Research Journal

Vadim Smolyakov (vss@mit.edu)

June 30, 2017

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:

$$p(w|X, y) \propto \exp\{-\frac{1}{2\sigma_n^2}||y - Xw||^2\} \exp\{-\frac{1}{2} w^T \Sigma_p^{-1} w\} \tag{3}$$

$$\propto \exp\{-\frac{1}{2}(w - \bar{w})(\frac{1}{\sigma_n^2} XX^T + \Sigma_p^{-1})(w - \bar{w})\} \tag{4}$$

$$\sim N(\frac{1}{\sigma_n^2} A^{-1} Xy, A^{-1}) \tag{5}$$

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:

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

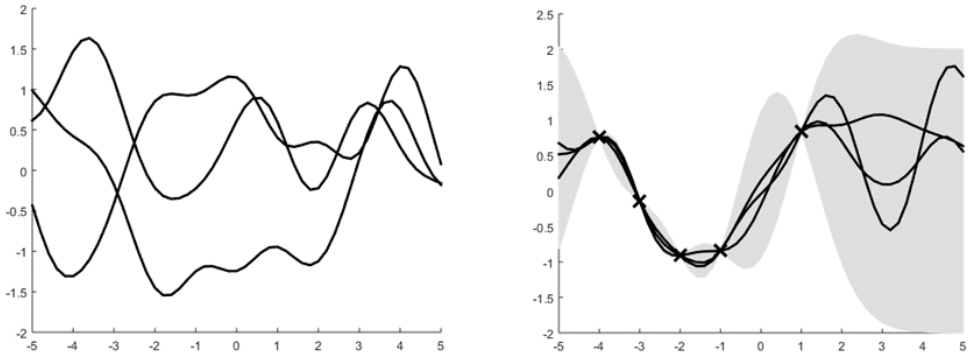

Figure 1: Gaussian process prior (left) and posterior (right).

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 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:

$$p(f_*|X_*, X, y) \sim N(f_*|\mu_*, \Sigma_*) \tag{12}$$

$$\mu_* = K_*^T K_y^{-1} y \tag{13}$$

$$\Sigma_* = K_{**} - K_*^T K_y^{-1} K_* \tag{14}$$

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.

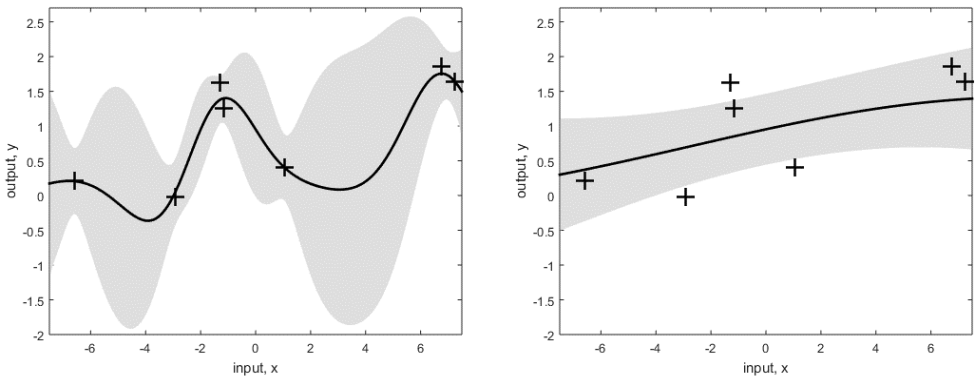

Figure 2: The effect of varying GP hyperparameters

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.

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:

$$(G(A_1), ..., G(A_r)) \sim \text{Dir}(\alpha H(A_1), ..., \alpha H(A_r)) \tag{16}$$

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

$$p(x_1, ..., x_K) = \frac{\Gamma(\sum_k \alpha_k)}{\prod_k \Gamma(\alpha_k)} \prod_{k=1}^{K} x_k^{\alpha_k - 1} \tag{17}$$

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. Let $\theta_1, ..., \theta_n$ be a sequence of independent draws from $G \sim DP(\alpha, H)$. We

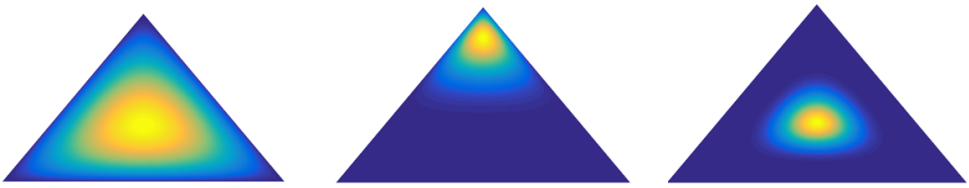

Figure 3: Dirichlet distribution: $\text{Dir}(\alpha_1, \alpha_2, \alpha_3)$ over the simplex.

