF is that it will be maximized by the feature vector yielding the most marginal uncertainty (high-entropy posterior predictive) but for which the likelihood demonstrates certainty (low entropy) for any given setting of the parameters. Or as Houlsby et al. (2011) say: it "seek[s] the $\boldsymbol{x}$ for which the parameters under the posterior disagree about the outcome the most,"—hence the word *disagreement* in the name BALD.

Batch Active Learning While AL that acquires single points greedily can be near-optimal in certain cases (Golovin and Krause, 2011; Dasgupta, 2005), it becomes severely limited in large-scale settings. One reason is the burden of re-training the model after every acquired data point: re-training a deep neural networks thousands of times is clearly impractical. Even if computation was not a concern, adding just one single point to the labeled set will often result in a negligible change to the updated parameters (Sener and Savarese, 2018). Moreover, since changes in the model will be small, subsequent AL steps will result in acquiring very similar points.

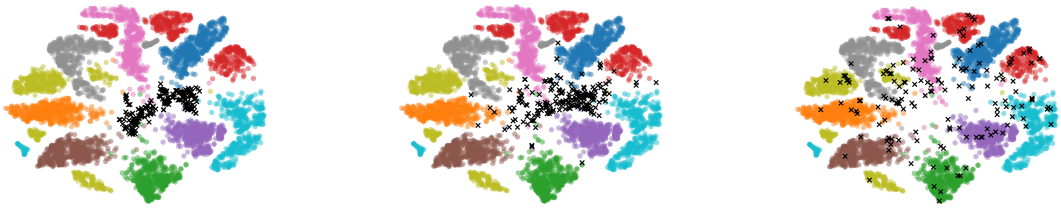
Due to these limitations of single-point acquisition, there is wide interest in *batch* AL methodologies. Given a maximum batch size of B, we can write the batch AL AF as:

$$\boldsymbol{X}^* = \underset{\{\boldsymbol{x}_b\}_{b=1}^B \in \mathcal{X}_p^t}{\arg\max} \mathcal{A}\left(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_B; \boldsymbol{\theta}_t\right), \tag{38}$$

where $\boldsymbol{X}^* = \{\boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_B^*\}$ are the B points that maximize the joint AF $\mathcal{A}\left(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_B; \boldsymbol{\theta}_t\right)$. Unsurprisingly, this general formulation of batch AL is quite challenging, as it is a combinatorial optimization problem that requires checking the AF for all B-sized subsets of the pool set. Often a greedy approximation is made in which the AF is assumed to decompose point-wise:

$$\mathcal{A}\left(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_B; \boldsymbol{\theta}_t\right) \approx \sum_{b=1}^{B} \mathcal{A}\left(\boldsymbol{x}_b; \boldsymbol{\theta}_t\right). \tag{39}$$

This approximation is comparatively easy to implement: compute the AF for each point in the pool set and choose the B points who have the highest values of their AF. However, such naive batch construction methods still result in highly correlated queries Sener and Savarese (2018). We demonstrate this in Figure 1, where Subfigure (a) shows a the batch (black dots) collected by maximizing point-wise entropy and Subfigure (b) shows a batch collected by point-wise BALD. On the other hand, Subfigure (c) shows a batch AL method (Pinsler et al., 2019) that does encourage diversity across the batch. See Kirsch et al. (2019) for a batch AL extension of BALD.



(a) Entropy (b) BALD (c) Bayesian Coreset

Figure 1: Batch construction of different AL methods on *cifar10*, shown as a t-SNE projection (Maaten and Hinton, 2008). Given 5000 labeled points (colored by class), a batch of 200 points (black crosses) is queried.

Active Learning with Noisy Labels So far we have assumed that an oracle mechanism exists that can provide the true label y^*. As discussed in the previous section on crowdsourcing, such a mechanism does not always—and usually does not—exist. Thus, at each acquisition step, we may be provided with one or more noisy labels: $y_1, \ldots, y_L$, where L is the number of annotators. Thus this situation would require marrying the consensus-making techniques from crowdsourcing with the AL loop. This synthesis can be done most simply by using a naive aggregation mechanism over the labels, such as majority voting. In this case, the label with the most votes would then be added to the label set as the ground-truth. On the other hand, if we are using a more sophisticated model from Section 2, the first step would be to update the label model $p(\mathrm{y}|\cdot)$. Then the updated posterior over the truth, $p(\mathrm{t}|\mathrm{y})$, can be used in several ways. One would be to take the mode of $p(\mathrm{t}|\mathrm{y})$ and use that as y^*. Note that one major departure from the usual AL framework is that, in the case of noisy labelers, we are learning more about the label noise at every AL iteration, and thus we may want to use the new observations to update the truth inference for points obtained during previous iterations. In other words, the labeled set $\mathcal{D}_t$—which we usually assume is fixed except for the newly collected point / batch—can have arbitrary changes over time, driven by the updated model of label noise. If obtaining the labels is especially costly, we may want to go a step further and only query one of multiple labelers for each AL iteration. This is the setting considered and addressed by Yan et al. (2011). They use the conditional label model $p_l(\mathrm{y}_l|\mathrm{x}, \mathrm{t})$ from Yan et al. (2010) to query the (estimated) best labeler at each AL step.

3 Imitation Learning

Imitation learning (IL) is a variant of RL in which the reward function ($\mathcal{R}$) is unknown, and instead, observations of human behavior are provided instead. It is assumed that this human behavior is drawn from a policy that is nearly (but not exactly) optimal under the unknown reward function. Below I describe variants of IL, including reductions to supervised learning and ones that allow multiple human queries.

3.1 Behavior Cloning

Behavior cloning (BC) is the simplest of IL strategies, essentially reducing the problem to that of traditional supervised learning. BC assumes access to a data set:

$$\mathcal{D} \;=\; \{(s_n, a_n)\}_{n=1}^{N}\,, \qquad a_t \sim \pi_E(\mathrm{a}|s_t) \tag{40}$$

where π_E is the human expert's policy. These state-action pairs do not have to be sequential in time. BC then fits a policy $\pi_{\boldsymbol{\theta}}(\mathrm{a}|\mathrm{s})$ to this dataset using traditional supervised learning:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}\in\Theta}{\arg\min} \sum_{n=1}^{N} \ell\left(a_n, \pi_{\boldsymbol{\theta}}(\cdot|s_n)\right) \tag{41}$$

where ℓ is a suitable loss function that quantifies the distance between the expert's action and the one suggested by the policy. For example, one could use maximum likelihood estimation:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}\in\Theta}{\arg\min}\; \mathbb{E}_{\mathrm{s}}\left[\mathbb{KLD}\left[\,\pi_E(\mathrm{a}|\mathrm{s})\;||\;\pi_{\boldsymbol{\theta}}(\mathrm{a}|\mathrm{s})\,\right]\right] \;\approx\; \underset{\boldsymbol{\theta}\in\Theta}{\arg\max} \sum_{n=1}^{N} \log \pi_{\boldsymbol{\theta}}(a_n|s_n) \;+\; \text{const.}$$

Notice that the expectation over states is defined by the expert interacting with the environment—not the policy we aim to learn—this is a problem in that our policy is disconnected with the underlying MDP. Thus, while easy to implement, BC suffers from some obvious limitations:

- Training the policy is disconnected from the underlying MDP. In other words, the policy never 'sees' states that are generated by itself.
- The efficacy of BC depends on having excellent coverage of the state space.
- The policy could have good supervised learning performance, but this does not guarantee good performance under the MDP. In fact, theory has shown that supervised learning error can be small, but the policy still has quadratic error w.r.t. the reward.

