Research Journal

Vadim Smolyakov (vss@csail.mit.edu)

February 17, 2021

Contents

1 Bayesian Non-parametrics

1.1 Gaussian Process

Gaussian processes (GPs) define a prior over functions that can be updated to a posterior once we have observed data. In a supervised setting, the function gives a mapping between the data points x_i and the target value y_i: $y_i = f(x_i)$. Gaussian processes infer a distribution over functions given the data $p(f|x, y)$ and then use it to make predictions given new data. A GP assumes that the function is defined at a finite and arbitrary chosen set of points $x_1, ..., x_n$, such that $p(f(x_1), ..., f(x_n))$ is jointly Gaussian with mean $\mu(x)$ and covariance $\Sigma(x)$, where $\Sigma_{ij} = \kappa(x_i, x_j)$ and κ is a positive definite kernel function.

Supervised learning can be divided into regression (prediction of continuous quantities) and classification (prediction of discrete class labels). Consider a simple regression problem:

$$f(x) = x^T w \quad y = f(x) + \epsilon \quad \epsilon \sim N(0, \sigma_n^2) \tag{1}$$

Assuming independent and identically distributed noise, we can write down the likelihood function:

$$p(y|X, w) = \prod_{i=1}^{n} p(y_i|x_i, w) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_n} \exp\{-\frac{(y_i - x_i^T w)^2}{2\sigma_n^2}\} \sim N(Xw, \sigma_n^2 I) \tag{2}$$

In Bayesian framework, we need to specify a prior over the parameters: $w \sim N(0, \Sigma_p)$ Writing only the terms of the likelihood and the prior which depend on the weights, we get:

$$\begin{aligned} p(w|X, y) &\propto \exp\{-\frac{1}{2\sigma_n^2}||y - Xw||^2\} \exp\{-\frac{1}{2} w^T \Sigma_p^{-1} w\} & (3)\\ &\propto \exp\{-\frac{1}{2}(w - \bar{w})(\frac{1}{\sigma_n^2} XX^T + \Sigma_p^{-1})(w - \bar{w})\} & (4)\\ &\sim N(\frac{1}{\sigma_n^2} A^{-1} Xy, A^{-1}) & (5) \end{aligned}$$

where $A = \sigma_n^{-2} XX^T + \Sigma_p^{-1}$. Thus, we have a closed form posterior distribution over the parameters w. To make predictions using this equation, we need to invert the matrix A. Assuming the observations are noiseless, we want to predict the function outputs $y_* = f(x_*)$. Consider the following joint GP distribution:

$$\begin{pmatrix} f \\ f_* \end{pmatrix} \sim N\left(\begin{pmatrix} \mu \\ \mu_* \end{pmatrix}, \begin{pmatrix} K & K_* \\ K_*^T & K_{**} \end{pmatrix} \right) \tag{6}$$

where $K = \kappa(X, X)$, $K_* = \kappa(X, X_*)$ and $K_{**} = \kappa(X_*, X_*)$. Using standard rules for conditioning Gaussians, the posterior has the following form:

$$\begin{aligned} p(f_*|X_*, X, f) &\sim N(f_*|\mu_*, \Sigma_*) & (7)\\ \mu_* &= \mu(X_*) + K_*^T K^{-1}(f - \mu(X)) & (8)\\ \Sigma_* &= K_{**} - K_*^T K^{-1} K_* & (9) \end{aligned}$$

Figure 1 shows three functions drawn at random from a GP prior (left) and GP posterior (right) after observing five data points in the case of noise-free observations. The shaded area corresponds to two times the standard deviation around the mean (95% confidence region). We

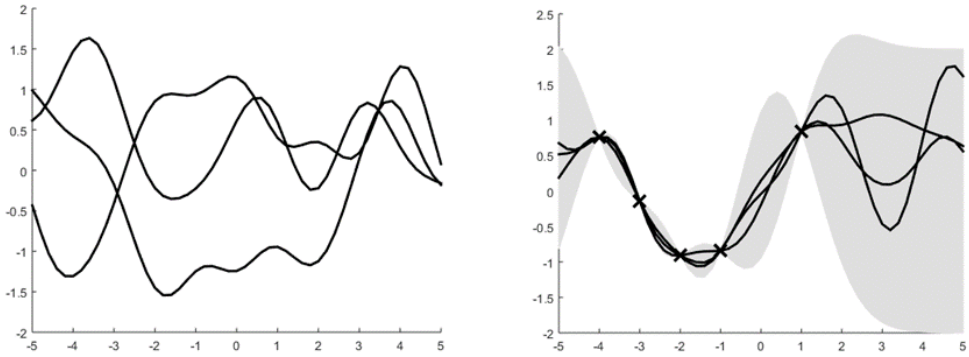

Figure 1: Gaussian process prior (left) and posterior (right).

can see that the model perfectly interpolates the training data and that the predictive uncertainty increases as we move further away from the observed data.

Since our algorithm is defined in terms of inner products in the input space, it can be lifted into feature space by replacing the inner products with $k(x, x')$, this is often referred to as the *kernel trick*. The kernel measures similarity between objects and it doesn't require preprocessing them into feature vector format. For a example, a common kernel function is a *radial basis function*:

$$k(x, x') = \exp\left(- \frac{||x - x'||^2}{2\sigma^2} \right) \tag{10}$$

In the case of a Gaussian kernel, the feature map lives in an infinite dimensional space. In this case, it is clearly infeasible to explicitly represent the feature vectors. Another commonly used kernel in Gaussian process regression is the *matern kernel*:

$$\kappa(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu} r}{l} \right)^{\nu} K_{\nu} \left(\frac{\sqrt{2\nu} r}{l} \right) \tag{11}$$

where $r = ||x - x'||$, $\nu > 0$, $l > 0$, and K_{ν} is a modified Bessel function. As $\nu \to \infty$, this approaches the squared exponential kernel, if $\nu = \frac{1}{2}$, the kernel simplifies to $\kappa(r) = \exp(-r/l)$.

Assuming the observations are noisy, $y = f(x) + \epsilon$, where $\epsilon \sim N(0, \sigma_y^2)$, GP is not required to interpolate the data but rather it must be close to the observed data. The covariance of the observed noisy responses is $\text{cov}(y|X) = K + \sigma_y^2 I = K_y$, where we assumed that the noise terms were independently added to each observation. Hence, the posterior predictive density is:

$$\begin{aligned} p(f_*|X_*, X, y) &\sim N(f_*|\mu_*, \Sigma_*) && (12) \\ \mu_* &= K_*^T K_y^{-1} y && (13) \\ \Sigma_* &= K_{**} - K_*^T K_y^{-1} K_* && (14) \end{aligned}$$

In multiple dimensions, we can write down the *square exponential kernel* as follows:

$$\kappa_y(x_p, x_q) = \sigma_f^2 \exp\big(- \frac{1}{2}(x_p - x_q)^T M (x_p - x_q) \big) + \sigma_y^2 \delta_{pq} \tag{15}$$

where $M = l^{-2} I$, l is the horizontal scale over which the function changes, σ_f^2 controls the vertical scale and σ_y^2 is the observation noise variance.

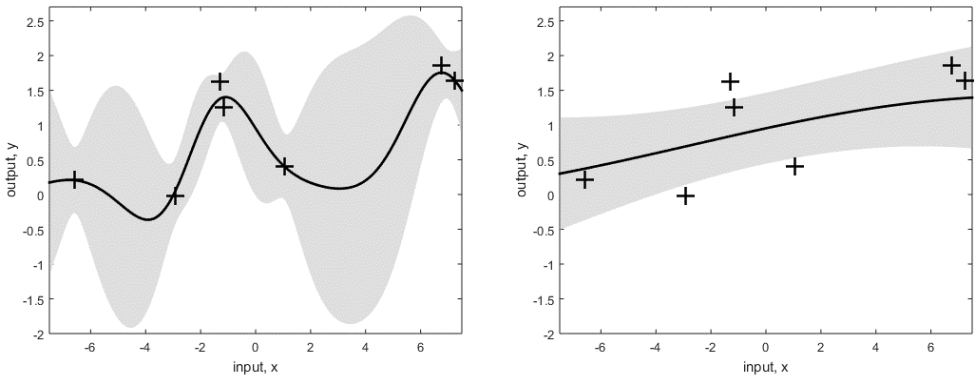

Figure 2: The effect of varying GP hyperparameters

Figure 2 shows the importance of choosing kernel hyperparameters and their impact on GP regression. We fix $\sigma_f^2 = 1$ and plot GP regression after 7 noisy observations. On the left, the length scale and noise variance were set to $(l, \sigma_y^2) = (1, 0.2)$. We can see that the mean function appears wiggly and has a small bias due to observation noise. On the right, the kernel parameters were set to $(l, \sigma_y^2) = (10, 0.8)$. The curve is much smoother due to large l and also shows higher observation noise.

We can use MCMC to learn GP hyperparameters by placing a prior on scale l, function variance σ_f^2 and noise variance σ_y^2. We choose uninformative priors for GP hyper-parameters:

$$l \sim \text{Unif}(0, 10) \tag{16}$$

$$\sigma_f^2 \sim \exp\{\text{Unif}(-10, 5)\} \tag{17}$$

$$\sigma_y^2 \sim \exp\{\text{Unif}(-10, 5)\} \tag{18}$$

Using the No-U-Turn Sampler (NUTS) we obtain the following trace plots shown in Figure 3. We can see that posterior mode is close to ground truth hyper-parameters shown by solid red lines.

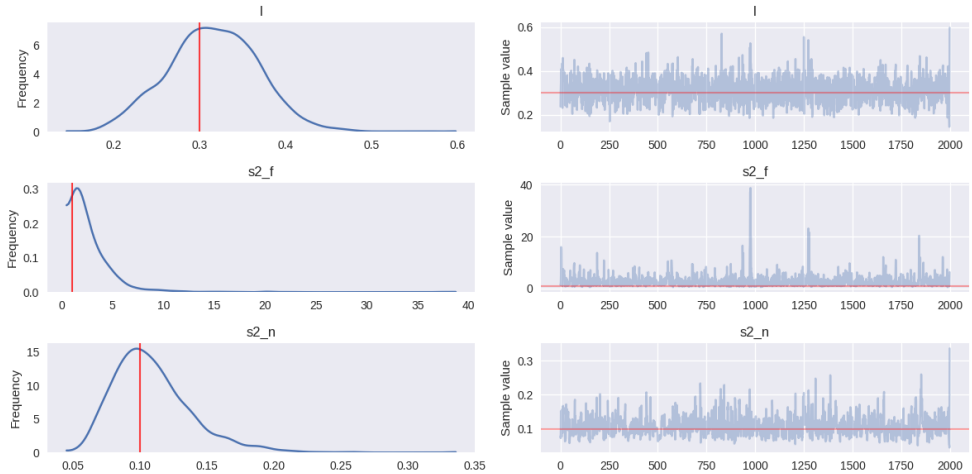

Figure 3: MCMC trace plots for GP hyperparameters.

Gaussian Processes (GPs) can be used for classification if the output of a GP is mapped

to the range $[0, 1]$. In the binary case, we define the model as $p(y_i|x_i) = \sigma(f(x_i))$, where we let $\sigma(z) = (1 + \exp(-z))^{-1}$. As with Bayesian logistic regression, the main difficulty in fitting GP classifier is that the Gaussian prior is not conjugate to the multinoulli likelihood. As a result several approximation algorithms can be used: Gaussian approximation, Expectation Propagation, variational and MCMC.

1.2 Dirichlet Process

The Dirichlet process is a stochastic process used in Bayesian non-parametric models. Each draw from a Dirichlet process is a discrete distribution. For a random distribution G to be distributed according to a DP, its finite dimensional marginal distributions have to be Dirichlet distributed. Let H be a distribution over Θ and α be a positive real number. We say that G is a Dirichlet process distributed with *base distribution* H and *concentration parameter* α if:

$$(G(A_1), ..., G(A_r)) \sim \text{Dir}(\alpha H(A_1), ..., \alpha H(A_r)) \tag{19}$$

for every finite measurable partition A_1,...,A_r of Θ, where Dirichlet distribution is defined as:

$$p(x_1, ..., x_K) = \frac{\Gamma(\sum_k \alpha_k)}{\prod_k \Gamma(\alpha_k)} \prod_{k=1}^{K} x_k^{\alpha_k - 1} \tag{20}$$

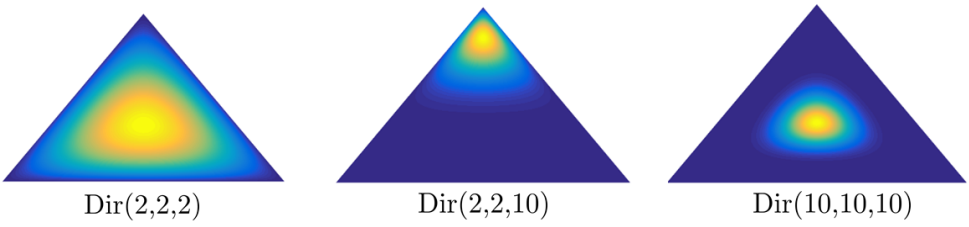

Figure 4: Dirichlet distribution: $\text{Dir}(\alpha_1, \alpha_2, \alpha_3)$ over the simplex.

The base distribution is the mean of the DP: $E[G(A)] = H(A)$, whereas the concentration parameter is the inverse variance: $V[G(A)] = H(A)(1 - H(A))/(\alpha + 1)$ for any measureable set $A \subset \Theta$. The larger the α, the smaller the variance, and the DP will concentrate more of its mass around the mean as shown in Figure 5.

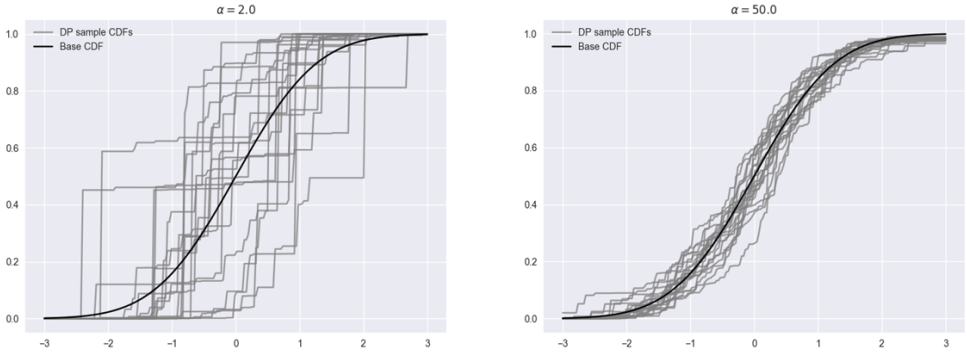

Figure 5: Base distribution CDF for $\alpha = 2$ and $\alpha = 50$.

Let $\theta_1, ..., \theta_n$ be a sequence of independent draws from $G \sim DP(\alpha, H)$. We are interested in the posterior distribution of G given observed values of $\theta_1, ..., \theta_n$. Let n_k be the number of

observed values in A_k, the by conjugacy the Dirichlet and multinomial distributions we have:

$$(G(A_1), ..., G(A_r))|\theta_1, ..., \theta_n \sim \text{Dir}(\alpha H(A_1) + n_1, ..., \alpha H(A_r) + n_r) \tag{21}$$

The posterior is also a DP with an updated concentration parameter and base distribution [48]:

$$G|\theta_1, ..., \theta_n \sim DP(\alpha + n, \frac{\alpha}{\alpha + n} H + \frac{n}{\alpha + n} \frac{\sum_{i=1}^{n} \delta_{\theta_i}}{n}) \tag{22}$$

Note that the posterior base distribution is a weighted average between the prior base distribution H and the empirical distribution $\frac{1}{n}\sum_{i=1}^{N} \delta_{\theta_i}$. Therefore, α can be interpreted as the strength of the prior.

1.3 Construction of the DP

The Dirichlet Process can be represented in different ways and the representation influences the inference procedure.

1.3.1 Blackwell-MacQueen Urn Scheme

The Blackwell-MacQueen Urn Scheme, also commonly known as the Polya urn scheme, provides a way of constructing a sequence of draws $\theta_1, \theta_2, ..., \theta_n \sim G$ where $G \sim DP(\alpha, H)$ without explicitly having to represent G. The posterior distribution with G marginalized out and $\theta_1 \sim H$ can be written as:

$$\theta_{n+1}|\theta_1, ..., \theta_n \sim \frac{1}{\alpha + n}\left(\alpha H + \sum_{i=1}^{n} \delta_{\theta_i}\right) \tag{23}$$

The above equation describes a sequential process for drawing θ_i from G, which can be described by the following urn metaphor. First draw $\theta_1 \sim H$, paint a ball with that color and put it in the urn. Then, at each subsequent step n, either draw $\theta_n \sim H$ with probability $\alpha/(\alpha + n)$ and put a ball of that color in the urn, or with probability $n/(\alpha + n)$, randomly draw a ball from the urn, set θ_n to its color, then paint a new ball in that color and return both balls to the urn.

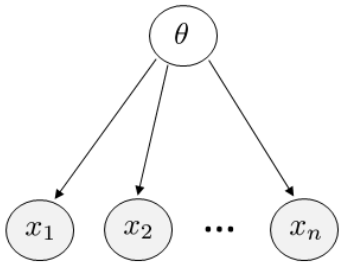

Figure 6: Naive Bayes graphical model

The Blackwell-MacQueen urn model has been used to show the existence of the DP. Starting from (23) we can construct a distribution over draws as follows:

$$P(\theta_1, ..., \theta_n) = \prod_{i=1}^{n} P(\theta_i|\theta_1, ..., \theta_{i-1}) \tag{24}$$

This random sequence is infinitely exchangeable: the probability of generating $\theta_1, ...\theta_n$ in that order is equal to the probability of generating them in any alternative order. Thus, for a given permutation σ, we have

$$P(\theta_1, ..., \theta_n) = P(\theta_{\sigma(1)}, ..., \theta_{\sigma(2)}) \tag{25}$$

The strength of infinite exchangeability lies in the following theorem:

Theorem 1.1 *(De Finetti, 1930s) A sequence of random variables $x_1, x_2, .., x_n$ is infinitely exchangeable iff for all n:*

$$p(x_1, x_2, ..., x_n) = \int p(\theta) \prod_{i=1}^{n} p(x_i|\theta) d\theta \tag{26}$$

Therefore, if we have exchangeable data, there must exist a parameter θ, a likelihood $p(x|\theta)$ and a distribution P on θ such that conditioned on θ the observations are independent as shown in Figure 6. In our setting, the prior over the random distribution $p(\theta)$ is the Dirichlet Process $DP(\alpha, H)$, thus establishing existence.

1.3.2 Chinese Restaurant Process

Chinese Restaurant Process (CRP) is a discrete-time stochastic process that is based on the clustering property of a DP. Let $\theta_1^*, ..., \theta_m^*$ be unique values of draws $\theta_1, ..., \theta_n$ and n_k be the number of repetitions of θ_k^*. Then, the predictive distribution can be written as:

$$\theta_{n+1}|\theta_1, ..., \theta_n \sim \frac{1}{\alpha + n} \left(\alpha H + \sum_{k=1}^{m} n_k \delta_{\theta_k^*} \right) \tag{27}$$

Notice that θ_k^* will be repeated in proportion to n_k, the number of times it has already been observed. This is a *rich-gets-richer* phenomenon: the larger the cluster, the faster it will grow. The Chinese Restaurant Process takes its name from a metaphor describing its construction: imagine a Chinese restaurant which has an infinite number of tables, each of which can accomodate an infinite number of clusters. The first customer enters the restaurant and sits at the first table. The $n+1$st customer either joins an already occupied table k with probability proportional to n_k of customers already there or sits at a new table with probability proportional to α. Representing customers with integers and tables with clusters, CRP defines a partition on $[n]$. The number of occupied tables K approaches $\alpha \log(n)$ as n approaches infinity. And therefore, DP is a non-parametric prior that favors models whose complexity grows with the amount of data.

1.3.3 Stick-Breaking Construction

The stick breaking construction [41] represents draws $G \sim DP(\alpha, H)$ as a weighted sum of atoms (point masses). It is given as follows:

$$\beta_k \sim \text{Beta}(1, \alpha) \qquad \theta_k^* \sim H \tag{28}$$

$$\pi_k = \beta_k \prod_{l=1}^{k-1} (1 - \beta_l) \qquad G = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k^*} \tag{29}$$

This construction guarantees that $G \sim DP(\alpha, H)$. The name stick-breaking comes from the fact that one can interpret the π_k as the lengths broken off a unit length stick as shown in Figure 7 for $\alpha = 5$ and $\alpha = 10$.

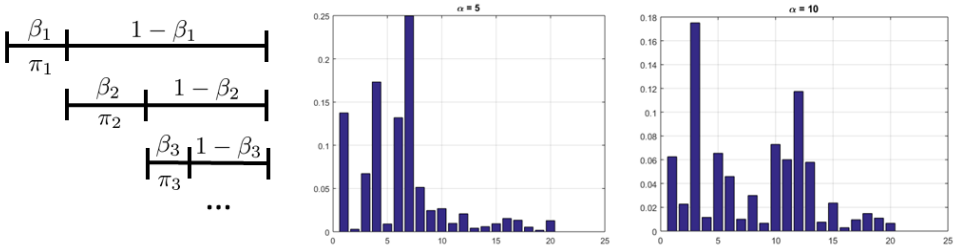

Figure 7: Stick-Breaking Weights

1.4 Hierarchical Dirichlet Process (HDP)

Hierarchical Dirichlet Processes model problems involving groups of data, where each observation within a group is a draw from a mixture model and mixture components are shared between groups. In each group, the number of components is learned from data using a Dirichlet Process prior. In addition, the base measure for the child Dirichlet processes is itself distributed according to a Dirichlet process. Since a draw from the global DP is a discrete distribution, the group-level DPs share mixture components. The HDP model can be summarized as follows and is shown in Figure 8:

$$G_0|\gamma, H \sim DP(\gamma, H) \tag{30}$$

$$G_j|\alpha, G_0 \sim DP(\alpha, G_0) \tag{31}$$

The distribution G_0 varies around the prior H, with the amount of variability given by γ. The

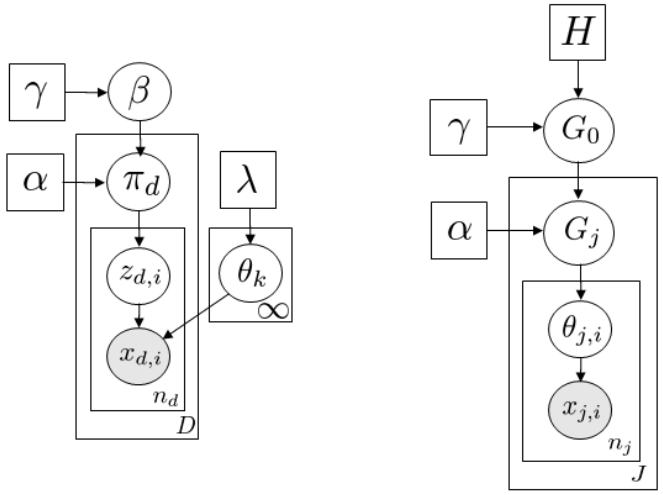

Figure 8: HDP Graphical Model

actual distribution G_j over $\theta_{j,i}$ in j^{th} group deviates from G_0 with the amount of variability given by α. If we expect the amount of variability between groups to be different, we can use a separate concentration parameter α_j for each group j.

Given the global DP prior G_0, we can express it using a stick-breaking representation: $G_0 = \sum_{k=1}^{\infty} \beta_k \delta_{\phi_k}$, where $\phi_k \sim H$ and $\beta_k \sim \text{GEM}(\gamma)$. Since G_0 has support at the points ϕ_k, each G_j has support at these points as well and can be written as: $G_j = \sum_{k=1}^{\infty} \pi_{jk} \delta_{\phi_k}$. The group weights π_j are related to global weights β as follows [47]:

$$\beta'_k \sim \text{Beta}(1, \gamma) \qquad \beta_k = \beta'_k \prod_{l=1}^{k-1}(1 - \beta'_l) \tag{32}$$

$$\pi'_{jk} \sim \text{Beta}(\alpha\beta_k, \alpha(1 - \sum_{l=1}^{k}\beta_l)) \qquad \pi_{jk} = \pi'_{jk} \prod_{l=1}^{k-1}(1 - \pi'_{jl}) \tag{33}$$

An analog of the Chinese restaurant process for HDPs is the Chinese restaurant franchise (CRF): the metaphor is extended to allow multiple restaurants which share a set of dishes. In this interpretation, each group defines a separate restaurant in which customers (observations) x_{ji} sit at tables (clusters) t_{ji}. Each table shares a single dish (parameter) θ_{ji}, which is ordered from a menu G_0 shared among restaurants (groups) as shown in Figure 9

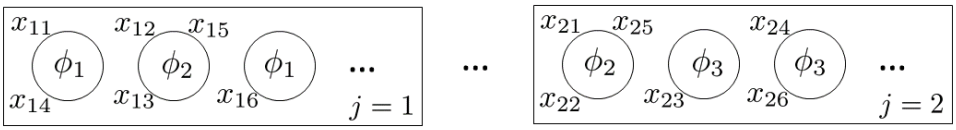

Figure 9: Chinese Restaurant Franchise (CRF). Each restaurant is represented by a rectangle. Customers x_{ji} are seated at tables. At each table a dish ϕ_k is served from a global menu.

Let $\phi_1, ..., \phi_K$ denote the K atoms distributed according to H: this is the global menu of dishes. Let ψ_{jt} represent table-specific choice of dishes; in particular ψ_{jt} is the dish served at table t in restaurant j. Note that each θ_{ji} is associated with one ψ_{jt} and each ψ_{jt} is associated with one ϕ_k. In the CRF metaphor, customer i in restaurant j sat at table t_{ji}, while table t in restaurant j served dish k_{jt}.

The number of tables in restaurant j serving dish k is denoted m_{jk}, and the number of customers in restaurant j at table t eating dish k is n_{jtk}. Marginal counts are represented with dots. For example, $n_{j..}$ and $m_{j.}$ represent the number of customers and tables, respectively, in restaurant j.

We can compute the marginals under HDP when G_0 and G_j are integrated out using (22):

$$\theta_{ji}|\theta_{j1}, ..., \theta_{jn}, \alpha, G_0 \sim \sum_{t=1}^{m_{j.}} \frac{n_{jt.}}{n + \alpha}\delta_{\psi_{jt}} + \frac{\alpha}{n + \alpha}G_0 \tag{34}$$

If a term in the first summation is chosen, we set $\theta_{ji} = \psi_{jt}$. If the second term is chosen then we increment $m_{j.}$ by one and draw $\psi_{jm_{j.}} \sim G_0$. To integrate out G_0, we can use (22) again:

$$\psi_{jt}|\psi_{11}, \psi_{12}, ..., \psi_{21}, ..., \gamma, H \sim \sum_{k=1}^{K} \frac{m_{.k}}{m_{..} + \gamma}\delta_{\phi_k} + \frac{\gamma}{m_{..} + \gamma}H \tag{35}$$

To summarize, for each j and i, we first sample θ_{ji} using (34). If a new sample from G_0 is needed, we use (35) to obtain a new sample ψ_{jt}.

An on-line variational inference algorithm for Hierarchical Dirichlet Process [51] was used to fit a topic model on the 20newsgroups dataset. The dataset consists of $11,314$ documents and over $100K$ unique tokens. Standard text pre-processing was used including tokenization, stop-word removal and stemming. A compressed dictionary of $4K$ words was constructed by

filtering out tokens that appear in less than 5 documents and more than 50% of the corpus. The top-level truncation was set to $T = 20$ topics and the second level truncation was set to $K = 8$ topics. The concentration parameters were chosen as $\gamma = 1.0$ at the top-level and $\alpha = 0.1$ at the group level to yield a broad range of shared topics that are concentrated at the group level. Figure 10 shows a sample of the global level topics inferred by online variational HDP algorithm. We can find topics about autos, politics and for sale items that correspond to the target labels of the 20newsgroups dataset.

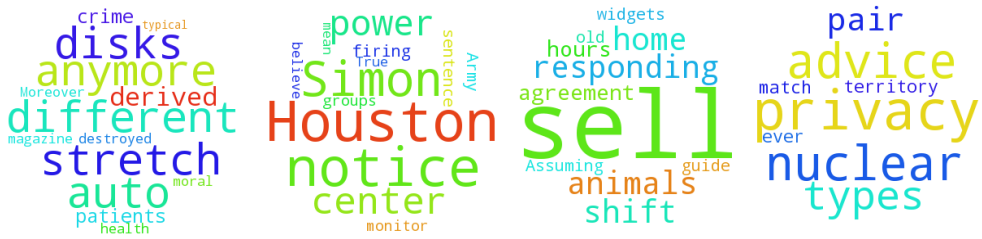

Figure 10: Sample of HDP topics inferred using online variational bayes algorithm on 20newsgroups dataset.

1.4.1 HDP Hidden Markov Models

The Hierarchical Dirichlet Process (HDP) can be used to define a prior distribution on transition matrices over countably infinite state spaces. The HDP-HMM is known is an infinite Hidden Markov Model where the number of states is inferred automatically. To consider a non-parametric variant of the HMM, we must consider a set of DPs, one for each value of the current state. In addition, the DPs must be linked because we want the same set of next states to be reachable from each of the current states. The relates directly to HDP, where the atoms associated with state-conditional DPs are shared.

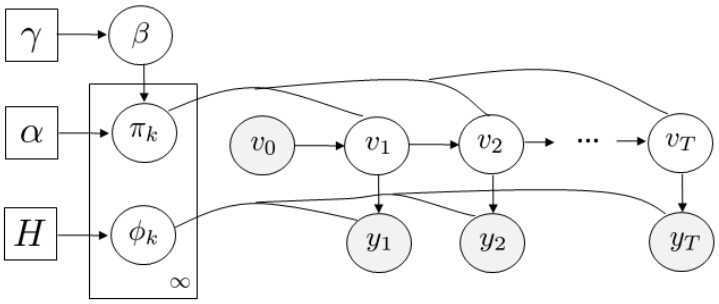

Figure 11: HDP-HMM graphical model

We can describe the HDP-HMM model in Figure 11 using the stick-breaking construction. The parameters have the following distribution:

$$\beta|\gamma \sim \text{GEM}(\gamma) \quad \pi_k|\alpha, \beta \sim DP(\alpha, \beta) \quad \phi_k|H \sim H \tag{36}$$

for each $k = 1, 2, ...$ while for time steps $t = 1, ..., T$ the state and observation distributions are:

$$v_t|v_{t-1}, \pi_k \sim \pi_{v_{t-1}} \qquad y_t|v_t, \phi_k \sim F(\phi_{v_t}) \tag{37}$$

given a starting state v_0. Each π_j is a DP draw and is interpreted as the transition distribution from state j. The π_js are linked through DP draws parameterized by the same discrete measure β. Therefore, the states are shared between different DP draws.

Besides Gibbs sampling, one way to compute the posterior for infinite HMM is using a *beam sampling* algorithm [50]. Beam sampling combines two ideas: slice sampling and dynamic programming to sample whole state trajectories. Slice sampling is applied to limit the number of states to a finite number in each time step of the iHMM. The underlying idea is to limit the beam search to a small number of states so that a good trajectory can be found. Beam sampling is an MCMC method that guarantees convergence to the true posterior.

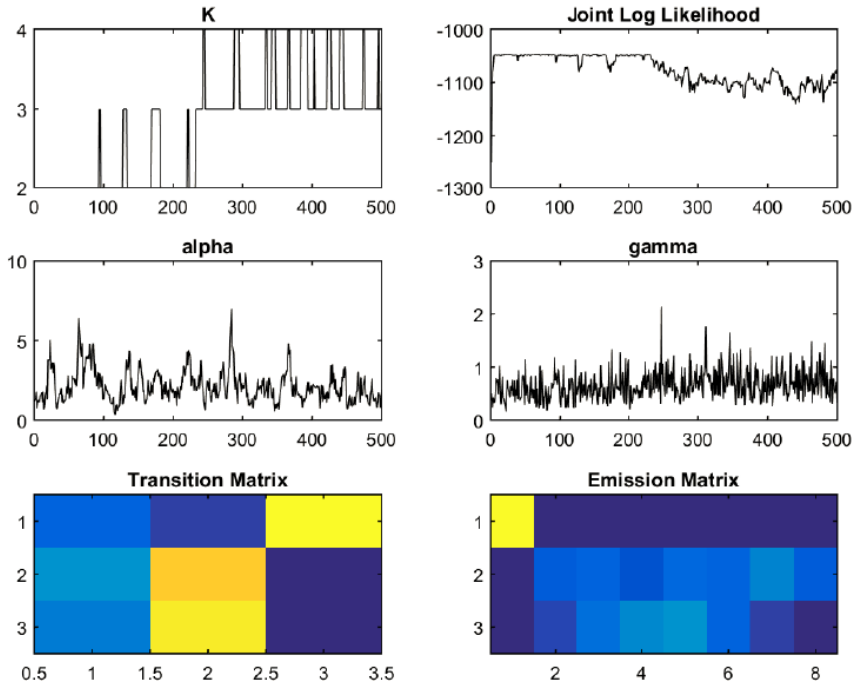

Figure 12: HDP-HMM inference results using the beam sampling algorithm [50].

Figure 12 shows posterior inference results for synthetically generated time-series data. The ground truth data was generated using 4 unique states for given transition and emition matrices. From the top-left plot in Figure 12, we can see that the beam sampler discovered three transition states at iteration 250. We can also see the samples of α and γ concentration parameters for the HDP-HMM as well as the transition and emission matrices.

1.5 Dependent Dirichlet Process (DDP)

The earlier part of Bayesian non-parametric literature focused on problems where a single distribution is assigned a non-parametric prior. However, in many applications, the objective is modelling a collection of distributions used in temporal and spatial processes. The Dirichlet process assumes that observations are exchangeable and therefore the data points have no inherent ordering that influences their labelling. This assumption is invalid for modelling termporal and spatial processes in which the order of data points plays a critical role in creating meaningful clusters.

The dependent Dirichlet process (DDP) originally formulated by MacEachern [29] provides a non-parametric prior over evolving mixture models. A construction of the DDP built on Poisson process [28] led to the development of the DDP mixture model (DDPMM) which generalizes DPMM by including birth, death and transition processes for the clusters in the model. In

addition, a low-variance approximations to DDPMM have been derived leading to a dynamic clustering algorithm [6].

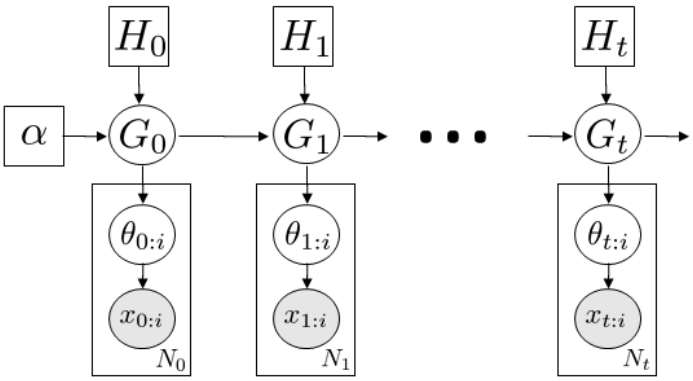

Figure 13: Dependent Dirichlet Process (DDP) graphical model.

Under time-varying setting, it is natural to introduce different DP priors for different time steps as shown in Figure 13. The generative model can be written as:

$$D_t \sim DP(\alpha, H_t) \tag{38}$$

$$\theta_{t:i}|D_t \sim D_t \quad for i = 1, ..., n_t, \ t = 0, ..., T \tag{39}$$

$$x_{t:i}|\theta_{t:i} \sim F(\theta_{t:i}) \quad for i = 1, ..., n_t, \ t = 0, ..., T \tag{40}$$

A Poisson-based construction of DDP [28] exploits the connection between Poisson and Dirichlet processes. In particular, by applying operations that preserve *complete randomness* to the underlying Poisson processes: superposition, subsampling and point transition, a new Poisson and therefore a new Dirichlet process is produced. Therefore, a Markov chain of Dirichlet processes can be written as:

$$D_t = T(S_q(D_{t-1})) \bigoplus H_t, \quad where H_t \sim DP(\alpha, H) \tag{41}$$

where S_q is an acceptance function, T is a probabilistic transition, combined with new terms from innovation process H_t to form D_t.