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:

$$(G(A_1), ..., G(A_r))|\theta_1, ..., \theta_n \sim \text{Dir}(\alpha H(A_1) + n_1, ..., \alpha H(A_r) + n_r) \tag{18}$$

The posterior is also a DP with an updated concentration parameter and base distribution [18]:

$$G|\theta_1, ..., \theta_n \sim DP(\alpha + n, \frac{\alpha}{\alpha + n} H + \frac{n}{\alpha + n} \frac{\sum_{i=1}^{n} \delta_{\theta_i}}{n}) \tag{19}$$

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:

$$\theta_{n+1}|\theta_1, ..., \theta_n \sim \frac{1}{\alpha + n} \left(\alpha H + \sum_{i=1}^{n} \delta_{\theta_i} \right) \tag{20}$$

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.

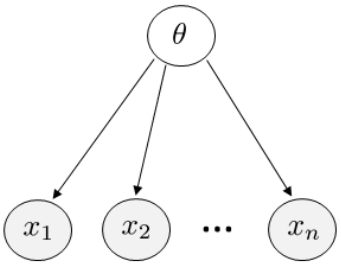

Figure 4: Naive Bayes graphical model

The Blackwell-MacQueen urn model has been used to show the existence of the DP. Starting from (20) we can construct a distribution over draws as follows:

$$P(\theta_1, ..., \theta_n) = \prod_{i=1}^{n} P(\theta_i | \theta_1, ..., \theta_{i-1}) \tag{21}$$

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

$$P(\theta_1, ..., \theta_n) = P(\theta_{\sigma(1)}, ..., \theta_{\sigma(2)}) \tag{22}$$

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:*

$$p(x_1, x_2, ..., x_n) = \int p(\theta) \prod_{i=1}^{n} p(x_i | \theta) d\theta \tag{23}$$

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 4. 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:

$$\theta_{n+1} | \theta_1, ..., \theta_n \sim \frac{1}{\alpha + n} \left(\alpha H + \sum_{k=1}^{m} n_k \delta_{\theta_k^*} \right) \tag{24}$$

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

$$\beta_k \sim \text{Beta}(1, \alpha) \qquad \theta_k^* \sim H \tag{25}$$

$$\pi_k = \beta_k \prod_{l=1}^{k-1}(1 - \beta_l) \qquad G = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k^*} \tag{26}$$

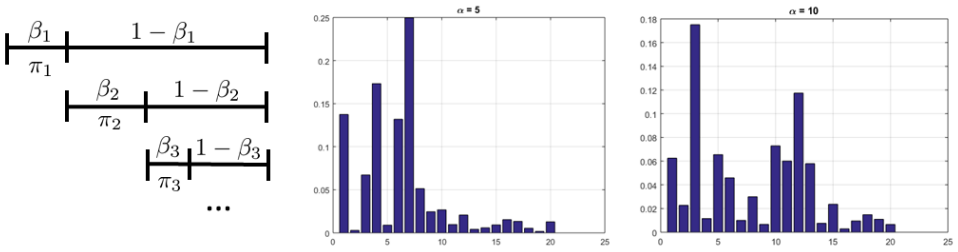

Figure 5: Stick-Breaking Weights

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 5 for $\alpha = 5$ and $\alpha = 10$.

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 6:

$$G_0 | \gamma, H \sim DP(\gamma, H) \tag{27}$$

$$G_j | \alpha, G_0 \sim DP(\alpha, G_0) \tag{28}$$

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

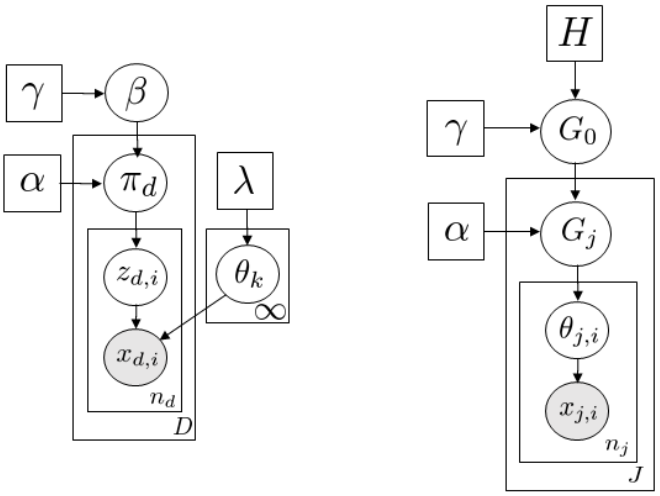

Figure 6: HDP Graphical Model

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 [17]:

$$\beta'_k \sim \text{Beta}(1, \gamma) \qquad \beta_k = \beta'_k \prod_{l=1}^{k-1} (1 - \beta'_l) \tag{29}$$

$$\pi'_{jk} \sim \text{Beta}(\alpha\beta_k, \alpha(1 - \sum_{l=1}^{k} \beta_l)) \qquad \pi_{jk} = \pi'_{jk} \prod_{l=1}^{k-1} (1 - \pi'_{jl}) \tag{30}$$

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 7

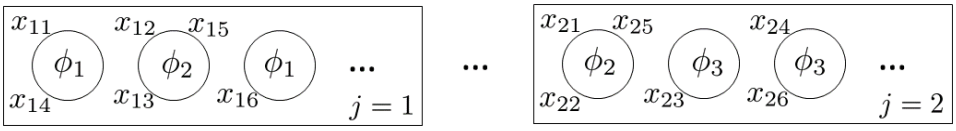

Figure 7: Chinese Restaurant Franchise (CRF). Each restaurant is represented by a rectangle. Customers x_{ji} are seated at tables. At each table a dish ϕ_k is served from a global menu.