Hence, BC seems best suited for fine-tuning policies that are already quite performant.

We can quantify BC's sub-optimality as follows. Assume that supervised learning has succeeded such that the optimal policy and the learned policy agree in their top-ranked action with high probability:

$$\mathbb{E}_{\mathrm{s}\sim d(\pi^*)}\left[\mathbb{I}\left[\underset{a\in\mathcal{A}}{\arg\max}\,\pi^*(\mathrm{a}|\mathrm{s}) \neq \underset{a\in\mathcal{A}}{\arg\max}\,\hat{\pi}(\mathrm{a}|\mathrm{s})\right]\right] = P(\pi^* \neq \hat{\pi}|s) \leq \epsilon \tag{42}$$

where ϵ is a small non-negative constant $\epsilon \in \mathbb{R}^{\geq 0}$ and the expectation is taken w.r.t. π^*'s stationary distribution over states. We are then interested in the difference in value functions under the optimal and learned policies, which can be founded as follows:

Proposition 3.1. Assume a normalized reward function $r_t \in [-1, 1]$ and both policies are deterministic such that they choose the top-ranked action. Moreover, assume that $P(\pi^* \neq \hat{\pi}|s) \leq \epsilon$ for $\epsilon \in \mathbb{R}^{\geq 0}$. The difference in value functions is then:

$$V^{\pi^*}(s) \;-\; V^{\hat{\pi}}(s) \;\leq\; \frac{2}{(1-\gamma)^2} \cdot \epsilon$$

where $\gamma \in [0, 1)$ is the discount factor.

The upper bound is linear in the supervised learning error ϵ but quadratic in the discount, meaning that the more future rewards are considered (i.e. γ closer to one), the worse the performance gap will be when the policy is deployed in the MDP. Thus we can say $2/(1-\gamma)^2$ 'amplifies' the supervised learning error ϵ.

Before going to the proof, first note that the maximum value of the value function is when the reward is maximized at one at every state: $\sum_{t=1}^{\infty} \gamma^t r_t = \sum_{t=1}^{\infty} \gamma^t = 1/(1-\gamma)$, where the infinite series converges due to it being a standard geometric series. Similarly, the value functions can take a minimum of $-1/(1-\gamma)$. Thus $2/(1-\gamma) \geq V^{\pi^*}(s) - V^{\hat{\pi}}(s) \geq 0$ and so we'd like an upper bound that's a function of ϵ. The proof is then:

$$\begin{aligned}
V^{\pi^*}(s) - V^{\hat{\pi}}(s) &= \frac{1}{1-\gamma} \cdot \mathbb{E}_{\mathrm{s}\sim d(\pi^*)} \left[A^{\hat{\pi}}(\mathrm{s}, \arg\max_{a\in\mathcal{A}} \pi^*(a|\mathrm{s})) \right] \\
&= \frac{1}{1-\gamma} \cdot \mathbb{E}_{\mathrm{s}\sim d(\pi^*)} \left[Q^{\hat{\pi}}(\mathrm{s}, \arg\max_{a\in\mathcal{A}} \pi^*(a|\mathrm{s})) \;-\; V^{\hat{\pi}}(\mathrm{s}) \right] \\
&= \frac{1}{1-\gamma} \cdot \mathbb{E}_{\mathrm{s}\sim d(\pi^*)} \left[Q^{\hat{\pi}}(\mathrm{s}, \arg\max_{a\in\mathcal{A}} \pi^*(a|\mathrm{s})) - \sum_{a'\in\mathcal{A}} \mathbb{I}[a' = \arg\max_{a\in\mathcal{A}} \hat{\pi}(a|\mathrm{s})] \cdot Q^{\hat{\pi}}(\mathrm{s}, a') \right] \\
&= \frac{1}{1-\gamma} \cdot \mathbb{E}_{\mathrm{s}\sim d(\pi^*)} \left[Q^{\hat{\pi}}(\mathrm{s}, \arg\max_{a\in\mathcal{A}} \pi^*(a|\mathrm{s})) - Q^{\hat{\pi}}(\mathrm{s}, \arg\max_{a\in\mathcal{A}} \hat{\pi}(a|\mathrm{s})) \right] \\
&\leq \frac{1}{1-\gamma} \cdot \mathbb{E}_{\mathrm{s}\sim d(\pi^*)} \Bigg[\mathbb{I}\left[\arg\max_{a\in\mathcal{A}} \pi^*(\mathrm{a}|\mathrm{s}) \neq \arg\max_{a\in\mathcal{A}} \hat{\pi}(\mathrm{a}|\mathrm{s}) \right] \cdot \frac{2}{1-\gamma} \\
&\qquad\qquad + \mathbb{I}\left[\arg\max_{a\in\mathcal{A}} \pi^*(\mathrm{a}|\mathrm{s}) = \arg\max_{a\in\mathcal{A}} \hat{\pi}(\mathrm{a}|\mathrm{s}) \right] \cdot 0 \Bigg] \\
&= \frac{1}{1-\gamma} \cdot \mathbb{E}_{\mathrm{s}\sim d(\pi^*)} \left[\mathbb{I}\left[\arg\max_{a\in\mathcal{A}} \pi^*(\mathrm{a}|\mathrm{s}) \neq \arg\max_{a\in\mathcal{A}} \hat{\pi}(\mathrm{a}|\mathrm{s}) \right] \cdot \frac{2}{1-\gamma} \right] \\
&= \frac{1}{1-\gamma} \cdot \frac{2}{1-\gamma} \cdot \mathbb{E}_{\mathrm{s}\sim d(\pi^*)} \left[\mathbb{I}\left[\arg\max_{a\in\mathcal{A}} \pi^*(\mathrm{a}|\mathrm{s}) \neq \arg\max_{a\in\mathcal{A}} \hat{\pi}(\mathrm{a}|\mathrm{s}) \right] \right] \\
&= \frac{1}{1-\gamma} \cdot \frac{2}{1-\gamma} \cdot \epsilon \\
&= \frac{2}{(1-\gamma)^2} \cdot \epsilon.
\end{aligned}$$

where the first identify is the *performance difference lemma*. The upper-bound created in the fifth line arises from the fact that if the policies choose the same actions ($\pi^* = \pi$), then the difference in the Q-functions is zero. Otherwise, we assume the difference is maximal at $2/(1-\gamma)$. This then reduces the problem into the probability of the policies being equal, which we have already defined to be ϵ. Perhaps a tighter bound exists by making a stronger assumption than that the difference in returns will be less than maximal.