1.6 Indian Buffet Process

The Indian Buffet Process(IBP) is a stochastic process defining a probability distribution over sparse binary matrices with a finite number of rows and an infinite number of columns. This distribution is suitable to use as a prior for models with potentially infinite number of features. The form of the prior ensures that only a finite number of features will be present in any finite set of observations but more features may appear as more data points are observed.

Let Z be a $N \times K$ binary matrix indicating the presence or absence of a latent feature. The IBP places the following prior on Z [18]:

$$p(Z) = \frac{\alpha^K}{\prod_{i=1}^{N} K_1^{(i)}!} \exp\{-\alpha H_N\} \prod_{k=1}^{K} \frac{(N - m_k)!(m_k - 1)!}{N!} \tag{42}$$

where K is the number of nonzero columns in Z, m_k is the number of ones in column k of Z, H_N is the N^{th} harmonic number, and K_h is the number of occurences of the non-zero binary vector h among the columns in Z. The parameter α controls the expected number of features present in each observation.

In the Indian Buffet Process (IBP), the rows of Z correspond to customers and the columns correspond to dishes in an infinitely long buffet. The first customer takes the first Poisson(α) dishes. The ith customer than takes dishes that have been previously sampled with probability m_k/i, where m_k is the number of people who have already sampled dish k. He also takes Poisson(α/i) new dishes. Then, z_{nk} is one if customer n tried kth dish and zero otherwise.

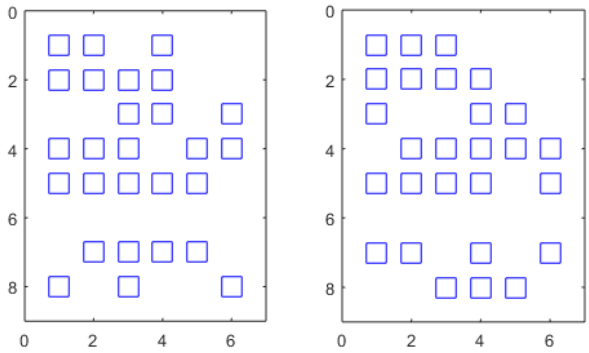

Figure 14: Original binary feature matrix (left) and its left-ordered form (right)

This process is infinitely exchangeable for an equivalence class of binary matrices defined by a left-ordered many-to-one function. $lof(Z)$ is obtained by ordering the columns of the binary matrix Z from left to right by the magnitude of the binary number expressed by that column, taking the first row as the most significant bit. The left ordering of a binary matrix is shown in Figure 14

In this section, we focus on variational inference procedures for the linear-Gaussian likelihood model [11]. Let X be a $N \times D$ matrix where each of the N rows contains a D-dimensional observation. We focus on a model where X can be approximated as:

$$X_{N\times D} = Z_{N\times K} \times A_{K\times D} + \epsilon \tag{43}$$

where Z is a binary feature matrix, the values for feature k are stored in row k of A, and ϵ is measurement noise. Figure 15 shows the predicted binary feature matrix Z and lower bound on marginal likelihood using variational inference algorithm for IBP [11]. The concentration parameter was set to $\alpha = 1$ and the feature truncation level was set to $K = 6$.

1.7 Infinite Mixture of Experts

Just as Dirichlet process mixtures can be thought of as infinite mixture models that select the number of active components as part of inference, infinite mixture of experts can be thought of selecting active components from an infinite number of regressions. This flexibility makes it a powerful tool for nonparametric Bayesian data analysis.

Infinite mixture of experts generalizes the Dirichlet Process Mixture Model by allowing the

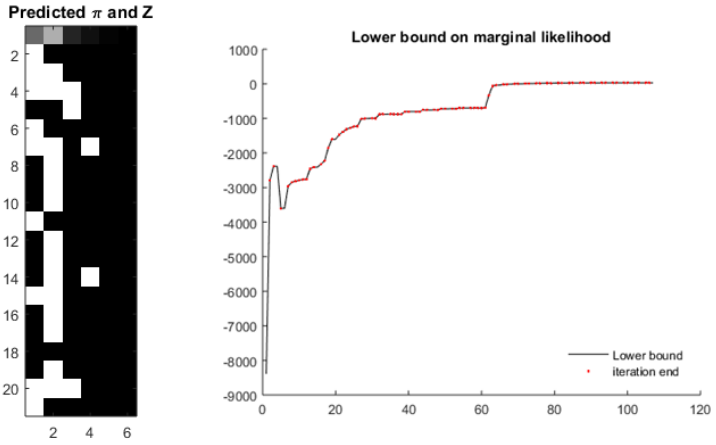

Figure 15: Predicted IBP binary feature matrix Z using variational inference.

mixture weights and component means to vary according to the input predictor x:

$$y|x \sim \sum_{i=1}^{\infty} w_i|x \times N(\mu_i|x, \tau_i^{-1}) \tag{44}$$

To determine the conditional mixture weights, we can use probit stick-breaking process:

$$v_i|x = \Phi(\alpha_i + \beta_i x) \tag{45}$$

$$w_i|x = v_i|x \times \prod_{j=1}^{i-1}(1 - v_j|x) \tag{46}$$

where Φ is the cumulative distribution function of the standard normal. To model multiple regressions, we define the mean of the mixture of experts as:

$$\mu_i|x \sim \gamma_i + \delta_i x \tag{47}$$

Figure 16 shows an infinite mixture of experts applied to LIDAR dataset that shows locally linear relationship between range and log ratio as well as heteroskedastic observation noise. The infinite mixture was truncated at $K = 20$ components which is reasonable considering that there are only three linear regressions having non-zero posterior expected weight.

Normal priors were set on regression coefficients for mixture mean and mixture weights, in addition $\tau_i \sim \text{Gamma}(1, 1)$ prior was set on the precision of the Gaussian mixture. The figure also shows a 95% credible interval that captures well the time-varying nature of observation noise. The flexibility of infinite mixture of experts and the ability to modularly specify the conditional mixture weights and component densities makes it a very useful nonparametric Bayesian model.

1.8 DPMM

The Dirichlet Process Mixture Models (DPMM) belong to a class of *infinite mixture models*, in which we do not impose any prior knowledge on the number of clusters K. DPMM models learn the number of clusters from the data using a non-parametric prior based on the Dirichlet Process (DP). Automatic model selection leads to computational savings of cross validating the model for multiple values of K.

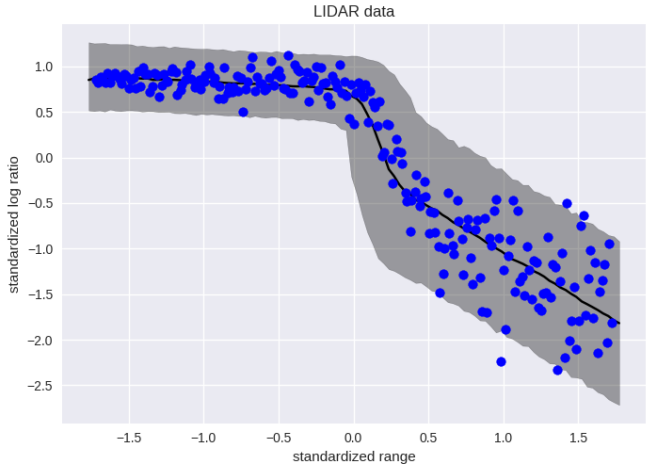

Figure 16: Infinite Mixture of Experts Regression.

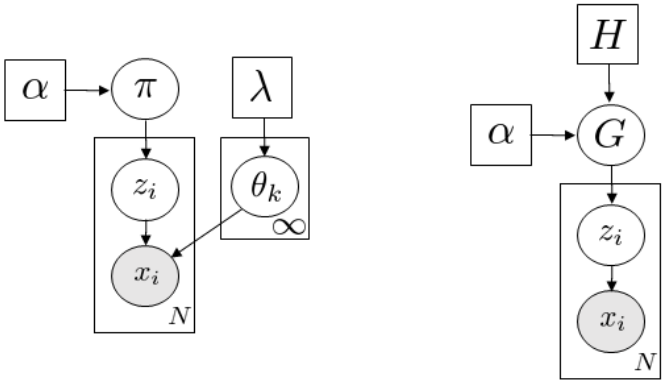

Figure 17: DPMM Graphical Model

Consider the graphical model of the DPMM in Figure 17. For each data point x_i, there's a corresponding label z_i that assigns the point to one of the clusters with mixture weight π_k and parameters $\theta_k = \{\mu_k, \Sigma_k\}$. The generative model can be written as follows:

$$\begin{aligned} p(x_i|z_i = k, \theta) &= p(x_i|\theta_k) && (48)\\ p(z_i = k|\pi) &= \pi_k && (49)\\ p(\pi|\alpha) &= \text{Dir}(\pi|\alpha) && (50)\\ p(\theta_k|\beta) &= \text{NIW}(\mu_k, \Sigma_k|m_0, \kappa_0, \nu_0, S_0) && (51) \end{aligned}$$

An equivalent representation of this model can be written as:

$$G(\theta) = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k} \tag{52}$$

where $\pi \sim \text{Dir}(\alpha_1, ..., \alpha_K)$ and $\theta_k \sim H$. Therefore, G is an infinite mixture of cluster functions of delta functions. In practice, we can construct G using a *stick-breaking construction*:

$$\beta_k \sim \text{Beta}(1, \alpha) \tag{53}$$

$$\pi_k = \beta_k \prod_{l=1}^{k-1}(1 - \beta_l) = \beta_k(1 - \sum_{l=1}^{k-1} \pi_l) \tag{54}$$

This is commonly denoted as $\pi \sim \text{GEM}(\alpha)$. The number of generated mixture components increases with α. The hyper-parameter α controls the expected number of clusters:

$$E[K] = \alpha \times \log(1 + n/\alpha) \tag{55}$$

$$VAR[K] = \alpha \times \log(1 + n/\alpha) \tag{56}$$

Thus, the number of clusters grows logarithmically with the number of data points and in direct proportion to α as shown in Figure 18

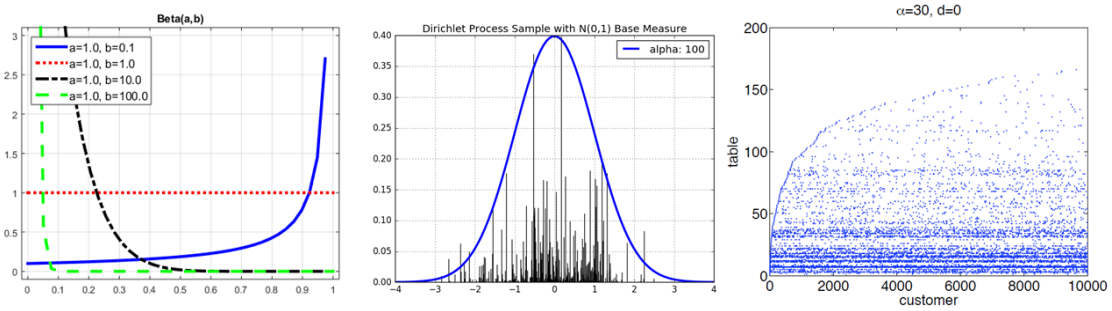

Figure 18: a) Beta$(1, \alpha)$ b)DP samples c) DP cluster growth

1.9 Fitting a DPMM model

One way to fit a DPMM is to modify the collapsed Gibbs sampler for a finite mixture model [31]:

$$p(z_i = k|z_{-i}, x, \alpha, \beta) \propto p(z_i = k|z_{-i}, \alpha)p(x_i|x_{-i}, z_i = k, z_{-i}, \beta) \tag{57}$$

The first term is given by the Chinese Restaurant Process (CRP):

$$p(z_i = k|z_{-i}, \alpha) = \begin{cases} \frac{N_k}{N+\alpha-1} & \text{if } k \text{ is occupied} \\ \frac{\alpha}{N+\alpha-1} & \text{if } k \text{ is a new cluster} \end{cases} \tag{58}$$

The second term is the posterior predictive and can be computed as follows:

$$p(x_i|x_{-i}, z_i = k, z_{-i}, \beta) = p(x_i|x_{k\setminus i}, \beta) = \frac{p(x_k|\beta)}{p(x_{k\setminus i}|\beta)} \tag{59}$$

We can compute both the numerator and the denominator above if we can find an expression for the marginal $p(x)$:

$$p(x) = \int_\mu \int_\Sigma p(x, \mu, \Sigma) d\mu d\Sigma = \int_\mu \int_\Sigma p(x|\mu, \Sigma) p(\mu, \Sigma|\beta) d\mu d\Sigma \tag{60}$$

$$= (2\pi)^{-ND/2} \frac{Z_{NIW}(D, \kappa_N, \nu_N, S_N)}{Z_{NIW}(D, \kappa_0, \nu_0, S_0)} = \pi^{-ND/2} \frac{\kappa_0^{D/2} |S_0|^{\nu_0/2}}{\kappa_N^{D/2} |S_N|^{\nu_N/2}} \prod_{i=1}^{D} \frac{\Gamma(\frac{\nu_N+1-i}{2})}{\Gamma(\frac{\nu_0+1-i}{2})} \tag{61}$$

Therefore, we can write the second term as follows:

$$\log p(x_i|x_{k\setminus i}) = z(D, N+1, \kappa_{N+1}, \nu_{N+1}, S_{Ni}) - z(D, N, \kappa_N, \nu_N, S_N) \tag{62}$$

$$z(D, N, \kappa, \nu, S) = -\frac{ND}{2} \log \pi - \frac{D}{2} \log \kappa - \frac{\nu}{2} \log |S| + \sum_{i=1}^{D} \log \Gamma(\frac{\nu + i - 1}{2}) \tag{63}$$

We can summarize, the collapsed Gibbs sampler for an infinite Gaussian mixture model in Algorithm 1.

Algorithm 1 Collapsed Gibbs for DPMM

```
Initialize labels z
for t = 1,2,...,T do
    for i = 1,2,...,N do
        remove x_i's statistics from component z_i
        delete empty component
        for k = 1,2,...,K+1 do
            compute log p(z_i = k|z_{-i}, x, α, β) ∼
            log p(z_i = k|z_{-i}, α) + log p(x_i|x_{k\i}, β)
        sample z_i = k_new from p(z_i = k|z_{-i}, x, α, β)
        create a new cluster if z_i = K + 1
        add x_i's statistics to component z_i = k_new
    end for
end for
```

Figure 19 shows the clustering results of DPMM collapsed gibbs sampler with Gaussian base measure and $\alpha = 1$ after 100 iterations on a synthetic dataset of $1K$ points in 2D. The sampler was initialized with $K_{init} = 2$ clusters and correctly identified all 5 clusters in the data.

Figure 20 shows the clustering results of DPMM on categorical data. Documents from a reduced NIPS dataset with 300 docs, 5K vocab and 130K words were grouped into 11 clusters. The top words for the first 4 clusters are displayed in Figure 20. The results indicate meaningful division of articles by topics about neural networks, gaussian mixtures, computer vision and reinforcement learning. The gibbs sampler was initialized with $K_{init} = 2$ and $\alpha = 1$ and coverged after 20 iterations through the corpus.

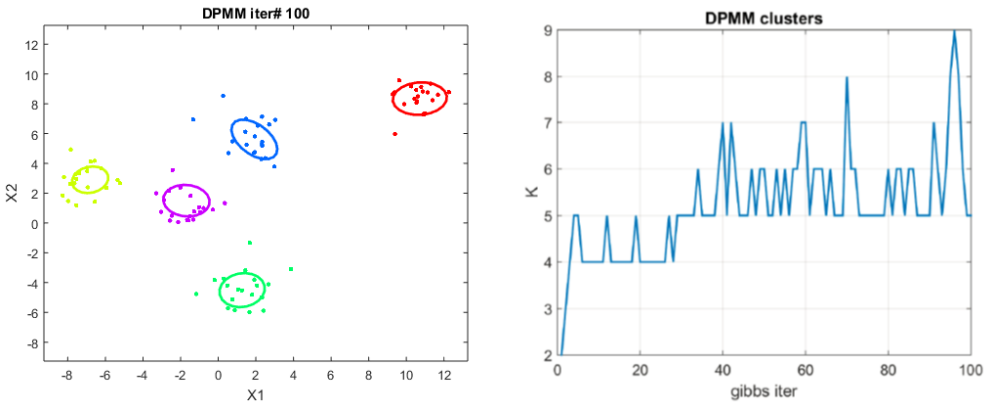

Figure 19: DPMM Clustering Results for Gaussian data

cluster 1:	cluster 2:	cluster 3:	cluster 4:
units	distribution	visual	state
hidden	gaussian	neurons	policy
weights	mixture	eye	action
state	probability	system	reinforcement
unit	likelihood	activity	actions
layer	hidden	cells	system
error	variables	position	sutton
weight	bayesian	units	states
recurrent	log	response	optimal
architecture	posterior	direction	step
system	em	cortex	control

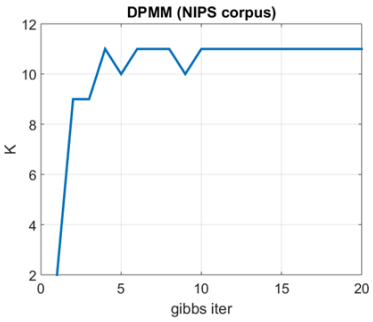

Figure 20: DPMM Clustering Results for Categorical data

2 Bayesian Inference

This section focuses on derivation of Bayesian inference algorithms for a variety of graphical models. The purpose of this section to practice derivation and implementation of Bayesian inference algorithms focusing on Markov-Chain Monte-Carlo (MCMC) sampling techniques from high dimensional posterior distributions.

2.1 Maximum Likelihood Estimation

The most common approach to parameter estimation is to pick the parameters that assign highest probability to the training data. This is known as Maximum Likelihood Estimation (MLE).

$$\begin{aligned}\hat{\theta}_{mle} &= \arg\max_{\theta} P(D|\theta) = \arg\max_{\theta} \prod_{n=1}^{N} p(y_n|x_n,\theta) = \arg\max_{\theta} \sum_{n=1}^{N} \log p(y_n|x_n,\theta) \\ &= \arg\min \mathrm{NLL}(\theta) = \arg\min_{\theta} -\sum_{n=1}^{N} \log p(y_n|x_n,\theta) \end{aligned} \tag{64}$$

where x_n and y_n are the input and output data, respectively. In the unsupervised case:

$$\hat{\theta}_{mle} = \arg\min_{\theta} -\sum_{n=1}^{N} \log p(y_n|\theta) \tag{65}$$

In *Empirical Risk Minimization* (ERM), we replace NLL with any loss and take expectation wrt empirical distribution. For example, a 0-1 loss or a surrogate loss often chosen to be the maximally tight convex upper bound which is then easy to minimize.

2.1.1 Naive Bayes

Consider discrete-valued features $x \in \{1, ..., K\}^D$ where D is the number of features and K is the number of distinct values they can take on. The Naive Bayes model assumes conditional independence of random variables x given the class labels y, i.e.

$$p(x|y = c, \theta) = \prod_{j=1}^{D} p(x_j|y = c, \theta_{jc}) \tag{66}$$

The class conditional density above can take several forms: (1) Gaussian $N(x_j|\mu_{jc}, \sigma_{jc}^2)$, (2) Bernoulli $\text{Ber}(x_j|\mu_j c)$, (3) Categorical $\text{Cat}(x_j|\mu_{jc})$.

Let's assume that we have a Bernoulli class conditional density, which means that $x_{i,j} \sim \text{Ber}(x_{i,j}|\theta_{jc})$ are binary indicators for presence ($x_{i,j} = 1$) of word j in document i or absence ($x_{i,j} = 0$) of the word, parameterized by θ_{jc}. Note that in the context of Naive Bayes classification of text, a data point $x_i = \{x_{i,1}, ..., x_{i,j}, ..., x_{i,D}\}$ is a document with $j = \{1, ..., D\}$ features (vocabulary words) and an associated document class label y_i.

We can write down the joint as follows:

$$p(x_i, y_i|\theta) = p(y_i|\pi) \prod_{j=1}^{D} p(x_{ij}|y_i = c, \theta_{jc}) = \prod_{c=1}^{C} \pi_c^{1[y_i=c]} \prod_{j=1}^{D} \prod_{c=1}^{C} p(x_{ij}|y_i = c, \theta_{jc})^{1[y_i=c]} \tag{67}$$

As a result, the log-likelihood for the entire corpus is:

$$\begin{aligned}
\log p(D|\theta) &= \log \prod_{i=1}^{n} p(x_i, y_i|\theta) = \sum_{i=1}^{n} \log p(x_i, y_i|\theta) \\
&= \sum_{i=1}^{N} \sum_{c=1}^{C} 1[y_i = c] \log \pi_c + \sum_{i=1}^{n} \sum_{j=1}^{D} \sum_{c=1}^{C} 1[y_i = c] \log p(x_{ij}|y_i = c, \theta_{jc}) \\
&= \sum_{c=1}^{C} N_c \log \pi_c + \sum_{j=1}^{D} \sum_{c=1}^{C} \sum_{i:y_i=c} \log \text{Ber}(x_{ij}|\theta_{jc})
\end{aligned} \tag{68}$$

There is a problem if some of the words have zero counts. To solve this problem, we can introduce a prior on our latent variables that acts as α-smoothing. For simplicity, we'll use a factored prior:

$$p(\theta) = p(\pi) \prod_{j=1}^{D} \prod_{c=1}^{C} p(\theta_{jc}) = \text{Dir}(\alpha_1, ..., \alpha_C) \prod_{j=1}^{D} \prod_{c=1}^{C} \text{Beta}(\beta_0, \beta_1) \tag{69}$$

Combining the factored prior with the factored likelihood, we get:

$$\begin{aligned}
p(\theta|D) &= p(\pi|D) \prod_{j=1}^{D} \prod_{c=1}^{C} p(\theta_{jc}|D) \\
&= \text{Dir}(\pi|\text{N}_1 + \alpha_1, ..., \text{N}_\text{C} + \alpha_\text{C}) \prod_{j=1}^{D} \prod_{c=1}^{C} \text{Beta}(\theta_{jc}|(N_c - N_{jc}) + \beta_0, N_{jc} + \beta_1)
\end{aligned} \tag{70}$$

We can use Calculus to find MAP estimates of model parameters. For π, we want to maximize the Lagrangian:

$$l(\pi, \lambda) = \log \prod_{c=1}^{C} \pi_c^{\alpha_k + N_k - 1} + \lambda\big(1 - \sum_c \pi_c\big) = \sum_{c=1}^{C} (\alpha_c + N_c - 1) \log \pi_c + \lambda\big(1 - \sum_c \pi_c\big) \tag{71}$$

Taking partial derivative with respect to π_c, we get:

$$\frac{\partial l(\pi, \lambda)}{\partial \pi_c} = \frac{\alpha_c + N_c - 1}{\pi_c} - \lambda = 0 \implies \lambda \pi_c = \alpha_c + N_c - 1 \tag{72}$$

Using the sum to 1 constraint:

$$\lambda \sum_{c=1}^{C} \pi_c = \sum_{c=1}^{C} \alpha_c + N_c - 1 \implies \lambda = \alpha_0 + N - C \tag{73}$$

Therefore, the MAP estimate is:

$$\pi_c = \frac{N_c + \alpha_c - 1}{N + \sum_c \alpha_c - C} \tag{74}$$

We can use a similar argument to derive the MAP estimate of θ_{jc}. Taking the log of the posterior and isolating the term of interest, we can find the mode of the Beta distribution using Calculus and as a result the MAP estiamte is:

$$\theta_{jc} = \frac{N_c - N_{jc} + \beta_0 - 1}{N_c - N_{jc} + \beta_0 + N_{jc} + \beta_1 - 2} = \frac{N_c - N_{jc} + \beta_0 - 1}{N_c + \beta_0 + \beta_1 - 2} \tag{75}$$

Note, if we set $\beta_0 = \beta_1 = 1$, we get the MLE estimate and if we set $\beta_0 = \beta_1 = 2$, we get add 1 Laplacian smoothing.

Learning the model parameters is the first step. We would like to next use the learned parameters to make a prediction about the class label of a test document. Using Bayes rule, we have:

$$p(y = c|x, D) \propto p(y = c|D) \prod_{j=1}^{D} p(x_j|y = c, D) \tag{76}$$

Substituting the distributions and taking the log, we get:

$$\log p(y = c|x, D) \propto \log \hat{\pi}_c + \sum_{j=1}^{D} \Big(1[x_{ij} = 1] \log \hat{\theta}_{jc} + 1[x_{ij} = 0] \log(1 - \hat{\theta}_{jc}) \Big) \tag{77}$$

Thus, we can plug-in the MAP estimates $\hat{\pi}_c$ and $\hat{\theta_{jc}}$ in the expression above to compute the log probability of class label for a new document.

2.2 Expectation Maximization (EM)

The EM algorithm provides a way of computing ML/MAP estimates when we have unobserved latent variables and/or missing data. EM exploits the fact that if the data were fully observed, then the ML/MAP estimates would be easy to compute. In particular, EM is an iterative algorithm which alternates between inferring the missing values given the parameters (E step) and

then optimizing the parameters given filled in data (M step).

In the EM algorithm, we define the complete data log-likelihood $l_c(\theta)$ where x_i are the observed random variables and z_i are unobserved. Since we don't know z_i we can't compute $p(x_i, z_i|\theta)$ but we can compute an expectation of $l_c(\theta)$ wrt to parameters $\theta^{(k-1)}$ from the previous iteration.

$$l_c(\theta) = \sum_{i=1}^{N} \log p(x_i, z_i|\theta) \tag{78}$$

The goal of the E-step is to compute $Q(\theta, \theta^{(k-1)})$ on which the ML/MAP estimates depend. The goal of the M-step is to re-compute θ by finding the ML/MAP estimates:

$$\text{E} - \text{step} \quad : \quad Q(\theta, \theta^{(k-1)}) = E_{\theta^{(k-1)}}\big[l_c(\theta)|D, \theta^{(k-1)}\big] \tag{79}$$

$$\text{M} - \text{step} \quad : \quad \theta^{(k)} = \arg\max_{\theta} Q(\theta, \theta^{(k-1)}) + \log p(\theta) \tag{80}$$

2.2.1 Gaussian Mixture Model (GMM)

To derive the EM algorithm for GMM, we first need to compute the expected complete data log-likelihood:

$$\begin{aligned}
Q(\theta, \theta^{(k-1)}) &= E\Big[\sum_i \log p(x_i, z_i|\theta)\Big] \\
&= \sum_i E\Big[\log\Big[\prod_{k=1}^{K} (\pi_k p(x_i|\theta_k))^{1[z_i=k]}\Big]\Big] && (81) \\
&= \sum_i \sum_k E\big[1[z_i = k]\big] \log\big[\pi_k p(x_i|\theta_k)\big] && (82) \\
&= \sum_i \sum_k p(z_i = k|x_i, \theta^{(k-1)}) \log\big[\pi_k p(x_i|\theta_k)\big] && (83) \\
&= \sum_i \sum_k r_{ik} \log \pi_k + \sum_i \sum_k r_{ik} \log p(x_i|\theta_k) && (84)
\end{aligned}$$

where $r_{ik} = p(z_i = k|x_i, \theta^{(k-1)})$ is the soft assignment of point x_i to cluster k.

E-step. Given $\theta^{(k-1)}$, we want to compute the soft assignments:

$$r_{ik} = p(z_i = k|x_i, \theta^{(k-1)}) = \frac{p(z_i = k, x_i|\theta^{(k-1)})}{\sum_{k=1}^{K} p(z_i = k, x_i|\theta^{(k-1)})} = \frac{p(x_i|z_i = k, \theta^{(k-1)})\pi_k}{\sum_{k=1}^{K} p(x_i|z_i = k, \theta^{(k-1)})\pi_k} \tag{85}$$

where $\pi_k = p(z_i = k)$ are the mixture proportions.

M-step. In the M step, we maximize Q with respect to model parameters π and θ_k. First, let's find π that maximizes the Lagrangian:

$$\frac{\partial Q}{\partial \pi_k} = \frac{\partial}{\partial \pi_k}\Big[\sum_i \sum_k r_{ik} \log \pi_k + \lambda(1 - \sum_k \pi_k)\Big] = \sum_i r_{ik} \frac{1}{\pi_k} - \lambda = 0 \tag{86}$$

Substituting the above expression into the constraint, we get

$$\sum_k \pi_k = 1 \implies \sum_k \frac{1}{\lambda} \sum_i r_{ik} = 1 \implies \lambda = \sum_i \sum_k r_{ik} = \sum_i 1 = N \tag{87}$$

Therefore, $\pi_k = \frac{1}{\lambda}\sum_i r_{ik} = \frac{1}{N}\sum_i r_{ik}$. To find the optimum parameters $\theta_k = \{\mu_k, \Sigma_k\}$, we want to optimize the terms of Q that depend on θ_k:

$$\begin{aligned} l(\mu_k, \Sigma_k) &= \sum_i r_{ik} \log p(x_i|\theta_k) \\ &\propto -\frac{1}{2}\sum_i r_{ik}\Big[\log|\Sigma_k| + (x_i-\mu_k)^T\Sigma_k^{-1}(x_i-\mu_k)\Big] \end{aligned} \tag{88}$$

To find the optimimum μ_k, we differentiate the above expression. First, focusing on the second term inside the sum, we can make a substitution $y_i = x_i - \mu_k$:

$$\frac{\partial}{\partial\mu_k}(x_i-\mu_k)^T\Sigma_k^{-1}(x_i-\mu_k) = \frac{\partial}{\partial y_i}y_i^T\Sigma^{-1}y_i\frac{\partial y_i}{\partial\mu_k} = -1\times(\Sigma^{-1}+\Sigma^{-T})y_i \tag{89}$$

Substituting the above expression, we get:

$$\frac{\partial}{\partial\mu_k}l(\mu_k,\Sigma_k) \propto -\frac{1}{2}\sum_i r_{ik}\Big[-2\Sigma^{-1}(x_i-\mu_k)\Big] = \Sigma^{-1}\sum_i r_{ik}(x_i-\mu_k) = 0 \tag{90}$$

which implies that

$$\sum_i r_{ik}(x_i-\mu_k) = 0 \implies \mu_k = \frac{\sum_i r_{ik}x_i}{\sum_i r_{ik}} \tag{91}$$

To compute optimum Σ_k, we can use the trace identity:

$$x^TAx = \mathrm{tr}(x^TAx) = \mathrm{tr}(xx^TA) = \mathrm{tr}(Axx^T) \tag{92}$$

Using $\Lambda = \Sigma^{-1}$ notation, we have:

$$\begin{aligned} l(\Lambda) &\propto -\frac{1}{2}\sum_i r_{ik}\log|\Lambda| - \frac{1}{2}\sum_i r_{ik}\mathrm{tr}\Big[(x_i-\mu)(x_i-\mu)^T\Lambda\Big] \\ &= -\frac{1}{2}\sum_i r_{ik}\log|\Lambda| - \frac{1}{2}\mathrm{tr}(S_\mu\Lambda) \end{aligned} \tag{93}$$

Taking matrix derivative, we get:

$$\begin{aligned} \frac{\partial l(\Lambda)}{\partial\Lambda} &= -\frac{1}{2}\sum_i r_{ik}\Lambda^{-T} - \frac{1}{2}S_\mu^T = 0 \\ \Lambda^{-1}\sum_i r_{ik} &= S_\mu^T \implies \Sigma = \frac{S_\mu^T}{\sum_i r_{ik}} \\ \Sigma &= \frac{\sum_i r_{ik}(x_i-\mu_k)(x_i-\mu_k)^T}{\sum_i r_{ik}} = \frac{\sum_i r_{ik}x_ix_i^T}{\sum_i r_{ik}} - \mu_k\mu_k^T \end{aligned} \tag{94}$$

These equations make intuitive sense, the mean of cluster k is a weighted by r_{ik} average of all points assigned to cluster k, while the covariance is the weighted empirical scatter matrix.

2.2.2 Latent Dirichlet Allocation (LDA)

The EM algorithm maximizes the log-likelihood of the observed data:

$$\theta^* = \arg\max_\theta \sum_{i=1}^n \log p(x_i|\theta) = \arg\max_\theta \sum_{i=1}^n \log\big[\sum_z p(x_i,z_i|\theta)\big] \tag{95}$$

where $p(x_i, z_i|\theta)$ is the joint distribution between the observed words x_i and unobserved (latent) asssignments z_i parameterized by θ. In the case of LDA topic model, $\theta = \{\theta_d, \beta_{w|z}\}$ where $\theta_d \sim \text{Dir}(\alpha)$ are document level topic proportions and $\beta_{w|z} \sim \text{Dir}(\eta)$ are corpus level topics.

E step. Given $\theta^{(k)}$, we want to find soft counts $n(z)$ and $n(w, z)$. First, we compute the posterior over the topic assignments:

$$p(z|w, \theta^{(k)}) = \frac{p(z, w|\theta^{(k)})}{\sum_z p(z, w|\theta^{(k)})} = \frac{\theta_{z|d}\beta_{w|z}}{\sum_z \theta_{z|d}\beta_{w|z}} \tag{96}$$

Given the posterior $p(z|w, \theta^{(k)})$, we can compute the soft counts:

$$n(z) = \sum_{i=1}^{n_d} p(z|w_i, \theta^{(k)}) \tag{97}$$

$$n(w, z) = \sum_{d=1}^{D}\sum_{i=1}^{n_d} p(z|w_i, \theta^{(k)})\delta(w, w_i) \tag{98}$$

where $\delta(w, w_i)$ is an indicator which is equal to 1 when $w = w_i$ and 0 otherwise.

M step. Given the soft counts $n(z)$ and $n(w, z)$, we want to update $\theta^{(k)}$:

$$\theta_{z|d}^{(k+1)} = \frac{n(z)}{n_d} \tag{99}$$

$$\beta_{w|z}^{(k+1)} = \frac{n(w, z)}{\sum_{w \in V} n(w, z)} \tag{100}$$

where n_d is the total number of words in document d and V is our vocabulary. Thus, we can initialize $\theta^{(0)}$ at random and iterate between the E-step and the M-step until convergence. The EM algorithm may require several restarts to avoid local optima. As a result of several iterations, the log-likelihood of observed words under the topic model will increase.