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 (19):

$$\theta_{ji}|\theta_{j1},...,\theta_{jn},\alpha,G_0 \sim \sum_{t=1}^{m_{j.}} \frac{n_{jt.}}{n+\alpha}\delta_{\psi_{jt}} + \frac{\alpha}{n+\alpha}G_0 \tag{31}$$

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 (19) again:

$$\psi_{jt}|\psi_{11},\psi_{12},...,\psi_{21},...,\gamma,H \sim \sum_{k=1}^{K} \frac{m_{.k}}{m_{..}+\gamma}\delta_{\phi_k} + \frac{\gamma}{m_{..}+\gamma}H \tag{32}$$

To summarize, for each j and i, we first sample θ_{ji} using (31). If a new sample from G_0 is needed, we use (32) to obtain a new sample ψ_{jt}.

An on-line variational inference algorithm for Hierarchical Dirichlet Process [20] 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 8 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.

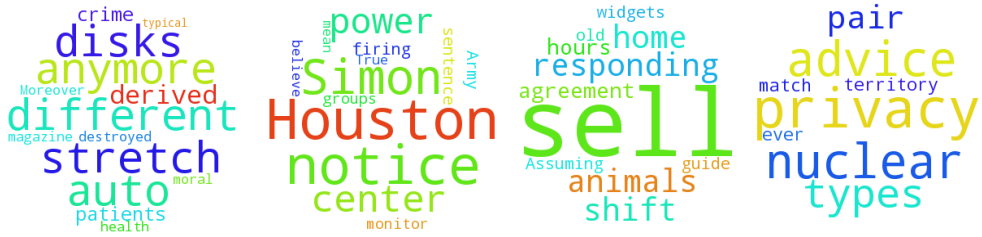

Figure 8: Sample of HDP topics inferred using online variational bayes algorithm on 20newsgroups dataset.

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.

We can describe the HDP-HMM model in Figure 9 using the stick-breaking construction. The parameters have the following distribution:

$$\beta|\gamma \sim \text{GEM}(\gamma) \quad \pi_k|\alpha,\beta \sim DP(\alpha,\beta) \quad \phi_k|H \sim H \tag{33}$$

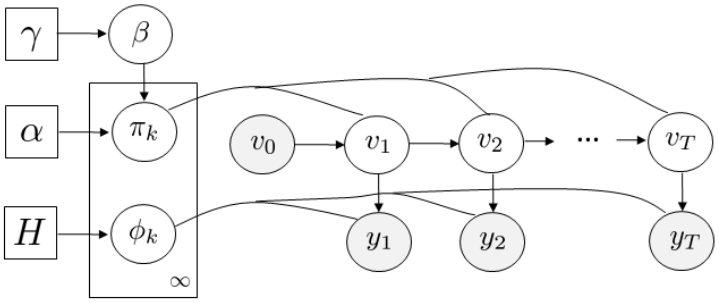

Figure 9: HDP-HMM graphical model

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

$$v_t|v_{t-1}, \pi_k \sim \pi_{v_{t-1}} \qquad y_t|v_t, \phi_k \sim F(\phi_{v_t}) \tag{34}$$

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 [19]. 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.