3.2 Policy Learning via an Interactive Demonstrator

Policy Learning via an Interactive Demonstrator (PLID) is an extension of Behavior Cloning that allows the expert to be queried multiple times. Again we start with a set of demonstrations:

$$\mathcal{D}_0 \;=\; \{(s_{0,n}, a_{0,n})\}_{n=1}^{N}, \qquad a_{0,t} \sim \pi_E(\mathrm{a}|s_{0,t}). \tag{43}$$

Moreover, assume that we have used BC to fit a policy $pi_0(\mathrm{a}|\mathrm{s})$ to $\mathcal{D}_0$. PLID then proceeds with the following loop:

1. For $m = [1, M]$:
2. Rollout $\pi_{m-1}(\mathrm{a}|\mathrm{s})$ to collect a sequence of states $\{s_{m,n}\}_{n=1}^{N}$.
3. Query expert to gather corresponding actions for the observed states:

$$\mathcal{D}_m \;=\; \{(s_{m,n}, a_{m,n})\}_{n=1}^{N}, \qquad a_{m,t} \sim \pi_E(\mathrm{a}|s_{m,t}).$$

4. Apply BC to fit a policy $\pi_m(\mathrm{a}|\mathrm{s})$ to $\mathcal{D}_{0:m} = \mathcal{D}_0 \cup \ldots \cup \mathcal{D}_m$.

The PLID formulation above is known as *DAGGER* (Ross et al., 2011), as it aggregates the data collected from each loop. Alternatively, policies could be trained individually at each loop and then some form of model fusion performed. This is a better approach when $\mathcal{D}_m$ is large such that re-training on the combined data set takes a long time.

PLID solves BC's problems of being disconnected from sequential decision making (by rolling out the current policy at step #2) and possibly having limited state coverage (by re-querying the expert). We can see this explicitly by considering the behavior cloning objective after one step of interactive demonstration:

$$\hat{\boldsymbol{\theta}}_1 = \underset{\boldsymbol{\theta}\in\Theta}{\arg\min}\; \mathbb{E}_{\mathrm{s}\sim\pi_0}\left[\mathbb{KLD}\left[\; \pi_E(\mathrm{a}|\mathrm{s}) \;||\; \pi_{\boldsymbol{\theta}_0}(\mathrm{a}|\mathrm{s}) \;\right]\right] \;\approx\; \underset{\boldsymbol{\theta}\in\Theta}{\arg\max} \sum_{n=1}^{N} \log \pi_{\boldsymbol{\theta}_0}(a_n|s_n) \;+\; \text{const.}$$

Unlike above, where the expectation over states was under rollouts of the expert's policy, here they are states obtained by rolling out π_0 (the policy we're learning). This comes at the price, of course, of needing much more participation and effort from the expert. Moreover, if the state-space is continuous, it may be difficult for the expert to provide a demonstration exactly at the state found during the rollout. Imagine the case of finding a policy for driving a car: the car's position and the exact configuration of the controls must re-created and the expert thrust into the task of driving at that exact instant.

3.3 Distribution Matching

Distribution matching (DM) aims to solve BC's problems (namely, disconnection from sequential decision making / the underlying MDP) while not having PLID's limitation of intensive expert supervision. The key insight that differentiates DM is to consider the joint distribution over actions *and* states, instead of just the conditional distribution of actions given states. Call a T-length sequence of states an actions a *trajectory*, denoted as

$\tau = (s_0, a_1, s_1, \ldots, a_T, s_T)$, and consider a distribution over trajectories:

$$p_\pi(\boldsymbol{\tau}) = p(\mathrm{s}_0) \prod_{t=1}^{T} \pi(\mathrm{a}_t|\mathrm{s}_{t-1})\ \mathbb{P}(\mathrm{s}_t|\mathrm{s}_{t-1}, \mathrm{a}_t) \tag{44}$$

where π is a policy and $\mathbb{P}(\mathrm{s}_{t+1}|\mathrm{s}_t, \mathrm{a}_t)$ is the MDP's transition probability. Let the probability of a trajectory under the model be denoted $p_{\pi(\boldsymbol{\theta})}(\boldsymbol{\tau})$ and the expert demonstrator's distribution be denoted $p_E(\boldsymbol{\tau})$. We will not have an exact analytical form for $p_E(\boldsymbol{\tau})$; rather we see only samples

$$\mathcal{D} = \{(s_{n,0}, a_{n,1}, s_{n,1}, \ldots, a_{n,T}, s_{n,T})\}_{n=1}^{N}, \quad a_{j,t} \sim \pi_E(\mathrm{a}|s_{j,t}),\ s_{j,t+1} \sim \mathbb{P}(\mathrm{s}_{t+1}|s_{j,t}, a_{j,t}),$$

with the expert's policy $\pi_E(\mathrm{a}|s)$ again being assumed to be near optimal.

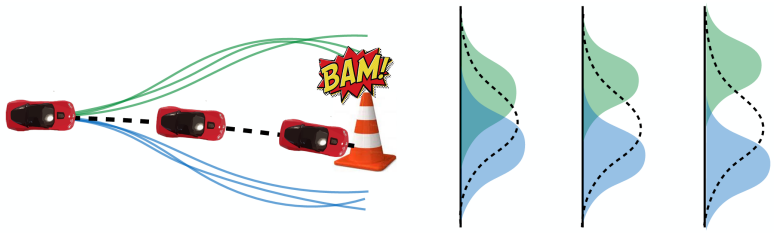


Figure 2: Imitation learning can fail when the target policy is multi-modal and the learned policy is not sufficiently expressive to cover multiple modes. Image reproduced from Ke et al. (2021).

Divergence Function Consider fitting a policy $\pi_{\boldsymbol{\theta}}$ by defining a divergence function that measures a notion of distance or dissimilarity between the model's distribution over trajectories and the expert's:

$$\mathbb{D}\left[\ p_{\pi(\boldsymbol{\theta})}(\boldsymbol{\tau})\ ||\ p_E(\boldsymbol{\tau})\ \right].$$

Specifically, Englert et al. (2013) proposes the Kullback–Leibler divergence:

$$\begin{aligned} \mathbb{KLD}\left[\ p_E(\boldsymbol{\tau})\ ||\ p_{\pi(\boldsymbol{\theta})}(\boldsymbol{\tau})\ \right] &= \sum_{\boldsymbol{\tau} \in (\mathcal{S}\times\mathcal{A})^T} p_E(\boldsymbol{\tau})\ \log \frac{p_E(\boldsymbol{\tau})}{p_{\pi(\boldsymbol{\theta})}(\boldsymbol{\tau})} \\ &\approx \sum_{n=1}^{N} -\log p_{\pi(\boldsymbol{\theta})}(\boldsymbol{\tau}_n)\ +\ \text{const.} \end{aligned} \tag{45}$$

where the second line corresponds to the negative log-likelihood computed under N samples. This formulation can be thought of as similar to BC (with a log-likelihood objective) but different in that the distribution over states is modeled as well as the conditional distribution over actions. However, sampling whole trajectories can be statistically difficult (for large