2.3 Gibbs Sampling

2.3.1 Latent Dirichlet Allocation

The LDA graphical model associates each word $w_{i,d}$ with a topic label $z_{i,d} \in \{1, 2, ..., K\}$. Each document is associated with topic proportions θ_d. The topics ϕ_k are shared across all documents. The hyper-parameters α and η define our rior knowledge of topic proportions and topics, respectively. The full generative model can be summarized as follows:

$$\theta_d|\alpha \sim \text{Dir}(\alpha) \tag{101}$$

$$z_{i,d}|\theta_d \sim \text{Cat}(\theta_d) \tag{102}$$

$$\phi_k|\eta \sim \text{Dir}(\eta) \tag{103}$$

$$w_{i,d}|z_{i,d} = k, \phi \sim \text{Cat}(\phi_k) \tag{104}$$

In order to derive Gibbs sampling updates, we focus on the joint distribution $p(w, z) = p(w|z)p(z)$ and we analytically integrate out θ and ϕ parameters. This is known as *collapsed Gibbs sampling* and it tends to be more efficient because we are sampling in a lower dimensional space. The process of sampling z and integrating out θ is known as *Rao-Blackwellization* named after the following theorem:

Theorem 2.1 *(Rao-Blackwell) Let z and θ be dependent random variables and $f(z,\theta)$ be some scalar function, then:*

$$\text{var}_{z,\theta}\big[f(z,\theta)\big] \geq \text{var}_{z}\big[E_{\theta}[f(z,\theta)|z]\big] \tag{105}$$

Joint distribution $p(w,z)$.
We proceed with collapsed Gibbs sampling as follows:

$$\begin{aligned} p(w,z;\alpha,\eta) &= p(w|z)p(z) = \int p(w,\phi|z)d\phi \times \int p(z,\theta)d\theta \\ &= \int p(w|\phi,z)p(\phi)d\phi \times \int p(z|\theta)p(\theta)d\theta = I_1 \times I_2 \end{aligned} \tag{106}$$

Let's compute I_2 first. We know that,

$$p(\theta) = \prod_{d=1}^{D} p(\theta_d|\alpha) = \prod_{d=1}^{D} \text{Dir}(\theta_d|\alpha) = \prod_{d=1}^{D} \frac{1}{Z(\alpha)} \prod_{k=1}^{K} \theta_{d,k}^{\alpha-1}, \quad \text{where} \quad Z(\alpha) = \frac{\prod \Gamma(\alpha_i)}{\Gamma(\sum \alpha_i)} \tag{107}$$

where we assume that $\alpha_1 = \alpha_2 = \cdots = \alpha_k = \alpha$. Similarly, let's write out the distribution for $p(z|\theta)$:

$$\begin{aligned} p(z|\theta) &= \prod_{d=1}^{D}\prod_{j=1}^{N_d} p(z_{d,j}|\theta_d) = \prod_{d=1}^{D}\prod_{j=1}^{N_d} \text{Cat}(\theta_d) = \prod_{d=1}^{D}\prod_{j=1}^{N_d} \theta_{d,k}^{x_k^j} \\ &= \prod_{d=1}^{D}\prod_{k=1}^{K} \theta_{d,k}^{\sum_{j=1}^{N_d} x_k^j} = \prod_{d=1}^{D}\prod_{k=1}^{K} \theta_{d,k}^{n_{d,k}} \end{aligned} \tag{108}$$

where $n_{d,k}$ is a count of words in document d assigned to topic k, and x_k^j is a one-hot encoded vector indicating the presence of topic k assigned to word j. Combining the equations above, we get the following expression for I_2:

$$\begin{aligned} p(z) &= \int p(z|\theta)p(\theta)d\theta = \int \Big[\prod_{d=1}^{D}\prod_{k=1}^{K} \theta_{d,k}^{n_{d,k}}\Big]\Big[\prod_{d=1}^{D} \frac{1}{Z(\alpha)} \prod_{k=1}^{K} \theta_{d,k}^{\alpha-1}\Big]d\theta \\ &= \frac{1}{Z(\alpha)} \prod_{d=1}^{D} \Big[\int \prod_{k=1}^{K} \theta_{d,k}^{\alpha+n_{d,k}-1} d\theta_d\Big] = \frac{1}{Z(\alpha)} \prod_{d=1}^{D} Z(\alpha + n_{d,k}) \end{aligned} \tag{109}$$

where we used the identity for the Dirichlet distribution:

$$\int \prod_{i=1}^{k} x_i^{\alpha_i - 1} dx = Z(\alpha) = \frac{\prod_{i=1}^{n} \Gamma(\alpha_i)}{\Gamma(\sum_i \alpha_i)} \tag{110}$$

Let's compute I_1 next.

$$p(\phi) = \prod_{k=1}^{K} p(\phi_k|\eta) = \prod_{k=1}^{K} \text{Dir}(\phi_k|\eta) = \prod_{k=1}^{K} \frac{1}{Z(\eta)} \prod_{v=1}^{V} \phi_{k,v}^{\eta-1}, \quad \text{where} \quad Z(\eta) = \frac{\prod \Gamma(\eta_i)}{\Gamma(\sum \eta_i)} \tag{111}$$

where we assume that $\eta_1 = \eta_2 = \cdots = \eta_v = \eta$. Similarly, let's write out the distribution for $p(w|\phi,z)$:

$$\begin{aligned} p(w|\phi,z) &= \prod_{d=1}^{D}\prod_{j=1}^{Nd}\prod_{k=1}^{K} p(w_{d,j}|\phi_k) = \prod_{d=1}^{D}\prod_{j=1}^{Nd}\prod_{k=1}^{K} \text{Cat}(\phi_k) \\ &= \prod_{d=1}^{D}\prod_{j=1}^{Nd}\prod_{k=1}^{K}\prod_{v=1}^{V} \phi_{k,v}^{x_v^{d,j}} = \prod_{k=1}^{K}\prod_{v=1}^{V} \phi_{k,v}^{\sum_d \sum_j x_v^{d,j}} = \prod_{k=1}^{K}\prod_{v=1}^{V} \phi_{k,v}^{n_{k,v}} \end{aligned} \tag{112}$$

where $n_{k,v}$ is a count of the number of times word v was assigned to topic k, and $x_v^{d,j}$ is a one-hot encoded vector indicating the presence of word v in document d at word location j. Combining the equations above, we get the following expression for I_1:

$$\begin{aligned} p(w|z) &= \int p(w|\phi,z)p(\phi)d\phi = \int \Big[\prod_{k=1}^{K}\prod_{v=1}^{V}\phi_{k,v}^{n_{k,v}}\Big]\Big[\prod_{k=1}^{K}\frac{1}{Z(\eta)}\prod_{v=1}^{V}\phi_{k,v}^{\eta-1}\Big] \\ &= \frac{1}{Z(\eta)}\prod_{k=1}^{K}\Big[\int\prod_{v=1}^{V}\phi_{k,v}^{n_{k,v}+\eta-1}\Big] = \frac{1}{Z(\eta)}\prod_{k=1}^{K}Z(\eta+n_{k,v}) \end{aligned} \tag{113}$$

Finally, taking a product of I_1 and I_2, we can compute the joint distribution:

$$p(w,z) = p(w|z)p(z) = \Big[\frac{1}{Z(\eta)}\prod_{k=1}^{K}Z(\eta+n_{k,v})\Big]\Big[\frac{1}{Z(\alpha)}\prod_{d=1}^{D}Z(\alpha+n_{d,k})\Big] \tag{114}$$

Posterior of θ and ϕ.
We have integrated out θ and ϕ for collapsed Gibbs sampling, however, we still need to obtain the posterior distribution over these latent variables. In the derivation below, we condition on the assignments $z_{d,j}$ to compute the posterior distribution:

$$\begin{aligned} p(\theta_d|z_d) &\propto p(z_d|\theta_d)p(\theta_d) = \prod_{j=1}^{N_d}p(z_{d,j}|\theta_d)\prod_{k=1}^{K}\theta_{d,k}^{\alpha-1} = \prod_{k=1}^{K}\theta_{d,k}^{n_{d,k}}\prod_{k=1}^{K}\theta_{d,k}^{\alpha-1} \\ &= \prod_{k=1}^{K}\theta_{d,k}^{n_{d,k}+\alpha-1} \sim \mathrm{Dir}(\alpha+n_{d,k}) \end{aligned} \tag{115}$$

Similarly, $\phi_{k,v} \sim \mathrm{Dir}(\eta+n_{k,v})$.

Collapsed Gibbs updates.
We need to compute the full conditional distribution $p(z_i|z_{-i},w)$. We can proceed as follows:

$$\begin{aligned} p(z_i=k|z_{-i},w) &= \frac{p(z_i,z_{-i},w)}{p(z_{-i},w)} = \frac{p(w|z)p(z)}{p(z_{-i},w_{-i},w_i)} = \frac{p(w|z)p(z)}{p(w_{-i}|z_{-i},w_i)p(z_{-i})p(w_i)} \\ &\propto \frac{p(w|z)p(z)}{p(w_{-i}|z_{-i})p(z_{-i})} = \frac{\prod_{k=1}^{K}Z(\eta+n_{k,v})}{\prod_{k=1}^{K}Z(\eta+n_{-k,v})}\times\frac{\prod_{d=1}^{D}Z(\alpha+n_{d,k})}{\prod_{d=1}^{D}Z(\alpha+n_{d,-k})} \end{aligned} \tag{116}$$

For all docs other than $d=d'$, the numerator and denominator of the second factor are exactly the same:

$$\begin{aligned} \frac{\prod_{d=1}^{D}Z(\alpha+n_{d,k})}{\prod_{d=1}^{D}Z(\alpha+n_{d,-k})} &= \frac{Z(\alpha+n_{d',k})}{Z(\alpha+n_{d',-k})} = \frac{\prod_{k=1}^{K}\Gamma(\alpha+n_{d',k})}{\prod_{k=1}^{K}\Gamma(\alpha+n_{d',-k})}\times\frac{\Gamma(\sum_{k=1}^{K}\alpha+n_{d',-k})}{\Gamma(\sum_{k=1}^{K}\alpha+n_{d',k})} \\ &= \frac{\Gamma(\alpha+n_{d',-k'}+1)}{\Gamma(\alpha+n_{d',-k'})}\times\frac{\Gamma(\sum_{k=1}^{K}\alpha+n_{d',-k})}{\Gamma(\sum_{k=1}^{K}\alpha+n_{d',-k}+1)} \\ &= \frac{\alpha+n_{d',-k'}}{\sum_{k=1}^{K}\alpha+n_{d',-k}} \end{aligned} \tag{117}$$

where we used the identity: $\Gamma(x+1) = x! = x(x-1)! = x\Gamma(x)$. Similarly, we can simplify the first factor to:

$$\frac{\prod_{k=1}^{K}Z(\eta+n_{k,v})}{\prod_{k=1}^{K}Z(\eta+n_{-k,v})} = \frac{\eta+n_{-k',v'}}{\sum_{v=1}^{V}\eta+n_{-k',v}} \tag{118}$$

Putting the two factors together, our full conditional distribution is:

$$p(z_i = k'|z_{-i}, w) \propto \left[\frac{\eta + n_{-k',v'}}{\sum_{v=1}^{V} \eta + n_{-k',v}}\right]\left[\frac{\alpha + n_{d',-k'}}{\sum_{k=1}^{K} \alpha + n_{d',-k}}\right] \tag{119}$$

In the expression above, the first ratio is a probability of word w_i (for which we are sampling the label z_i) under topic k' and the second ratio is the probability of topic k' in document d'.

2.4 Metropolis-Hastings (MH) Sampling

2.4.1 Gaussian Mixture Model (GMM)

2.5 Slice sampling

2.6 HMC sampling

2.7 Annealed Importance Sampling

Annealed Importance Sampling (AIS) is a method that combines independent importance sampling with Markov chain methods. It can be used to sample from multi-modal posterior distributions as well as to estimate ratios of normalizing constants (used in Bayesian model selection).

2.7.1 AIS algorithm

Consider a sequence of distributions $p_n(x), p_{n-1}(x), ..., p_1(x), p_0(x)$, where $p_0(x)$ is our target distribution of interest (a posterior) and $p_n(x)$ is a starting distribution (a prior). Our goals is to draw samples from a multi-modal $p_0(x)$ by using a sequence of intermediate distributions $p_j(x)$:

$$p_j(x) \propto p_0(x)^{\beta_j} p_n(x)^{1-\beta_j}, \quad \text{where} \quad 0 = \beta_n < \beta_{n-1} < \cdots < \beta_0 = 1 \tag{120}$$

We assume that we can draw samples from intermediate distributions $p_j(x)$ via a sequence of Markov chains $T_j(x, x')$ that model a transition from state x to x'. $T_j(x, x')$ can be any of the Markov chain methods that satisfy detailed balance, i.e. Gibbs, Slice or Metropolis-Hastings (MH) samplers. The AIS algorithm can be summarized as follows:

Algorithm 2 Annealed Importance Sampling (AIS) [32]

1: Define a target distribution $p_0(x)$, a prior $p_n(x)$, and an intermediate $p_j(x)$
2: Define a range of β values: $0 = \beta_n < \beta_{n-1} < \cdots < \beta_0 = 1$
3: Define the transitions $T_j(x, x')$
4: **for** sample $s = 1$ to S **do**
5: init $w = 0$
6: sample $x_{prev} \sim p_n(x)$
7: **for** $j = n - 1$ to 1:
8: $x_{new} \sim T_j(x_{prev}, \cdot)$
9: $x_{prev} = x_{new}$
10: $w = w + \log(p_{j-1}(x_{prev})) - \log(p_j(x_{prev}))$
11: **end for**
12: samples$[s] = x_{prev}$
13: weights$[s] = \exp(w)$
14: **end for**
15: **return** samples, weights

According to Algorithm 2, we obtain 1 sample from the target $p_0(x)$ and the associated importance weight at the cost of n samples from intermediate distributions $p_j(x)$. The algorithm is using an equation for computing the importance weights, let's derive it below.

AIS can be seen as importance sampling in an extended z-space $\{z_0, ..., z_{n-1}\}$. We can represent the target distribution as follows:

$$p(z) \propto f(z) = f_0(z_0)\tilde{T}_1(z_0, z_1)\tilde{T}_2(z_1, z_2) \times \cdots \times \tilde{T}_{n-1}(z_{n-2}, z_{n-1}) \tag{121}$$

where $\tilde{T}_j$ is the reversal of T_j. We assume the samples satisfy detailed balance:

$$\begin{aligned} p_j(z)\tilde{T}_j(z, z') &= p_j(z')T_j(z', z) && (122) \\ \tilde{T}_j(z, z') &= T_j(z', z)\frac{p_j(z')}{p_j(z)} = T_j(z', z)\frac{f_j(z')}{f_j(z)} && (123) \end{aligned}$$

As a result, we can re-write the target distribution as follows:

$$p(z) \propto f_0(z_0)\left[T_1(z_1, z_0)\frac{f_1(z_1)}{f_1(z_0)}\right]\left[T_2(z_2, z_1)\frac{f_2(z_2)}{f_2(z_1)}\right] \times ... \times \left[T_{n-1}(z_{n-1}, z_{n-2})\frac{f_{n-1}(z_{n-1})}{f_{n-1}(z_{n-2})}\right] \tag{124}$$

Similarly, the proposal distribution is given by:

$$q(z) \propto g(z) = f_n(z_{n-1})T_{n-1}(z_{n-1}, z_{n-2}) \times \cdots \times T_2(z_2, z_1)T_1(z_1, z_0) \tag{125}$$

Thus, the importance weights are given by:

$$\begin{aligned} w &= \frac{f(z_0, ..., z_{n-1})}{g(z_0, ..., z_{n-1})} \\ &= \frac{f_0(z_0)\left[T_1(z_1, z_0)\frac{f_1(z_1)}{f_1(z_0)}\right]\left[T_2(z_2, z_1)\frac{f_2(z_2)}{f_2(z_1)}\right] \times \cdots \times \left[T_{n-1}(z_{n-1}, z_{n-2})\frac{f_{n-1}(z_{n-1})}{f_{n-1}(z_{n-2})}\right]}{T_1(z_1, z_0)T_2(z_2, z_1) \times \cdots \times T_{n-1}(z_{n-1}, z_{n-2})f_n(z_{n-1})} \\ &= \frac{f_{n-1}(z_{n-1})}{f_n(z_{n-1})} \times \frac{f_{n-2}(z_{n-2})}{f_{n-1}(z_{n-2})} \times \cdots \times \frac{f_1(z_1)}{f_2(z_1)} \times \frac{f_0(z_0)}{f_1(z_0)} \end{aligned} \tag{126}$$

We can see a telescoping product of ratios of distributions which is equal to standard importance sampling when $n = 1$.

AIS can also be used to approximate the ratio of normalizing constants. Suppose, we are interested in computed an expected value of $f(x)$. We can proceed as follows:

$$\begin{aligned} E[f(x)] &= \int f(x)p(x)dx = \int f(x)\frac{q(x)}{q(x)}p(x)dx = \frac{Z_q}{Z_p}\int f(x)\frac{\tilde{p}(x)}{\tilde{q}(x)}q(x)dx \\ &= \frac{Z_q}{Z_p}\frac{1}{S}\sum_{s=1}^{S}\tilde{w}(s)f(s), \text{ where } s \sim q(x) \text{ and } \tilde{w}(s) = \frac{\tilde{p}(s)}{\tilde{q}(s)} \end{aligned} \tag{127}$$

We can compute the ratio of normalizing constants as follows:

$$\frac{Z_p}{Z_q} = \frac{1}{Z_q}\int \tilde{p}(x)dx = \frac{1}{Z_q}\int \frac{\tilde{p}(x)}{q(x)}q(x)dx = \int \frac{\tilde{p}(x)}{\tilde{q}(x)}q(x)dx = \frac{1}{S}\sum_{s=1}^{S}\tilde{w}(s) \tag{128}$$

Combining the two expression above, we get:

$$E[f] = \frac{Z_q}{Z_p}\frac{1}{S}\sum_{s=1}^{S}\tilde{w}(s)f(s) = \frac{\frac{1}{S}\sum_s \tilde{w}(s)f(s)}{\frac{1}{S}\sum_s \tilde{w}(s)} = \sum_{s=1}^{S}w(s)f(s) \tag{129}$$

2.7.2 AIS estimator

This section studies the properties of AIS estimator and computes quantities relevant to characterizing the performance of AIS. The AIS estimator can be defined as follows:

$$\hat{a} = E[a(x)] = \frac{\frac{1}{n}\sum_{i=1}^{n} \tilde{w}^i a(x_i)}{\frac{1}{n}\sum_{i=1}^{n} \tilde{w}^i} = \frac{1}{n}\sum_{i=1}^{n} w_*^i a(x_i), \quad \text{where} \quad w_*^i = \tilde{w}^i / n^{-1} \sum_i \tilde{w}^i \tag{130}$$

By the Weak Law of Large Numbers (WLLN), assuming independent samples x_i, we know that the average of iid samples converges in probability to the true mean:

$$a = E[w_*(X_i)a(X_i)] \to b(\hat{a}) = E[\hat{a}] - a = 0 \tag{131}$$

where X_i are assumed to be iid random variables distributed according to $q(x)$. As a result, AIS is in theory an unbiased estimator. However, due to samples from intermediate distributions being not always independent (due to correlations in Markov Chain using Metropolis-Hastings sampler), there can exist a bias associated with AIS estimates.

The variance of the AIS estimator is given by:

$$\text{VAR}(\hat{a}) = \frac{\sigma^2}{n} = \frac{1}{n}\frac{\sum_{i=1}^{n}\left(\tilde{w}^i(a(x_i) - \hat{a})\right)^2}{\left(\sum_{i=1}^{n}\tilde{w}^i\right)^2} \tag{132}$$

To check whether AIS estimate is consistent, we need to show that as we increase the number of samples n, the estimate $\hat{a}$ converges to the true value a in probability:

$$\lim_{n\to\infty} P(|\hat{a}_n - a| > \epsilon) = 0 \tag{133}$$

Using Chebyshev's inequality, we can bound the deviation from a as follows:

$$\lim_{n\to\infty} P(|\hat{a} - a| > \epsilon) \leq \frac{\text{VAR}(\hat{a})}{\epsilon^2} = \frac{\sigma^2}{n\epsilon^2} = 0 \tag{134}$$

As a result, the AIS estimator is consistent, here we didn't need to use an assumption that samples are independent but we needed to assume that the variance of importance weights is finite. Thus, we expect the variance around the estimate to shrink as we add more samples. One factor that affects the variance of the estimator is the magnitude of importance weights. Since $w(x_i) = p(x_i)/q(x_i)$, the weights are small if $q(x_i)$ is similar to $p(x_i)$ and has heavy tails. This is the desired case that leads to a small variance in our estimate.

We would like to have a way of telling when the importance weights are problematic, e.g. when only a couple of weights dominate the weighted sum. When samples are correlated the variance is σ^2/n_{eff}, where n_{eff} is the effective sample size. To compute an expression for effective sample size, we set the variance of weighted average equal to the variance of unweighted average:

$$\text{VAR}\left(\frac{1}{n}\sum_{i=1}^{n} a(x_i)\right) = \frac{\sigma^2}{n_{eff}} = \text{VAR}\left(\frac{\frac{1}{n}\sum_{i=1}^{n}\tilde{w}^i a(x_i)}{\frac{1}{n}\sum_{i=1}^{n}\tilde{w}^i}\right) \quad \text{where} \quad \text{VAR}(a(x_i)) = \sigma^2 \tag{135}$$

Solving for n_{eff}, we get

$$n_{eff} = \frac{(\sum_{i=1}^{n} w_i)^2}{\sum_{i=1}^{n} w_i^2} = \frac{n}{1 + \text{VAR}(w_*^i)} \tag{136}$$

We can use n_{eff} is a kind of diagnostics for the importance sampler where the target number n_{eff} depends on the application. We can also compute a confidence interval for our estimate. Given the variance expression above, an approximate 99% confidence interval for a is $\hat{a} \pm 2.58\sigma_a/\sqrt{n}$.

2.7.3 AIS experiments

Minimum Working Example To test how well AIS can estimate the quantities of interest such as posterior mean and normalization constant, we design the following experiment. Let $p_n = N(0, 1)$ be the standard normal prior distribution and $p_0 \propto N(-5, 2)$ be the unnormalized posterior. We let β linearly vary in the interval $[0, 1]$ divided into 100 steps, i.e. we have 100 intermediate distributions formed by taking a weighted geometric average between the prior and the posterior.

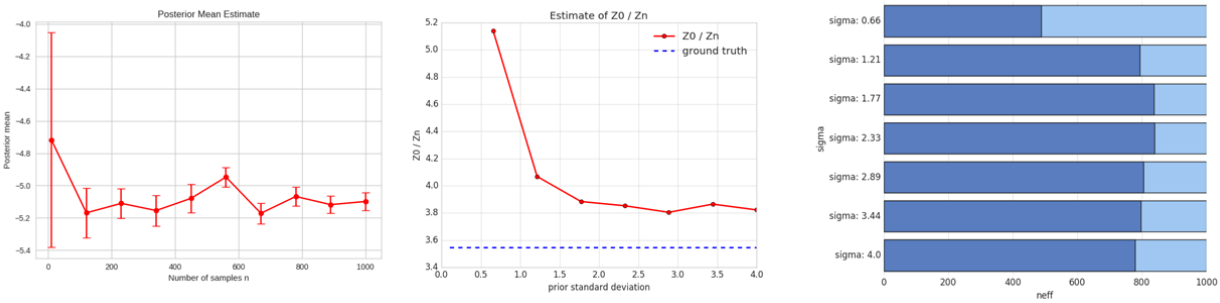

Figure 21: AIS posterior mean estimate (left). AIS normalizer estimate (middle). AIS effective sample size (right).

We use $n = 10$ step Metropolis-Hastings (MH) sampler to sample from intermediate distributions and compute importance weights as described in Algorithm 2. The experimental results are summarized in Figure 21. As expected, as the number of AIS samples increases the variance of our estimate drops. However, the AIS estimator is not completely unbiased as we can see from Figure 21 (left) since the ground truth value is -5. Similarly, there's evidence of bias in the normalization constant estimate as can be seen in Figure 21 (middle), since the ground truth value is equal to $Z_0/Z_n = \sqrt{4\pi} = 3.545$. This bias is most likely caused by the fact that samples are not completely independent (due to MH). We can also measure our sample efficiency by observing the number of effective samples Figure 21 (right) as we vary the variance of our prior distribution. The solid bars indicate the effective number of samples out of a total of 1000.

3 Variational Inference

3.1 Introduction

This section focuses on a class of approximate inference algorithms based on variational inference. The basic idea is to choose an approximation $q(x)$ from a tractable family of distributions and then make this approximation as close as possible to the true posterior $p^*(x)$. This reduces inference to an optimization problem.

We can use KL divergence to measure the distance between distributions. In particular, we use reverse KL to make the computation tractable.

$$KL(q||p^*) = \sum_x q(x) \log \frac{q(x)}{p^*(x)} \tag{137}$$

Let $\tilde{p}(x) = p^*(x)Z$ be the un-normalized distribution, then our objective function:

$$J(q) = KL(q||\tilde{p}) = \sum_x q(x) \log \frac{q(x)}{p^*(x)Z} = \sum_x q(x) \log \frac{q(x)}{p^*(x)} - \log Z = KL(q||p^*) - \log Z \tag{138}$$

Since KL divergence is non-negative, $J(q)$ is an upper bound on the marginal likelihood:

$$J(q) = KL(q||p^*) - \log Z \geq -\log Z = -\log p(D) \tag{139}$$

when $q(x)$ equals the true posterior $p^*(x)$, the KL divergence vanishes and the optimal value $J(q^*)$ equals the log partition function and for all other values of q it yields a bound. $J(q)$ is called the *variational free energy* and can be written as:

$$\min_q J(q) = E_q[\log q(x)] + E_q[-\log \tilde{p}(x)] = -H(q) + E_q[E(x)] \tag{140}$$

The variational objective function (140) is closely related to energy minimization in statistical physics. The first term acts as a regularizer by encouraging maximum entropy, while the second term is the expected energy and encourages the variational distribution q to explain the data.

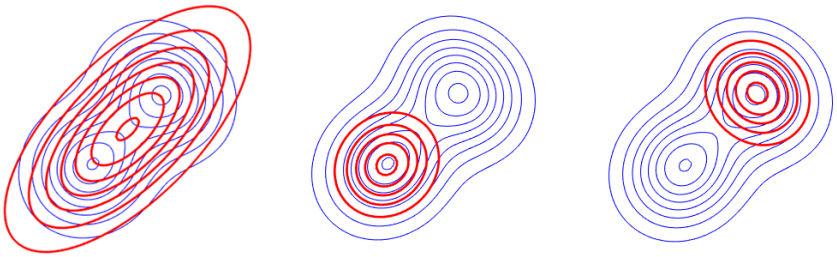

Figure 22: Forward vs reverse KL on a bimodal distribution. The blue contours is the true distribution $p(x)$. The red contours is approximate distribution $q(x)$.

The reverse KL that acts as a penalty term in the variational objective is also known as I-projection or information projection. In the reverse KL, $q(x)$ will typically under-estimate the support of $p(x)$ and will lock onto one of its modes. This is due to $q(x) = 0$ whenever $p(x) = 0$ to make sure the KL divergence stays finite. On the other hand, the forward KL, known as M-projection or momement projection is zero avoiding for $q(x)$ and will over-estimate the support

of $p(x)$ as shown in Figure 22.

We can use the Jensen's inequality to derive the Evidence Lower BOund (ELBO), an objective that we can maximize to learn the variational parameters of our model. Let x be our data and z be the latent variables, then we can derive our ELBO objective as follows:

$$\log p(x) = \log \sum_z p(x,z) = \log \sum_z \frac{q(z)}{q(z)} p(x,z) = \log E_{q(z)}\left[\frac{p(x,z)}{q(z)}\right] \tag{141}$$

$$\geq E_{q(z)}\left[\log \frac{p(x,z)}{q(z)}\right] = E_{q(x)}\left[\log p(x,z)\right] - E_{q(x)}\left[\log q(x)\right] = \text{ELBO} \tag{142}$$

Notice that the first term is the average negative energy and the second term is the entropy. Thus, a good posterior must assign most of its probability mass to regions of low energy (i.e. high joint probability density) while also maximizing the entropy of $q(z)$. Thus, variational Bayes, in constrast to MAP, prevents $q(z)$ from collapsing to an atom.

One form of ELBO emphasizes that the lower bound becomes tighter as the variational distribution better approximates the posterior:

$$\text{ELBO} = E_{q(z)}\left[\log \frac{p(x,z)}{q(z)}\right] = E_{q(z)}\left[\log \frac{p(z|x)p(x)}{q(z)}\right] = -KL(q(z)||p(z|x)) + \log p(x) \tag{143}$$

Therefore, we can improve the ELBO by improving the model log evidence $\log p(x)$ through the prior $p(z)$ or the likelihood $p(z|x)$ or by improving the variational posterior approximation $q(z)$.

Finally, we can write the ELBO as follows:

$$\text{ELBO} = \frac{1}{n}\sum_{i=1}^{n}\left[E_{q(z_i)}\big[\log p(x_i|z_i)\big] - KL(q(z_i)||p(z_i))\right] \tag{144}$$

this version emphasizes a likelihood term for the i-th observation and KL divergence term between each approximating distribution and the prior.

One of the most popular forms of variational inference is called the *mean field* approximation, where we assume that the posterior is a fully factorized approximation of the form:

$$q(x) = \prod_i q_i(x_i) \tag{145}$$

where we optimize over the parameters of each marginal distribution $q_j(x_j)$. Our goal is to minimize variational free energy $J(q)$ or equivalently, maximize the lower bound:

$$L(q) = -J(q) = \sum_x q(x) \log \frac{\tilde{p}(x)}{q(x)} \tag{146}$$

We can re-write the objective for each marginal distribution q_j, keeping the rest of the terms as

constants:

$$
\begin{aligned}
L(q_j) &= \sum_x \prod_i q_i(x_i) \Big[\log \tilde{p}(x) - \sum_k \log q_k(x_k) \Big] && (147)\\
&= \sum_{x_j} \sum_{x_{-j}} q_j(x_j) \prod_{i \neq j} q_i(x_i) \Big[\log \tilde{p}(x) - \sum_k \log q_k(x_k) \Big] && (148)\\
&= \sum_{x_j} q_j(x_j) \log f_j(x_j) - \sum_{x_j} q_j(x_j) \sum_{x_{-j}} \prod_{i \neq j} q_i(x_i) \Big[\sum_{k \neq j} \log q_k(x_k) + \log q_j(x_j) \Big] && (149)\\
&= \sum_{x_j} q_j(x_j) \log f_j(x_j) - \sum_{x_j} q_j(x_j) \log q_j(x_j) + \text{const} && (150)
\end{aligned}
$$

where we defined $\log f_j(x_j) = \sum_{x_{-j}} \prod_{i \neq j} q_i(x_i) \log \tilde{p}(x) = E_{-q_j}[\log \tilde{p}(x)]$. Since we are replacing the values by their mean value, the method is known as mean field. We can re-write $L(q_j) = -KL(q_j || f_j)$ and therefore maximize the objective by setting $q_j = f_j$ or equivalently:

$$\log q_j(x_j) = \log f_j(x_j) = E_{-q_j}[\log \tilde{p}(x)] \tag{151}$$

where the functional form of q_j will be determined by the type of variables x_j and their probability model.

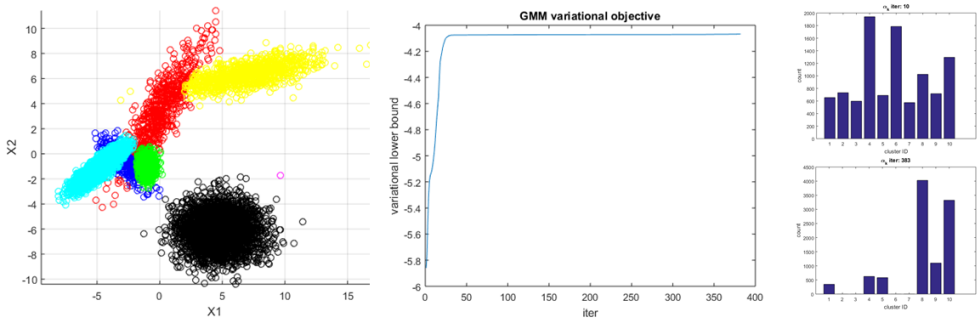

Figure 23: Variational Bayes EM applied to a Mixture of Gaussians

Figure 23 shows the posterior clustering assignment as a result of running Variational Bayes EM algorithm on a Gaussian Mixture. The algorithm correctly identified $K = 6$ clusters using 10 clusters as the starting point. We can see the rapid increase in the variational objective when the algorithm figures out that it could increase the objective by removing unnecessary clusters in early iterations, while the plateau results in moving the clusters around. This is also evident from the prior and posterior parameter values for mixing proportions α_k. As the number of iterations increase the counts for unnecessary mixture components drop to zero.

3.2 Application: Topic Models

A topic model is a latent variable model for discrete data such as text documents. Latent Dirichlet Allocation (LDA) is a topic model that represents each document as a finite mixture of topics, where a topic is a distribution over words. The objective is to learn the shared topic distribution and topic proportions for each document. LDA assumes a bag of words model in which the words are exchangeable and as a result sentence structure is not preserved, i.e. only the word counts matter. Thus, each document is reduced to a vector of counts over the vocabulary V and the entire corpus of D documents is summarized in a *term-document* matrix $A_{V \times D}$.

LDA can be seen as a non-negative matrix factorization problem that takes the term-document matrix and factorizes it into a product of topics $W_{V\times K}$ and topic proportions $H_{K\times D}$: $A = WH$.

A common method for adjusting the word counts is *tf-idf* that logarithmically drives down to zero word counts that occur frequently across documents: $A_{t,d} \log \frac{D}{n_t}$, where D is the total number of documents in the corpus and n_t is the number of documents where term t appears. The *tf-idf* smoothing identifies the sets of words that are discriminative for documents and leads to better model performance. The term-document matrix generalizes from counts of individual words (unigrams) to larger structural units such as n-grams. In the case of n-grams different smoothing techniques (such as Laplace smoothing) are used to address the lack of observations in a very large feature space.

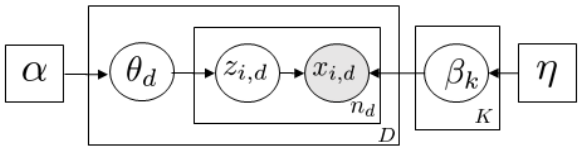

Figure 24: Latent Dirichlet Allocation (LDA) graphical model

Figure 24 shows the graphical model for the Latent Dirichlet Allocation (LDA). The LDA topic model associates each word $x_{i,d}$ with a topic label $z_{i,d} \in \{1, 2, ..., K\}$. Each document is associated with topic proportions θ_d that could be used to measure document similarity. The topics β_k are shared across all documents. The hyper-parameters α and η capture our prior knowledge of topic proportions and topics, respectively, e.g. from past on-line training of the model. The full generative model can be specified as follows:

$$\theta_d|\alpha \sim \text{Dir}(\alpha) \tag{152}$$
$$z_{i,d}|\theta_d \sim \text{Cat}(\theta_d) \tag{153}$$
$$\beta_k|\eta \sim \text{Dir}(\eta) \tag{154}$$
$$x_{i,d}|z_{i,d} = k, \beta \sim \text{Cat}(\beta_k) \tag{155}$$

The joint distribution for a single document d can be written as follows [5]:

$$p(x, z, \theta, |\alpha, \beta) = p(\theta_d|\alpha) \prod_{i=1}^{n_d} p(z_{i,d}|\theta_d) p(x_{i,d}|z_{i,d}, \beta) \tag{156}$$

The parameters α and β are corpus-level parameters, the variable θ_d is sampled once every document, while $z_{i,d}$ and $x_{i,d}$ are word-level variables sampled once for each word in each document. Unlike a multinomial clustering model where each document is associated with a single topic, LDA represents each document as a mixture of topics.