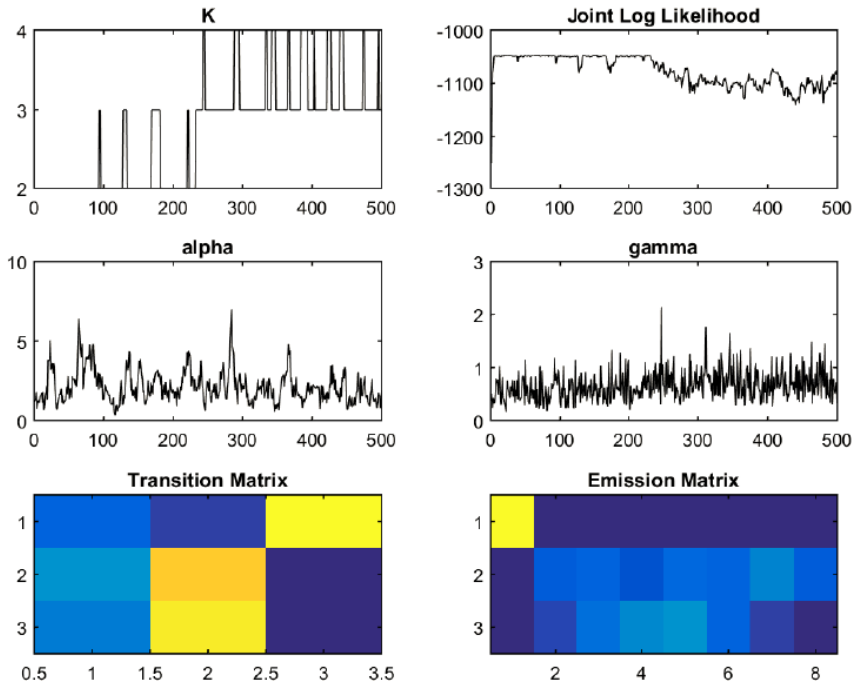

Figure 10: HDP-HMM inference results using the beam sampling algorithm [19].

Figure 10 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 10, 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 [11] provides a non-parametric prior over evolving mixture models. A construction of the DDP built on Poisson process [10] 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 [2].

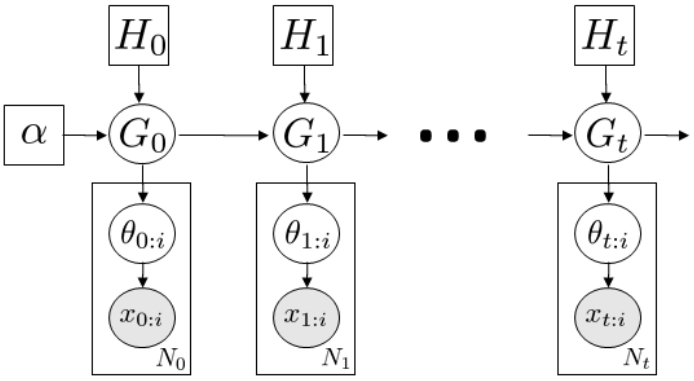

Figure 11: Dependent Dirichlet Process (DDP) graphical model.

Under time-varying setting, it is natural to introduce different DP priors for different time steps as shown in Figure 11. The generative model can be written as:

$$D_t \sim DP(\alpha, H_t) \tag{35}$$

$$\theta_{t:i}|D_t \sim D_t \quad fori = 1, ..., n_t,\ t = 0, ..., T \tag{36}$$

$$x_{t:i}|\theta_{t:i} \sim F(\theta_{t:i}) \quad fori = 1, ..., n_t,\ t = 0, ..., T \tag{37}$$

A Poisson-based construction of DDP [10] 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:

$$D_t = T(S_q(D_{t-1})) \bigoplus H_t, \quad whereH_t \sim DP(\alpha, H) \tag{38}$$

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 [6]:

$$p(Z) = \frac{\alpha^K}{\prod_{i=1}^{N} K_1^{(i)}!} \exp\{-\alpha H_N\} \prod_{k=1}^{K} \frac{(N - m_k)!(m_k - 1)!}{N!} \tag{39}$$

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.

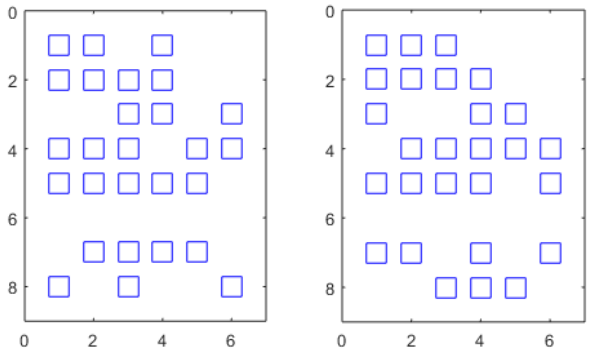

Figure 12: Original binary feature matrix (left) and its left-ordered form (right)

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 12

In this section, we focus on variational inference procedures for the linear-Gaussian likelihood model [4]. 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:

$$X_{N\times D} = Z_{N\times K} \times A_{K\times D} + \epsilon \tag{40}$$

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 13 shows the predicted binary feature matrix Z and lower bound

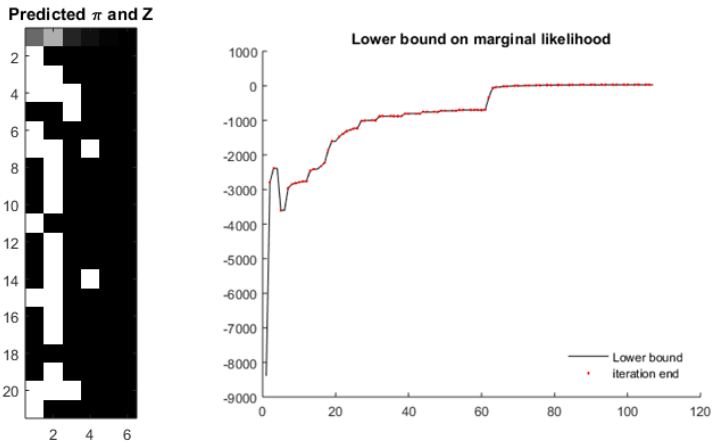

Figure 13: Predicted IBP binary feature matrix Z using variational inference.

on marginal likelihood using variational inference algorithm for IBP [4]. The concentration parameter was set to $\alpha = 1$ and the feature truncation level was set to $K = 6$.