T), and thus a factorization over state-action pairs is often assumed (Englert et al., 2013):

$$\begin{aligned}\mathbb{KLD}\left[\, p_E(\boldsymbol{\tau}) \;||\; p_{\pi(\boldsymbol{\theta})}(\boldsymbol{\tau}) \,\right] \;&\approx\; \sum_{t=1}^{T} \mathbb{KLD}\left[\, p_E(\mathrm{s}_t, \mathrm{a}_t) \;||\; p_{\pi(\boldsymbol{\theta})}(\mathrm{s}_t, \mathrm{a}_t) \,\right] \\ &\approx\; \sum_{n=1}^{N}\sum_{t=1}^{T} -\left\{\log \pi_{\boldsymbol{\theta}}\left(a_{n,t} | s_{n,t-1}\right) \;+\; \log q_{\boldsymbol{\theta}}\left(s_{n,t}\right)\right\} \;+\; \text{const.}\end{aligned} \tag{46}$$

where $q_{\boldsymbol{\theta}}$ is a marginal distribution over states. We can think of this objective as a form of regularized BC, as we don't just want to fit the policy but also a regularizer that ensures the distribution over states matches that of the expert's trajectories. Despite the success of the above formulation, Ke et al. (2021) point out a general limitation in using the Kullback–Leibler divergence from expert to model, as p_π will be forced to place support everywhere that p_E does, and if our model is not sufficiently expressive, then we will learn solutions that try to interpolate across modes but capture none of them. Figure **??** demonstrates this problem: due to the unimodal distribution's need to cover both modes, the model tries to satisfy both modes, which causes the car to crash directly into the thing it was trying to avoid. One may then wish to turn to the reverse KLD, which is mode seeking since the expectation is now taken w.r.t. $p_{\pi(\boldsymbol{\theta})}$:

$$\mathbb{KLD}\left[\, p_{\pi(\boldsymbol{\theta})}(\boldsymbol{\tau}) \;||\; p_E(\boldsymbol{\tau}) \,\right] \;=\; \sum_{\boldsymbol{\tau} \in (\mathcal{S}\times\mathcal{A})^T} p_{\pi(\boldsymbol{\theta})}(\boldsymbol{\tau}) \;\log \frac{p_{\pi(\boldsymbol{\theta})}(\boldsymbol{\tau})}{p_E(\boldsymbol{\tau})}. \tag{47}$$

However, this objective is hard to optimize due to needing to evaluate probability density / mass for p_π *and* p_E—the latter of which we only can observe through samples.

Adversarial Imitation Learning *Generative Adversarial Imitation Learning* (GAIL)—inspired by the similarly named *Generative Adversarial Networks* (GANs)—are one scalable approach to estimate high-dimensional, mode-seeking divergences. They operate by changing the problem to one of two-sample testing: given samples from the model $\boldsymbol{\tau}_j \sim p_\pi$ and expert $\boldsymbol{\tau}_n \sim p_E$, can a binary classifier correctly predict the source of each?

$$\begin{aligned}\mathbb{J}_{\boldsymbol{\psi}}\left(\, p_{\pi(\boldsymbol{\theta})},\; p_E \,\right) \;&=\; \mathbb{E}_E\left[-\log h\left(\boldsymbol{\tau};\boldsymbol{\psi}\right)\right] \;+\; \mathbb{E}_\pi\left[-\log(1 - h\left(\boldsymbol{\tau};\boldsymbol{\psi}\right))\right] \\ &\approx\; \left(\frac{1}{N}\sum_{n=1}^{N} -\log h\left(\boldsymbol{\tau}_n;\boldsymbol{\psi}\right)\right) \;+\; \left(\frac{1}{J}\sum_{j=1}^{J} -\log(1 - h\left(\boldsymbol{\tau}_j;\boldsymbol{\psi}\right))\right)\end{aligned} \tag{48}$$

where $h\left(\boldsymbol{\tau};\boldsymbol{\psi}\right) : \tau \mapsto (0,1)$ represents the binary classifier. $\mathbb{J}_{\boldsymbol{\psi}}$ will be minimized when the model's and expert's sample trajectories are perfectly discriminated. GAIL can also be formulated over state-action pairs, if full trajectories are too challenging, as done above:

$$\approx\; \left(\frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T} -\log h\left(s_{n,t}, a_{n,t};\boldsymbol{\psi}\right)\right) \;+\; \left(\frac{1}{J}\sum_{j=1}^{J}\sum_{t=1}^{T} -\log(1 - h\left(s_{j,t}, a_{j,t};\boldsymbol{\psi}\right))\right). \tag{49}$$

With the adversarial objective in hand, we can use it to optimize the policy $\pi_{\boldsymbol{\theta}}$ by an adversarial max-min problem where the classifier seeks to minimize the classification loss while the policy seeks to maximize it (so that the samples look to be from indistinguishable

sources):

$$(\boldsymbol{\theta}^*, \boldsymbol{\psi}^*) = \arg\max_{\boldsymbol{\theta}} \ \arg\min_{\boldsymbol{\psi}} \ \mathbb{J}_{\boldsymbol{\psi}} \left(\ p_{\pi(\boldsymbol{\theta})}, \ p_E \ \right). \tag{50}$$

The optimal discriminator $\boldsymbol{\psi}^*$ satisfies:

$$\log h\left(\boldsymbol{\tau}; \boldsymbol{\psi}^*\right) \ - \ \log\left(1 - h\left(\boldsymbol{\tau}; \boldsymbol{\psi}^*\right)\right) \ = \ \log \frac{p_E(\boldsymbol{\tau})}{p_{\pi(\boldsymbol{\theta})}(\boldsymbol{\tau})}, \tag{51}$$

and hopefully the ratio $p_E(\boldsymbol{\tau})/p_{\pi(\boldsymbol{\theta}^*)}(\boldsymbol{\tau}) \approx 1$, meaning that the distributions have been successfully matched. In summary, GAIL is an attractive method as it does not require any instantiation of a density / mass function and can operate solely using samples from the expert and policy (i.e. rollouts). This comes at some cost to sample efficiency and a more difficult optimization problem, but those tradeoffs seem to not be a limitation in practice.

3.4 Inverse Reinforcement Learning

Inverse RL (IRL) aims to learn the *reward function* from expert demonstrations. Hence the term *inverse* since RL usually assumes the reward function is given. A policy is learned as well, using the learned reward function. Thus, learning both functions could be a challenge, but IRL assumes that learning the reward function is statistically easier than learning the policy. Specifically, we assume the reward is a parameterized function: $\mathcal{R}(\mathrm{s}_t) \approx R_{\boldsymbol{\phi}}(\mathrm{s}_t)$ where $\boldsymbol{\phi}$ are the parameters of the model used to learn the reward function. For simplicity, we assume the reward is only a function of the current state. Now the RL optimization objective is a function of both the policy's parameters ($\boldsymbol{\theta}$) and reward model's ($\boldsymbol{\phi}$):

$$\begin{aligned} \mathcal{J}(\boldsymbol{\theta}, \boldsymbol{\phi}) \ &= \ \sum_{s \in \mathcal{S}} d^{\pi(\boldsymbol{\theta})}(s) \sum_{a \in \mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \cdot \mathbb{E}_{\pi(\boldsymbol{\theta})} \left[\ G_{\boldsymbol{\phi}}\left(\mathrm{s}_t; \{\mathrm{a}_t, \mathrm{a}_{t+1}, \ldots\}\right) \ | \ \mathrm{s}_t = s, \mathrm{a}_t = a \ \right] \\ &= \ \sum_{s \in \mathcal{S}} d^{\pi(\boldsymbol{\theta})}(s) \sum_{a \in \mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \cdot \mathbb{E}_{\pi(\boldsymbol{\theta})} \left[\ \sum_{t'=1}^{\infty} \gamma_{t'} \cdot R_{\boldsymbol{\phi}}\left(\mathrm{s}_{t+t'}\right) \ | \ \mathrm{s}_t = s, \mathrm{a}_t = a \ \right]. \end{aligned} \tag{52}$$

As we will see below, solving this difficult optimization problem will require assuming some constraints on the policy and/or reward function. Yet in general, IRL follows the algorithmic sketch below.