The key inference problem that we need to solve in order to use LDA is that of computing the posterior distribution of the latent variables for a given document: $p(\theta, z|x, \alpha, \beta)$. The posterior can be approximated with the following variational distribution:

$$q(\theta, z|\gamma, \phi) = q(\theta|\gamma) \prod_{i=1}^{n} q(z_i|\phi_i) \tag{157}$$

The variational parameters are optimized to maximize the Evidence Lower BOund (ELBO):

$$\log p(x|\alpha,\eta) \geq L(x,\phi,\gamma,\lambda) = E_q[\log p(x,z,\theta,\beta|\alpha,\eta)] - E_q[\log q(z,\theta,\beta)] \tag{158}$$

We choose a fully factored distribution q of the form:

$$q(z_{id}=k) = \phi_{dwk}; \quad q(\theta_d) \sim \mathrm{Dir}(\theta_d|\gamma_d); \quad q(\beta_k) \sim \mathrm{Dir}(\beta_k|\lambda_k) \tag{159}$$

We can expand the lower bound by using the factorizations of p and q:

$$L(\gamma,\phi;\alpha,\beta) = \mathbb{E}_q[\log p(\theta|\alpha)] + \mathbb{E}_q[\log p(z|\theta)] + \mathbb{E}_q[\log p(w|z,\beta)] - \mathbb{E}_q[\log q(\theta)] - \mathbb{E}_q[\log q(z)]$$

Each of the five terms in $L(\gamma,\phi;\alpha,\beta)$ can be expanded [5] as follows:

$$\begin{aligned}
L(\gamma,\phi;\alpha,\beta) &= \log\Gamma(\sum_{j=1}^{k}\alpha_j) - \sum_{i=1}^{k}\log\Gamma(\alpha_i) + \sum_{i=1}^{k}(\alpha_i-1)(\Psi(\gamma_i) - \Psi(\sum_{j=1}^{k}\gamma_j)) && (160)\\
&+ \sum_{n=1}^{N}\sum_{i=1}^{k}\phi_{ni}(\Psi(\gamma_i) - \Psi(\sum_{j=1}^{k}\gamma_j)) && (161)\\
&+ \sum_{n=1}^{N}\sum_{i=1}^{k}\sum_{j=1}^{V}\phi_{ni}w_n^j\log\beta_{ij} && (162)\\
&- \log\Gamma(\sum_{j=1}^{k}\gamma_j) + \sum_{i=1}^{k}\log\Gamma(\gamma_i) - \sum_{i=1}^{k}(\gamma_i-1)(\Psi(\gamma_i) - \Psi(\sum_{j=1}^{k}\gamma_j)) && (163)\\
&- \sum_{n=1}^{N}\sum_{i=1}^{k}\phi_{ni}\log\phi_{ni} && (164)
\end{aligned}$$

where $\Psi(x) = \frac{d}{dx}\log\Gamma(x)$ is the digamma function. $L(\gamma,\phi;\alpha,\beta)$ can be maximized using coordinate ascent over the variational parameters ϕ, γ, λ [5]:

$$\begin{aligned}
\phi_{dwk} &\propto \exp\{E_q[\log\theta_{dk}] + E_q[\log\beta_{kw}]\} && (165)\\
\gamma_{dk} &= \alpha + \sum_w n_{dw}\phi_{dwk} && (166)\\
\lambda_{kw} &= \eta + \sum_d n_{dw}\phi_{dwk} && (167)
\end{aligned}$$

where the expectations under q of $\log\theta$ and $\log\beta$ are:

$$E_q[\log\theta_{dk}] = \Psi(\gamma_{dk}) - \Psi(\sum_{i=1}^{K}\gamma_{di}) \quad E_q[\log\beta_{kw}] = \Psi(\lambda_{kw}) - \Psi(\sum_{i=1}^{W}\lambda_{ki}) \tag{168}$$

The variational parameter updates in (165) can be used in an online setting that does not require a full pass through the entire corpus at each iteration. An online update of variational parameters enables topic analysis for very large datasets including streaming data. Online VB for LDA is described in Algorithm 3.

As the t-th vector of word counts n_t is observed, we perform an E step to find locally optimal values of γ_t and ϕ_t, holding λ fixed. We then compute $\tilde{\lambda}$ that would be optimal if our entire corpus consisted of the single document n_t repeated D times. We then update λ as

Algorithm 3 Online variational Bayes for LDA [21]

Define $\rho_t = (\tau_0 + t)^{-\kappa}$
Initialize λ randomly
for t = 1 to ∞ do
    E step:
    Initialize $\gamma_{tk} = 1$
    repeat
        Set $\phi_{twk} \propto \exp\{E_q[\log \theta_{tk}] + E_q[\log \beta_{kw}]\}$
        Set $\gamma_{tk} = \alpha + \sum_w \phi_{twk} n_{tw}$
    until $\frac{1}{K} \sum_k |\Delta \gamma_{tk}| < \epsilon$
    M step:
    Compute $\tilde{\lambda}_{kw} = \eta + D n_{tw} \phi_{twk}$
    Set $\lambda = (1 - \rho_t)\lambda + \rho_t \tilde{\lambda}$
end for

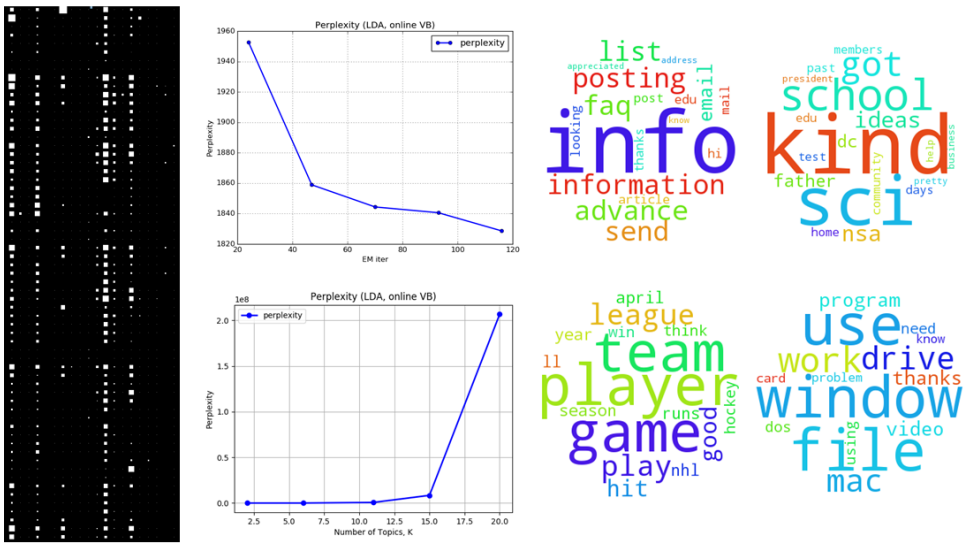

Figure 25: Online Variational Bayes Inference Results for LDA.

a weighted average of its previous value and $\tilde{\lambda}$, where the weight is given by the learning parameter $\rho_t = (\tau_0 + t)^{-\kappa}$ for $\kappa \in (0.5, 1]$, controlling the rate at which old values of $\tilde{\lambda}$ are forgotten.

Figure 25 shows the inference results for the online VB LDA algorithm on 20newsgroups dataset consisting of $11,314$ documents and a compressed vocabulary size of $1K$ words. The number of topics was set to $K = 20$ and the batch size (number of documents to use in each EM iteration) was set to 512. Figure 25 shows the Hinton diagram of the inferred topic matrix $W_{V \times K}$ where only the first 64 rows are shown. The perplexity plots show the improvement in model ability to explain the data as the number of EM iterations increases, where perplexity is defined as follows:

$$\text{Perplexity}(\text{w}_{\text{test}}) = \exp\{-\frac{1}{D_{test}} \sum_d \frac{1}{n_d} \sum_{w \in n_d} \log p(w_{test})\} \tag{169}$$

Model selection can be done by evaluating perplexity for different values of K. Guided by the fact that there are 20 different newsgroups, the value of K was set to 20. Finally, the top 20 words for a random sample of 4 topics are shown in Figure 25. The four topics are about information, sports, computers and school.

3.3 Application: Image Denoising in Ising Model

The Ising model is an example of a Markov Random Field (MRF) and has its origins in statistical physics. The Ising model assumes that we have a grid of nodes, where each node can be in one of two possible states. The state of each node depends on the neighboring nodes through interaction potentials. In the case of images, this translates to a smoothness constraint, i.e. a pixel prefers to be of the same color as the neighboring pixels. In the image denoising problem, we assume that we have a 2-D grid of noisy pixel observations of an underlying true image and we would like to recover the true image.

Let y_i be noisy observations of binary latent variables $x_i \in \{-1, +1\}$. We can write down the joint distribution as follows:

$$p(x, y) = p(x)p(y|x) = \prod_{(s,t) \in E} \Psi_{st}(x_s, x_t) \prod_{i=1}^{n} p(y_i|x_i) = \prod_{(s,t) \in E} \exp\{x_s w_{st} x_t\} \prod_{i=1}^{n} N(y_i|x_i, \sigma^2) \tag{170}$$

where the interaction potentials are represented by Ψ_{st} for every pair of nodes x_s and x_t in a set of edges E and the observations y_i are Gaussian with mean x_i and variance σ^2. Here, w_{st} is the coupling strength and assumed to be constant and equal to $J > 0$ indicating a preference for the same state as neighbors (i.e. the potential $\Psi(x_s, x_t) = \exp\{x_s J x_t\}$ is higher when x_s and x_t are both either $+1$ or -1).

To fit the model parameters using variational inference, we want to maximize the ELBO:

$$\text{ELBO} = E_{q(x)}\Big[\log p(x, y)\Big] - E_{q(x)}\Big[\log q(x)\Big] = \tag{171}$$

$$= E_{q(x)}\Big[\sum_{(s,t) \in E} x_s w_{st} x_t + \sum_{i=1}^{n} \log N(x_i; \sigma^2)\Big] - \sum_{i=1}^{n} E_{q_i(x)}\Big[\log q_i(x)\Big] \tag{172}$$

where we are using the mean-field assumption of a fully-factored approximation $q(x)$:

$$q(x) = \prod_{i=1}^{n} q(x_i; \mu_i) \tag{173}$$

Using the previously derived result, we state that $q(x_i; \mu_i)$ that minimizes the KL divergence is given by:

$$q_i(x_i) = \frac{1}{Z_i} \exp\left[E_{-q_i}\{\log p(x)\}\right] \tag{174}$$

where E_{-q_i} denotes the expectation over every q_j except for $j = i$. To compute $q_i(x_i)$, we only care bout the terms that involve x_i, i.e. we can isolate them as follows:

$$E_{-q_i}\{\log p(x)\} = E_{-q_i}\{x_i \sum_{j \in N(i)} w_{ij} x_j + \log N(x_i, \sigma^2) + \text{const}\} = \tag{175}$$

$$= x_i \sum_{j \in N(i)} J \times \mu_j + \log N(x_i, \sigma^2) + \text{const} \tag{176}$$

where $N(i)$ denotes the neighbors of node i and μ_j is the mean of a binary random variable:

$$\mu_j = E_{q_j}[x_j] = q_j(x_j = +1) \times (+1) + q_j(x_j = -1) \times (-1) \tag{177}$$

Figure 26 shows the parametric form of our mean-field approximation for the Ising model:

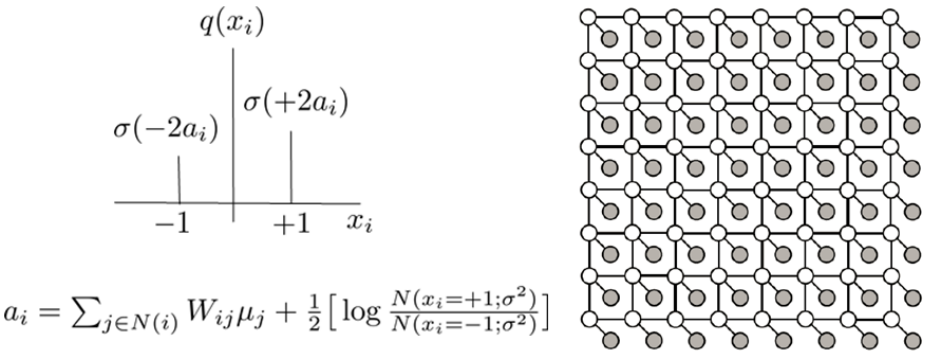

Figure 26: Mean-field VI approximation (left) and Ising model (right).

In order to compute this mean, we need to know the values of $q_j(x_j = +1)$ and $q_j(x_j = -1)$. Let $m_i = \sum_{j \in N(i)} w_{ij}\mu_j$ be the mean value of neighbors and let $L_i^+ = N(x_i = +1, \sigma^2)$ and $L_i^- = N(x_i = -1, \sigma^2)$, then we can compute the mean as follows:

$$q_i(x_i = +1) = \frac{\exp\{m_i + L_i^+\}}{\exp\{m_i + L_i^+\} + \exp\{-m_i + L_i^-\}} \tag{178}$$

$$= \frac{1}{1 + \exp\{-2m_i + L_i^- - L_i^+\}} = \frac{1}{1 + \exp\{-2a_i\}} = \sigma(2a_i) \tag{179}$$

where $a_i = m_i + 1/2(L_i^+ - L_i^-)$ and $\sigma(x)$ is a sigmoid function. Since $q_i(x_i = -1) = 1 - q_i(x_i = +1) = 1 - \sigma(2a_i) = \sigma(-2a_i)$, we can write the mean of our variational approximation $q_i(x_i)$ as follows:

$$\mu_i = E_{q_i}[x_i] = \sigma(2a_i) - \sigma(-2a_i) = \tanh(a_i) \tag{180}$$

In other words, our mean-field updates of the variational parameters μ_i at iteration k are computed as follows:

$$\mu_i^{(k)} = \tanh\left(\sum_{j \in N(i)} w_{ij}\mu_j^{(k-1)} + \frac{1}{2}\left[\log\frac{N(x_i=+1,\sigma^2)}{N(x_i=-1,\sigma^2)}\right]\right) \times \lambda + (1-\lambda) \times \mu_i^{(k-1)} \tag{181}$$

where we added a learning rate parameter $\lambda \in (0, 1]$. We further note that we can simplify the computation of ELBO term by term as follows:

$$\sum_{(s,t)\in E} E_{q(x)}[x_s w_{st} x_t] = \frac{1}{2}\sum_{i=1}^{n}\sum_{j\in N(i)}\left(\sum_{x_i\in\{-1,+1\}}\sum_{x_j\in\{-1,+1\}} q_i(x_i)q_j(x_j)x_i J x_j\right) = \tag{182}$$

$$\frac{1}{2}\sum_{i=1}^{n}\sum_{j\in N(i)}\Big(q_i(x_i=+1)JE[x_j] - q_i(x_i=-1)JE[x_j]\Big) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j\in N(i)} E[x_i]JE[x_j] \tag{183}$$

Similarly,

$$E_{q(x)}[\log N(x_i,\sigma^2)] = \sum_{i=1}^{n}\left[\sum_{x_i\in\{-1,+1\}} q_i(x_i)\log N(x_i,\sigma^2)\right] = \tag{184}$$

$$\sum_{i=1}^{n}\Big[\sigma(2a_i)\log N(x_i=+1,\sigma^2) + \sigma(-2a_i)\log N(x_i=-1,\sigma^2)\Big] \tag{185}$$

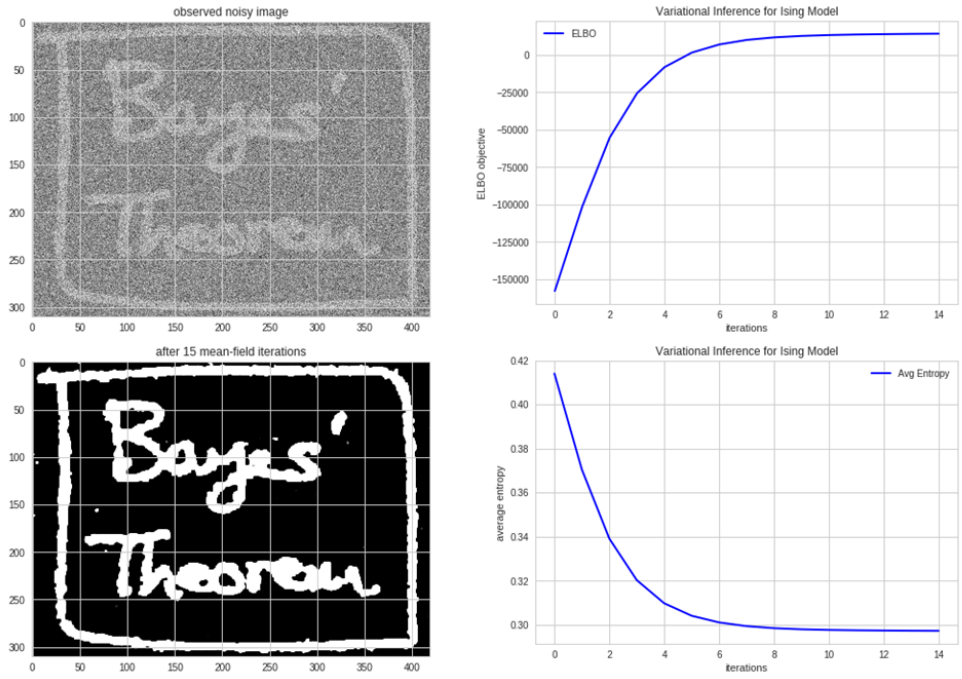

Figure 27: Image Denoising for Ising model.

Figure 27 shows experimental results for binary image denoising via mean-field variational inference. The noisy observed image is shown in the top-left obtained by adding Gaussian noise to each pixel and binarizing the image based on mean threshold. We then set the variational inference parameters such as the coupling strength $J = 1$, noise level $\sigma = 2$, smoothing rate $\lambda = 0.5$ and max number of iterations to 15. The resulting de-noised binary image is shown in

the bottom-left corner of Figure 27. We can also see an increase in the ELBO objective (top-right) and a decrease in the average entropy of our binary random variables $q_i(x_i)$ representing the value of each pixel (bottom-right) as the number of mean-field iterations increases.

The 2-D Ising model can be extended in multiple ways, for example: 3-D grids and K-states per node (aka Potts model).

3.4 Stochastic Variational Inference

One limitation of LDA is that the number of topics is fixed ahead of time. A commonly used approach to finding the number of topics K is cross-validation. However, for very large data-sets this approach may not be practical. We can address this issue with a Bayesian non-parametric topic model where the number of topics is learned from data: the Hierarchical Dirichlet Process (HDP) topic model.

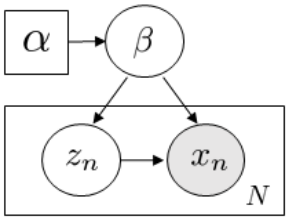

Figure 28: Graphical model with local and global latent variables.

While tranditional algorithms require repeatedly analyzing the whole data set before updating the variational parameters, we focus on analyzing randomly sampled subsets. First, we derive the Stochastic Variational Inference (SVI) algorithm for a class of graphical models with global and local laten variables as shown in Figure 28. The join distribution factorizes into a product of a global term β and local term z_n:

$$p(x, z, \beta|\alpha) = p(\beta|\alpha) \prod_{n=1}^{N} p(x_n, z_n|\beta) \tag{186}$$

Our goal is to approximate the posterior distribution of the hidden variables given the observations $p(\beta, z|x)$. We use a variational distribution to approximate the posterior as measured by KL divergence. Variational inference minimizes KL divergence or alternatively maximizes the evidence lower bound (ELBO):

$$\begin{aligned}
\log p(x) &= \log \int p(x, z, \beta) dz d\beta && (187)\\
&= \log \int p(x, z, \beta) \frac{q(z, \beta)}{q(z, \beta)} dz d\beta && (188)\\
&= \log \left(E_q \left[\frac{p(x, z, \beta)}{q(z, \beta)} \right] \right) && (189)\\
&\geq E_q[\log p(x, z, \beta)] - E_q[\log q(z, \beta)] && (190)\\
&= L(q) && (191)
\end{aligned}$$

The ELBO contains two terms: the first term is the expected log joint $E_q[\log p(x, z, \beta)]$ and the second term is the entropy of the variational distribution $-E_q[\log q(z, \beta)]$. We restrict $q(z, \beta)$ to be in a tractable family of distributions in order to efficiently compute the expectations in the ELBO. We then find the member of the family that maximizes the ELBO and use the optimized distribution as a proxy for the posterior. Solving this maximization problem is equivalent to finding the member of the family that is closest in KL divergence to the posterior:

$$\begin{aligned} \min KL(q(z, \beta)||p(z, \beta|x)) &= E_q[\log q(z, \beta)] - E_q[\log p(z, \beta|x)] && (192) \\ &= E_q[\log q(z, \beta)] - E_q[\log p(x, z, \beta)] + \log p(x) && (193) \\ &= -L(q) + \text{const} && (194) \end{aligned}$$

where $\log p(x)$ is replaces by a constant because it does not depend on q. The simplest variational family of distributions is the fully factored mean-field family. In this family, each hidden variable is independent and governed by its own parameter:

$$q(z, \beta) = q(\beta|\lambda) \prod_{n=1}^{N} \prod_{j=1}^{J} q(z_{nj}|\phi_{nj}) \tag{195}$$

To specify the form of the distribution, we choose $q(\beta|\lambda)$ and $q(z_{nj}|\phi_{nj})$ to be in the exponential distribution with the natural parameters λ and ϕ_{nj}:

$$\begin{aligned} q(\beta|\lambda) &= h(\beta) \exp\{\lambda^T t(\beta) - a_g(\lambda)\} && (196) \\ q(z_{nj}|\phi_{nj}) &= h(z_{nj}) \exp\{\phi_{nj}^T t(z_{nj} - a_l(\phi_{nj}))\} && (197) \end{aligned}$$

The mean-field family has several computational advantages. For example the entropy term in the ELBO objective function decomposes:

$$-E_q[\log q(z, \beta)] = -E_\lambda[\log q(\beta)] - \sum_{n=1}^{N} \sum_{j=1}^{J} E_{\phi_{nj}}[\log q(z_{nj})] \tag{198}$$

In traditional mean-field variational inference, we maximize $L(q)$ with coordinate ascent: we iteratively optimize each variational parameter while holding the other parameters fixed. Given our assumptions that each conditional is an exponential family, we can optimize each coordinate in closed form. We first derive the coordinate update for the global parameter λ. Keeping only the terms of $L(q)$ that depend on λ, we get:

$$\begin{aligned} L(\lambda) &= E_q[\log p(x, z, \beta)] - E_q[\log q(\beta)] + \text{const} && (199) \\ &= E_q[\log p(\beta|x, z)] - E_q[\log q(\beta)] + \text{const} && (200) \end{aligned}$$

where we used the factorization $p(x, z, \beta) = p(\beta|x, z)p(x, z)$ and absorbed $E_q[p(x, z)]$ into the constant that does not depend on λ. To derive the coordinate ascent update, we take the gradient [22]:

$$\nabla_\lambda L = \nabla_\lambda^2 a_g(\lambda)(E_q[\eta_g(x, z, \alpha)] - \lambda) \tag{201}$$

We can set the gradient to zero by setting $\lambda = E_q[\eta_g(x, z, \alpha)]$. This sets the global variational parameter equal to the expected natural parameter of its complete conditional distribution. We now turn to the local parameters ϕ_{nj}. The gradient is similar to the global case:

$$\nabla_{\phi_{nj}} L = \nabla_{\phi_{nj}}^2 a_l(\phi_{nj})(E_q[\eta_l(x_n, z_{n,-j}, \beta)] - \phi_{nj}) \tag{202}$$

Algorithm 4 Coordinate Ascent SVI [22]

Initialize $\lambda^{(0)}$ randomly
repeat
  for each local variational parameter ϕ_{nj} **do**
    Update $\phi_{nj}^{(t)} = E_{q^{(t-1)}}[\eta_{l,j}(x_n, z_{n,-j}, \beta)]$
  end for
  Update the global variational parameters: $\lambda^{(t)} = E_{q^{(t)}}[\eta_g(z, x)]$
until the ELBO converges

We can set the gradient to zero by choosing $\phi_{nj} = E_q[\eta_l(x_n, z_{n,-j}, \beta)]$. The variational updates form the algorithm for coordinate ascent variational inference, iterating between updates of each local parameter and the global parameter. The full algorithm is described in Algorithm 4 which is guaranteed to find a local optimum of the ELBO.

We now turn the Hierarchical Dirichlet Process (HDP) topic model. The HDP topic model couples a set of document-level DPs via a single top-level DP [47]. The base distribution H of the top-level DP is a symmetric Dirichlet over the vocabulary simplex. We draw once from this DP: $G_0 \sim DP(\omega, H)$. In the second level, we use G_0 as a base measure for a document level DP: $G_d \sim DP(\alpha, G_0)$. As a result, the global topics are shared between documents with different mixing proportions.

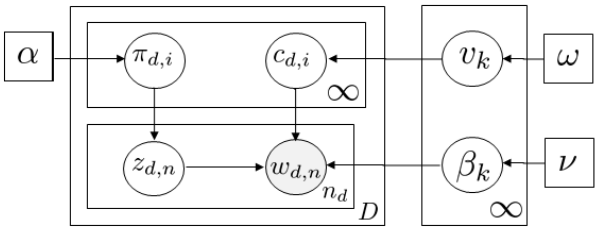

Figure 29: HDP graphical model for SVI inference.

Figure 29 shows a graphical model for the HDP topic model. The generative process of the HDP topic model can be described as follows.

1. Draw global topics, $\beta_k \sim \text{Dir}(\eta)$
2. Draw mixing proportions, $\nu_k \sim \text{Beta}(1, \omega)$
3. For each document d:
   (a) Draw document-level topic indices, $c_{di} \sim \text{Mult}(\nu)$
   (b) Draw document-level mixing proportions, $\pi_{di} \sim \text{Beta}(1, \alpha)$
   (c) For each word n:
      i. Draw topic assignment $z_{dn} \sim \text{Mult}(\pi_d)$
      ii. Draw word $w_n \sim \text{Mult}(\beta_{c_{d,z_{dn}}})$

In order to implement an infinite number of topics at corpus and document levels we need to truncate our representation to K topics at the corpus level and T topics at the document level.

This ways we are optimizing a finite number of variational parameters. We can write down the joint variational family distribution as:

$$q(\beta, \nu, z, \pi) = \left(\prod_{k=1}^{K} q(\beta_k|\lambda_k) q(\nu_k|a_k) \right) \left(\prod_{d=1}^{D} \prod_{i=1}^{T} q(c_{di}|\zeta_{di}) q(\pi_{di}|\gamma_{di}) \prod_{n=1}^{N} q(z_{dn}|\phi_{dn}) \right) \tag{203}$$

The Stochastic Variational Inference (SVI) algorithm is summarized in Algorithm 5.

Algorithm 5 Stochastic Variational Inference for HDP [22]

Initialize $\lambda^{(0)}$ randomly. Set $a^{(0)} = 1$ and $b^{(0)} = \omega$
Set the step-size schedule ρ_t
repeat
  Sample a document w_d uniformly from the dataset
  For $i \in \{1, ..., T\}$, $k \in \{1, ..., K\}$: $\zeta_{di}^k \propto \exp\{\sum_{n=1}^{N} E[\log \beta_{k,w_{dn}}]\}$
  For $n \in \{1, ..., N\}$, $i \in \{1, ..., T\}$: $\phi_{dn}^i \propto \exp\{\sum_{k=1}^{K} \zeta_{di}^k E[\log \beta_{k,w_{dn}}]\}$
  repeat
    For $i \in \{1, ..., T\}$ set
      $\gamma_{di}^{(1)} = 1 + \sum_{n=1}^{N} \phi_{dn}^i$
      $\gamma_{di}^{(2)} = \alpha + \sum_{n=1}^{N} \sum_{j=i+1}^{T} \phi_{dn}^j$
      $\zeta_{di}^k \propto \exp\{E[\log \sigma_k(V)] + \sum_{n=1}^{N} \phi_{dn}^i E[\log \beta_{k,w_{dn}}]\}$, $k \in \{1, ..., K\}$
    For $n \in \{1, ..., N\}$ set
      $\phi_{dn}^i \propto \exp\{E[\log \sigma_i(\pi_d)] + \sum_{k=1}^{K} \zeta_{di}^k E[\log \beta_{k,w_{dn}}]\}$, $i \in \{1, ..., T\}$
  until local parameters converge
  For $k \in \{1, ..., K\}$ set intermediate topics:
    $\hat{\lambda}_{kv} = \eta + D \sum_{i=1}^{T} \zeta_{di}^k \sum_{n=1}^{N} \phi_{dn}^i w_{dn}$
    $\hat{a}_k = 1 + D \sum_{i=1}^{T} \zeta_{di}^k$
    $\hat{b}_k = \omega + D \sum_{i=1}^{T} \sum_{l=k+1}^{K} \zeta_{di}^l$
  Set
    $\lambda^{(t)} = (1 - \rho_t)\lambda^{(t-1)} + \rho_t \hat{\lambda}$
    $a^{(t)} = (1 - \rho_t) a^{(t-1)} + \rho_t \hat{a}$
    $b^{(t)} = (1 - \rho_t) b^{(t-1)} + \rho_t \hat{b}$
until end of documents

where the expectations used in Algorithm 5 are computed as follows:

$$E[\log \beta_{kv}] = \Psi(\lambda_{kv}) - \Psi(\sum_{v'} \lambda_{kv'}) \tag{204}$$

$$E[\log \sigma_k(V)] = \Psi(a_k) - \Psi(a_k + b_k) + \sum_{l=1}^{k-1} [\Psi(b_l) - \Psi(a_l + b_l)] \tag{205}$$

The on-line variational inference algorithm for Hierarchical Dirichlet Process was used to fit a topic model on the 20newsgroups dataset. The dataset consists of $11,314$ documents and over $100K$ unique tokens. Standard text pre-processing was used including tokenization, stop-word removal and stemming. A compressed dictionary of $4K$ words was constructed by filtering out tokens that appear in less than 5 documents and more than 50% of the corpus. The top-level truncation was set to $T = 20$ topics and the second level truncation was set to $K = 8$ topics. The concentration parameters were chosen as $\gamma = 1.0$ at the top-level and $\alpha = 0.1$ at the group level to yield a broad range of shared topics that are concentrated at the group level. Figure 30

shows a sample of the global level topics inferred by online variational HDP algorithm. We can find topics about autos, politics and for sale items that correspond to the target labels of the 20newsgroups dataset.

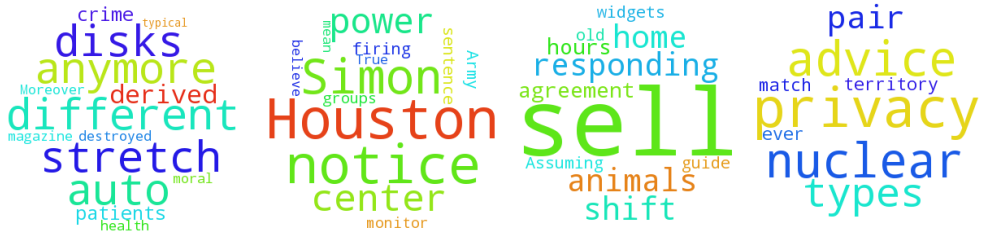

Figure 30: Sample of HDP topics inferred using online variational bayes algorithm on 20newsgroups dataset.

3.5 Normalizing Flows

A normalizing flow describes the transformation of a probability density through a sequence of invertible mappings [38]. The target density is approximated through a flow of the prior distribution through a sequence of invertible (one-to-one) transformations resulting in the target posterior density.

The basic rule for transformation of densities considers an invertible, smooth mapping $f : \mathbb{R}^d \to \mathbb{R}^d$. Recall, that in the case of discrete random variables, for a one-to-one $y = f(x)$ where $X \sim p(x)$, the transformed density is $p_y(y) = \sum_{\{x:y=f(x)\}} p_x(x)$. If X is a continuous random variable, we can differentiate the CDF to obtain an expression for the transformed density:

$$F_y(y) = P(Y \leq y) = P(f(X) \leq y) = P(X \leq f^{-1}(y)) \tag{206}$$

$$p_y(y) = \frac{d}{dy}F_y(y) = \frac{d}{dy}P(X \leq f^{-1}(y)) = \frac{dx}{dy}\frac{d}{dx}P(X \leq f^{-1}(y)) = p_x(x)|\frac{dx}{dy}| \tag{207}$$

The above expression generalizes for high dimensional transformations using a Jacobian:

$$p_y(y) = p_x(x)|\det\left(\frac{\partial x}{\partial y}\right)| \tag{208}$$

Let's examine a k step transformation:

$$z_k = f_k \circ f_{k-1} \circ \cdots \circ f_2 \circ f_1(z_0) = f_k(f_{k-1}(\cdots f_2(f_1(z_0)))) \tag{209}$$

For $k = 1$, we have $z_1 = f_1(z_0)$, where $z_0 \sim q_0(z_0)$ and

$$q_1(z_1) = q_0(z_0)|\det\left(\frac{\partial z_0}{\partial z_1}\right)| = q_0(z_0)|\det\left(\frac{\partial f_1^{-1}}{\partial z_1}\right)| = q_0(z_0)|\det\left(\frac{\partial f_1}{\partial z_0}\right)|^{-1} \tag{210}$$

For $k = 2$, we have $z_2 = f_2 \circ f_1(z_0) = f_2(f_1(z_0)) = f_2(z_1)$ that we can re-write as:

$$q_2(z_2) = q_1(z_1)|\det\left(\frac{\partial f_2}{\partial z_1}\right)|^{-1} = q_0(z_0)|\det\left(\frac{\partial f_1}{\partial z_0}\right)|^{-1} \times |\det\left(\frac{\partial f_2}{\partial z_1}\right)|^{-1} \tag{211}$$

Generalizing for k steps, we have:

$$q_k(z_k) = q_0(z_0)|\det\left(\frac{\partial f_1}{\partial z_0}\right)|^{-1} \times |\det\left(\frac{\partial f_2}{\partial z_1}\right)|^{-1} \times \cdots \times |\det\left(\frac{\partial f_k}{\partial z_{k-1}}\right)|^{-1} \tag{212}$$

Taking the log, we can represent this compactly as:

$$\log q(z_k) = \log q(z_0) - \sum_{i=1}^{K} \log |\det \Big(\frac{\partial f_i}{\partial z_{i-1}} \Big)| \tag{213}$$

Note the above density is parametrized by one-to-one, invertible transformations $f_k(\cdot)$. In order to learn the flow parameters, we consider the following variational inference objective.

$$\log p(x) \geq E_q \big[\log \frac{p(x,z)}{q(z|x)} \big] = \text{ELBO} = -F(x) \tag{214}$$

Thus, we are interested in maximizing the ELBO or equivalently minimizing the free energy $F(x)$. Treating $F(x)$ as a loss function, we can expand it as follows:

$$\begin{aligned} \min F(x) &= E_{q(z|x)} \Big[\log \frac{q(z|x)}{p(x,z)} \Big] = E_{q_0(z_0)} \Big[\log q_k(z_k) - \log p(x, z_k) \Big] \\ &= E_{q_0(z_0)} \Big[\log q_0(z_0) \Big] - E_{q_0(z_0)} \Big[\sum_{k=1}^{K} \log |\det \Big(\frac{\partial f_i}{\partial z_{i-1}} \Big)| \Big] - E_{q_0(z_0)} \Big[\log p(x, z_k) \Big] \end{aligned} \tag{215}$$

where $E_{q_0(z_0)}$ is Monte Carlo expectation with respect to the prior distribution $q_0(z)$. Note that we can ignore the first term in the objective function because it doesn't depend on the flow paramters. Also, we swapped the expectation wrt to $q(z)$ to an expectation with respect to the prior $q_0(z)$, this is true in general for normalizing flows since we can fix the flow parameters and can compute expectations with respect to any of the intermediate distributions by drawing samples from the prior:

$$E_{q_k}[h(z)] = \int q_k(z) h(z) dz = \frac{1}{n} \sum_{i=1}^{n} h(z_i^k), \quad \text{where} \quad z_i^k \sim q_k(z) \tag{216}$$

$$= \frac{1}{n} \sum_{i=1}^{n} h(f_k \circ f_{k-1} \circ \cdots \circ f_1(z_i^0)), \quad \text{where} \quad z_i^0 \sim q_0(z) \tag{217}$$

$$= E_{q_0}[h(f_k \circ f_{k-1} \circ \cdots \circ f_1(z_i^0))] \tag{218}$$

For NF inference, we need to identify a class of invertible transformations and efficient algorithms for computing determinant of the Jacobian. *Planar* and *radial* flows were designed in a way that makes their determinant of Jacobian easy to compute. Other extensions include auto-regressive flows. A planar transform can be defined as follows:

$$f(z) = z + uh(w^t z + b) \tag{219}$$

parameterized by $\lambda = \{w \in \mathbb{R}^d, u \in \mathbb{R}^d, b \in \mathbb{R}\}$, where h is a smooth, point-wise non-linearity (such as tanh) with derivative h'. For a planar flow of this form, we can compute the logdet Jacobian term in $O(D)$ time:

$$|\det \frac{\partial f}{\partial z}| = |\det(I + uh'(w^t z + b)w^t)| = |1 + u^t h'(w^t z + b)w| \tag{220}$$

As a result, we are interested in optimizing the non-convex objective:

$$\min_{\{w_k, u_k, b_k\}} -E_{q_0} \Big[\sum_{k=1}^{K} \log |1 + u_k^t h'(w_k^t z_{k-1} + b_k) w_k| \Big] - E_{q_0} \Big[\log p(x, z_k) \Big] \tag{221}$$

Alternatively, we can consider a family of transformations that modify an initial density q_0 around a reference point z_0, known as a radial flow:

$$f(z) = z + \beta h(\alpha, r)(z - z_0) \tag{222}$$

where $r = |z - z_0|$ and $h(\alpha, r) = 1/(\alpha + r)$ and the parameters of the map are $\lambda = \{z_0 \in \mathbb{R}^d, \alpha \in \mathbb{R}^+, \beta \in \mathbb{R}\}$. This family also allows for linear computation of the determinant:

$$|\det \frac{\partial f}{\partial z}| = [1 + \beta h(\alpha, r)]^{d-1}[1 + \beta h(\alpha, r) + \beta h'(\alpha, r)r] \tag{223}$$

The effect of invertible flows can be understood as a sequence of expansions or contractions on the initial density. The resuling distribution is transformed by increasing density outside its original region (expansion) or concentrating in its original region (contraction). With an appropriate choice of transformation f_k we can use a simple prior distribution such as factorized Gaussian and by applying normalizing flows of different lengths, we obtain increasingly complex and multi-modal distributions. One question one may ask is how to choose K the length of the flow that invites a non-parametric way of looking at normalizing flows.