1.7 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.

Consider the graphical model of the DPMM in Figure 14. For each data point x_i, there's a

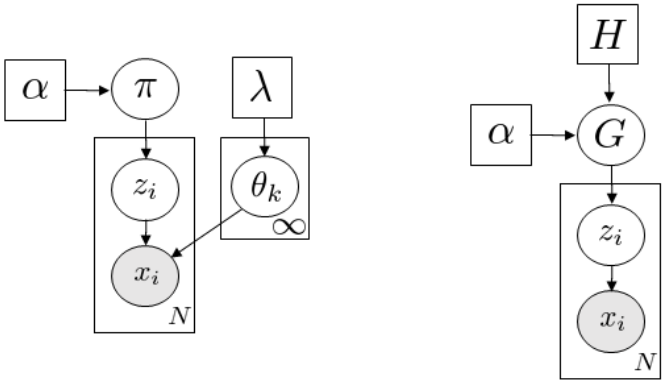

Figure 14: DPMM Graphical Model

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:

$$\begin{aligned} p(x_i|z_i = k, \theta) &= p(x_i|\theta_k) && (41)\\ p(z_i = k|\pi) &= \pi_k && (42)\\ p(\pi|\alpha) &= \text{Dir}(\pi|\alpha) && (43)\\ p(\theta_k|\beta) &= \text{NIW}(\mu_k, \Sigma_k|m_0, \kappa_0, \nu_0, S_0) && (44) \end{aligned}$$

An equivalent representation of this model can be written as:

$$G(\theta) = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k} \tag{45}$$

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*:

$$\beta_k \sim \text{Beta}(1, \alpha) \tag{46}$$

$$\pi_k = \beta_k \prod_{l=1}^{k-1}(1 - \beta_l) = \beta_k(1 - \sum_{l=1}^{k-1} \pi_l) \tag{47}$$

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:

$$E[K] = \alpha \times \log(1 + n/\alpha) \tag{48}$$

$$VAR[K] = \alpha \times \log(1 + n/\alpha) \tag{49}$$

Thus, the number of clusters grows logarithmically with the number of data points and in direct proportion to α as shown in Figure 15

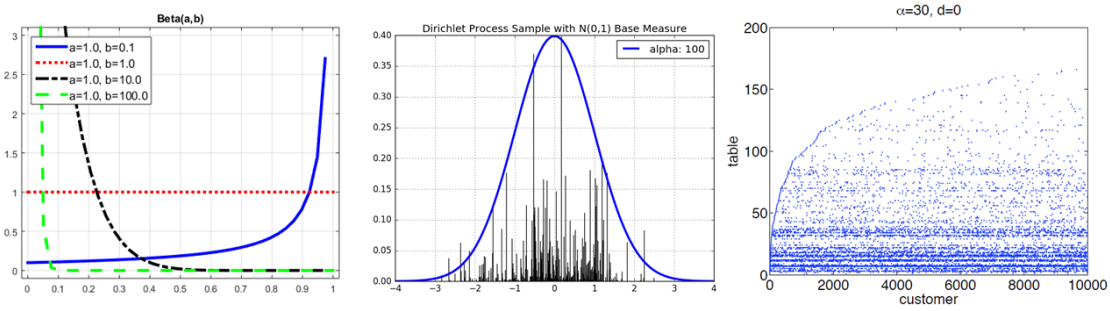

Figure 15: a) Beta$(1, \alpha)$ b)DP samples c) DP cluster growth

1.8 Fitting a DPMM model

One way to fit a DPMM is to modify the collapsed Gibbs sampler for a finite mixture model [12]:

$$p(z_i = k|z_{-i}, x, \alpha, \beta) \propto p(z_i = k|z_{-i}, \alpha)p(x_i|x_{-i}, z_i = k, z_{-i}, \beta) \tag{50}$$

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

$$p(z_i = k|z_{-i}, \alpha) = \begin{cases} \frac{N_k}{N+\alpha-1} & \text{if } k \text{ is occupied} \\ \frac{\alpha}{N+\alpha-1} & \text{if } k \text{ is a new cluster} \end{cases} \tag{51}$$

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

$$p(x_i|x_{-i}, z_i = k, z_{-i}, \beta) = p(x_i|x_{k\setminus i}, \beta) = \frac{p(x_k|\beta)}{p(x_{k\setminus i}|\beta)} \tag{52}$$