1. Fit $R_{\boldsymbol{\phi}}$ to the expert demonstrations.
2. Given $R_{\boldsymbol{\phi}}$, fit $\pi_{\boldsymbol{\theta}}$ using traditional RL.
3. Compare $\pi_{\boldsymbol{\theta}}$ vs π_E, the model and expert policies.
4. If the difference in policies is sufficiently large, repeat the loop.

This sketch should make clear that IRL can be computationally expensive, since traditional RL is an *inner loop* of the procedure. Yet we must pay some price for training under a sequential setting and not assuming repeated queries to the human.

Linear Reward Function Let's start simple, by considering a linear reward function:

$$R_{\boldsymbol{\phi}}(\mathrm{s}) = \boldsymbol{\phi}^\top \boldsymbol{\psi}(\mathrm{s})$$

where $\boldsymbol{\phi} \in \mathbb{R}^D$, such that $|\boldsymbol{\phi}|_1 \leq 1$, are the parameters and $\boldsymbol{\psi} : \mathcal{S} \mapsto [0,1]^D$ is a binary vector of features that describes state $\mathbf{s}$. Under this assumption, the value function is also linear:

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}\left[\, G\left(\mathrm{s}_t; \{\mathrm{a}_t, \mathrm{a}_{t+1}, \ldots\}\right) \mid \mathrm{s}_t = s \,\right] \\ &= \mathbb{E}_{\pi}\left[\, \sum_{t'=1}^{\infty} \gamma_{t'} \cdot R_{\boldsymbol{\phi}}\left(\mathrm{s}_{t+t'}\right) \mid \mathrm{s}_t = s \,\right] \\ &= \mathbb{E}_{\pi}\left[\, \sum_{t'=1}^{\infty} \gamma_{t'} \cdot \boldsymbol{\phi}^{\top} \boldsymbol{\psi}\left(\mathrm{s}_{t+t'}\right) \mid \mathrm{s}_t = s \,\right] \\ &= \boldsymbol{\phi}^{\top} \underbrace{\mathbb{E}_{\pi}\left[\, \sum_{t'=1}^{\infty} \gamma_{t'} \cdot \boldsymbol{\psi}\left(\mathrm{s}_{t+t'}\right) \mid \mathrm{s}_t = s \,\right]}_{\boldsymbol{\mu}_{\pi}(s)} \end{aligned}$$

where $\boldsymbol{\mu}_{\pi}$ is a feature vector quantifying the states expected to be visited under policy π. For the expert, we see a finite set of states and thus can compute the demonstrator's empirical embedding as:

$$\boldsymbol{\mu}_E = \frac{1}{N} \sum_{n=1}^{N} \sum_{t'=1}^{\infty} \gamma_{t'} \cdot \boldsymbol{\psi}\left(s_{n,t'}\right).$$

Writing the IRL objective from Equation 52, we have:

$$\begin{aligned} \mathcal{J}(\boldsymbol{\theta}, \boldsymbol{\phi}) &= \sum_{s \in \mathcal{S}} d^{\pi(\boldsymbol{\theta})}(s) \cdot V^{\pi}(s) \\ &= \sum_{s \in \mathcal{S}} d^{\pi(\boldsymbol{\theta})}(s)\ \boldsymbol{\phi}^{\top}\ \mathbb{E}_{\pi}\left[\, \sum_{t'=1}^{\infty} \gamma_{t'} \cdot \boldsymbol{\psi}\left(\mathrm{s}_{t+t'}\right) \mid \mathrm{s}_t = s \,\right] \\ &= \boldsymbol{\phi}^{\top}\ \sum_{s \in \mathcal{S}} d^{\pi(\boldsymbol{\theta})}(s)\ \mathbb{E}_{\pi}\left[\, \sum_{t'=1}^{\infty} \gamma_{t'} \cdot \boldsymbol{\psi}\left(\mathrm{s}_{t+t'}\right) \mid \mathrm{s}_t = s \,\right] \\ &= \boldsymbol{\phi}^{\top}\ \sum_{s \in \mathcal{S}} d^{\pi(\boldsymbol{\theta})}(s)\ \boldsymbol{\mu}_{\pi}\left(s\right) \\ &= \boldsymbol{\phi}^{\top}\ \mathbb{E}_{d(s;\boldsymbol{\theta})}\left[\boldsymbol{\mu}_{\pi}\left(s\right)\right] \end{aligned} \tag{53}$$

We can then consider the difference between the objective achieved by the expert and model:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\phi}) &= \left|\ \mathcal{J}(\boldsymbol{\theta}^{*}_{E}, \boldsymbol{\phi})\ -\ \mathcal{J}(\boldsymbol{\theta}, \boldsymbol{\phi})\ \right| \\ &= \left|\ \boldsymbol{\phi}^{\top}\ \boldsymbol{\mu}_E\ -\ \boldsymbol{\phi}^{\top}\ \mathbb{E}_{d(s;\boldsymbol{\theta})}\left[\boldsymbol{\mu}_{\pi}\left(s\right)\right]\ \right| \\ &\leq \left|\left|\ \boldsymbol{\mu}_E\ -\ \mathbb{E}_{d(s;\boldsymbol{\theta})}\left[\boldsymbol{\mu}_{\pi}\left(s\right)\right]\ \right|\right|_2^2 \end{aligned} \tag{54}$$

where the inequality arises from the assumption that $|\boldsymbol{\phi}|_1$ is bounded. Firstly, this inequality demonstrates a fundamental connection to distribution matching, as under these assumptions, matching the first moment of the state features upper-bounds $\mathcal{L}(\boldsymbol{\phi})$. Thus, minimizing the difference between the expected state features (i.e. moment matching) will minimize the gap in between the expert's and model's objectives $\mathcal{J}$, for any choice of $\boldsymbol{\phi}$.