3.5.1 Annealed NF

Annealed Normalizing Flows explores the hypothesis that if we don't take the steepest descent path but take one guided by AIS, we can arrive at a better solution (non-greedy). The annealed NF algorithm is summarized in Algorithm 6

Algorithm 6 Annealed Normalizing Flows

1: Initialize ϕ randomly
2: **for** $t = 1$ to T **do**
3: $x_0 \sim N(0, I)$ //sample a mini-batch of data of size m
4: $x_k = f_k \circ f_{k-1} \circ \cdots \circ f_1(x_0)$ //NF forward pass
5: $\beta_t = \beta(t, a) = 1 - \exp\big(-at\big)$ //update beta
6: $\log p_t(x_k) = (1 - \beta_t) \log N(0, I) + \beta_t \log p(x_k)$ //compute annealed target distribution
7: $\log q(x_k; \phi) = \log q(z_0) - \sum_{i=1}^{K} \log |\det \left(\frac{\partial f_i}{\partial x_{i-1}} \right)|$ //compute variational approximation
8: $\text{Loss}(x_k) = E_{q0}\Big[\log q(x_k; \phi) - \log p_t(x_k)\Big]$ // compute variational loss (negative ELBO)
9: $\phi = \phi - \epsilon_t \times g$, where $g = \frac{1}{m} \sum_{i=1}^{m} \nabla_\phi \text{Loss}(x_k^{(i)}; \phi)$ //update NF parameters (backward pass)
10: **end for**

Geometric average of the prior and target distributions used to create an intermediate $p_t(x_k)$ minimizes a weighted sum of the KL divergences to the prior p_a and target p_b:

$$p_t(x_k) = \arg\min_q (1 - \beta) D_{KL}(q||p_a) + \beta D_{KL}(q||p_b) \tag{224}$$

$$\text{s.t.} \sum_x q(x) = 1 \tag{225}$$

The objective function above is minimized by $p_t(x_k)$ that allocates mass in the intersection of p_a and p_b (due to reverse KL), i.e. $p_t(x_k)$ is high in regions that are likely under both the prior

and the target distributions as shown in Figure 31. To see why the expression above is true we can include the constraints as part of the objective and optimize the Lagrangian:

$$\begin{aligned} L(q) &= \lambda(1 - \sum_x q(x)) + (1-\beta)\sum_x q(x)\big[\log q(x) - \log p_a(x)\big] \\ &+ \beta \sum_x q(x)\big[\log q(x) - \log p_b(x)\big] \quad (226) \\ &= \lambda(1 - \sum_x q(x)) + \sum_x q(x)\log q(x) \\ &- \sum_x q(x)\big[(1-\beta)\log p_a(x) + \beta \log p_b(x)\big] \quad (227) \end{aligned}$$

By taking the derivative of $L(q)$ and setting it to zero, we get:

$$\begin{aligned} \frac{dL(q)}{dq(x)} &= -\lambda + \log q(x) + 1 - \big[(1-\beta)\log p_a(x) + \beta \log p_b(x)\big] = 0 \quad (228) \\ \log q(x) &= \lambda - 1 + \log p_a^{1-\beta}(x) + \log p_b^{\beta}(x) \quad (229) \\ q(x) &\propto p_a^{1-\beta}(x) p_b^{\beta}(x) \quad (230) \end{aligned}$$

for some constant λ, s.t. $\sum_x q(x) = 1$. If p_a and p_b belong to an exponential family P with natural parameters θ_{p_a} and θ_{p_b}, the optimum distribution is also in exponential family with natural parameters $\theta_\beta = (1-\beta)\theta_{p_a} + \beta\theta_{p_b}$.

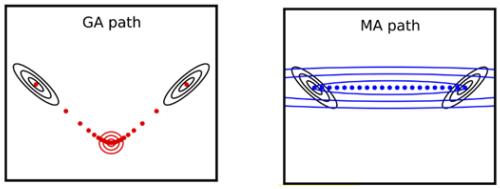

Figure 31: Geometric Average (GA) path (left) and Moment Average (MA) path (right). Figure source: [19].

If instead we minimize the forward KL objective with respect to natural parameters θ of the approximating distribution, we obtain the Moment Average (MA) path [19].

$$\arg\min_\theta (1-\beta) D_{KL}(p_a || q(x;\theta)) + \beta D_{KL}(p_b || q(x;\theta)) \quad (231)$$

We can re-write the objective as follows:

$$\begin{aligned} J(\theta) &= (1-\beta)\sum_x p_a(x)\big[\log p_a(x) - \log q(x;\theta)\big] \\ &+ \beta \sum_x p_b(x)\big[\log p_b(x) - \log q(x;\theta)\big] \quad (232) \\ &= \text{const} - \sum_x \big[(1-\beta)p_a(x) + \beta p_b(x)\big]\log q(x) \quad (233) \\ &= \text{const}' + \log Z(\theta) - \sum_x \big[(1-\beta)p_a(x) + \beta p_b(x)\big]\theta^T \phi(x) \quad (234) \end{aligned}$$

By setting the derivative to zero, we get:

$$\begin{aligned} \frac{\partial J(\theta)}{\partial \theta_i} &= E_q[\phi_i(x)] - \sum_x \big[(1-\beta)p_a(x) + \beta p_b(x)\big]\phi_i(x) = 0 \quad (235) \\ E_q[\phi_i(x)] &= (1-\beta)E_{p_a}[\phi_i(x)] + \beta E_{p_b}[\phi_i(x)] \quad (236) \end{aligned}$$

The above expression can be interpreted as drawing $(1-\beta)$ fraction of points from p_a and β fraction of points from p_b. Notice that because we are optimizing the forward KL, the approximating distribution includes the support of both p_a and p_b as shown in Figure 31.

3.5.2 NF experiments

To evaluate how well NF can approximate posterior densities, we designed the following experiment. Let $p(x) \propto \exp\{-E(x)\}$ be our target distribution with energy $E(x)$ defined as follows:

$$E(x) = \frac{1}{2}\left(\frac{||z|| - 2}{0.4}\right)^2 - \log\left(e^{-\frac{1}{2}\left[\frac{z_1-2}{0.6}\right]^2} + e^{-\frac{1}{2}\left[\frac{z_1+2}{0.6}\right]^2}\right) \tag{237}$$

We restrict ourselves to K planar flows and define the training epoch to be 512 iterations with a mini-batch size of 512 samples drawn from standard normal prior. The NF algorithm is implemented in PyTorch using CUDA GPU.

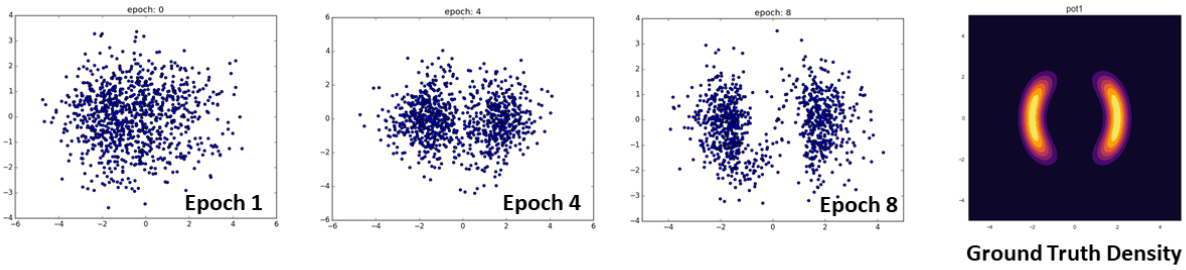

Figure 32: Samples from NF approximation to ground truth density.

Figure 32 shows samples drawn from NF model trained to minimize the free energy loss $F(x)$:

$$\log p(x) \geq E_q\left[\log \frac{p(x,z)}{q(z|x)}\right] = \text{ELBO} = -F(x) \tag{238}$$

We can see that as the number of iterations progresses the NF approximations starts to resemble the ground truth density.

Since the ground truth density is 2 dimensional, we can compute the ground truth value of $\log p(x)$ using numerical integration as well as Annealed Importance Sampling (AIS).

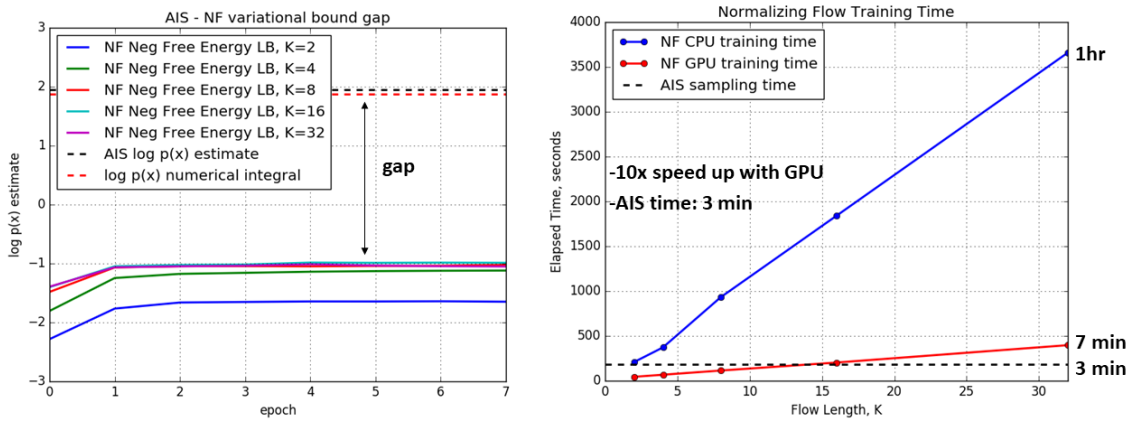

Figure 33: NF variational objective gap (left). NF timing experiments (right).

Figure 33 (left) shows the variational objective gap between the negative free energy and the ground truth value estimated by numerical integration:

$$\log p(x) = \text{ELBO} + D_{KL}(q(z)||p(z|x)) \tag{239}$$

We can see that as we increase the number of planar flows K, we get a more accurate representation of the posterior. Timing experiments in Figure 33 (right) show that GPU provides about $10x$ speed up in comparison with CPU and linear increase with the number of flows K.

In addition, we can estimate the distance between samples from NF and ground truth distributions using Wasserstein distance with entropy regularization:

$$\begin{aligned} d_M(a,b) &= \min_P \langle P, M \rangle_F - \lambda H(P) && (240) \\ \text{s.t.} \quad & Pe = a \\ & P^T e = b \\ & P_{ij} \geq 0 && (241) \end{aligned}$$

where P is the transport matrix and p_{ij} is the amount of mass moved from point i in distribution a to point j in distribution b. P is doubly stochastic and non-negative. M is the cost matrix, a pair-wise distance matrix between samples from distribution a and samples from distribution b. The distribution a consists of 1024 samples from trained NF network and distribution b consists of 1024 samples based on grid-discretized ground truth density. Note that the entropy term makes P less sparse, while we want to preserve sparsity (i.e. avoid having to move many points in a go to many points in b). However, this formulation leads to efficient matrix scaling algorithm based on Sinkhorn iterations [9].

Figure 34 shows the affect of the prior covariance on samples from NF distribution. We set the prior to $N([4,0]^T, \Sigma)$ where in one case $\Sigma = I$ (left) and int he second case $\Sigma = 10I$ (right). We compute Wasserstein distances $d_M(a,b)$ between the NF samples and the ground truth in each case. We can see that for a more concentrated prior, only one mode of the target distribution is discovered with $d_M(a,b) = 0.069$, while in the case of $\Sigma = 10I$, both modes are discovered and $d_M(a,b) = 0.0087$.

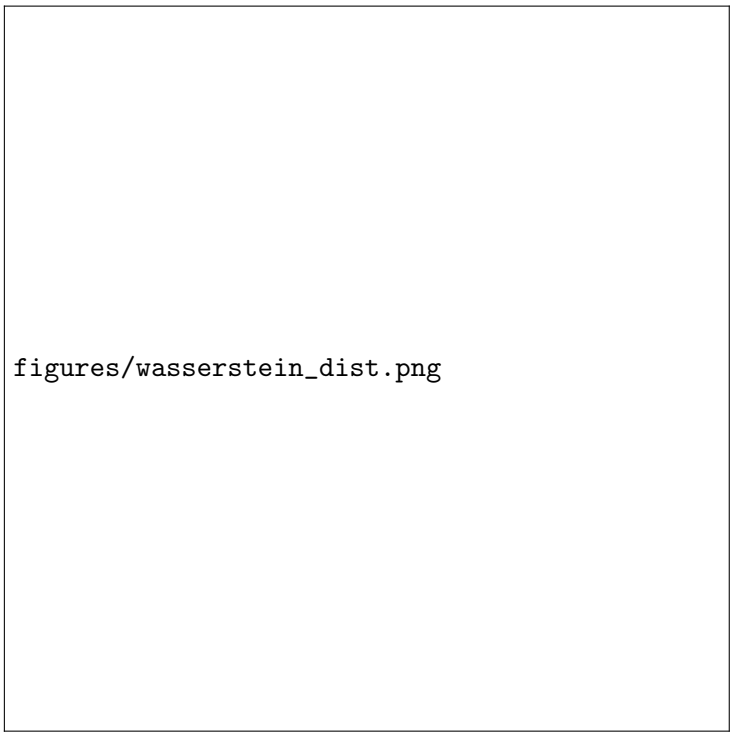

Figure 34: Impact of prior distribution $N([4,0]^T, \Sigma)$. Left: $\Sigma = I$, only one mode of the target distribution is discovered with $d_M(a,b) = 0.069$. Right: $\Sigma = 10I$, both modes are recovered with $d_M(a,b) = 0.0087$

4 Optimization

4.1 Natural Gradient Descent

Let's define the score function $s(\theta) = \nabla_\theta \log p(x|\theta)$, the Fisher information matrix is given by $I(\theta) = -E\Big[\nabla^2_\theta \log p(x|\theta)\Big]$ or in the matrix form:

$$\Big[I(\theta)\Big]_{ij} = -E\Big[\frac{\partial^2}{\partial\theta_i \partial\theta_j} \log p(x|\theta)\Big] \tag{242}$$

The second order Taylor expansion of KL divergence is

$$KL(p_\theta||p_{\theta+d}) = KL(p_\theta||p_\theta) - E_{p(x|\theta)}[\nabla_\theta \log p(x|\theta)]^T d + \frac{1}{2} d^T I(\theta) d \approx \frac{1}{2} d^T I(\theta) d \tag{243}$$

We would like to find a vector d that minimizes the loss function $L(\theta)$:

$$d^* = \arg \min_{ds.t. KL(p_\theta||p_{\theta+d})=c} L(\theta + d) \tag{244}$$

We can re-write the above constrained problem as a Lagrangian:

$$\begin{aligned} d^* &= \arg\min_d L(\theta + d) + \lambda(KL(p_\theta||p_{\theta+d}) - c) && (245) \\ &\approx \arg\min L(\theta) + \nabla_\theta L(\theta)^T d + \frac{1}{2} d^T I(\theta) d - \lambda c && (246) \end{aligned}$$

Setting the derivative wrt to d to zero, we get:

$$0 = \nabla_\theta L(\theta) + \lambda I(\theta) d \tag{247}$$

$$d = -\frac{1}{\lambda} I(\theta)^{-1} \nabla_\theta L(\theta) \tag{248}$$

Thus, our optimal d is in the opposite direction of the gradient preconditioned by inverse of Fisher information matrix. The natural gradient is defined as

$$\tilde{\nabla}_\theta L(\theta) = I(\theta)^{-1} \nabla_\theta L(\theta) \tag{249}$$

It's common to use Taylor series expansion of the objective function. We can define two important properties that characterize the speed of convergence: α-strong convexity and β-smoothness. Let $\delta = x - x_0$, then 2nd order Taylor expansion is:

$$f(x+\delta) = f(x) + \nabla^T f(x)\delta + \frac{1}{2}\delta^T H(x)\delta \tag{250}$$

By subtracting the linear approximation from the original $f(x)$, we define the following lower and upper bounds:

$$\frac{\alpha}{2}||\delta||_2^2 \leq f(x+\delta) - \Big[f(x) + \nabla^T f(x)\delta\Big] \leq \frac{\beta}{2}||\delta||_2^2 \tag{251}$$

The coefficient α bounds the degree of flatness of our objective function $f(x)$ and is related to amount of progress gradient descent can make along the objective. The coefficient β bounds the degree of smoothness a function can have (e.g. think of the corner in the absolute value function) Together, α-strong convexity and β-smoothness characterize the speed of convergence of gradient descent.

4.2 Non-Convex Optimization

Non-convex problems are NP hard to solve and in general no algorithmic technique is expected to succeed at finding global optima solutions to these problems. However, non-convex problems that have a certain structure can be solved in polynomial time, such as projections on non-convex constraints and structural properties of the objective funcions (e.g. marginal convexity) [23]

4.2.1 Projected Gradient Descent

Exploiting the projected gradient descent algorithm with non-convex problems requires projections onto non-convex sets:

$$\Pi_C(z) = \arg\min_{x \in C} ||x - z||_2 \tag{252}$$

The projection problem above can itself be NP hard. However, for several well-structured sets, projections can be carried out efficiently despite the sets being non-convex. For example, projecting into sparse vectors:

$$\hat{w} = \arg\min_{||w||_0 \leq s} \sum_{i=1}^{n} \big(y_i - x_i^T w\big)^2 \tag{253}$$

The projection above can be done by sorting co-ordinates of the z vector in descending order and setting all except that top s coordinates to zero. Similarly, projections into low-rank matrices as used in recommendation systems:

$$\hat{A_{lr}} = \arg\min_{rank(X) \leq r} \sum_{(i,j) \in \Omega} \big(X_{ij} - A_{ij}\big)^2 \tag{254}$$

The projection above can be done by applying singular value decomposition of the matrix A and keeping the top r singular values and vectors. Non-convex projections are used in the generalized Projected Gradient Descent (PGD) summarized in Algorithm 7.

Algorithm 7 Generalized Projected Gradient Descent

Input: objective function f, constraint set C, step length η
Output: A point $\hat{x} \in C$ with near-optimal objective value
$x^1 \leftarrow 0$
for t = 1,2,...,T **do**
    $z^{t+1} = x^t - \eta \nabla f(x^t)$
    $x^{t+1} = \arg\min_{x \in C} ||x - z^{t+1}||$
end for
option 1: **return** $\hat{x}_{final} = x^T$
option 2: **return** $\hat{x}_{avg} = \frac{1}{T} \sum_t x^t$
option 3: **return** $\hat{x}_{best} = \arg\min_{t \in [T]} f(x^t)$

We can see two alternating steps of moving in the direction opposite to the gradient and projecting the resulting vector on our non-convex constrain set.

4.2.2 Alternating Minimization

Alternating Minimization (AM) is a widely used non-convex optimization technique. The popular K-means clustering algorithm and the EM algorithm for latent variable models are problem-specific variants of the general alternating minimization principle. The alternating minimization principle is most often used in settings where the optimization problem consists of two or more groups of variables. For example, in a collaborative filtering recommendation system, the objective function is not jointly convex, however, it is *marginally convex*:

$$\hat{A}_{lv} = \arg \min_{U \in R^{m \times r}, V \in R^{n \times r}} \sum_{(i,j) \in \Omega} \left(U_i^T V_j - A_{i,j} \right)^2 \tag{255}$$

For a group of two variables, we can write down alternative minimization as in Algorithm 8

Algorithm 8 Alternating Minimization

Input: objective function $f : X \times Y \to R$
Output: A point $\hat{x} \in X \times Y$ with near-optimal objective value
$(x^1, y^1) \leftarrow$ INIT()
for t = 1,2,...,T **do**
    $x^{t+1} = \arg\min_{x \in X} f(x, y^t)$
    $y^{t+1} = \arg\min_{y \in Y} f(x^{t+1}, y)$
end for
return (x^T, y^T)

The idea of solving several intermediate marginal optimization problems instead of a single big optimization problem is the key to practical success of alternating minimization algorithm. Algorithms similar to alternating minimization are commonly known as Coordinate Descent (CD) and are very popular for large scale convex optimization problems. The coordinate descent treats a single p dimensional variable $x \in R^p$ as p one-dimensional variables $\{x_1, ..., x_p\}$ and executes AM like steps resulting in intermediate problems being uni-dimensional. A variant

of Coordinate Descent splits variables into blocks of multi-dimensional variables and minimizes over one block of variables at each time step.

4.2.3 Stochastic Optimization Techniques

One of the main hurdles in achieving local optimality in high dimensional problems are saddle points. Saddle points are characterized by zero gradient and Hessian matrix with both positive and negative eigen values. They stall the progress of optimization methods such as gradient descent due to the vanishing gradient while not characterizing the function in any useful way in contrast to local optima. However, saddle points are unstable and a slight perturbation is likely to cause gradient descent to roll down the function surface. This motivates stochastic optimization techniques.

Saddle points can often be caused by symmetry of the objective function and they increase exponentially with the dimension of the problem. This motivates the application of Noisy Gradient Descent (NGD) Algorithm 9.

Algorithm 9 Noisy Gradient Descent

Input: objective f, constraint set C, max step length $\eta_{\max}$, tolerance ϵ
Output: A locally optimal point $\hat{x} \in R^p$
$x^1 \leftarrow$ INIT()
Set $T = 1/\eta^2$, where $\eta = \min\{\epsilon^2/\log^2(1/\epsilon), \eta_{\max}\}$
for t = 1,2,...,T **do**
    sample perturbation $\zeta^t \sim S^{p-1}$ //random point on a unit sphere
    $g^t = \nabla f(x^t) + \zeta^t$
    $x^{t+1} = \Pi_C(x^t - \eta g^t)$ //project onto constraint set
end for
return x^T

The technique of using randomly pertrubed (stochastic) gradients to escape saddle ponts is part of a more general framework of Langevin Dynamics which studies the case when the perturbations are not isotropic or are applied at a scale that adapts to the problem.

4.2.4 Simulated Annealing

Simulated Annealing (SA) is a stochastic algorithm that attempts to find the global optimum of an objective function $f(x)$. The method is inspired by statistical physics, in particular the Boltzmann distribution that specifies the probability of being in a particular state x:

$$p(x) \propto \exp\{-f(x)/T\} \tag{256}$$

where $f(x)$ is the energy of the system and T is the temperature. As the temperature approaches zero, the system spends more and more time in its minimum energy (most probable) state. As the temperature decreases, the largest peaks become larger and the smallest peaks dissappear. By cooling slowly enough, it is possible to track the largest peak and therefore find the global optimum.

Simulated annealing is closely related to the Metropolis-Hastings algorithm for generating samples from a probability distribution. At each step of the algorithm, we sample a new state according to a proposal distribution $x' \sim q(|x_k)$, such as a random walk proposal:

$$x' = x_k + \epsilon_k, \quad \text{where } \epsilon_k \sim N(0, \Sigma) \tag{257}$$

Having proposed a new state, we compute α as in Algorithm 10. Thus, if a new state has

Algorithm 10 Simulated Annealing

$\alpha = \exp\{(f(x) - f(x'))/T\}$
$r = \min(1, \alpha)$
$u \sim \text{Unif}(0, 1)$
if $u < r$
    $x_{k+1} = x'$
else
    $x_{k+1} = x_k$
end if

lower energy (higher probability), we will definitely accept it but if it has higher energy (lower probability), we might still accept it depending on the temperature. Therefore, the algorithm allows downhill moves in probability space but less frequently as the temperature drops. In practice it is common to use an exponential cooling schedule: $T_k = T_0 C^k$, where $T_0 \sim 1$ is the initial temperature and $C \sim 0.8$ is the cooling rate. Cooling too quickly can result in getting stuck in local optima, while cooling too slowly wastes time. The optimum cooling schedule is difficult to determine.

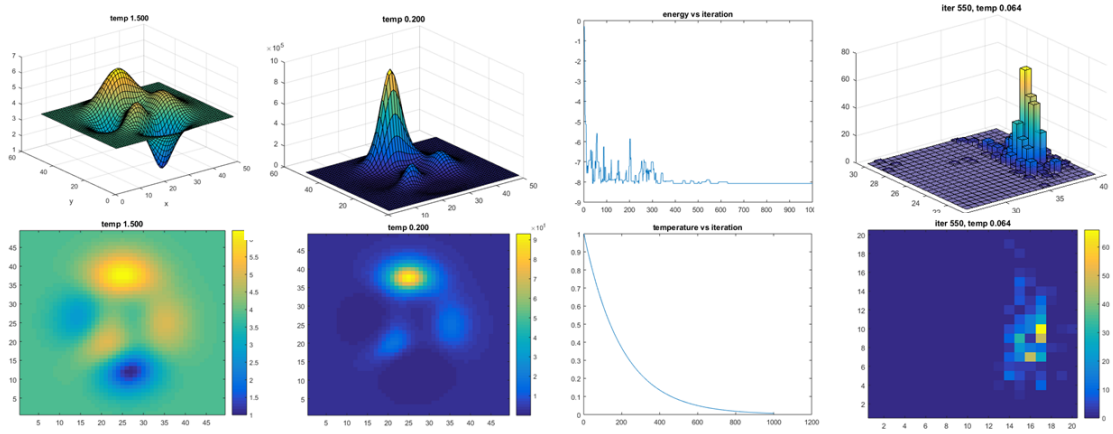

Figure 35: Simulated Annealing

Figure 35 shows the objective $f(x)$ at two different temperatures (left). The function appears more peaky when the temperature is lower. We can also see that the method stochastically reduces the energy over time for the given cooling schedule (middle). Finally a histogram of samples shows that most samples are concentrated near the global maximum (right).

4.3 Bayesian Optimization

Machine learning algorithms frequenty require careful tuning of hyperparameters. Often exhaustive and computationally expensive methods such as grid search cross validation are used to find model parameters that optimize a suitable performance objective. An alternative to grid search is randomized parameter optimization that samples parameter settings from a distribution over possible parameter values. This has two main benefits over the exhaustive search: the number of iterations can be chosen independent of the number of parameters and adding parameters that do not influence performance does not decrease efficiency. Rather than exploring the parameter space randomly (according to a chosen distribution), it would be great to adapt an active

learning approach that selects parameter values in a way that reduces uncertainty and provides a balance between exploration and exploitation. Bayesian optimization provides an automated Bayesian framework by utilizing Gaussian Processes (GPs) to model algorithm's generalization performance [45].

Bayesian optimization assumes that a suitable performance function was sampled from a Gaussian Process and maintains a posterior distribution for this function as observations are made: $f(x) \sim \text{GP}(m(x), \kappa(x, x'))$. To choose which hyperparameters to explore next, one can optimize the Expected Improvement (EI) over the current best result or the Gaussian process Upper Confidence Bound (UCB). EI and UCB have been shown to be efficient in the number of function evaluations required to find the global optimum of multi-modal black-box functions. Bayesian optimization uses all of the information available from previous evaluations of the objective function as opposed to relying on local gradient and Hessian approximations. This results in an automated procedure that can find an optimum of non-convex functions with relatively few evaluations, at the cost of performing more computation to determine the next point to try. This is particularly useful when evaluations are expensive to perform such as in selecting hyperparameters for deep neural networks.

To determine what point should be evaluated next, we need to choose and acquisition function which is used to construct a utility function from the GP posterior. In general, the acquisition function depends on previous observations as well as GP hyperparameters that we denote as $a(x; \{x_n, y_n\}, \theta)$, then $x_{next} = \arg\max_x a(x)$. Let $\mu(x; \{x_n, y_n\}, \theta)$ be the predictive GP mean function, $\sigma^2(x; \{x_n, y_n\}, \theta)$ be the predictive GP variance function and $\Phi(x)$ be the cumulative distribution function of the standard normal. Then we can define the following acquisition functions.

Probability of Improvement. This strategy maximizes the probability of improving over the best current value. This can be computed as follows:

$$a_{PI}(x; \{x_n, y_n\}, \theta) = \Phi(\gamma(x)), \qquad \gamma(x) = \frac{f(x_{best}) - \mu(x; \{x_n, y_n\}, \theta)}{\sigma(x; \{x_n, y_n\}, \theta)} \tag{258}$$

Expected Improvement. Alternatively, one could choose to maximize the expected improvement (EI) over the current best.

$$a_{EI}(x; \{x_n, y_n\}, \theta) = \sigma(x; \{x_n, y_n\}, \theta)(\gamma(x)\Phi(\gamma(x)) + N(\gamma(x); 0, 1)) \tag{259}$$

Upper Confidence Bound. UCB is the idea of exploiting upper confidence bounds to construct acquisition functions that minimize regret over the course of their optimization. These acquisition functions have the following form:

$$a_{UCB}(x; \{x_n, y_n\}, \theta) = \mu(x; \{x_n, y_n\}, \theta) - \kappa\sigma(x; \{x_n, y_n\}, \theta) \tag{260}$$

where κ is a tunable acquisition parameter to balance exploration and exploitation. The optima of acquisition functions are located where the uncertainty in GP posterior is large (exploration) and/or where the GP prediction is high (exploitation). Since acquisition functions have analytical forms that are simple to evaluate, they are easier to optimize then the original objective function.

An important practical consideration for Bayesian optimization is an appropriate choice of GP kernel and its associated hyperparameters. Squared exponential kernel is often a default

choice for Gaussian process regression. However, sample functions with this kernel are too smooth for practical optimization problems. Instead the following ARD Matern 5/2 kernel is proposed:

$$K_{M52}(x,x') = \theta_0\left(1+\sqrt{5r^2(x,x')}+\frac{5}{3}r^2(x,x')\right)\exp\{-\sqrt{5r^2(x,x')}\} \quad (261)$$

$$r^2(x,x') = \sum_{d=1}^{D}(x_d - x'_d)^2/\theta_d^2 \quad (262)$$

This kernel results in sample functions that are twice-differentiable without requiring the smoothness of the squared exponential kernel. The associated kernel hyperparameters are D length scales $\theta_{1:D}$, the covariance amplitude θ_0, the observation noise ν and a constant mean m. An additional assumption made by GP regression is that the underlying process is stationary, i.e. we can re-write the kernel $k(x,x')$ as a function of $x-x'$. Intuitively, a function whose length scale does not change throughout the input space will be modelled well by a GP with a stationary kernel.

Bayesian Optimization algorithm is summarized in Algorithm 11.

Algorithm 11 Bayesian Optimization

1: **for** $n = 1, 2, \ldots$ **do**
2: select new x_{n+1} by optimizing acquisition function α
$x_{n+1} = \arg\max_x \alpha(x; D_n, \theta)$
3: query objective function to obtain $y_{n+1} = f(x_{n+1})$
4: augment data $D_{n+1} = \{D_n, (x_{n+1}, y_{n+1})\}$
5: update GP posterior and acquisition function
6: **end for**

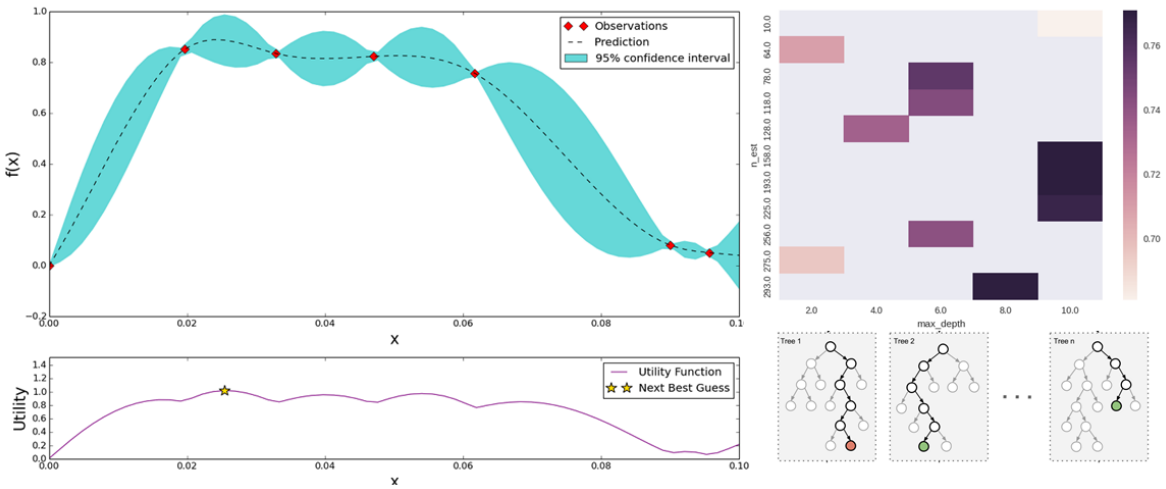

Figure 36: Bayesian Optimization of SVM and Random Forest parameters

Figure 36 shows Baeysian optimization applied to SVM and Random Forest. F1 score was used as performance objective function for a classification task. The figure on the left shows Bayesian optimization of F1 score as a function of the gamma parameter of the SVM RBF kernel: $K(x,x') = \exp\{-\gamma||x-x'||^2\}$. We can see that after only 7 iterations we have discovered the

gamma parameter that gives the maximum F1 score. The peak of EI utility function at the bottom tells us which experiment to perform next. The figure on the right shows Bayesian optimization of F1 score as a function of maximum depth and the number of estimators of a Random Forest classifier. From the heatmap, we can tell that the maximum F1 score is achieved for 158 estimators with depth equal to 10.

4.4 Active Learning

The key idea behind active learning is that a machine learning algorithm can achieve greater accuracy with fewer training labels if it is allowed to choose the data from which it learns. Active learning is well motivated in many modern machine learning problems where unlabeled data may be abundant but labels are expensive to obtain. Active learning is sometimes called query learning or optimal experimental design because an active learner poses queries in the form of unlabelled data instances to be labeled by an oracle. In this way, the active learner seeks to achieve high accuracy using as few labeled instances as possible [42].

We focus on *pool-based sampling* that assumes that there is a small set of labeled data L and a large pool of unlabelled data U. Queries are selectively drawn from the pool according to an informativeness measure. Pool based methods rank the entire collection of unlabelled data to select the best query. Therefore, for very larget data-sets, stream-based sampling maybe more appropriate where the data is scanned sequentially and query decisions are evaluated individually.

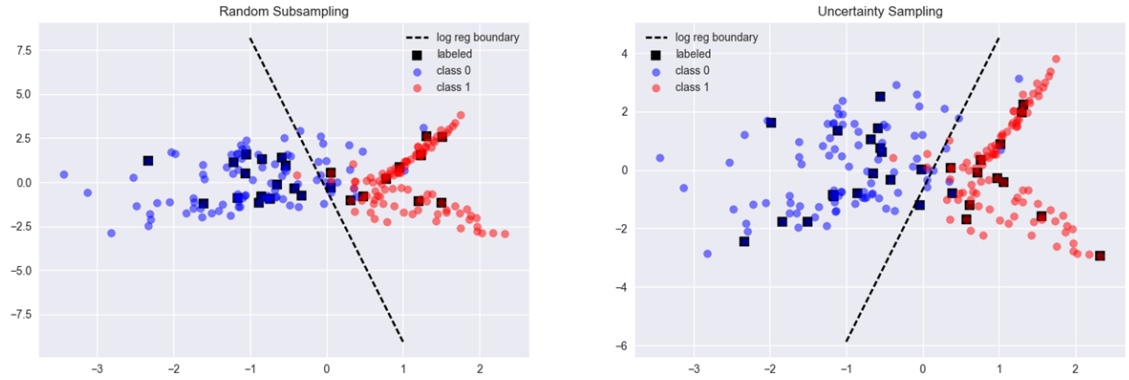

Figure 37: Random Subsampling vs Uncertainty Sampling

Figure 37 shows an example of pool-based active learning based on binary classification of a synthetic dataset with two balanced classes using Logistic Regression (LR). On the left, we can see the LR decision boundary as a result of training on a randomly subsampled set of 30 labels that achieves a classification accuracy of 90% on held out data. On the right, we can see the LR decision boundary as a result of training on 30 queries selected by uncertainty sampling based on entropy. Uncertainty sampling achieves a higher classification accuracy of 92.5% on the held out set.