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

$$p(x) = \int_\mu \int_\Sigma p(x, \mu, \Sigma) d\mu d\Sigma = \int_\mu \int_\Sigma p(x|\mu, \Sigma) p(\mu, \Sigma|\beta) d\mu d\Sigma \tag{53}$$

$$= (2\pi)^{-ND/2} \frac{Z_{NIW}(D, \kappa_N, \nu_N, S_N)}{Z_{NIW}(D, \kappa_0, \nu_0, S_0)} = \pi^{-ND/2} \frac{\kappa_0^{D/2} |S_0|^{\nu_0/2}}{\kappa_N^{D/2} |S_N|^{\nu_N/2}} \prod_{i=1}^{D} \frac{\Gamma(\frac{\nu_N + 1 - i}{2})}{\Gamma(\frac{\nu_0 + 1 - i}{2})} \tag{54}$$

Therefore, we can write the second term as follows:

$$\log p(x_i|x_{k\setminus i}) = z(D, N+1, \kappa_{N+1}, \nu_{N+1}, S_{Ni}) - z(D, N, \kappa_N, \nu_N, S_N) \tag{55}$$

$$z(D, N, \kappa, \nu, S) = -\frac{ND}{2} \log \pi - \frac{D}{2} \log \kappa - \frac{\nu}{2} \log |S| + \sum_{i=1}^{D} \log \Gamma(\frac{\nu + i - 1}{2}) \tag{56}$$

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 16 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 17 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 17. 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.

2 Variational Inference

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

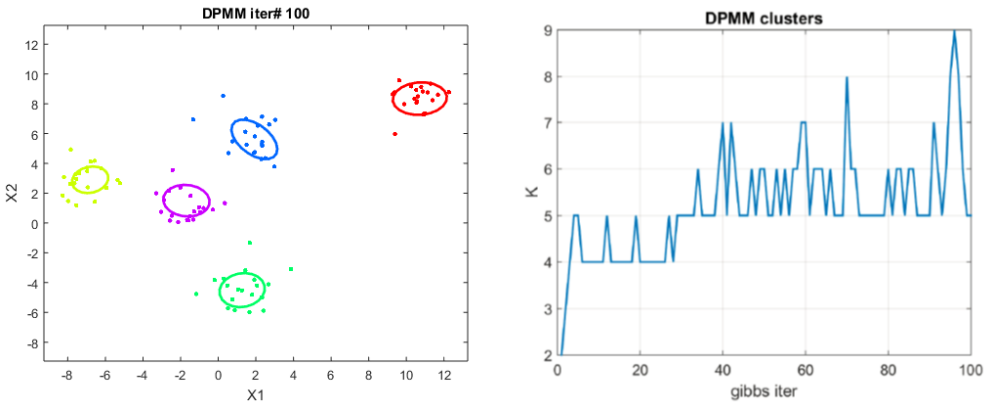

Figure 16: DPMM Clustering Results for Gaussian data

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

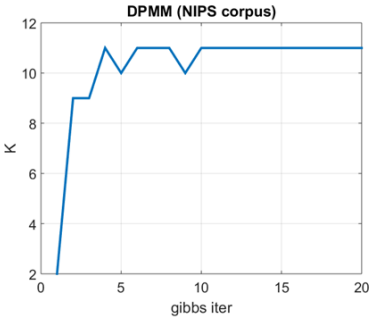

Figure 17: DPMM Clustering Results for Categorical data

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.

$$KL(q||p^*) = \sum_x q(x) \log \frac{q(x)}{p^*(x)} \tag{57}$$

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

$$J(q) = KL(q||\tilde{p}) = \sum_x q(x) \log \frac{q(x)}{p^*(x)Z} = \sum_x q(x) \log \frac{q(x)}{p^*(x)} - \log Z = KL(q||p^*) - \log Z \tag{58}$$

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

$$J(q) = KL(q||p^*) - \log Z \geq -\log Z = -\log p(D) \tag{59}$$

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:

$$\min_q J(q) = E_q[\log q(x)] + E_q[-\log \tilde{p}(x)] = -H(q) + E_q[E(x)] \tag{60}$$

The variational objective function (60) 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.

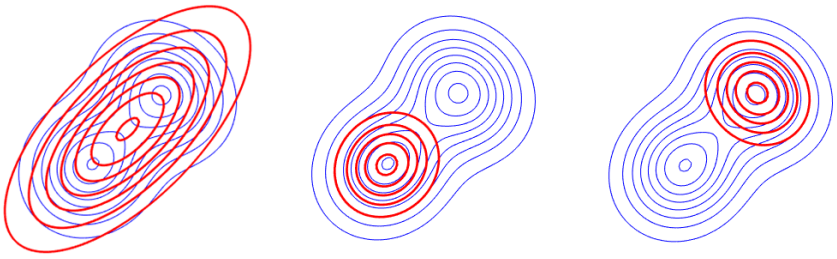

Figure 18: Forward vs reverse KL on a bimodal distribution. The blue contours is the true distribution $p(x)$. The red contours is approximate distribution $q(x)$.