However, simply examining the state distribution neglects learning $\boldsymbol{\phi}$, and in turn, obtaining a form for the reward function. One could question why a form for the reward is

even needed, if obtaining the policy simply by distribution matching will do, but if it is, then $\boldsymbol{\phi}$ can be obtained as follows. Abbeel and Ng (2004) propose an iterative max-margin approach that first fits $\boldsymbol{\phi}$, runs RL to find $\boldsymbol{\theta}$, and repeats. Yet a more general approach was proposed by Syed and Schapire (2007) based on an adversarial game:

$$\boldsymbol{\theta}^* \;=\; \arg\max_{\boldsymbol{\theta}} \; \min_{\boldsymbol{\phi}} \; \left(\boldsymbol{\phi}^\top \; \mathbb{E}_{d(s;\boldsymbol{\theta})} \left[\boldsymbol{\mu}_\pi \left(s \right) \right] \;-\; \boldsymbol{\phi}^\top \; \boldsymbol{\mu}_E \right). \tag{55}$$

The intuition is that, for a fixed reward (i.e. given $\boldsymbol{\theta}$), the policy can be chosen so that the model achieves a better reward than the expert achieved. However, the environment is then free to change the reward in order to minimize the quantity, thus choosing in favor of the expert. This bares some similarity to GAIL, with $\boldsymbol{\phi}$ acting in the spirit of the discriminator. Yet, of course, now the 'discriminator' has the interpretation as a reward function.

Another approach to IRL of note is *maximum entropy inverse reinforcement learning* (MaxEnt IRL). This approach provides a more explicit bridge between distribution matching and previous IRL approach. Let the state features for one of the expert's trajectories be denoted $\boldsymbol{\mu}_{E,n} = \sum_{t'=1} \gamma_{t'} \cdot \boldsymbol{\psi}(s_{n,t'})$, such that $\boldsymbol{\mu}_E = (1/N) \sum_n \boldsymbol{\mu}_{E,n}$. MaxEnt IRL then models the probability of a trajectory by exponentiating the linear reward model from above:

$$p\left(\boldsymbol{\tau}_n; \boldsymbol{\phi}\right) = \frac{1}{Z(\boldsymbol{\phi})} \; \exp\left\{ \boldsymbol{\phi}^\top \boldsymbol{\mu}_{E,n} \right\}, \quad Z(\boldsymbol{\phi}) = \int_{\boldsymbol{\tau} \in (\mathcal{S},\mathcal{A})^T} \exp\left\{ \boldsymbol{\phi}^\top \boldsymbol{\mu}_{\boldsymbol{\tau}} \right\} \; d\boldsymbol{\tau} \tag{56}$$

where $Z(\boldsymbol{\phi})$ is known as the partition function (or normalizing constant) that ensures the probability is normalized by computing the exponentiated reward over all possible states. Now consider optimizing $\boldsymbol{\phi}$ via gradient ascent of the log-likelihood under the expert's demonstrations. The gradient calculation is:

$$\begin{aligned}
\nabla_{\boldsymbol{\phi}} \frac{1}{N} \sum_{n=1}^{N} \log p\left(\boldsymbol{\tau}_n; \boldsymbol{\phi}\right) \;&=\; \frac{1}{N} \nabla_{\boldsymbol{\phi}} \sum_{n=1}^{N} \left[\boldsymbol{\phi}^\top \boldsymbol{\mu}_{E,n} \;-\; \log Z(\boldsymbol{\phi}) \right] \\
&=\; \frac{1}{N} \sum_{n=1}^{N} \nabla_{\boldsymbol{\phi}} \left[\boldsymbol{\phi}^\top \boldsymbol{\mu}_{E,n} \right] \;-\; \frac{N}{N} \nabla_{\boldsymbol{\phi}} \log Z(\boldsymbol{\phi}) \\
&=\; \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{\mu}_{E,n} \;-\; \frac{1}{Z(\boldsymbol{\phi})} \int_{\boldsymbol{\tau} \in (\mathcal{S},\mathcal{A})^T} \nabla_{\boldsymbol{\phi}} \exp\left\{ \boldsymbol{\phi}^\top \boldsymbol{\mu}_{\boldsymbol{\tau}} \right\} \; d\boldsymbol{\tau} \\
&=\; \boldsymbol{\mu}_E \;-\; \frac{1}{Z(\boldsymbol{\phi})} \int_{\boldsymbol{\tau} \in (\mathcal{S},\mathcal{A})^T} \exp\left\{ \boldsymbol{\phi}^\top \boldsymbol{\mu}_{\boldsymbol{\tau}} \right\} \; \boldsymbol{\mu}_{\boldsymbol{\tau}} \; d\boldsymbol{\tau} \\
&=\; \boldsymbol{\mu}_E \;-\; \int_{\boldsymbol{\tau} \in (\mathcal{S},\mathcal{A})^T} p\left(\boldsymbol{\tau}; \boldsymbol{\phi}\right) \; \boldsymbol{\mu}_{\boldsymbol{\tau}} \; d\boldsymbol{\tau} \\
&=\; \boldsymbol{\mu}_E \;-\; \mathbb{E}_{\tau|\phi} \left[\boldsymbol{\mu}_{\boldsymbol{\tau}} \right]
\end{aligned} \tag{57}$$

where $\boldsymbol{\mu}_E$ is the expert's state features (over all trajectories) and $\mathbb{E}_{\tau|\phi} \left[\boldsymbol{\mu}_{\boldsymbol{\tau}} \right]$ is the expected state features under $p\left(\boldsymbol{\tau}; \boldsymbol{\phi}\right)$. Note the similarity to the upper-bound above where the expert's and policy's expected features are matched.

Yet also note that no policy has been introduced here. To parameterize a policy, MaxEnt IRL now assumes the following implied policy: $\pi(\mathrm{a}|\mathrm{s}) \;\propto\; Q^\pi(\mathrm{s}, \mathrm{a})$, which is simply the policy that chooses actions with probability proportional to their expected return. Now consider that $p\left(\boldsymbol{\tau}; \boldsymbol{\phi}\right)$ also places high probability on states with the highest rewards. Thus the

contribution of the partition function in the gradient is approximated with a policy as:

$$\mathbb{E}_{\tau|\phi}\left[\boldsymbol{\mu}_{\boldsymbol{\tau}}\right] \;\approx\; \mathbb{E}_{d(s;\boldsymbol{\theta})}\left[\boldsymbol{\mu}_{\pi}\left(s\right)\right] \tag{58}$$

where the RHS term is the expected state features under policy $\pi_{\boldsymbol{\theta}}(\mathrm{a}|\mathrm{s})$. Mechanistically, this is doing the correct thing as the gradient will be zero when $\mathbb{E}_{d(s;\boldsymbol{\theta})}\left[\boldsymbol{\mu}_{\pi}\left(s\right)\right]$ and $\boldsymbol{\mu}_E$ have been matched, i.e. their difference is zero. The full gradient update can be computed as:

$$\boldsymbol{\phi}_{t+1} \;=\; \boldsymbol{\phi}_t \;+\; \alpha \cdot \left(\boldsymbol{\mu}_E \;-\; \mathbb{E}_{d(s;\boldsymbol{\theta}_t)}\left[\boldsymbol{\mu}_{\pi}\left(s\right)\right]\right)$$

where $\boldsymbol{\theta}_t$ is obtained by running RL using the reward function $R_{\boldsymbol{\phi}_t}(\mathrm{s}) = \boldsymbol{\phi}_t^{\top}\boldsymbol{\psi}(\mathrm{s})$.