4.4.1 Query Strategies

Uncertainty Sampling. One of the simplest and most commonly used query framework is uncertainty sampling. In this framework, an active learner queries the label about which it is least

certain. For example, in binary logistic regression, uncertainty sampling queries points near the boundary where the probability of being positive is close to 0.5. For multi-class problems, uncertainty sampling can query points that are least confident:

$$x^*_{LC} = \arg\max_x 1 - P_\theta(\hat{y}|x) \tag{263}$$

where $\hat{y} \in \{1, ..., K\}$ is the class label with the highest posterior probability under the model θ. The criterion for the least confident strategy only considers information about the most probable label. We can use max margin sampling to preserve information about the remaining label distribution:

$$x^*_M = \arg\min_x P_\theta(\hat{y}_1|x) - P_\theta(\hat{y}_2|x) \tag{264}$$

where $\hat{y}_1$ and $\hat{y}_2$ are the first and second most probable class labels under the model, respectively. Intuitively, instances with large margins are easy to classify. Thus, points with small margins are ambiguous and knowing their labels would help the model to more effectively discriminate between them. However, for multi-class problems with very large label sets, the margin strategy still ignores much of the output distribution for the remaining classes. A more general uncertainty sampling strategy is based on entropy:

$$x^*_H = \arg\max_x -\sum_i P_\theta(y_i|x) \log P_\theta(y_i|x) \tag{265}$$

By learning labels that have highest entropy we can reduce label uncertainty. Uncertainty sampling also works for regression problems, in which case the learner queries the point with highest output variance in its prediction.

Query by Committee. Another query selection framework is the Query By Committee (QBC) algorithm that involves maintaining a committee $C = \{\theta^{(1)}, ..., \theta^{(C)}\}$ of models which are all trained on the current labeled set L but represent competing hypotheses. Each committee member is then allowed to vote on the labelings of query candidates and the most informative query is considered to be an instance about which they most disagree. The objective of QBC is to minimize a set of hypotheses that are consistent with the current labeled training data L. For measuring the level of disagreement two main approaches have been proposed [42]: the vote entropy and KL divergence. The vote entropy is defined as follows:

$$x^*_{VE} = \arg\max_x -\sum_i \frac{V(y_i)}{C} \log \frac{V(y_i)}{C} \tag{266}$$

where $y_i \in \{1, ..., K\}$ is the class label, $V(y_i)$ is the number of votes that a label received from the committee members and C is the size of the committee. The KL divergence for QBC voting is defined as follows:

$$x^*_{KL} = \arg\max_x \frac{1}{C} \sum_{c=1}^{C} KL(P_{\theta^{(c)}} || P_C) \tag{267}$$

$$KL(P_{\theta^{(c)}} || P_C) = \sum_i P_{\theta^{(c)}}(y_i|x) \log \frac{P_{\theta^{(c)}}(y_i|x)}{P_C(y_i|x)} \tag{268}$$

$$P_C(y_i|x) = \frac{1}{C} \sum_{c=1}^{C} P_{\theta^{(c)}}(y_i|x) \tag{269}$$

where $\theta^{(c)}$ represents a member model of the committee and $P_C(y_i|x)$ is the consensus probability that y_i is the correct label. The KL divergence metric considers the most informative

query to be the one with the largest average difference between the label distributions of any one committee member and the consensus distribution.

Variance Reduction. We can reduce the generalization error by minimizing output variance. Consider a regression problem for which the learning objective is to minimize the mean squared error. Let $\bar{\theta} = E[\hat{\theta}]$ be the expected value of the parameter estimate $\hat{\theta}$ and let θ^* be the ground truth, then

$$\begin{aligned} \text{MSE} &= E\Big[(\hat{\theta} - \theta^*)^2\Big] = E\Big[\big[(\hat{\theta} - \bar{\theta}) + (\bar{\theta} - \theta^*)\big]^2\Big] && (270) \\ &= E\Big[(\hat{\theta} - \bar{\theta})^2\Big] + 2(\bar{\theta} - \theta^*)E\Big[\hat{\theta} - \bar{\theta}\Big] + (\bar{\theta} - \theta^*)^2 && (271) \\ &= E\Big[(\hat{\theta} - \bar{\theta})^2\Big] + (\bar{\theta} - \theta^*)^2 && (272) \\ &= \text{VAR}[\hat{\theta}] + \text{bias}^2(\hat{\theta}) && (273) \end{aligned}$$

This is called the **bias-variance tradeoff**. Thus, it is possible to achieve lower MSE with a biased estimator as long as it reduces the variance. A natural question is how low can the variance be? The answer is given by the Cramer-Rao lower bound that provides a lower bound on the variance of any unbiased estimator.

Theorem 4.1 *(Cramer-Rao Lower Bound) Assuming $p(x|\theta)$ satisfies the regularity condition, the variance of any unbiased estimator $\hat{\theta}$ satisfies:*

$$\text{VAR}(\hat{\theta}) \geq \frac{1}{-E\big[\frac{\partial^2 \log p(x|\theta)}{\partial \theta^2}\big]} = \frac{1}{I(\theta)} \tag{274}$$

where $I(\theta)$ is the Fisher information matrix.

Thus, the Minimum Variance Unbiased (MVU) estimator achieves the minimum variance equal to the inverse of the Fisher information matrix. To minimize variance of parameter estimates, an active learner should select data that maximizes its Fisher information. For multi-variate models with K parameters, Fisher information takes the form of a $K \times K$ matrix:

$$[I(\theta)]_{ij} = -E\left[\frac{\partial^2 \log p(x|\theta)}{\partial \theta_i \partial \theta_j}\right] \tag{275}$$

As a result, there are several options for minimizing the inverse information matrix: A-optimality minimizes the trace: $Tr(I^{-1}(\theta))$, D-optimality minimizes the determinant: $|I^{-1}(\theta)|$ and E-optimality minimizes the maximum eigenvalue: $\lambda_{max}[I^{-1}(\theta)]$.

However, there are some computational disadvantages to the variance reduction methods. Estimating output variance requires inverting a $K \times K$ matrix for each unlabeled instance, resulting in a time complexity of $\mathcal{O}(UK^3)$, where U is the size of the query pool. As a result variance reduction methods are empirically slower than simpler query strategies like uncertainty sampling.

Active learning and *semi-supervised learning* both try to make the most of unlabeled data. For example, a basic semi-supervised technique is self-training in which the learner is first trained with a small amount of labeled data and then used to classify the unlabeled data. The most

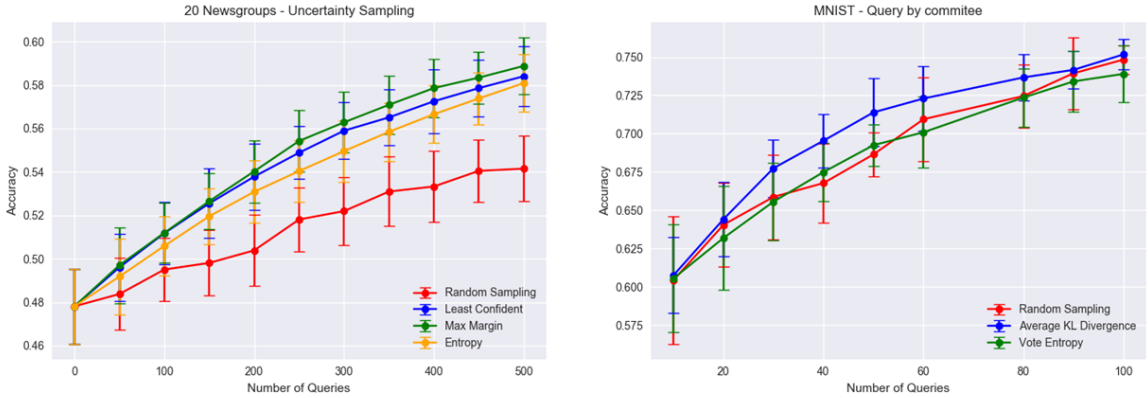

Figure 38: Active Learning: Uncertainty Sampling and Query By Committee.

confident unlabeled instances together with their predicted labels are added to the training set and the process repeats.

Figure 38 compares 3 uncertainty sampling techniques: least confident, max margin and entropy with a random subsampling baseline on 20 newsgroups dataset classified with Logistic Regression. All three methods achieve higher accuracy in comparison to baseline that highlights the benefit of active learning. On the right, we can see the performance of Query By Committee strategy applied to MNIST dataset. The committee consists of 5 instances of Logistic Regression. Two methods are compared against the random subsampling baseline: vote entropy and average KL divergence. We can see that average KL divergence achieves highest classification accuracy. All experiments were repeated 10 times.

4.5 Submodular Optimization

Many observation selection problems satisfy an intuitive **diminishing returns** property: adding an observation helps more if we made few observations so far, and helps less if we already made a lot of observations. From a sensing perspective, combining two sets of information has less gain than the sum of information gains of the individual sets. This concept is formalized by the property of submodularity.

Submodularity is a property of *set functions* $f : 2^V \to \mathbb{R}$ that assign each subset $S \subseteq V$ a value $f(S)$. F is called *submodular* if for all $A, B \subseteq V$ it holds that

$$F(A) + F(B) \geq F(A \cup B) - F(A \cap B) \tag{276}$$

Note that if F is non-negative ($F(A) \geq 0$ for all A) submodularity implies subadditivity: $f(A \cup B) \leq f(A) + f(B)$. An alternative definition of submodular functions is the following. A set function is sub-modular iff for all $A \subseteq B \subseteq V$ and $s \in V \setminus B$, it holds that

$$F(A \cup \{s\}) - F(A) \geq F(B \cup \{s\}) - F(B) \tag{277}$$

We can usually check more easily for monotonicity and submodularity of a function via its incremental definition, where the increment function $f(A|B)$ is defined as $f(A|B) = f(A \cup B) - f(B)$. Thus, for a single element $\{e\} \in V \setminus B$ and $A \subseteq B \subseteq V$, we can define sub-modularity as:

$$f(e|A) \geq f(e|B) \tag{278}$$

An important subclass of submodular functions are those which are *monotone* where enlarging the argument set cannot cause the function to decrease. A function $f : 2^V \to \mathbb{R}$ is monotone if for every $A \subseteq B \subseteq V$, we have $f(A) \leq f(B)$.

Submodularity is closed under the operations of non-negative addition, restriction and conditioning.

Theorem 4.2 *(Submodularity is closed under conic combination). Given submodular functions $f_1, f_2, ..., f_n$ and $\alpha_i \geq 0$, their conic combination (non-negative addition) is submodular.*

$$f(A) = \sum_{i=1}^{n} \alpha_i f_i(A) \tag{279}$$

Proof. Since every function f_i is submodular and $\alpha_i \geq 0$, we have for $A \subseteq B$:

$$f_i(j|A) \geq f_i(j|B) \Rightarrow \sum_{i=1}^{n} \alpha_i f_i(j|A) \geq \sum_{i=1}^{n} \alpha_i f_i(j|B) \Rightarrow f(j|A) \geq f(j|B) \tag{280}$$

Theorem 4.3 *(Entropy is submodular). Let V be the index set of a set of random variables, then the following function is submodular:*

$$f(A) = H(X_A) = -\sum_{X_A} p(X_A) \log p(X_A) \tag{281}$$

Proof. For $A \subseteq B \subseteq V$ and $e \in V \setminus B$, we have:

$$\begin{aligned} F(A \cup e) - F(A) &= H(X_A, X_e) - H(X_A) = H(X_e|X_A) && (282)\\ &\geq H(X_e|X_A, X_B) = H(X_e|X_B) = H(X_B, X_e) - H(X_B) && (283)\\ &= F(B \cup e) - F(B) && (284) \end{aligned}$$

where the inequality is due to the fact that conditioning reduces entropy.

Theorem 4.4 *(Mutual Information is submodular). MI is a submodular function if observed variables are conditionally independent given the latent state.*

Proof. MI can be expressed as a set function $f(A) = I(X; Y_A)$. We make a distinction between latent variables X and observed variables Y. For any pair $I, J \subseteq V$ such that $I \cap J = \emptyset$, we assume Y_I is independent of Y_J given X. Let $A \subseteq B \subseteq V$, and $j \in V \setminus B$, we have:

$$\begin{aligned} I(Y_j; Y_{B \setminus A}|Y_A) &\geq 0 && (285)\\ H(Y_j|Y_A) - H(Y_j|Y_{(B \setminus A) \cup A}) &\geq 0 && (286)\\ H(Y_j|Y_A) &\geq H(Y_j|Y_B) && (287) \end{aligned}$$

which is the submodularity for entropy. Continuing by using conditional independence assumption and subtracting $H(Y_j|X) = H(Y_j|X, Y_A) = H(Y_j|X, Y_B)$ from both sides:

$$\begin{aligned} H(Y_j|Y_A) - H(Y_j|X) &\geq H(Y_j|Y_B) - H(Y_j|X) && (288)\\ H(Y_j|Y_A) - H(Y_j|X, Y_A) &\geq H(Y_j|Y_B) - H(Y_j|X, Y_B) && (289)\\ I(Y_j; X|Y_A) &\geq I(Y_j; X|Y_B) && (290)\\ I(X; Y_j|Y_A) &\geq I(X; Y_j|Y_B) && (291) \end{aligned}$$

4.5.1 Greedy Maximization

Submodular functions arise in many applications and therefore it is natural to study submodular optimization. In particular, we focus on greedy maximization of submodular functions. We are interested in solving problems of the form:

$$\max_{S \subseteq V} f(S) \tag{292}$$

subject to some constraints on S. We will focus on two kinds of constraints: *cardinality constraint* $|S| \leq k$ and the *budget constraint* $\text{cost}(S) \leq B$. An example of the former is a sensor placement problem where we want to find the k best sensor locations. An example of the latter is a knapsack problem where we would like to maximize the value of items given a total budget of their weight.

A simple greedy approach to maximizing submodular functions is a greedy selection summarized in Algorithm 12 that starts with an empty set S_0 and at iteration i, adds the element maximizing the increment function:

$$S_i = S_{i-1} \cup \{\arg\max_e f(e|S_{i-1})\} \tag{293}$$

Algorithm 12 Greedy Algorithm for Maximizing a Submodular Function

1: Init $A_0 = \emptyset$, k
2: **for** $j = 1, 2, ..., k$ **do**
3: $\quad a_j = \arg\max_{e \in V} f(e|A_{j-1})$
4: $\quad$ set $A_j = A_{j-1} \cup \{a_j\}$
5: $\quad$ set $V = V \setminus \{a_j\}$
6: **end for**

A celebrated result by Nemhauser [33] proves that that the greedy algorithm provides a good approximation to the optimal solution of the NP-hard optimization problem.

Theorem 4.5 *Let $f : 2^V \to \mathbb{R}$ be a non-negative, monotone submodular function and let $\{S_i\}_{i \geq 0}$ be the greedily selected sets. Then for all positive integers k and l:*

$$f(S_l) \geq \left(1 - e^{-l/k}\right) \max_{S:|S| \leq k} f(S) \tag{294}$$

Proof. Let $S^* = \arg\max\{f(S) : |S| \leq k\}$ be an optimal set of size k. Order the elements of S^* arbitrarily as $\{v_1^*, ..., v_k^*\}$. Then, we have the following sequence of inequalities for all $i < l$:

$$\begin{aligned}
f(S^*) &\leq f(S^* \cup S_i) && (295)\\
&= f(S_i) + \sum_{j=1}^{k} f(v_j^* | S_i \cup \{v_1^*, ..., v_{j-1}^*\}) && (296)\\
&\leq f(S_i) + \sum_{v \in S^*} f(v|S_i) && (297)\\
&\leq f(S_i) + \sum_{v \in S^*} (f(S_{i+1}) - f(S_i)) && (298)\\
&\leq f(S_i) + k(f(S_{i+1}) - f(S_i)) && (299)
\end{aligned}$$

where the first inequality follows from monotonicity of f, the second equality is a telescoping sum, the third inequality is follows from the submodularity of f, the fourth holds because S_{i+1} is built greedily from S_i, and the last inequality reflects the fact that $|S^*| \leq k$. Therefore,

$$f(S^*) - f(S_i) \leq k(f(S_{i+1}) - f(S_i)) \tag{300}$$

Define $\delta_i = f(S^*) - f(S_i)$, we can then re-write the above expression as $\delta_i \leq k(\delta_i - \delta_{i+1})$, that can be re-arranged to yield:

$$\delta_{i+1} \leq \left(1 - \frac{1}{k}\right)\delta_i \tag{301}$$

Therefore, $\delta_l \leq (1 - \frac{1}{k})^l \delta_0$. Using the inequality $1 - x \leq e^{-x}$, we have

$$\delta_l \leq \left(1 - \frac{1}{k}\right)^l \delta_0 \leq e^{-l/k} f(S^*) \tag{302}$$

where we used the fact that $\delta_0 = f(S^*) - f(\emptyset) \leq f(S^*)$ since f is non-negative by assumption. Substituting $\delta_l = f(S^*) - f(S_l)$ and re-arranging, we get:

$$f(S_l) \geq (1 - e^{-l/k}) f(S^*) \tag{303}$$

Therefore, if we let the greedy algorithm pick $l = k$ sensors (compared to the optimal set of size k), the greedy solution is no worse than $1 - 1/e \approx 0.632$ of the optimal value!

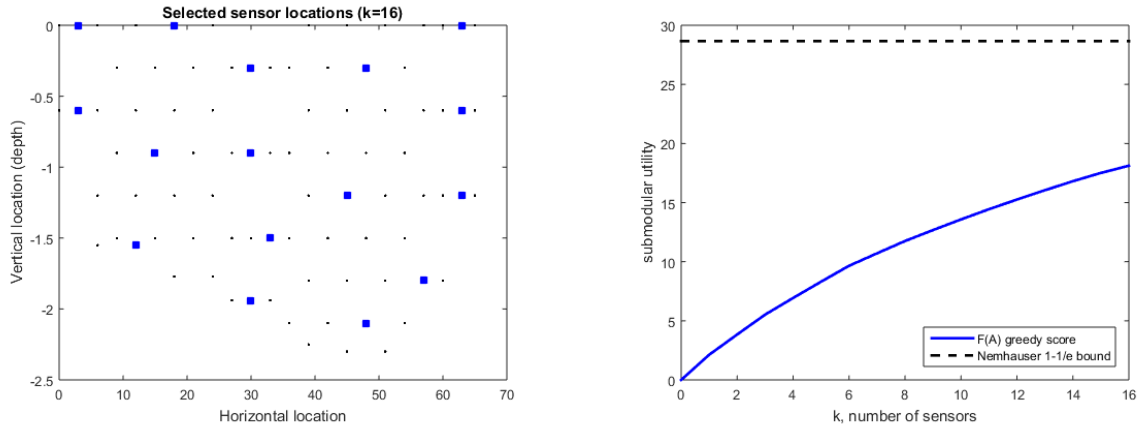

Figure 39: Submodular function optimization subject to cardinality constraint.

Figure 39 shows a sensor selection problem that maximizes mutual information $F_{mi}(A) = H(V \setminus A) - H(V \setminus A|A)$ over the ground set V subject to cardinality constraint: $|S| \leq k$. We can see that the selected sensors evenly cover the geographic area. Also shown are submodular utility scores along with Nemhouser bound obtained by taking the last score and dividing it by $(1 - 1/e)$.

In many applications, the elements $v \in V$ may have non-uniform costs $c(v) \geq 0$ and we may want to maximize f subject to a budget B that the total cost cannot exceed:

$$\max_S f(S) s.t. \sum_{v \in S} c(v) \leq B \tag{304}$$

The standard (uniform cost) greedy algorithm can perform arbitrarily badly since it ignores the cost. Therefore, we need to modify the selection step by normalizing the increment function by the element's cost:

$$S_{i+1} = S_i \cup \left\{ \arg \max_{v \in V \setminus S_i : c(v) \leq B - c(S_i)} \frac{f(v|S_i)}{c(v)} \right\} \tag{305}$$

Thus, we choose an element that has highest incremental gain and lowest cost. Note that the sensor selection problem can be thought of as a special case of the budget constrained problem if we assign unit costs to all measurements and set the budget $B = k$. Let S_{uc} be the uniform cost solution and S_{cb} be the cost-benefit greedy solution described above. It can be shown that:

$$\max\{f(S_{uc}), f(S_{cb})\} \geq \frac{1 - 1/e}{2} \text{OPT} \tag{306}$$

The submodular maximization algorithm with a budget constraint also known as Cost-Effective Lazy Forward-selection (CELF) Algorithm is summarized in Algorithm 13.

Algorithm 13 Cost-Effective Lazy Forward-selection (CELF)

1: Input: Reward function F, cost function C, budget B
2: $A_{UC} = \text{LazyForward}(F, C, B, 'UC')$
3: $A_{CB} = \text{LazyForward}(F, C, B, 'CB')$
4: return $\arg\max\{F(A_{UC}), F(A_{CB})\}$
5: LazyForward(F,C,B,type)
6:
7: **end for**

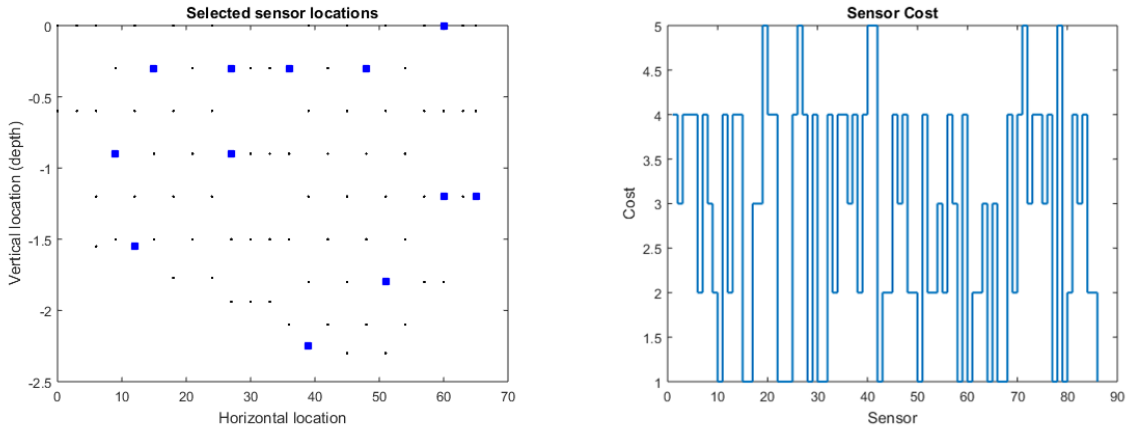

Figure 40: Submodular function optimization subject to budget constraint.

Algorithm 14 Generic Information Planning

1: **Input:** data $x \in \{D_{train}, D_{pool}\}$, model M, acquisition function $a(x, M)$
2: **Output:** trained model $M^{(T)}$
3: **for** $t = 1, 2, ..., T$ **do**
4: train $M^{(t)}$ on D_{train}
5: select $x^* = \arg\max_{x \in D_{pool}} a(x, M^{(t)})$
6: update $D_{train} = D_{train} \cup x^*$; $D_{pool} = D_{pool} \setminus x^*$
7: **end for**

5 Information Planning

Information planning enables faster learning with fewer training examples. It is particularly applicable when training examples are costly to obtain. This work examines the advantages of information planning for text data by focusing on three supervised models: Naive Bayes, supervised LDA and deep neural networks. We show that planning based on entropy and mutual information outperforms random selection baseline and therefore accelerates learning.

5.1 Introduction

Information planning involves making decisions based on information measures [8]. Information planning is closely related to active learning [42], [34] and optimum experiment design [12], [7] in which labeled data is expensive to obtain. The key idea behind information planning is that a model can learn better with fewer training examples if the training examples are carefully selected to maximize the information gain for a particular task. In other words, given an abundance of unlabelled data, we would like to rank the unlabelled examples according to their usefulness in training of the model. The top K most informative training examples are labelled by expert annotators and added to the training set.

There are three main scenarios in which the learner may be able to ask queries: membership queries, stream-based selective sampling and pool-based sampling. In membership query, the learner may request labels for any unlabelled instance in the input space. In stream-based selective sampling, the active learning is done sequentially as each unlabelled instance is drawn one at a time from the data source and the learner must decide whether to query or discard it. Finally, in pool-based sampling, the entire set of unlabelled data is ranked and the most informative subset is labelled and added to the training set. In this work, we focus on pool based sampling as summarized in Algorithm 14.

The decisions whether or not to query instances of unlabelled examples are based on a measure of informativeness or a query strategy. The simplest and most commonly used query framework is *uncertainty sampling* [27]. In this framework, an active learner queries instances about which it is least certain about the label. For example, in a binary classification task, uncertainty sampling queries points near the decision boundary where the class probabilities are close to 1/2, i.e. highest entropy. Uncertainty sampling also works for regression problems, in which case the learner queries points with highest output variance in their prediction. In general, uncertainty sampling chooses a subset x^* from a set of unlabelled examples $x \in D_{pool}$ that maximizes label entropy:

$$x^* = \arg \max_{x \in D_{pool}} -\sum_c p(y = c|x) \log p(y = c|x) \tag{307}$$

An alternative measure that not only looks at the uncertainty of the label but also at the

informativeness is mutual information:

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) \tag{308}$$

For random variables X and Y, to maximize the *information gain* $I(X;Y)$, we want to maximize $H(X)$ (uncertainty about X) and minimize $H(X|Y)$, i.e. we want Y to be informative about X. The choice of random variables X and Y depends on the specific application and probabilistic model. As we will see, in complex models, often one direction of mutual information is easier to compute compared to the other.

Other query selection strategies include ensemble methods such as query-by-committee [43] in which an ensemble of diverse and accurate models are trained on the labeled dataset. Each ensemble member is then allowed to vote on the labels of query candidates and the most informative query is considered to be an instance about which they most disagree. In this work, we focus on uncertainty sampling and mutual information as the primary query strategies.

In application to natural language processing, active learning has been successfully applied to information extraction [40], named entity recognition [1], text categorization [49], part-of-speech tagging [39] and parsing [2]. In this work, we study the benefits of information planning for text data on three different tasks: text classification, supervised topic modeling and sentiment classification. [1] To demonstrate the advantage of information planning, we compare performance of active learning against random selection. As the size of unlabelled dataset increases, we expect active learning to perform better.

5.2 Naive Bayes

We consider the task of text classification using a Naive Bayes graphical model as shown in Figure 41.

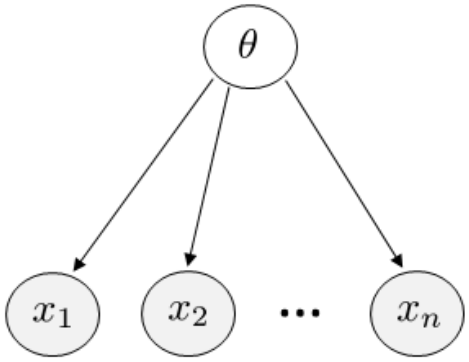

Figure 41: Naive Bayes graphical model.

Let x_{ij} be Bernoulli random variables indicating the presence ($x_{ij} = 1$) or absence ($x_{ij} = 0$) of a word $j \in \{1, ..., D\}$ in document $i \in \{1, ..., n\}$, parameterized by θ_{jc} for a given class label $y_i \in \{1, ..., C\}$. In addition, let π be a Dirichlet distribution representing the prior over the class labels. Thus, the total number of parameters is $|\theta| + |\pi| = \mathcal{O}(DC) + \mathcal{O}(C) = \mathcal{O}(DC)$. Due to the small number of paramters, the Naive Bayes model is immune to over-fitting [31].

[1] all code is available on-line

The choice of Bernoulli Naive Bayes formulation is important because it leads itself to word-based information planning. By associating each word in the dictionary with a binary random variable, we are able to compute the influence of individual words on class label distribution.

Consider words x_i in a single document i:

$$p(x_i, y_i|\theta) = p(y_i|\pi) \prod_{j=1}^{D} p(x_{ij}|y_i, \theta) = \prod_{c=1}^{C} \pi_c^{1[y_i=c]} \prod_{j=1}^{D} \prod_{c=1}^{C} p(x_{ij}|\theta_{jc})^{1[y_i=c]} \tag{309}$$

We can write-down the Naive Bayes inference algorithm by maximizing the log-likelihood objective:

$$\begin{aligned} \log p(D|\theta) &= \log \prod_{i=1}^{n} p(x_i, y_i|\theta) = \sum_{i=1}^{n} \log p(x_i, y_i|\theta) \\ &= \sum_{c=1}^{C} N_c \log \pi_c + \sum_{j=1}^{D} \sum_{c=1}^{C} \sum_{i:y_i=c} \log p(x_{ij}|\theta_{jc}) \end{aligned} \tag{310}$$

During test time, we would like to predict the class label y given the training data D and the learned model parameters. Applying the Bayes rule:

$$p(y = c|x_{i,1}, ..., x_{i,D}, D) \propto p(y = c|D) \prod_{j=1}^{D} p(x_{ij}|y = c, D) \tag{311}$$

Substituting $p(x_{ij}|y = c, D)$ and taking the log, we get:

$$\log p(y = c|x, D) \propto \log p(y = c|D) + \sum_{j=1}^{D} \big(1[x_{ij} = 1] \log \theta_{jc} + 1[x_{ij} = 0] \log(1 - \theta_{jc})\big) \tag{312}$$

In entropy based planning we want to rank test documents according to the entropy of their class label. In the case of class distribution, the entropy can be computed in closed form: $H(y) = -\sum_{c=1}^{C} p(y = c) \log p(y = c)$.

In the case of mutual information, we need to choose the random variables we want to use for planning. Since we are interested in classifying documents, it makes sense to choose y_i (the class label for document i) as one of the variables. Our choice of the second variable depends on the quantity of interest. We can choose one of the global variables θ_{jc} (probability that word j is present in class c) or π_c (probability of class c). In this example, we consider estimating $I(y_i; \theta)$, i.e. we are interested in measuring the information gain about the word distribution θ given the class label y_i for a test document. Since both variables are discrete, the mutual information can be estimated in closed form:

$$I(y_i; \theta) = \mathrm{KL}\big(p(y_i, \theta)||p(y_i)p(\theta)\big) = \sum_{y \in C} \sum_{\theta} p(\theta|y_i)p(y_i) \log \frac{p(\theta|y_i)}{p(\theta)} \tag{313}$$

We can compute $p(\theta) = \sum_{c \in C} p(x_j = 1|y = c)p(y = c)$ for $j \in \{1, ..., D\}$. In addition, we compute $I(x_j; \pi)$ to measure how informative each word x_j is to the global label distribution π.

We train our Naive Bayes classifier on a subset of the 20newsgroups dataset. In particular, we'll restrict ourselves to 4 classes: space, graphics, autos and hockey. We'll use a count vectorizer to produce a vector of word counts for each document while filtering stop and low frequency words. Figure 42 shows the posterior word distribution for the four classes.

Let's visualize the test document ranking based on entropy and mutual information in Figure 43. In particular, we are interested in outliers, i.e. documents that have highest entropy and mutual information. To visualize the difference between entropy and MI based planning, we can sort the documents according to entropy and use the same index to sort MI as shown in Figure 44. In Figure 44 documents that have high entropy but low MI are uncertain but not informative. Therefore, we expect MI to provide a better measure for information planning.

To evaluate the utility of information planning in comparison to random selection, we take a total of $1K$ documents and divide them into 100 labelled training documents, 700 unlabelled test documents, and holdout 200 documents for evaluating classification performance. We run the active learning pipeline, in which top K most informative documents are selected from the test set, labelled by annotators and added to the training set, at which point the modeled is re-trained from scratch and evaluated on the 200 heldout documents. Figure 45 shows the experimental results of this active learning pipeline. We can see a clear advantage of using information planning compared to random selection.

5.3 Supervised LDA

Latent Dirichlet Allocation (LDA) [5] is an unsupervised topic model that represents each document as a mixture of topics, where a topic is a distribution over words. The objective is to learn the shared topic distributions and their proportions for each document. However, it is often desirable to associate a quantity of interest (e.g. a score) with each document; this could be a five star rating for a movie review, a quality score of a report, or a number of times an on-line article is linked. Supervised LDA (sLDA) [4] is designed to jointly learn the topics and the score variable associated with each document. The graphical model of supervised LDA is shown in Figure 46.

sLDA associates each word x_{id} with a topic label $z_{id} \in \{1, ..., K\}$. Each document is associated with topic proportions θ_d that can also be used as a measure of document similarity. The topics β_k represented by the Dirichlet distribution are shared across all documents. The hyper-parameters α and η capture our prior knowledge of topic proportions and the topics, e.g. from past training of the model. Finally, y_d is the response variable that represents a score associated with each document. In this example, y_d is modelled as a regression between global coefficients w and a normalized histogram of topics present in a document: $\bar{z}$. Assuming a

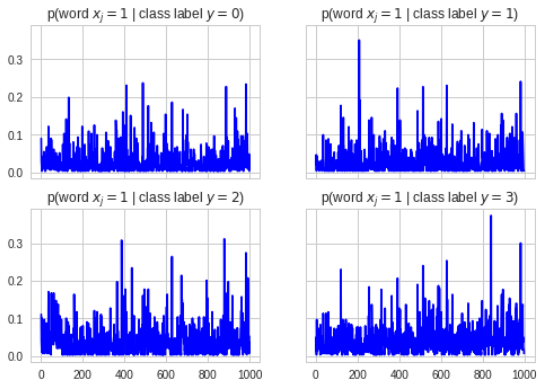

Figure 42: Naive Bayes class-conditional posterior.

Gaussian distribution, y_d can be expressed as follows:

$$y_d \sim N(w_1\bar{z}_1 + ... + w_k\bar{z}_k, \sigma^2) \tag{314}$$

$$\bar{z} = \frac{1}{N}\sum_{n=1}^{N} z_n \tag{315}$$

where z_n is a one-hot encoded topic assignment vector of size $1 \times K$. We use PyMC [36] Metropolis-Hastings sampler to infer the latent variables. Figure 47 shows the first 4 (out of 8) topics of sLDA model trained on the polarity sentiment dataset [35]. Notice, that the addition of the score variable y_d influences topic inference due to the coupling between w and β_k introduced by the $z_{i,d} \rightarrow y_d \leftarrow w$ and $z_{i,d} \rightarrow x_{i,d} \leftarrow \beta_k$ v-structures [25].

We can use a sample based estimator to compute the entropy of the score variable y_d: $H[y_d|x_d, D] = E[-\log p(y_d|x_d, D)]$.

Assuming we have topic samples:

$$\{(z_{1d}^{(i)}, ..., z_{Nd}^{(i)})\}_{i=1}^{M} \overset{MCMC}{\sim} p(z_{1d}, ..., z_{Nd}|x_d, D) \tag{316}$$

$$p(y_d|x_d, D) = \sum_{z_{1d}} ... \sum_{z_{Nd}} p(z_{1d}, ..., z_{Nd}|x_d, D)p(y_d|z_d) \approx \frac{1}{M}\sum_{i=1}^{M} p(y_d|z_d = z_d^{(i)}) = \hat{p}(y_d|x_d, D) \tag{317}$$

Once we have a sample estimate $\hat{p}(y_d|x_d, D)$, to compute entropy, we need to estimate the expectation:

$$E[-\log \hat{p}(y_d|x_d, D)] = -\int p(y_d|x_d, D)\log \hat{p}(y_d|x_d, D)dy_d \approx \frac{1}{M}\sum_{i=1}^{M} \log \hat{p}(y_d^{(i)}|x_d, D) \tag{318}$$

where $\{(y_d^{(i)})\}_{i=1}^{M} \overset{MCMC}{\sim} \hat{p}(y_d^{(i)}|x_d, D)$.

To evaluate the utility of information planning in comparison to random selection, we start with a corpus of 100 documents, from which we allocate 15 documents for training, 35 for testing and holdout 50 documents for evaluating sLDA performance. Figure 48 shows the test MSE of the Gaussian score variable evaluated against the ground truth review score on the heldout set of 50 documents.