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 18.

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:

$$q(x) = \prod_i q_i(x_i) \tag{61}$$

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:

$$L(q) = -J(q) = \sum_x q(x) \log \frac{\tilde{p}(x)}{q(x)} \tag{62}$$

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

$$\begin{aligned}
L(q_j) &= \sum_x \prod_i q_i(x_i) \Big[\log \tilde{p}(x) - \sum_k \log q_k(x_k) \Big] && (63)\\
&= \sum_{x_j} \sum_{x_{-j}} q_j(x_j) \prod_{i \neq j} q_i(x_i) \Big[\log \tilde{p}(x) - \sum_k \log q_k(x_k) \Big] && (64)\\
&= \sum_{x_j} q_j(x_j) \log f_j(x_j) - \sum_{x_j} q_j(x_j) \sum_{x_{-j}} \prod_{i \neq j} q_i(x_i) \Big[\sum_{k \neq j} \log q_k(x_k) + \log q_j(x_j) \Big] && (65)\\
&= \sum_{x_j} q_j(x_j) \log f_j(x_j) - \sum_{x_j} q_j(x_j) \log q_j(x_j) + \text{const} && (66)
\end{aligned}$$

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:

$$\log q_j(x_j) = \log f_j(x_j) = E_{-q_j}[\log \tilde{p}(x)] \tag{67}$$

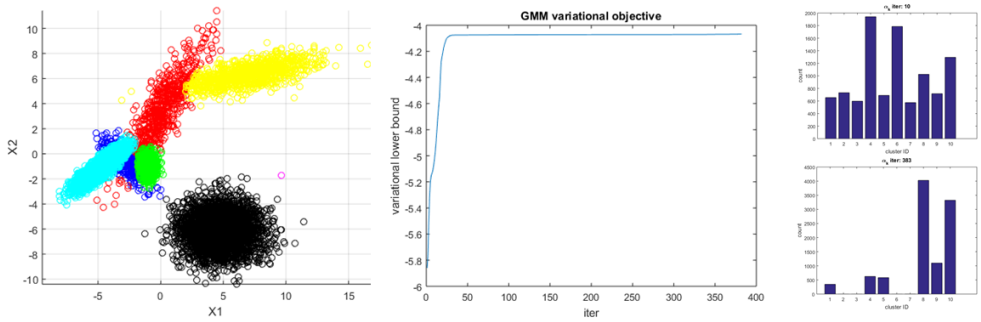

Figure 19: Variational Bayes EM applied to a Mixture of Gaussians

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

Figure 19 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.

2.1 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 20 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

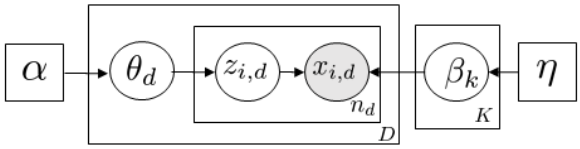

Figure 20: Latent Dirichlet Allocation (LDA) graphical model

model. The full generative model can be specified as follows:

$$\begin{aligned} \theta_d|\alpha &\sim \text{Dir}(\alpha) && (68)\\ z_{i,d}|\theta_d &\sim \text{Cat}(\theta_d) && (69)\\ \beta_k|\eta &\sim \text{Dir}(\eta) && (70)\\ x_{i,d}|z_{i,d}=k,\beta &\sim \text{Cat}(\beta_k) && (71) \end{aligned}$$

The joint distribution for a single document d can be written as follows [1]:

$$p(x,z,\theta,|\alpha,\beta) = p(\theta_d|\alpha)\prod_{i=1}^{n_d} p(z_{i,d}|\theta_d)p(x_{i,d}|z_{i,d},\beta) \tag{72}$$

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:

$$q(\theta,z|\gamma,\phi) = q(\theta|\gamma)\prod_{i=1}^{n} q(z_i|\phi_i) \tag{73}$$

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

$$\log p(x|\alpha,\eta) \geq L(x,\phi,\gamma,\lambda) = E_q[\log p(x,z,\theta,\beta|\alpha,\eta)] - E_q[\log q(z,\theta,\beta)] \tag{74}$$

We choose a fully factored distribution q of the form:

$$q(z_{id}=k) = \phi_{dwk}; \quad q(\theta_d) \sim \text{Dir}(\theta_d|\gamma_d); \quad q(\beta_k) \sim \text{Dir}(\beta_k|\lambda_k) \tag{75}$$