3.5 Reinforcement Learning with Human Feedback

Reinforcement Learning with Human Feedback (RLHF) (Christiano et al., 2017) is a form of IRL that has been made popular by its success in finetuning ChatGPT. Like previous methods, its core goal is to (i) learning a reward function from human demonstrations, and then (2) train (or more often, finetune an already trained model) using the learned reward function as the learning signal. Unlike previous imitation learning strategies discussed above, RLHF assumes we have access to ranked or paired demonstrations:

$$\mathcal{D}_{+,-} \;=\; \left\{\left(\boldsymbol{\tau}_n^{+}, \boldsymbol{\tau}_n^{-}\right)\right\}_{n=1}^{N}$$

where τ_n^{+} is a positive trajectory (i.e. a sequences of states and actions) that has been deemed by a human to encode more desirable behavior than the negative trajectory τ_n^{-}. These trajectory pairs could be describing a similar state in the environment or could simply be randomly paired by have two large batches of positive and negative demonstrations.

Given these ranked trajectories, the next step is to define a reward model $R_{\boldsymbol{\phi}} : \mathbf{T} \mapsto \mathbb{R}^{\geq 0}$ with parameters $\boldsymbol{\phi}$. We then define a *Bradley-Terry model* (Bradley and Terry, 1952) that encodes the probability of one trajectory being better than another, as a function of the reward model:

$$p_{\boldsymbol{\phi}}(\boldsymbol{\tau}^{+} \succ \boldsymbol{\tau}^{-}) \;=\; \sigma\left(R\left(\boldsymbol{\tau}^{+};\boldsymbol{\phi}\right) \;-\; R\left(\boldsymbol{\tau}^{-};\boldsymbol{\phi}\right)\right) \tag{59}$$

where $\sigma(\cdot)$ denotes the logistic function. Thus we see that the probability of one trajectory being preferable over another is simply the difference in their reward functions normalized to $(0, 1)$. Using this model, we can then define likelihood of the reward parameters $\boldsymbol{\phi}$:

$$\begin{aligned}
\ell\left(\boldsymbol{\phi};\mathcal{D}_{+,-}\right) \;&=\; \log\left\{\prod_{n=1}^{N} p_{\boldsymbol{\phi}}(\boldsymbol{\tau}_n^{+} \succ \boldsymbol{\tau}_n^{-})\right\} \\
&=\; \sum_{n=1}^{N} \log p_{\boldsymbol{\phi}}(\boldsymbol{\tau}_n^{+} \succ \boldsymbol{\tau}_n^{-}) \\
&=\; \sum_{n=1}^{N} \log \sigma\left(R\left(\boldsymbol{\tau}_n^{+};\boldsymbol{\phi}\right) \;-\; R\left(\boldsymbol{\tau}_n^{-};\boldsymbol{\phi}\right)\right).
\end{aligned}$$

Usually the reward function is taken to be a neural network of some form and $\ell\left(\boldsymbol{\phi};\mathcal{D}_{+,-}\right)$ is optimized with gradient ascent. After the reward model is fit, then $R\left(\boldsymbol{\tau}_n^{+};\boldsymbol{\phi}\right)$ can be

plugged into any suitable RL framework to learn a policy.

Difference from Traditional Inverse RL Notice that this setup is much simpler than the inverse RL formulations above, which usually require additional assumptions about the form of the reward function or how it can be learned in tandem with a policy. In other words, before we only has *positive* demonstrations and thus couldn't train a reward function on them directly since it has no exampled of what negative behaviors might look like. One had to assume that roughly everything not in the demonstration set was a negative behavior. Here have positive and negative trajectories allows the reward model to see both extremes and thus be learned directly without an inner-loop of RL / policy fitting.

Why won't Behavior Cloning suffice? Given that we have demonstrations from reliable humans, it is tempting to also consider a behavior cloning objective that uses both positive and negative examples:

$$\hat{\theta} = \arg\min_{\theta \in \Theta} \; \mathbb{E}_{\boldsymbol{\tau}^-} \left[\log \pi_{\theta}(\mathrm{a}^-|\mathrm{s}^-) \right] - \mathbb{E}_{\boldsymbol{\tau}^+} \left[\log \pi_{\theta}(\mathrm{a}^+|\mathrm{s}^+) \right] \tag{60}$$

where this objective aims to directly maximize the probability of the positive trajectories while minimizing the probability of the negative trajectories under the policy. This could work in the usual cases in which BC succeeds (such as a small state space). Yet, especially in domains such as language in which there are multiple equivalent solutions, interacting with the underlying MDP allows a policy to discover these symmetries instead of overfitting to precisely what's given in the positive demonstrations.

Direct Preference Optimization One method that bridges RLHF and behavior cloning is *direct preference optimization* (DPO) (Rafailov et al., 2023). It works as follows. Firstly, they notice that in RLHF, the final policy is usually trained with a regularized objective, to keep the final policy near the initial (assuming some good scheme exists for pre-training the policy, like next-token modeling with language):

$$\boldsymbol{\theta}^* \;=\; \arg\max_{\theta \in \Theta} \mathbb{E}_{\boldsymbol{\tau} \sim \pi_{\theta}} \left[R\left(\boldsymbol{\tau}; \boldsymbol{\phi}\right) \;-\; \mathbb{KLD}\left[\pi_{\theta}(\mathrm{a}|\mathrm{s}) || \pi_0(\mathrm{a}|\mathrm{s}) \right] \right].$$

Under this optimization problem, the optimal policy has the form:

$$\pi^*(\mathrm{a}|\mathrm{s}) \;=\; \pi_0(\mathrm{a}|\mathrm{s}) \cdot \frac{1}{\mathcal{Z}(\boldsymbol{\phi})} \cdot \exp\left\{ R\left(\boldsymbol{\tau}; \boldsymbol{\phi}\right) \right\}, \quad \mathcal{Z}(\boldsymbol{\phi}) \;=\; \sum_{\boldsymbol{\tau} \in \mathbf{T}} \exp\left\{ R\left(\boldsymbol{\tau}; \boldsymbol{\phi}\right) \right\} \sum_{(s,a) \in \boldsymbol{\tau}} \pi_0(a|s).$$

Solving for the reward function (outside of $\mathcal{Z}(\boldsymbol{\phi})$), we then have:

$$R\left(\boldsymbol{\tau}; \boldsymbol{\phi}\right) \;=\; \sum_{(\mathrm{s},\mathrm{a}) \in \boldsymbol{\tau}} \log \frac{\pi^*(\mathrm{a}|\mathrm{s})}{\pi_0(\mathrm{a}|\mathrm{s})} \;+\; \log \mathcal{Z}(\boldsymbol{\phi}).$$