We can see that active learning selection achieves smaller MSE in comparison to random selection baseline.

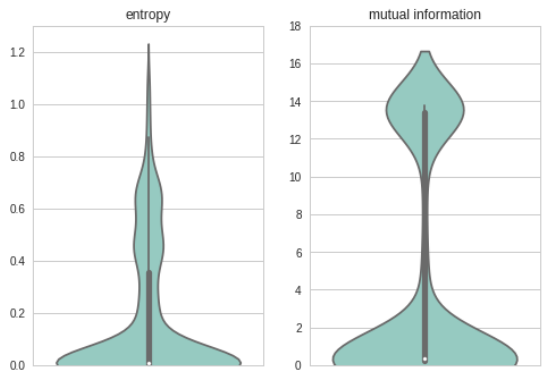

Figure 43: Violin plot showing the density of entropy and MI on test data.

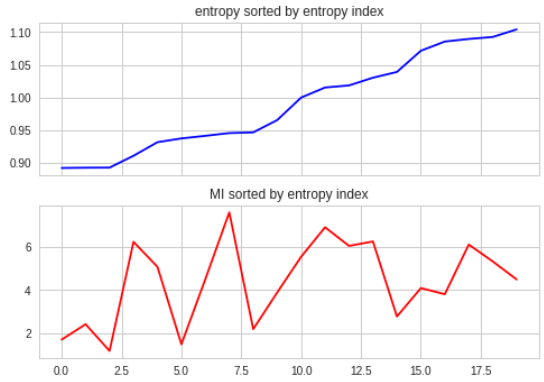

Figure 44: Test document entropy and MI sorted by entropy index.

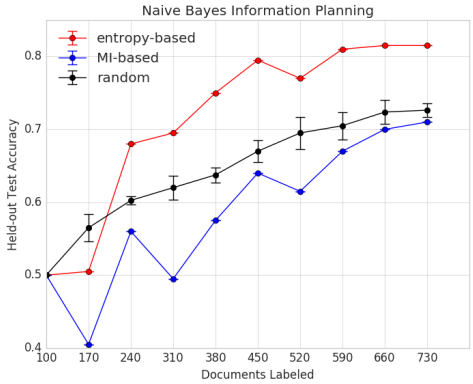

Figure 45: Classification accuracy as a function of labeled examples.

5.4 Deep Neural Network

We consider information planning in the case of deep neural networks. In particular, we examine two architectures: LSTM and CNN in application to sentence sentiment classification [17].

Recurrent Neural Networks (RNNs) provide a natural way of encoding a sentence as a sequence of words. Just as you are processing this sentence word by word while keeping a memory of what came before, RNNs maintain an internal model of the past and update it as soon as the new information arrives. Thus, RNNs contain an internal loop unrolled over the length of the input sequence. As a result, RNNs can capture more information in comparison to n-gram models. For example, in a n-gram language model (with $n = 3$), we can predict the next word

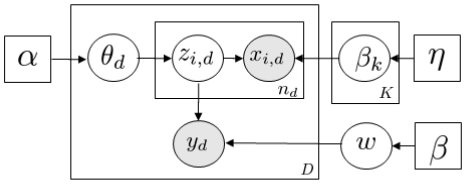

Figure 46: Supervised LDA [4] graphical model.

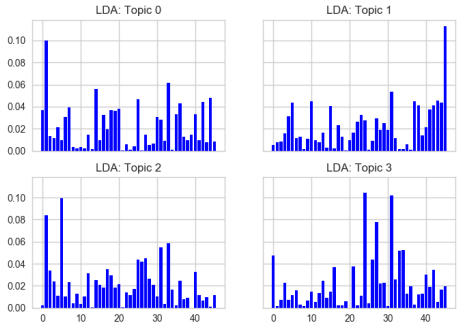

Figure 47: Supervised LDA topics inferred using PyMC [36].

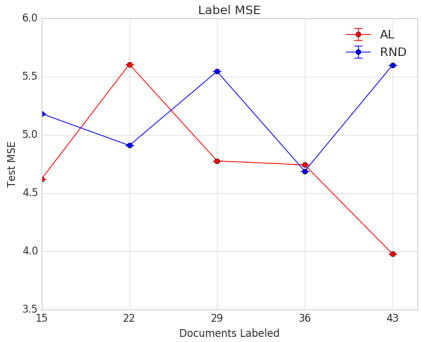

Figure 48: Supervised LDA test MSE.

given the previous two:

$$l_{sent} = \frac{1}{n} \log P(w_1, ..., w_n) = \frac{1}{n} \sum_{i=1}^{n} \log p(w_i | w_{i-1}, w_{i-2}) \tag{319}$$

The dependence in neural language model (LM) extends to the beginning of the sentence, i.e. $p(w_i|w_{i-1}, ..., w_1)$, while in a n-gram LM the conditioning is only on the previous n words and as n grows larger, sparsity of training data becomes a problem. In both cases, the word conditional distribution greatly reduces the output space of possible words and enables learning of word representations based on their context [30].

A sentence can be encoded by an RNN (such as LSTM) as either the output of its last layer or via average pooling of its intermediate outputs. In both cases, the word position information is preserved in contrast to averaging of word vectors. Figure 49 shows the LSTM architecture used for sentiment classification. We use a bi-directional LSTM that concatenates sequence representations of forward and reverse LSTMs. This enriches order-sensitive representation of the sentence and leads to a slight increase in accuracy. In addition to output layer dropout, we use recurrent dropout that applies the same dropout mask at every time-step (in contrast to a dropout mask that varies randomly between time-steps that could harm the learning process) [14].

Convolutional Neural Networks (CNNs) offer computational advantages due to their parallel nature and comparable performance with time-series data, especially in cases when recent past is not more informative than the distant past (due to translation invariance of CNNs). The same properties that make convnets excel at computer vision also make them highly relevant

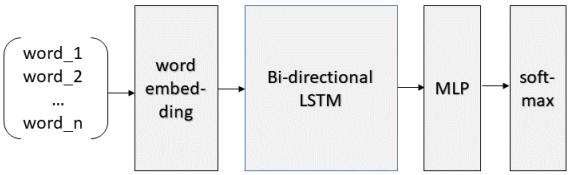

Figure 49: LSTM model architecture

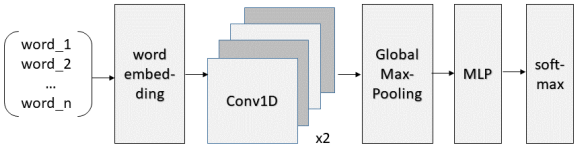

Figure 50: CNN model architecture

for sequence processing. Time can be treated as a spatial dimension like the height or width of a 2D image. For example, 1D convolutions can be used to extract patterns from local 1D patch sub-sequences of the input sequence. Because the same transformation is performed on every patch, a pattern learned at a certain position in the sentence can later be recognized at a different position, making 1D convnets translation invariant. While a word-level conv-net operating on word embeddings with a kernel size of 5 should be able to recognize words or word fragments of length 5 or less, a character-level 1D conv-net is able to learn about word morphology. In addition, we can learn hierarchical patterns by stacking several convolutional layers. Figure 50 shows the CNN architecture used for sentiment classification. It consists of a word embedding layer followed by a stack of two convolutional layers separated by a dropout layer. Finally, the activations across each of the filters are pooled via global max pooling and fed into a Multi-Layer Perceptron (MLP) classifier.

For both LSTM and CNN models, we choose pre-trained word embeddings because in an active learning setting, there's usually not enough training data to learn specialized word embeddings for a given task. In order to preserve the latent structure (that could otherwise be destroyed by large gradients from randomly initialized downstream layers), we configure the embeddings to be non-trainable. Both architectures use Glove pre-trained word embeddings [37] based on word co-occurence for dense representation of words.

As shown in [15] dropout [46] and other stochastic regularization techniques can be used for approximate inference in deep neural networks. This is accomplished by using dropout at test-time and computing Monte Carlo integration (referred to as MC dropout):

$$p(y=c|x,D) = \int p(y=c|x,w)p(w|D)dw \tag{320}$$

$$\approx \int p(y=c|x,w)q_\theta^*(w)dw \approx \frac{1}{T}\sum_{t=1}^{T} p(y=c|x,\hat{w}_t)$$

where $\hat{w}_t \sim q_\theta^*(w)$ and $q_\theta^*(w)$ is a variational distribution that minimizes KL divergence to the true model posterior $p(w|D)$:

$$\arg\min_\theta KL(q_\theta(w)||p(w|D)) \tag{321}$$

where θ are the variational parameters and w are the neural network weights. Averaging forward passes with MC dropout during test time is equivalent to Monte Carlo integration over a Gaussian process posterior approximation [16].

We consider two acquisition functions: one based on entropy and one based on mutual information. An *acquisition function* $a(x, M)$ determines which measurements to take next, and it's a function of the pool of unlabelled examples $x \in D_{pool}$ and the model M:

$$x^* = \arg\max_{x \in D_{pool}} a(x, M) \tag{322}$$

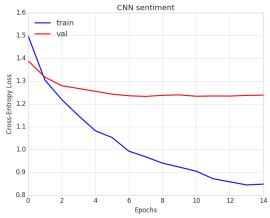

Figure 51: CNN: training loss

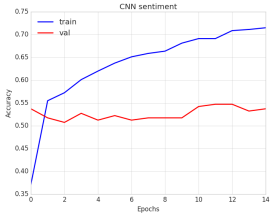

Figure 52: CNN: training acc

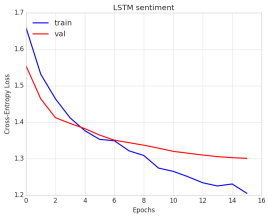

Figure 53: LSTM: training loss

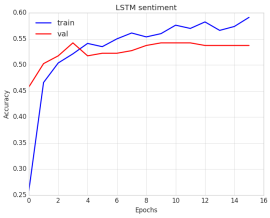

Figure 54: LSTM: training acc

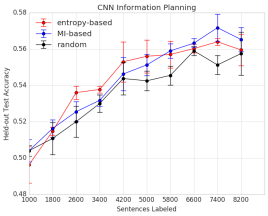

Figure 55: CNN: test accuracy

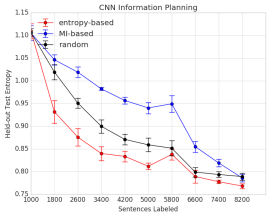

Figure 56: CNN: test entropy

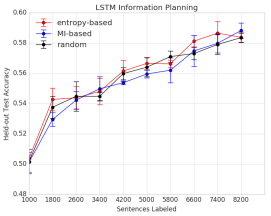

Figure 57: LSTM: test acc.

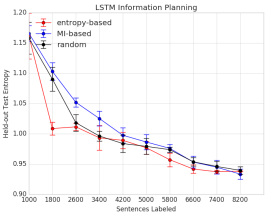

Figure 58: LSTM: test entropy

In the case of entropy, we want to choose data points that maximize:

$$H[y|x, D] = -\sum_c p(y = c|x, D) \log p(y = c|x, D) \tag{323}$$

In the case of mutual information, we want to choose data points that maximize the information gain between the predictions y and the model parameters w:

$$I(y; w|x, D) = H[y|x, D] - E_{p(w|D)}[H[y|w, x, D]] \tag{324}$$

where w in our case are the weights of the neural network and the expectation is computed using MC dropout. Thus, we are seeking data points x for which the model is both marginally most uncertain about y and most confident about the predicted label. Notice, that it's easier to compute MI as written above (as opposed to its alternative version) because the entropies are computed in a discrete, low-dimensional output space. This interpretation allows reasoning about uncertainty in deep learning and the application of Bayesian methods to analyze existing deep learning frameworks.

We evaluate the effectiveness of information planning on Kaggle movie review sentiment dataset [24]. The dataset consists of phrases from the Rotten Tomatoes dataset classified into 5 sentiment labels: negative, somewhat negative, neutral, somewhat positive and positive. Obstacles such as sentence negation, sarcasm, and language ambiguity make this a challenging task. We take the first 10K sentences, set aside 10% for training and holdout 10% for evaluating the performance. At each iteration, we select the top $K = 800$ most informative sentences to be annotated and added to the training set. Figures 55, 56, 57, 58 show active learning experimental results.

5.5 Discussion

In this section we discuss information planning results for each model.

Naive Bayes is the simplest model for studying information planning since all latent variables are discrete and information measures can be computed in closed form. By choosing the Bernoulli formulation of Naive Bayes we were able to do planning at the word level. In particular, the top-10 highest MI words as shown in Table 1 were found to be informative of the ground truth labels.

class 0	class 1	class 2	class 3	MI
get	think	first	nasa	teams
please	well	time	could	program
use	know	season	much	league
like	also	year	know	moon
one	cars	play	get	playoffs
anyone	get	hockey	also	orbit
thanks	like	one	like	players
graphics	one	would	one	earth
would	would	game	would	games
know	car	team	space	nasa

Table 1: Naive Bayes: top 10 words. Ground truth labels: graphics, autos, hockey, space.

From Figure 45, we can see that we achieve about 10% accuracy increase when we use uncertainty sampling in comparison to random selection. All experimentes were repeated 10 times.

In supervised LDA model, the score variable is learned jointly with the topics. Figure 48 shows that smaller MSE is achieved with uncertainty sampling in comparison to random selection. We also note that sampling based inference for sLDA is more suitable to stream queres in which documents are queried one at a time due considerable time it takes to run sampling for the entire pool of unlabelled examples.

In deep neural network setting we assume that computational expense of re-training the model from scratch is justified by the cost of obtaining the labels. From the training and validation loss in Figures 51, 53, we see no signs of over-fitting due regularization with dropout and weight decay. The validation loss plateaus early in the training indicating that model capacity can be increased in an attempt to achieve higher classification accuracy. Figures 52, 54 show that CNN and LSTM model achieve comparable accuracy on the sentiment classification task with validation accuracy close to 55%.

Figures 55, 57 show the benefit of information planning over the random basline. We can see that we achieve close to 1% improvement in accuracy in the case of CNN and a relatively smaller improvement in the case of LSTM, even thought LSTM achieves 2% higher held-out test accuracy in the end. It's also interesting to see, the decrease in test label entropy as shown in Figures 56, 58, where as expected the uncertainty sampling leads to highest reduction in entropy.

5.6 Set Cover Problem

We consider the problem of data driven sensor placement subject to placement constraints. In particular, we assume that we have n sub-sets of sensors that we would like to use to cover m potential locations. The number of locations m can be obtained as a discretization of the input space. We assume Gaussian Process (GP) environment in which our sensor measurements form

a jointly Gaussian distribution and every additional sub-set of sensors reduces the uncertainty.

This is known as a set cover problem [44], in which we would like to use minimum number of sub-sets whose union is equal to the universal set. A variation of this problem known as *max-k-cover* limits the number of possible sub-sets to k.

Greedy algorithms can be used to maximize the information gain objective with non-uniform cost [26]. However, greedy algorithms can fail to find global optima for certain inputs. Our claim is that simulated annealing (SA) can perform better than greedy. We study the computational and accuracy trade-offs of the two algorithms for the *max-k-cover* problem.

5.6.1 Greedy Algorithm

There exist performance guarantees for greedy algorithms with submodular objective function [26]. We select an increment in mutual information (MI) as our objective because it satisfies submodularity and improves upon the entropy maximization objective. Let A be a set of already selected sensor locations, y be a subset under consideration and $\bar{A}$ be a set representing unselected locations, i.e. $V = A \cup y \cup \bar{A}$. Then, define:

$$F_{MI}(A) = I(X_A; X_{V\setminus A}) = H(X_{V\setminus A}) - H(X_{V\setminus A}|X_A) \tag{325}$$

i.e. we consider maximizing mutual information between a set of selected sensors X_A and the remaining sensor locations $X_{V\setminus A}$. Then, the incremental MI can be written as:

$$\begin{aligned} F_{MI}(A \cup \{y\}) - F_{MI}(A) &= H(X_A \cup X_y) - H(X_A \cup X_y|X_{\bar{A}}) - \left[H(X_A) - H(X_A|X_{\bar{A}} \cup X_y)\right] \\ &= H(X_A \cup X_y) - H(X_V) + H(X_{\bar{A}}) - \left[H(X_A) - H(X_V) + H(X_{\bar{A}} \cup X_y)\right] \\ &= H(X_A \cup X_y) - H(X_A) + H(X_{\bar{A}}) - H(X_{\bar{A}} \cup X_y) \\ &= H(X_y|X_A) - H(X_y|X_{\bar{A}}) \end{aligned} \tag{326}$$

We can compute conditional entropies in closed form for the multivariate Gaussian distribution using the rules of conditioning [3]:

$$H(X_y|X_A) = \frac{1}{2}\log\left[(2\pi e)^{|y|}\det\Sigma_{X_y|X_A}\right] = \frac{1}{2}\log\det\Sigma_{X_y|X_A} + \text{const} \tag{327}$$

Given GP posterior mean and variance equations with mean μ and kernel K [31]:

$$\mu_{y|A} = \mu_y + \Sigma_{yA}\Sigma_{AA}^{-1}(x_A - \mu_A) \tag{328}$$

$$\Sigma_{X_y|X_A} = K(y,y) - \Sigma_{yA}\Sigma_{AA}^{-1}\Sigma_{Ay} \tag{329}$$

we can express our objective function as follows:

$$\begin{aligned} F_{MI}(A \cup \{y\}) - F_{MI}(A) &= H(X_y|X_A) - H(X_Y|X_{\bar{A}}) \\ &= \frac{1}{2}\log\det\Sigma_{X_y|X_A} - \frac{1}{2}\log\det\Sigma_{X_y|X_{\bar{A}}} \\ &= \frac{1}{2}\log\frac{\det|K(y,y) - \Sigma_{yA}\Sigma_{AA}^{-1}\Sigma_{Ay}|}{\det|K(y,y) - \Sigma_{y\bar{A}}\Sigma_{\bar{A}\bar{A}}^{-1}\Sigma_{\bar{A}y}|} \end{aligned} \tag{330}$$

We summarize the greedy algorithm for *max-k-cover* in Algorithm 15. The complexity of the greedy algorithm is $\mathcal{O}(n^4k)$: the covariance inversion step $\mathcal{O}(n^3)$ is computed for every $\mathcal{O}(n)$ sub-set y, $\mathcal{O}(k)$ times. Note, that we can compute the covariance matrix by choosing a kernel function that takes the coordinates of V sensors as input.

Algorithm 15 Greedy max-k-cover

Input: V, k, Σ_{VV}
Output: sensor placement A
$A = \{\}$
for $j = 1, 2, ..., k$ **do**
    for $y \in V \setminus A$ **do**
        $\delta_y = \frac{1}{2} \log \frac{\det |K(y,y) - \Sigma_{yA} \Sigma_{AA}^{-1} \Sigma_{Ay}|}{\det |K(y,y) - \Sigma_{y\bar{A}} \Sigma_{\bar{A}\bar{A}}^{-1} \Sigma_{\bar{A}y}|}$
    end for
    $y = \arg\max_{y \in V \setminus A} \delta_y$
    $A = A \cup y$
end for

5.6.2 Simulated Annealing

5.6.3 Dynamic Programming

6 Misc

6.1 Sequential Monte Carlo

Particle Filters (PF) is a Sequential Monte Carlo (SMC) method for estimating the internal states of a Switching Linear Dynamic System (SLDS). A set of particles is used to represent the posterior distribution of the states of the Markov process. At each iteration, the particles get re-samples and the ones that explain the observations best survive to the next iteration.

The Gaussian SLDS can be described as follows [10]:

$$\begin{aligned} z_t &\sim P(z_t|z_{t-1}) && (331)\\ x_t &= A(z_t)x_{t-1} + B(z_t)w_t + F(z_t)u_t && (332)\\ y_t &= C(z_t)x_t + D(z_t)v_t + G(z_t)u_t && (333) \end{aligned}$$

where $y_t \in \mathbb{R}$ denotes the observations, $x_t \in \mathbb{R}$ denotes latent Gaussian states, $z_t \in \mathbb{Z}$ denotes latent discrete states and u_t is a known control signal. The noise processes are iid Gaussian with $w_t \sim N(0, I)$ and $v_t \sim N(0, I)$. This model implies the continuous densities:

$$\begin{aligned} p(x_t|z_t, x_{t-1}) &\sim N(A(z_t)x_{t-1} + F(z_t)u_t, B(z_t)B(z_t)^T) && (334)\\ p(y_t|x_t, z_t) &\sim N(C(z_t)x_t + G(z_t)u_t, D(z_t)D(z_t)^T) && (335) \end{aligned}$$

along with initial states $x_0 \sim N(\mu_0, \Sigma_0)$ and $z_0 \sim P(z_0)$. The aim of the analysis is to compute the marginal posterior distribution of the discrete states $p(z_{0:t}, y_{1:t})$. The posterior density can be factorized as follows:

$$p(x_{1:t}, z_{1:t}|y_{1:t}) = p(x_{1:t}|y_{1:t}, z_{1:t})p(z_{1:t}|y_{1:t}) \tag{336}$$

where the density $p(x_{1:t}|y_{1:t}, z_{1:t})$ is Gaussian and can be computed analytically if we know the marginal posterior density $p(z_{1:t}|y_{1:t})$. We can re-write the marginal posterior recursively:

$$\begin{aligned} p(z_{1:t}|y_{1:t}) &= \frac{p(y_t|z_{1:t}, y_{1:t-1})p(z_{1:t}|y_{1:t-1})}{p(y_t|y_{1:t-1})} && (337)\\ &= \frac{p(y_t|z_t)p(z_t|z_{1:t-1}, y_{1:t-1})p(z_{1:t-1}|y_{1:t-1})}{p(y_t|y_{1:t-1})} && (338)\\ &\propto p(y_t|z_t)p(z_t|z_{t-1})p(z_{1:t-1}|y_{1:t-1}) && (339) \end{aligned}$$

which depends only on the current conditional distributions and previously computed $p(z_{1:t-1}|y_{1:t-1})$. The basic idea is to approximate the belief state of the entire state trajectory using a weighted set of particles:

$$p(z_{1:t}|y_{1:t}) \approx \sum_{s=1}^{S} \hat{w}_t^s \delta_{z_{1:t}^s}(z_{1:t}) \tag{340}$$

We update this belief using importance sampling. If the proposal has the form $q(z_{1:t}^s|y_{1:t})$ then the importance weights are given by:

$$w_t^s \propto \frac{p(z_{1:t}^s|y_{1:t})}{q(z_{1:t}^s|y_{1:t})} \propto \frac{p(y_t|x_t, z_t)p(x_t, z_t|x_{t-1}, z_{t-1})}{q(x_t, z_t|x_{t-1}, z_{t-1})} \tag{341}$$

We can choose the transition prior to be the proposal distribution:

$$q(x_t, z_t|x_{t-1}, z_{t-1}) = p(x_t, z_t|x_{t-1}, z_{t-1}) = p(x_t|x_{t-1}, z_{t-1})p(z_t|z_{t-1}) \tag{342}$$

Algorithm 16 Particle Filter Algorithm [10]

Sequential Importance Sampling Step
for i = 1 to N do
    $z_t^i \sim p(z_t|z_{t-1}^i)$
    $x_t^i \sim p(x_t|x_{t-1}^i, z_t^i)$ as in eq. (334)
end for
for i = 1 to N do
    $w_t^i \propto p(y_t|x_t^i, z_t^i)$
end for
$\hat{w}_t^i = \frac{w_t^i}{\sum_{s'} w_t^{s'}}$
Selection Step
    Multiply particles with respect to importance weights w_t^i
    to obtain N particles $\{x_{1:t}^i, z_{1:t}^i\}_{i=1}^N$
end for

then the importance weights are given by the likelihood function: $w_t \propto p(y_t|x_t, z_t)$. The particle filter algorithm is summarized in Algorithm 16.

The selection step modifies the weighted approximate density p_N to an unweighted density $\hat{p}_N$ by eliminating particles with low importance weights and by multiplying particles with high importance weights. Thus, $p_N(x) = \sum_{i=1}^N w_i \delta(x - x_i)$ is replaced by

$$\hat{p}_N(x) = \sum_{k=1}^N \frac{1}{N} \delta(x - x_k^*) = \sum_{i=1}^N \frac{n_i}{N} \delta(x - x_i) \tag{343}$$

There are many resampling schemes such as multinomial, stratified, systematic and residual. All these algorithms are unbiased and can be implemented in $O(N)$ time.

The Rao-Blackwellized Particle Filter (RBPF) is similar to the PF but we only sample the discrete states. Then for each sample of the discrete states, we update the mean and covariance of the continuous states using Kalman filter updates. In particular, we sample z_t^i and then propagate the mean μ_t^i and covariance Σ_t^i of x_t with a Kalman filter as follows:

$$\mu_{t|t-1}^i = A(z_t^i)\mu_{t-1|t-1}^i + F(z_t^i)u_t \tag{344}$$
$$\Sigma_{t|t-1}^i = A(z_t^i)\Sigma_{t-1|t-1}^i A(z_t^i)^T + D(z_t^i)D(z_t^i)^T \tag{345}$$
$$S_t^i = C(z_t^i)\Sigma_{t|t-1}^i C(z_t^i)^T + D(z_t^i)D(z_t^i)^T \tag{346}$$
$$y_{t|t-1}^i = C(z_t^i)\mu_{t|t-1}^i + G(z_t^i)u_t \tag{347}$$
$$\mu_{t|t}^i = \mu_{t|t-1}^i + \Sigma_{t|t-1}^i C(z_t^i)^T S_t^{-1}(y_t - y_{t|t-1}^i) \tag{348}$$
$$\Sigma_{t|t}^i = \Sigma_{t|t-1}^i - \Sigma_{t|t-1}^i C(z_t^i)^T S_t^{-1} C(z_t^i)\Sigma_{t|t-1}^i \tag{349}$$

The RBPF takes slightly longer to compute but results in more accurate predictions. Figure 59 shows the generated SLDS states (left) and the inferred states (middle) by Particle Filter (PF) and the Rao-Blackwellized version (RBPF). We can see that the inferred states closely correspond to the ground truth. Also shown is a particle resampling step (right) where only a fraction of the particles survive to the next iteration.

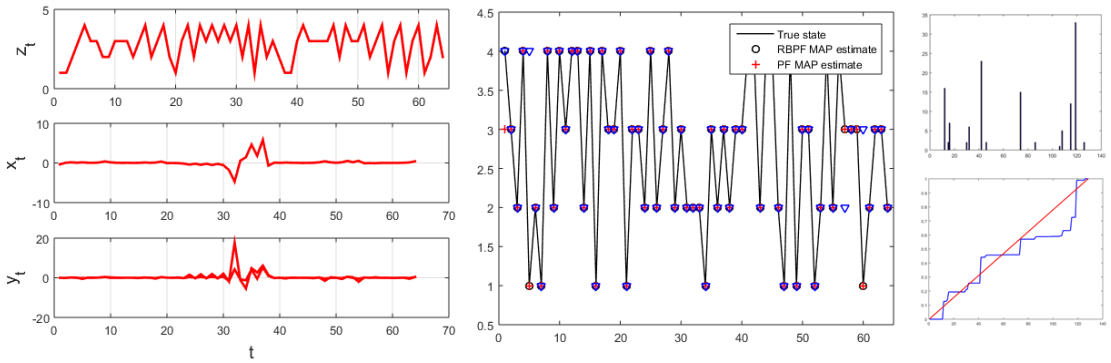

Figure 59: Particle Filter inference using PF and RBPF applied to Switching Linear Dynamic System.

6.2 Information Theory

Information theory addresses the problems of data representation and reliable transmission of information. In order to measure information content, we define several information measures below.

6.2.1 Entropy

The entropy of a random variable is a measure of its uncertainty. Let X be a discrete random variable with probability mass function $p(x)$, then its entropy is defined as:

$$H(X) = -\sum_x p(x) \log p(x) = -E[\log p(x)] \tag{350}$$

A maximum entropy discrete distribution is the uniform distribution. For a random variable with support K and $p(x) = 1/K$, the entropy is $H(X) = -\sum \frac{1}{K} \log \frac{1}{K} = \log_2 K$. Thus, entropy is measured in bits (using log base 2) or nats (using log base e). In contrast, the entropy of a deterministic random variable with $p(x) = \delta[x]$ is $H(X) = 1 \log 1 = 0$. In the special case of a binary random variable $X \in \{0, 1\}$ with $p(x = 1) = p$, the entropy is

$$H(X) = -p \log p - (1 - p) \log(1 - p) \tag{351}$$

It is a concave function with maximum uncertainty of 1 bit occuring at $p = \frac{1}{2}$, when $p = 0$ or $p = 1$, the entropy is 0. Since entropy is a concave function on the space of distributions, we have:

$$H(\lambda p_1 + (1 - \lambda)p_2) \geq \lambda H(p_1) + (1 - \lambda)H(p_2) \tag{352}$$

Proof. To prove that $H(p)$ is concave at p, we can use the log-sum inequality. Let $p_\lambda(y) = \lambda p_1(y) + (1 - \lambda)p_2(y)$then,

$$\begin{aligned} -H(p_\lambda) &= \sum_y p_\lambda(y) \log p_\lambda(y) \\ &= \sum_y \big[\lambda p_1(y) + (1 - \lambda)p_2(y)\big] \log \big[\lambda p_1(y) + (1 - \lambda)p_2(y)\big] \end{aligned} \tag{353}$$

Let's use the log-sum inequality with $K = 2$ and $a_1 = \lambda p_1(y)$, $a_2 = (1-\lambda)p_2(y)$, $b_1 = \lambda$ and $b_2 = (1-\lambda)$:

$$\begin{aligned} -H(p_\lambda) &\leq \lambda p_1(y) \log \frac{\lambda p_1(y)}{\lambda} + (1-\lambda)p_2(y) \log \frac{(1-\lambda)p_2(y)}{1-\lambda} \\ &= \lambda p_1(y) \log p_1(y) + (1-\lambda)p_2(y) \log p_2(y) \\ &= -\big[\lambda H(p_1) + (1-\lambda)H(p_2)\big] \end{aligned} \tag{354}$$

Since $0 \leq p(x) \leq 1$, we have $\log \frac{1}{p(x)} \geq 0$, and for a discrete X, the entropy is non-negative: $H(X) \geq 0$. We can find an upper bound for entropy using the Jensen's inequality:

$$H(X) = \sum_{x \in \mathcal{X}} p(x) \log \frac{1}{p(x)} \tag{355}$$

$$\leq \log \sum_{x \in \mathcal{X}} p(x) \times \frac{1}{p(x)} \tag{356}$$

$$= \log \sum_{x \in \mathcal{X}} 1 \tag{357}$$

$$= \log |\mathcal{X}| \tag{358}$$

We also note that entropy is independent of permutation or cyclical shifts of the support of our distribution, i.e. it only depends on the point masses $p(x)$.

The joint entropy of a pair of discrete random variables X and Y is defined as:

$$H(X,Y) = -\sum_x \sum_y p(x,y) \log p(x,y) = -E[\log p(x,y)] \tag{359}$$

when two variables are independent the joint entropy is additivie: $H(X,Y) = H(X) + H(Y)$ iff $p(x,y) = p(x)p(y)$. The conditional entropy is defined as:

$$H(X|Y) = \sum_y p(y) H(X|Y=y) = -\sum_y p(y) \sum_x p(x|y) \log p(x|y) \tag{360}$$

$$= -\sum_x \sum_y p(x,y) \log p(x|y) = -E[\log p(x|y)] \tag{361}$$

Note that conditioning on Y the uncertainty over X reduces on average: $H(X|Y) \leq H(X)$. The entropy of a pair of variables follows the chain rule:

$$H(X,Y) = H(X) + H(Y|X) = H(Y) + H(X|Y) \tag{362}$$

which follows from the chain rule for probability: $p(x,y) = p(x)p(y|x) = p(y)p(x|y)$. The chain rule can be generalized for multiple random variables $X_1, ..., X_N$:

$$H(X_1, ..., X_N) = \sum_{i=2}^{N} H(X_i|X_1, ..., X_{i-1}) + H(X_1) \leq \sum_{i=1}^{N} H(X_i) \tag{363}$$

where the inequality follows from the fact that conditioning reduces entropy. For continuous random variables, the multivariate Gaussian is the distribution with maximum differential en-

tropy:

$$
\begin{aligned}
h(X_1, ..., X_n) &= \int p(x) \log \frac{1}{p(x)} dx && (364)\\
&= \int p(x) \left[\frac{1}{2} \log(2\pi)^n |\Sigma| + \frac{1}{2} (x-\mu)^T \Sigma^{-1} (x-\mu) \right] dx && (365)\\
&= \frac{1}{2} \log(2\pi)^n |\Sigma| + \frac{1}{2} E[(x-\mu)^T \Sigma^{-1} (x-\mu)] && (366)\\
&= \frac{1}{2} \log(2\pi)^n |\Sigma| + \frac{1}{2} Tr\{\Sigma^{-1}\Sigma\} && (367)\\
&= \frac{1}{2} \log(2\pi)^n |\Sigma| + \frac{1}{2} n \log e = \frac{1}{2} \log \left[(2\pi e)^n |\Sigma| \right] && (368)
\end{aligned}
$$

In information theory, the analog of the law of large numbers is the Asymptotic Equipartition Property (AEP). The AEP states that:

Theorem 6.1 *(AEP) If X_1,X_2,...,X_n are iid $\sim p(x)$ then*

$$
-\frac{1}{n} \log p(X_1, X_2, ..., X_n) \to H(X) \quad \text{in probability} \tag{369}
$$

Proof.

$$
-\frac{1}{n} \log p(X_1, X_2, ..., X_n) = -\frac{1}{n} \sum_i \log p(X_i) \to -E[\log p(X)] = H(X) \tag{370}
$$

Thus, the probability $p(X_1, ..., X_n)$ assigned to an observed sequence will be close to $2^{-nH(X)}$. This enables us to classify the set of all sequences into a typical set, where the sample entropy is close to the true entropy and the nontypical set that contains all other sequences.

6.2.2 KL divergence

One way to measure the similarity between two probability distributions $p(x)$ and $q(x)$ is the Kullback-Leibler divergence or relative entropy. It is defined as follows:

$$
KL(p||q) = \sum_x p(x) \log \frac{p(x)}{q(x)} \tag{371}
$$

we can re-write it as:

$$
KL(p||q) = \sum_x p(x) \log p(x) - \sum_x p(x) \log q(x) = -H(p) + H(p, q) \tag{372}
$$

where $H(p, q)$ is the cross-entropy. The regular entropy can be written as $H(p, p)$ and therefore KL divergence can be seen as a penalty of extra bits needed to encode the data due to the fact that we used a distribution $q(x)$ to represent the data instead of the true distribution $p(x)$.

Theorem 6.2 *(Information Inequality) $KL(p||q) \geq 0$ with equality iff $p = q$.*

Proof.

$$
\begin{aligned}
-KL(p||q) &= -\sum_x p(x) \log \frac{p(x)}{q(x)} = \sum_x p(x) \log \frac{q(x)}{p(x)} && (373)\\
&\leq \log \sum_x p(x) \frac{q(x)}{p(x)} = \log \sum_x q(x) && (374)\\
&\leq \log 1 = 0 && (375)
\end{aligned}
$$

where we used the Jensen's inequality, which states that for any convex function $f(x)$, we have

$$f\Big(\sum_{i=1}^{n}\lambda_i x_i\Big) \leq \sum_{i=1}^{n}\lambda_i f(x_i) \tag{376}$$

where $\lambda_i \geq 0$ and $\sum_{i=1}^{n}\lambda_i = 1$. However, KL divergence is not a true distance between distributions because it is not symmetric ($KL(p||q) \neq KL(q||p)$) and it does not satisfy the triangle inequality. We can use the information inequality to derive an upper bound on entropy. Let $u(x) = 1/|\mathcal{X}|$ be the uniform distribution, then:

$$\begin{aligned} 0 &\leq KL(p||u) = \sum_x p(x)\log\frac{p(x)}{u(x)} && (377)\\ &= \sum_x p(x)\log p(x) - \sum_x p(x)\log u(x) && (378)\\ &= -H(X) + \log|\mathcal{X}| && (379)\end{aligned}$$

Thus, $H(X) \leq \log|\mathcal{X}|$ with equality iff $p(x) = u(x)$.