Plugging in this form for the reward function into the Bradley-Terry model, we have:

$$
\begin{aligned}
p_{\boldsymbol{\phi}}(\boldsymbol{\tau}^+ \succ \boldsymbol{\tau}^-) \;&=\; \sigma\left(R\left(\boldsymbol{\tau}^+;\boldsymbol{\phi}\right) \;-\; R\left(\boldsymbol{\tau}^-;\boldsymbol{\phi}\right)\right) \\
&=\; \sigma\left(\sum_{(\mathrm{s}^+,\mathrm{a}^+)\in\boldsymbol{\tau}^+} \log\frac{\pi^*(\mathrm{a}^+|\mathrm{s}^+)}{\pi_0(\mathrm{a}^+|\mathrm{s}^+)} \;+\; \log\mathcal{Z}(\boldsymbol{\phi}) \;-\; \sum_{(\mathrm{s}^-,\mathrm{a}^-)\in\boldsymbol{\tau}^-} \log\frac{\pi^*(\mathrm{a}^-|\mathrm{s}^-)}{\pi_0(\mathrm{a}^-|\mathrm{s}^-)} \;-\; \log\mathcal{Z}(\boldsymbol{\phi})\right) \\
&=\; \sigma\left(\sum_{(\mathrm{s}^+,\mathrm{a}^+)\in\boldsymbol{\tau}^+} \log\frac{\pi^*(\mathrm{a}^+|\mathrm{s}^+)}{\pi_0(\mathrm{a}^+|\mathrm{s}^+)} \;-\; \sum_{(\mathrm{s}^-,\mathrm{a}^-)\in\boldsymbol{\tau}^-} \log\frac{\pi^*(\mathrm{a}^-|\mathrm{s}^-)}{\pi_0(\mathrm{a}^-|\mathrm{s}^-)}\right).
\end{aligned}
\tag{61}
$$

where the difficult-to-compute term $\mathcal{Z}(\boldsymbol{\phi})$ cancels out. This equation is in terms of the optimal policy π^* for a specific (implied) reward model $R\left(\boldsymbol{\tau}^+;\boldsymbol{\phi}\right)$, but we can instead directly parameterize the policy to devise a learning objective that by-passes learning a reward function as an intermediate step:

$$
\begin{aligned}
\ell\left(\boldsymbol{\theta};\mathcal{D}_{+,-}\right) \;&=\; \sum_{n=1}^{N} \log p(\boldsymbol{\tau}_n^+ \succ \boldsymbol{\tau}_n^-) \\
&=\; \sum_{n=1}^{N} \log\sigma\left(\sum_{(s^+,a^+)\in\boldsymbol{\tau}_n^+} \log\frac{\pi_{\boldsymbol{\theta}}(a^+|s^+)}{\pi_0(\mathrm{a}^+|\mathrm{s}^+)} \;-\; \sum_{(s^-,a^-)\in\boldsymbol{\tau}_n^-} \log\frac{\pi_{\boldsymbol{\theta}}(a^-|s^-)}{\pi_0(a^-|s^-)}\right).
\end{aligned}
$$

Removing the pre-trained policy π_0, we have:

$$
\ell\left(\boldsymbol{\theta};\mathcal{D}_{+,-}\right) \;=\; \sum_{n=1}^{N} \log\sigma\left(\sum_{(s^+,a^+)\in\boldsymbol{\tau}_n^+} \log\pi_{\boldsymbol{\theta}}(a^+|s^+) \;-\; \sum_{(s^-,a^-)\in\boldsymbol{\tau}_n^-} \log\pi_{\boldsymbol{\theta}}(a^-|s^-)\right).
$$

Going back to the behavior cloning approach in Equation 60, we see that these objectives are very similar, and the only material difference is that DPO wraps the policy terms inside the logsistic function. We can get a better understanding of the mechanics by looking at the gradient:

$$
\begin{aligned}
&\nabla_{\boldsymbol{\theta}}\ell\left(\boldsymbol{\theta};\mathcal{D}_{+,-}\right) \;= \\
&\sum_{n=1}^{N}(1-p(\boldsymbol{\tau}_n^+ \succ \boldsymbol{\tau}_n^-))\left[\sum_{(s^+,a^+)\in\boldsymbol{\tau}_n^+} \nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(a^+|s^+) \;-\; \sum_{(s^-,a^-)\in\boldsymbol{\tau}_n^-} \nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(a^-|s^-)\right],
\end{aligned}
$$

and thus we see the gradient of the difference in the policies is weighted by one minus the probability of the positive trajectory being preferred. Thus the policy is updated only when the implied reward model has the incorrect preference and cannot continue to optimize the policy forever to overfit on the positive trajectory. This is experimentally confirmed by Rafailov et al. (2023), as their experiments show the model does not perform well without the $(1-p(\boldsymbol{\tau}_n^+ \succ \boldsymbol{\tau}_n^-))$ term.

Bibliography

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1,

2004.

Ralph Allan Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39:324, 1952. URL `https://api.semanticscholar.org/CorpusID:125209808`.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Sanjoy Dasgupta. Analysis of a greedy active learning strategy. In *Advances in Neural Information Processing Systems*, pages 337–344, 2005.

Alexander Philip Dawid and Allan M Skene. Maximum likelihood estimation of observer error-rates using the em algorithm. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 28(1):20–28, 1979.

Peter Englert, Alexandros Paraschos, Jan Peters, and Marc Peter Deisenroth. Model-based imitation learning by probabilistic trajectory matching. In *2013 IEEE international conference on robotics and automation*, pages 1922–1927. IEEE, 2013.

Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 42:427–486, 2011.

Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. Bayesian active learning for classification and preference learning. *arXiv Preprint arXiv:1112.5745*, 2011.

Sham M. Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning*, 2002.

Liyiming Ke, Sanjiban Choudhury, Matt Barnes, Wen Sun, Gilwoo Lee, and Siddhartha Srinivasa. Imitation learning as f-divergence minimization. In *Algorithmic Foundations of Robotics XIV: Proceedings of the Fourteenth Workshop on the Algorithmic Foundations of Robotics 14*, pages 313–329. Springer, 2021.

Andreas Kirsch, Joost van Amersfoort, and Yarin Gal. BatchBALD: Efficient and diverse batch acquisition for deep Bayesian active learning. *arXiv preprint arXiv:1906.08158*, 2019.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

David JC MacKay. Information-based objective functions for active data selection. *Neural computation*, 4(4):590–604, 1992.

Robert Pinsler, Jonathan Gordon, Eric Nalisnick, and José Miguel Hernández-Lobato. Bayesian batch active learning as sparse subset approximation. *Advances in neural information processing systems*, 32, 2019.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Neural Information Processing Systems*, 2023.

Vikas C. Raykar, Shipeng Yu, Linda H. Zhao, Gerardo Hermosillo Valadez, Charles Florin, Luca Bogoni, and Linda Moy. Learning from crowds. *Journal of Machine Learning Research*, 11(43):1297–1322, 2010. URL `http://jmlr.org/papers/v11/raykar10a.html`.

Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

Nicholas Roy and Andrew McCallum. Toward optimal active learning through sampling estimation of error reduction. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 441–448, 2001.

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations*, 2018.

Burr Settles. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–114, 2012.

Umar Syed and Robert E Schapire. A game-theoretic approach to apprenticeship learning. *Advances in neural information processing systems*, 20, 2007.

Yan Yan, Romer Rosales, Glenn Fung, Mark Schmidt, Gerardo Hermosillo, Luca Bogoni, Linda Moy, and Jennifer Dy. Modeling annotator expertise: Learning when everybody knows a bit of something. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010.

Yan Yan, Glenn M Fung, Rómer Rosales, and Jennifer G Dy. Active learning from crowds. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 1161–1168, 2011.