Before discussing the structure / properties of $D(p||q)$, it's helpful to introduce the log-sum inequality.

Theorem 6.3 *Let $a_1, ..., a_n$ and $b_1, ..., b_n$ be non-negative numbers. Let $a = \sum_{i=1}^{n} a_i$ and $b = \sum_{i=1}^{n} b_i$ then*

$$\sum_{i=1}^{n} a_i \log\frac{a_i}{b_i} \geq a\log\frac{a}{b} \tag{380}$$

with equality iff $\frac{a_i}{b_i}$ are equal for all i.

Proof. Let's define $f(x) = x\log x$, notice that $f(x)$ is convex (since $f''(x) = \frac{1}{x} > 0$ for $x > 0$). Then, we have:

$$\begin{aligned} \sum_{i=1}^{n} a_i\log\frac{a_i}{b_i} &= \sum_{i=1}^{n} b_i f(\frac{a_i}{b_i}) = b\sum_{i=1}^{n}\frac{b_i}{b}f(\frac{a_i}{b_i}) && (381)\\ &\geq bf\Big(\sum_{i=1}^{n}\frac{b_i}{b}\frac{a_i}{b_i}\Big) = bf\Big(\frac{1}{b}\sum_{i=1}^{n}a_i\Big) = bf(\frac{a}{b})\\ &= a\log\frac{a}{b}\end{aligned}$$

Theorem 6.4 *$KL(p||q)$ is convex in the pair (p, q), i.e.*

$$KL(\lambda p_1 + (1-\lambda)p_2||\lambda q_1 + (1-\lambda)q_2) \leq \lambda KL(p_1||q_1) + (1-\lambda)KL(p_2||q_2) \tag{382}$$

Proof. Expanding the inequality above, we want to show that

$$\begin{aligned} &\sum_x(\lambda p_1(x) + (1-\lambda)p_2(x))\log\frac{\lambda p_1(x) + (1-\lambda)p_2(x)}{\lambda q_1(x) + (1-\lambda)q_2(x)} \leq && (383)\\ &\lambda\sum_x p_1(x)\log\frac{p_1(x)}{q_1(x)} + (1-\lambda)\sum_x p_2(x)\log\frac{p_2(x)}{q_2(x)} && (384)\end{aligned}$$

Then for a fixed x, it suffices to show that

$$\sum_{i=1,2}\lambda_i p_i\log\frac{\lambda_i p_i}{\lambda_i q_i} \geq \Big(\sum_{i=1,2}\lambda_i p_i\Big)\Big(\log\frac{\sum_i\lambda_i p_i}{\sum_i\lambda_i q_i}\Big) \tag{385}$$

where $\lambda_1 = \lambda$ and $\lambda_2 = 1 - \lambda$, which is exactly the log-sum inequality. Note that divergence being convex in (p, q) implies that it is convex in p and q individually, i.e. set $p_1 = p_2 = p$ or $q_1 = q_2 = q$.

In order to find a distribution $q(x) \in Q$ that is closest to $p(x) \in P$, we can minimize $KL(q||p)$ with respect to $q(x)$ known as I-projection or information projection:

$$(\text{I} - \text{projection}): \quad q(x) = \arg\min_{q \in Q} KL(q||p) \tag{386}$$

Since the KL divergence is not symmetric in its arguments, the above expression will give different behavior compared to minimizing $KL(p||q)$ known as M-projection or moment projection:

$$(\text{M} - \text{projection}): \quad q(x) = \arg\min_{q \in Q} KL(p||q) \tag{387}$$

Both the I-projection and the M-projection are projections of a probability distribution $p(x)$ onto a set of distributions Q. For I-projection, $q(x)$ will typically under-estimate the support of $p(x)$ and will lock onto one of its modes. This is due to $q(x) = 0$ whenever $p(x) = 0$ to make sure KL divergence stays finite. For M-projection, $q(x)$ will typically over-estimate the support of $p(x)$ and will cover all of its modes. This is due to $q(x) > 0$ whenever $p(x) > 0$ to make sure KL divergence stays finite. The I-projection is useful in setting up information geometry because of the following inequality:

$$KL(q||p) \geq KL(q||p^*) + KL(p^*||p) \tag{388}$$

The inequality can be interpreted as information-geometric version of Pythagoras' triangle inequality theorem, where KL divergence is viewed as squared distance in Euclidean space.
If we are interested in measuring the distance between two multivariate Gaussian distributions with means μ_1 and μ_2 and covariances Σ_1 and Σ_2, the KL divergence can be computed in closed form:

$$\begin{aligned}
KL(p||q) &= \int p(x) \log \frac{p(x)}{q(x)} = \int [\log p(x) - \log q(x)] p(x) dx \\
&= \int \left[\frac{1}{2} \log \frac{|\Sigma_2|}{|\Sigma_1|} - \frac{1}{2}(x - \mu_1)^T \Sigma_1^{-1}(x - \mu_1) + \frac{1}{2}(x - \mu_2)^T \Sigma_2^{-1}(x - \mu_2) \right] p(x) dx \\
&= \frac{1}{2} \log \frac{|\Sigma_2|}{|\Sigma_1|} - \frac{1}{2} Tr\{E[(x - \mu_1)(x - \mu_1)^T \Sigma_1^{-1}]\} + \frac{1}{2} E[(x - \mu_2)^T \Sigma_2^{-1}(x - \mu_2)] \\
&= \frac{1}{2} \log \frac{|\Sigma_2|}{|\Sigma_1|} - \frac{1}{2} Tr\{I_d\} + \frac{1}{2}(\mu_1 - \mu_2)^T \Sigma_2^{-1}(\mu_1 - \mu_2) + \frac{1}{2} Tr\{\Sigma_2^{-1}\Sigma_1\} \\
&= \frac{1}{2} \left[\log \frac{|\Sigma_2|}{|\Sigma_1|} - d + Tr(\Sigma_2^{-1}\Sigma_1) + (\mu_2 - \mu_1)^T \Sigma_2^{-1}(\mu_2 - \mu_1) \right]
\end{aligned} \tag{389}$$

6.2.3 Mutual Information

Mutual information (MI) is a measure of overlap of random variables: amount of information one random variable contains about another random variable. Mutual information measures how similar the joint distribution $p(x, y)$ is compared to the factorized distribution $p(x)p(y)$ when the two variables are independent:

$$I(X; Y) = KL(p(x, y)||p(x)p(y)) = \sum_x \sum_y p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \tag{390}$$

where $I(X;Y) \geq 0$ with equality iff the variables are independent: $p(x,y) = p(x)p(y)$. Mutual information between X and Y can be interpreted as the reduction in uncertainty about X after observing Y or by symmetry, the reduction in uncertainty about Y after observing X:

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) = H(X) + H(Y) - H(X,Y) \tag{391}$$

The relationship between entropy and MI is captured in a Venn diagram in Figure 60. Note

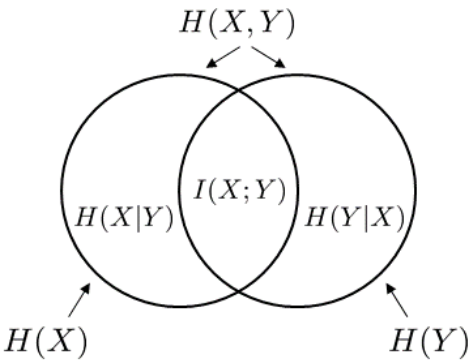

Figure 60: Relationship between entropy and mutual information

that entropy can be viewed as self-information $H(X) = I(X;X)$. The MI is the expected value of pointwise mutual information:

$$PMI(x,y) = \log \frac{p(x,y)}{p(x)p(y)} = \log \frac{p(x|y)}{p(x)} = \log \frac{p(y|x)}{p(y)} \tag{392}$$

This can be interpreted as the amount we learn from updating the prior $p(y)$ into the posterior $p(y|x)$. In fact, we can express MI as follows:

$$I(X;Y) = \sum_x p(x) \sum_y p(y|x) \log \frac{p(y|x)}{p(y)} = E_{p(x)}\big[KL(p(y|x)||p(y))\big] \tag{393}$$

Therefore, the MI between X and Y is equivalent to the expected KL distance between the posterior distribution $p(y|x)$ and the prior $p(y)$.

Theorem 6.5 *(Non-decreasing property of MI). As we add more random variables, the MI can increase or stay the same:*

$$I(X;Y_1) \leq I(X;Y_1,Y_2) \leq I(X;Y_1,...,Y_n) \tag{394}$$

where equality holds when Y_i's are independent of X.

We can compute a variational lower bound on MI using the Gibbs inequality.

Theorem 6.6 *(Variational MI bound). Let $p(x)$ be the original posterior distribution and $q(x)$ be the variational approximation to the posterior. The, the following inequality holds:*

$$I(X;Y) = \max_q \Big[H_p(x) + E_p\big[\log q(x)\big] \Big] \tag{395}$$

Proof. By non-negativity of KL divergence, we can obtain the Gibbs inequality.

$$\mathrm{KL}(p||q) = \sum_x p(x) \log p(x)q(x) = -H(p) + H(p,q) \geq 0 \tag{396}$$

As a result, we have

$$\begin{aligned} -H_p(x) &\geq -H_{p,q}(x) && (397) \\ I(X;Y) = H(X) - H_p(X|Y) &\geq H(X) - H_{p,q}(X) = H_p(x) + E_p\big[\log q(x)\big] && (398) \end{aligned}$$

Theorem 6.7 *(Data Processing Inequality). Let X, Y, Z form the following Markov chain: $X \to Y \to Z$, then the following inequality holds:*

$$I(X;Y) \geq I(X;Z) \tag{399}$$

Proof. By the chain rule, we can expand mutual information in two different ways:

$$I(X;Y,Z) = I(X;Z) + I(X;Y|Z) = I(X;Y) + I(X;Z|Y) \tag{400}$$

Since X is conditionally independent of Z given Y, we have $I(X;Z|Y) = 0$ and therefore

$$I(X;Y) = I(X;Z) + I(X;Y|Z) \tag{401}$$

Since $I(X;Y|Z) \geq 0$, we have the inequality: $I(X;Y) \geq I(X;Z)$. We can use the data processing inequality to show that *sufficient statistics* preserves mutual information. In particular consider the data generating process: $\theta \to X \to T(X)$. Given distribution parameters θ, we generate data by sampling from $f_\theta(x)$ and then we compute a statistic $T(X)$. The statistic $T(X)$ is called sufficient for θ if it contains all the information in X about θ, i.e. the data processing inequality is satisfied with equality:

$$I(\theta;X) = I(\theta;T(X)) \tag{402}$$

Hence sufficient statistics compresses the information about θ using sampled data.

For a bivariate Gaussian distribution we can compute $I(X;Y)$ as follows.
Let $\rho = cov(X,Y)/\sigma_x\sigma_y$, and $\sigma^2 = var(X) = var(Y)$ then $h(X) = h(Y) = \frac{1}{2}\log(2\pi e)\sigma^2$ and

$$h(X,Y) = \frac{1}{2}\log\big[(2\pi e)^2|\Sigma|\big] = \frac{1}{2}\log(2\pi e)^2\sigma^4(1-\rho^2) \tag{403}$$

Therefore,

$$I(X;Y) = h(X) + h(Y) - h(X,Y) = -\frac{1}{2}\log(1-\rho^2) \tag{404}$$

6.3 Imbalanced Learning

Most classification algorithms will only perform optimally when the number of samples in each class is roughly the same. Highly skewed datasets where the minority class is outnumbered by one or more classes commonly occur in fraud detection, medical diagnosis and computational biology. One way of addressing this issue is by re-sampling the dataset to offset the imbalance and arrive at a more robust and accurate decision boundary. Re-sampling techniques can be broadly divided into four categories: undersampling the majority class, over-sampling the minority class, combining over and under sampling, and creating ensemble of balanced datasets [20].

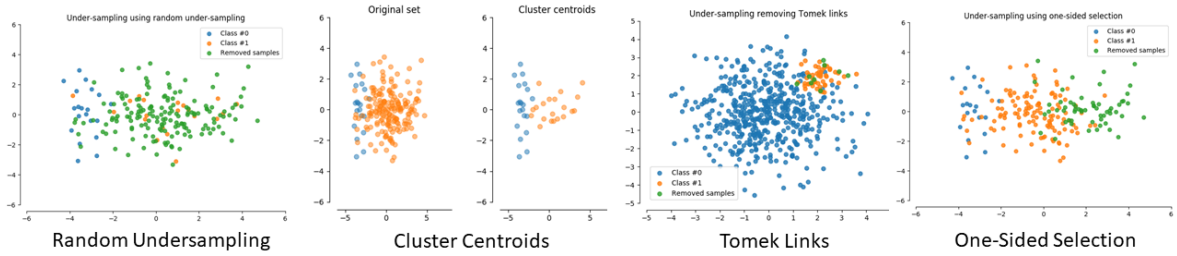

Figure 61: Four undersampling strategies: random, cluster centroids, Tomek links, one-sided selection.

Undersampling strategies. Undersampling methods remove data from the majority class of the original dataset as shown in Figure 61. Random Under Sampler simply removes data points from the majority class uniformly at random. Cluster Centroids is a method that replaces cluster of samples by the cluster centroid of a K-means algorithm, where the number of clusters is set by the level of undersampling. Tomek links remove unwanted overlap between classes where Tomek links are removed until all minimally distanced nearest neighbor pairs are of the same class. A Tomek link is defined as follows: given an instance pair (x_i, x_j), where $x_i \in S_{min}$, $x_j \in S_{maj}$ and $d(x_i, x_j)$ is the distance between x_i and x_j, then the (x_i, x_j) pair is called a Tomek link if there's no instance x_k such that $d(x_i, x_k) < d(x_i, x_j)$ or $d(x_j, x_k) < d(x_i, x_j)$. In this way, if two instances form a Tomek link then either one of these instances is noise or both are near a border. Therefore one can use Tomek links to clean up overlap between classes. By removing overlapping examples, one can establish well-defined clusters in the training set and lead to improved classification performance. The One Sided Selection (OSS) method selects a representative subset of the majority class E and combines it with the set of all minority examples S_{min} to form $N = \{E \cup S_{min}\}$. The reduced set N is further processed to remove all majority class Tomek links.

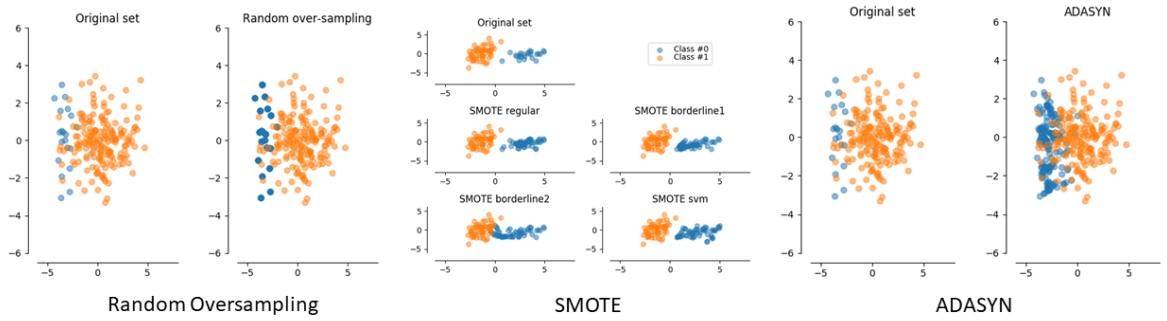

Figure 62: Three oversampling strategies: random, SMOTE, ADASYN.

Oversampling strategies. Oversampling methods append data to the minority class of the original dataset as shown in Figure 62. Random Over Sampler simply adds data points to the minority class uniformly at random. Synthetic Minority Oversampling Technique (SMOTE) generates synthetic examples by finding K nearest neighbors in the feature space and generating a new data point along the line segments joining any of the K minority class nearest neighbors. Synthetic samples are generated in the following way: take the difference between the feature vector (sample) under consideration and its nearest neighbor, multiply this difference by a random number between 0 and 1 and add it to the feature vector under consideration thus augmenting the dataset with a new data point. Adaptive Synthetic Sampling (ADASYN) uses

a weighted distribution for different minority class examples according to their level of difficulty in learning, where more synthetic data is generated for minority class examples that are harder to learn. As a result, the ADASYN approach improves learning of imbalanced dataset in two ways: reducing the bias introduced by class imbalance and adaptively shifting the classification decision boundary toward the difficult examples.

It's possible to combine over-sampling and under-sampling techniques into a hybrid strategy. Common examples include SMOTE and Tomek links or SMOTE and Edited Nearest Neighbors (ENN). Additional ways of learning on imbalanced datasets include weighing training instances, introducing different misclassification costs for positive and negative examples and bootstrapping.

6.4 Ensemble Methods

Ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model in order to decrease variance (bagging), bias (boosting), or improve predictions (stacking). Ensemble methods can be divided into two groups: *sequential* ensemble methods where the base learners are generated sequentially (e.g. AdaBoost) and *parallel* ensemble methods where the base learners are generated in parallel (e.g. Random Forest). The basic motivation of sequential methods is to exploit the dependence between the base learners since the overall performance can be boosted by weighing previously mislabeled examples with higher weight. The basic motivation of parallel methods is to exploit independence between the base learners since the error can be reduced dramatically by averaging.

Most ensemble methods use a single base learning algorithm to produce homogeneous base learners, i.e. learners of the same type leading to *homogeneous ensembles*. There are also some methods that use heterogeneous learners, i.e. learners of different types, leading to *heterogeneous ensembles*. In order for ensemble methods to be more accurate than any of its individual members the base learners have to be as accurate as possible and as diverse as possible.

6.4.1 Bagging

Bagging stands for bootstrap aggregation. One way to reduce the variance of an estimate is to average together multiple estimates. For example, we can train M different trees f_m on different subsets of the data (chosen randomly with replacement) and compute the ensemble:

$$f(x) = \frac{1}{M} \sum_{m=1}^{M} f_m(x) \tag{405}$$

Bagging uses bootstrap sampling to obtain the data subsets for training the base learners. For aggregating the outputs of base learners, bagging uses voting for classification and averaging for regression.

Figure 63 shows the decision boundary of a decision tree and k-NN classifiers along with their bagging ensembles applied to the Iris dataset. The decision tree shows axes parallel boundaries while the $k = 1$ nearest neighbors fits closely to the data points. The bagging ensembles were trained using 10 base estimators with 0.8 subsampling of training data and 0.8 subsampling of features. The decision tree bagging ensemble achieved higher accuracy in comparison to k-NN bagging ensemble because k-NN are less sensitive to perturbation on training samples and therefore they are called *stable learners*. Combining stable learners is less advantageous since

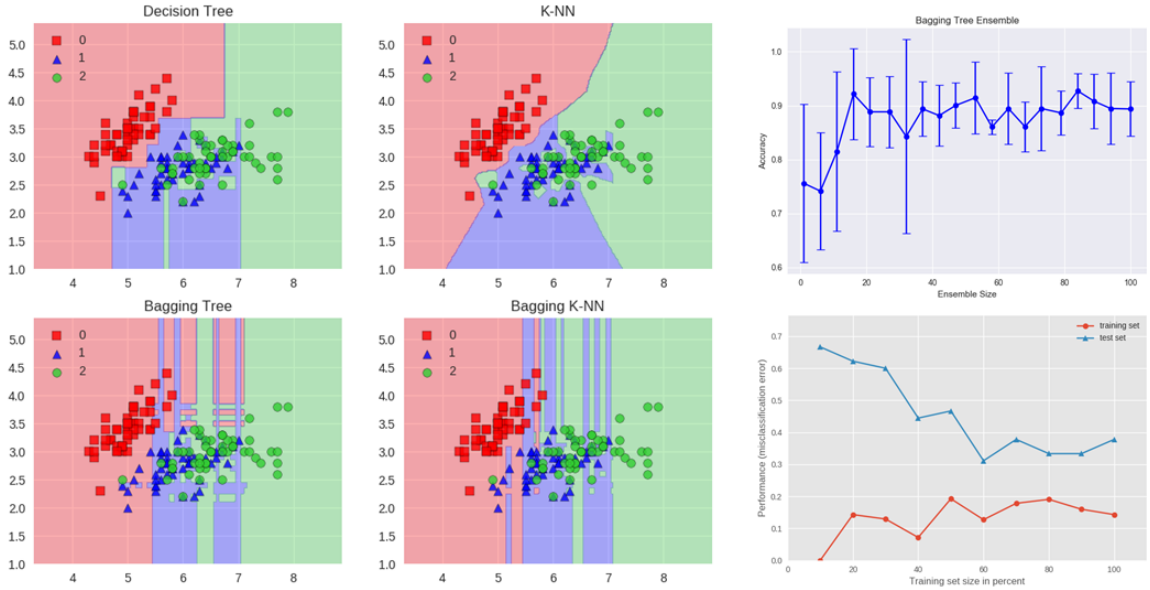

Figure 63: Bagging Ensemble with decision tree and k-NN as base estimator.

the ensemble will not help improve generalization performance. The figure also shows how the test accuracy improves with the size of the ensemble. Based on cross-validation results, we can see the accuracy increases until approximately 20 base estimators and then plateaus afterwards. Thus, adding base estimators beyond 20 only increases computational complexity without accuracy gains for the Iris dataset. The figure also shows learning curves for the bagging tree ensemble. We can see an average error of 0.15 on the training data and a U-shaped error curve for the testing data. The smallest gap between training and test errors occurs at around 80% of the training set size.

A commonly used class of ensemble algorithms are forests of randomized trees. In **random forests**, each tree in the ensemble is built from a sample drawn with replacement (i.e. a bootstrap sample) from the training set. In addition, instead of using all the features, a random subset of features is selected further randomizing the tree. As a result, the bias of the forest increases slightly but due to averaging of less correlated trees, its variance decreases resulting in an overall better model.

In **extremely randomized trees** algorithm randomness goes one step further: the splitting thresholds are randomized. Instead of looking for the most discriminative threshold, thresholds are drawn at random for each candidate feature and the best of these randomly-generated thresholds is picked as the splitting rule. This usually allows to reduce the variance of the model a bit more, at the expense of a slightly greater increase in bias.

6.4.2 Boosting

Boosting refers to a family of algorithms that are able to convert weak learners to strong learners. The main principle of boosting is to fit a sequence of weak learners (models that are only slightly better than random guessing, such as small decision trees) to weighted versions of the data, where more weight is given to examples that were mis-classified by earlier rounds. The predictions are then combined through a weighted majority vote (classification) or a weighted sum (regression)

to produce the final prediction. The principal difference between boosting and the committee methods such as bagging is that base learners are trained in sequence on a weighted version of the data.

Algorithm 17 describes the most widely used form of boosting algorithm called **AdaBoost** which stands for adaptive boosting [13].

Algorithm 17 AdaBoost

1: Init data weights $\{w_n\}$ to $1/N$
2: **for** $m = 1$ to M **do**
3: fit a classifier $y_m(x)$ by minimizing weighted error function J_m:
4: $J_m = \sum_{n=1}^{N} w_n^{(m)} 1[y_m(x_n) \neq t_n]$
5: compute $\epsilon_m = \sum_{n=1}^{N} w_n^{(m)} 1[y_m(x_n) \neq t_n] / \sum_{n=1}^{N} w_n^{(m)}$
6: evaluate $\alpha_m = \log\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$
7: update the data weights: $w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m 1[y_m(x_n) \neq t_n]\}$
8: **end for**
9: Make predictions using the final model: $Y_M(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(x)\right)$

We see that the first base classifier $y_1(x)$ is trained using weighting coefficients $w_n^{(1)}$ that are all equal. In subsequent boosting rounds, the weighting coefficients $w_n^{(m)}$ are increased for data points that are misclassified and decreased for data points that are correctly classified. The quantity ϵ_m represents a weighted error rate of each of the base classifiers. Therefore, the weighting coefficients α_m give greater weight to the more accurate classifiers.

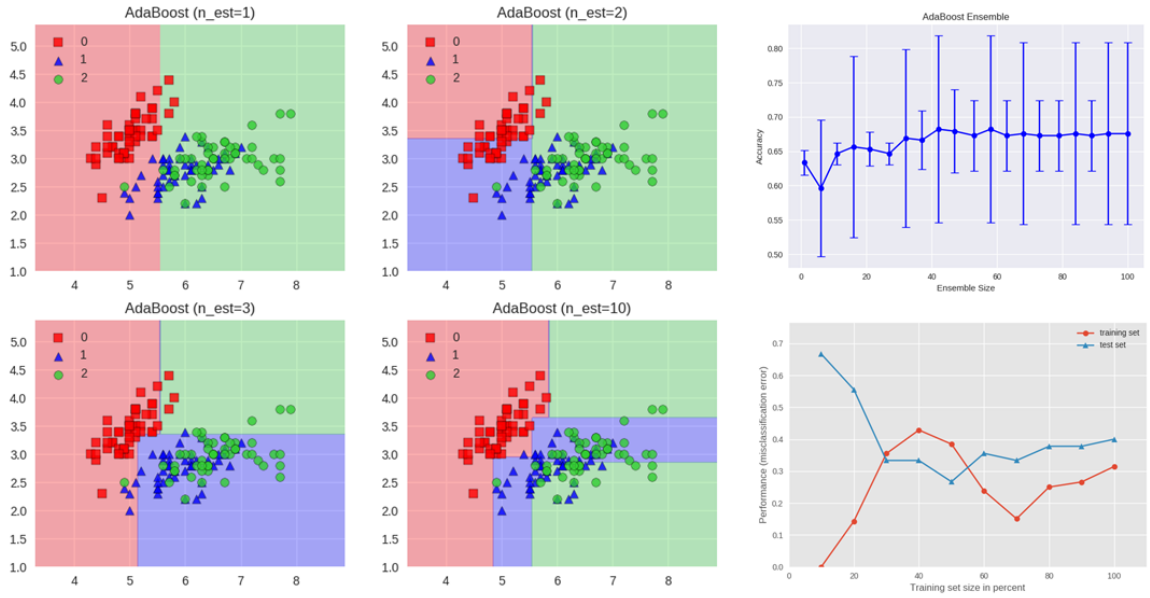

Figure 64: Boosting Ensemble with decision tree as base estimator.

The AdaBoost algorithm is illustrated in Figure 64. Each base learner consists of a decision tree with depth 1, thus classifying the data based on a feature threshold that partitions the space into two regions separated by a linear decision surface that is parallel to one of the axes. The figure also shows how the test accuracy improves with the size of the ensemble and the

learning curves for training and testing data.

Gradient Tree Boosting is a generalization of boosting to arbitrary differentiable loss functions. It can be used for both regression and classification problems. Gradient Boosting builds the model in a sequential way:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x) \tag{406}$$

At each stage the decision tree $h_m(x)$ is chosen to minimize a loss function L given the current model $F_{m-1}(x)$:

$$F_m(x) = F_{m-1}(x) + \arg\min_h \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + h(x_i)) \tag{407}$$

Gradient Boosting attempts to solve this minimization problem numerically via steepest descent. The steepest descent direction is the negative gradient of the loss function evaluated at the current model F_{m-1}:

$$F_m(x) = F_{m-1}(x) + \gamma_m \sum_{i=1}^{n} \nabla_F L(y_i, F_{m-1}(x_i)) \tag{408}$$

where the step size γ_m is chosen using line search. The algorithms for regression and classification differ in the type of loss function used.

6.4.3 Stacking

Stacking is an ensemble learning technique that combines multiple classification or regression models via a meta-classifier or a meta-regressor. The base level models are trained based on complete training set then the meta-model is trained on the outputs of base level model as features. The base level often consists of different learning algorithms and therefore stacking ensembles are often heterogeneous. Algorithm 18 summarizes stacking.

Algorithm 18 Stacking

Input: training data $D = \{x_i, y_i\}_{i=1}^m$
Ouput: ensemble classifier H
Step 1: learn base-level classifiers
for $t = 1$ to T **do**
    learn h_t based on D
end for
Step 2: construct new data set of predictions
for $i = 1$ to m **do**
    $D_h = \{x'_i, y_i\}$, where $x'_i = \{h_1(x_i), ..., h_T(x_i)\}$
end for
Step 3: learn a meta-classifier
learn H based on D_h
return H

The stacking ensemble is illustrated in Figure 65. It consists of k-NN, Random Forest and Naive Bayes base classifiers whose predictions are combined by Logistic Regression as a meta-classifier. We can see the blending of decision boundaries achieved by the stacking classifier. The

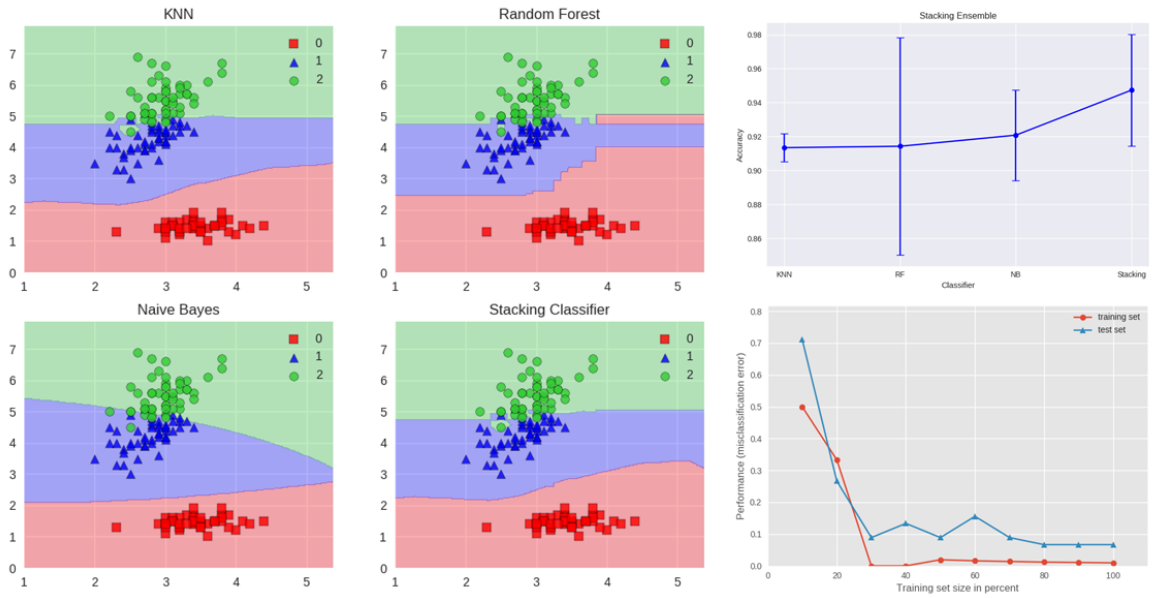

Figure 65: Stacking Ensemble with Logistic Regression as meta-classifier.

figure also shows that stacking achieves higher accuracy than individual classifiers and based on learning curves, it shows no signs of overfitting.

References

[1] M. Becker, B. Hachey, B. Alex, and C. Grover. Optimising selective sampling for bootstrapping named entity recognition. In *ICML*, 2005.

[2] M. Becker and M. Osborne. A two-stage method for active learning of statistical grammars. In *ACL*, 2002.

[3] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[4] D. Blei and J. McAuliffe. Supervised topic models. In *NIPS*, pages 1–8, 2010.

[5] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. In *JMLR*, pages 993–1022, 2003.

[6] T. Campbell, M. Liu, B. Kulis, J. P. How, and L. Carin. Dynamic clustering via asymptotics of the dependent dirichlet process. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[7] K. Chaloner and I. Verdinelli. Bayesian experimental design: a review. In *Statistical Science*, 1995.

[8] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley, 2006.

[9] M. Cuturi. Sinkhorn distances: lightspeed computation of optimal transport. In *NIPS*, 2013.

[10] N. de Freitas. Rao-blackwellized particle filtering for fault diagnosis. In *IEEE Aerospace Conference*, 2002.

[11] F. Doshi, K. T. Miller, J. Van Gael, and Y. W. Teh. Variational inference for the Indian buffet process. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 12, 2009.

[12] V. Federov. *Theory of Optimal Experiments*. Academic Press, 1972.

[13] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *J. Comput. Syst. Sci.*, pages 119–139, 1997.

[14] Y. Gal. Uncertainty in deep learning (phd thesis). In *arXiv*, 2016.

[15] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *ICML*, pages 1–12, 2016.

[16] Y. Gal, R. Islam, and Z. Ghahramani. Deep bayesian active learning with image data. In *arXiv*, pages 1–8, 2017.

[17] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016.

[18] T. L. Griffiths and Z. Ghahramani. The indian buffet process: An introduction and review. In *J. Mach. Learn. Res.*, pages 1185–1224, 2011.

[19] R. Grosse, C. Maddison, and R. Salakhutdinov. Annealing between distributions by averaging moments. In *NIPS*, 2013.

[20] H. He and E. A. Garcia. Learning from imbalanced data. In *IEEE Trans. on Knowl. and Data Eng.*, pages 1263–1284, 2009.

[21] M. Hoffman, F. R. Bach, and D. M. Blei. Online learning for latent dirichlet allocation. In *Advances in Neural Information Processing Systems*, pages 856–864, 2010.

[22] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. In *Journal of Machine Learning Research*, pages 1303–1347, 2013.

[23] P. Jain and P. Kar. *Non-convex Optimization for Machine Learning*. Now Publishers Inc, 2017.

[24] Kaggle. Sentiment analysis of movie reviews dataset. In *Kaggle Datasets*, 2015.

[25] Koller and Friedman. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, 2009.

[26] A. Kraus. Optimizing sensing: Theory and applications. In *PhD Thesis*, 2008.

[27] D. Lewis and W. Gale. A sequential algorithm for training text classifiers. In *ACM SIGIR*, 1994.

[28] D. Lin, W. Grimson, and J. W. Fisher III. Construction of dependent dirichlet processes based on compound poisson processes. In *Neural Information Processing Systems*, 2010.

[29] S. N. MacEachern. Dependent nonparametric processes. In *Proceedings of the Bayesian Statistical Science Section*, 1999.

[30] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *arXiv*, 2013.

[31] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[32] R. Neal. Annealed importance sampling. In *Computational Physics*, 1998.

[33] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions. In *Mathematical Programming*, pages 265–294, 1978.

[34] F. Olsson. A literature of active machine learning in the context of natural language processing. In *Technical Report*, 2009.

[35] B. Pang and L. Lee. A sentimental education: sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the ACL*, 2004.

[36] A. Patil, D. Huard, and C. Fonnesbeck. PyMC: Bayesian stochastic modelling in python. In *Journal of Statistical Software*, pages 1–81, 2010.

[37] J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1–12, 2014.

[38] D. Rezende and S. Mohamed. Variational inference with normalizing flows. In *ICML*, 2015.

[39] E. Ringger, P. McClanahan, R. Haertel, G. Busby, M. Carmen, J. Carroll, K. Seppi, and D. Lonsdale. Active learning for part-of-speech tagging: accelerating corpus annotation. In *ACL*, 2002.

[40] T. Scheffer, C. Decomain, and S. Wrobel. Query by committee. In *IDA*, 2001.

[41] J. Sethuraman. A constructive definition of Dirichlet priors. In *Statistica Sinica*, pages 639–650, 1994.

[42] B. Settles. Active learning literature survey. In *Computer Sciences Technical Report*, pages 1–67, 2009.

[43] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *ACM COLT*, 1992.

[44] S. Skiena. *The Algorithm Design Manual*. Springer, 2008.

[45] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[46] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. In *JMLR*, pages 1929–1958, 2014.

[47] Y. Teh, M. Jordan, M. J. Beal, and D. M. Blei. Hierarchical Dirichlet Processes. In *Journal of the American Statistical Association (JASA)*, 2005.

[48] Y. W. Teh. Dirichlet processes. In *Encyclopedia of Machine Learning*. Springer, 2010.

[49] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. In *JMLR*, 2002.

[50] J. Van Gael, Y. Saatci, Y. W. Teh, and Z. Ghahramani. Beam sampling for the infinite hidden Markov model. In *Proceedings of the International Conference on Machine Learning*, volume 25, 2008.

[51] C. Wang, J. Paisley, and D. Blei. Online variational inference for the hierarchical dirichlet process. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 752–760, 2011.