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 [1] as follows:

$$
\begin{aligned}
L(\gamma, \phi; \alpha, \beta) &= \log\Gamma(\sum_{j=1}^{k} \alpha_j) - \sum_{i=1}^{k} \log\Gamma(\alpha_i) + \sum_{i=1}^{k} (\alpha_i - 1)(\Psi(\gamma_i) - \Psi(\sum_{j=1}^{k} \gamma_j)) && (76)\\
&+ \sum_{n=1}^{N} \sum_{i=1}^{k} \phi_{ni} (\Psi(\gamma_i) - \Psi(\sum_{j=1}^{k} \gamma_j)) && (77)\\
&+ \sum_{n=1}^{N} \sum_{i=1}^{k} \sum_{j=1}^{V} \phi_{ni} w_n^j \log \beta_{ij} && (78)\\
&- \log\Gamma(\sum_{j=1}^{k} \gamma_j) + \sum_{i=1}^{k} \log\Gamma(\gamma_i) - \sum_{i=1}^{k} (\gamma_i - 1)(\Psi(\gamma_i) - \Psi(\sum_{j=1}^{k} \gamma_j)) && (79)\\
&- \sum_{n=1}^{N} \sum_{i=1}^{k} \phi_{ni} \log \phi_{ni} && (80)
\end{aligned}
$$

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 [1]:

$$
\begin{aligned}
\phi_{dwk} &\propto \exp\{E_q[\log \theta_{dk}] + E_q[\log \beta_{kw}]\} && (81)\\
\gamma_{dk} &= \alpha + \sum_{w} n_{dw} \phi_{dwk} && (82)\\
\lambda_{kw} &= \eta + \sum_{d} n_{dw} \phi_{dwk} && (83)
\end{aligned}
$$

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

$$
E_q[\log \theta_{dk}] = \Psi(\gamma_{dk}) - \Psi(\sum_{i=1}^{K} \gamma_{di}) \quad E_q[\log \beta_{kw}] = \Psi(\lambda_{kw}) - \Psi(\sum_{i=1}^{W} \lambda_{ki}) \tag{84}
$$

The variational parameter updates in (81) 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 2.

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 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 21 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 21 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:

$$
\text{Perplexity}(\text{w}_{\text{test}}) = \exp\{-\frac{1}{D_{test}} \sum_{d} \frac{1}{n_d} \sum_{w \in n_d} \log p(w_{test})\} \tag{85}
$$

Algorithm 2 Online variational Bayes for LDA [8]

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

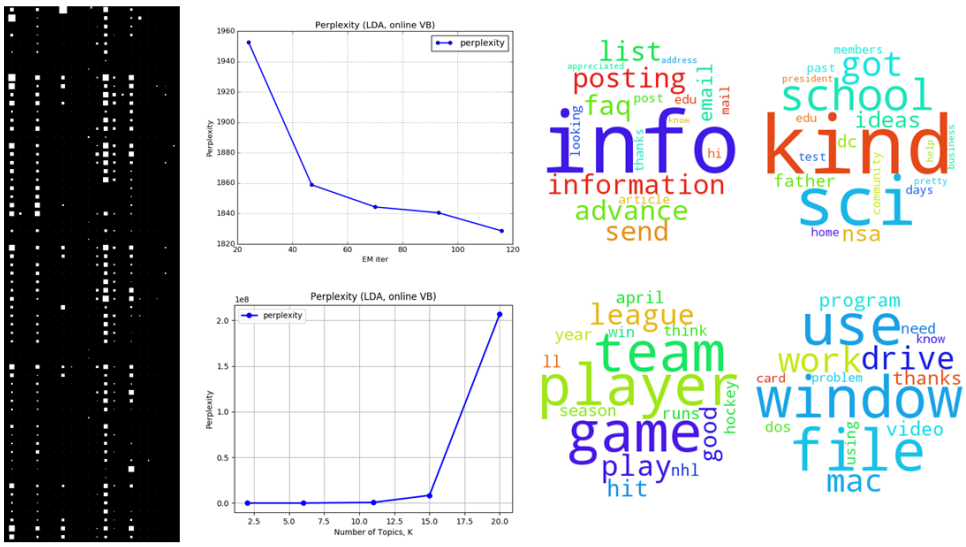

Figure 21: Online Variational Bayes Inference Results for LDA.

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 21. The four topics are about information, sports, computers and school.

2.2 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.

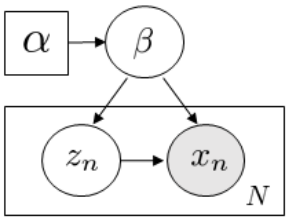

Figure 22: Graphical model with local and global latent variables.

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 22. The join distribution factorizes into a product of a global term β and local term z_n:

$$p(x, z, \beta|\alpha) = p(\beta|\alpha) \prod_{n=1}^{N} p(x_n, z_n|\beta) \tag{86}$$

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):

$$\begin{aligned}
\log p(x) &= \log \int p(x, z, \beta) dz d\beta && (87)\\
&= \log \int p(x, z, \beta) \frac{q(z, \beta)}{q(z, \beta)} dz d\beta && (88)\\
&= \log \left(E_q \left[\frac{p(x, z, \beta)}{q(z, \beta)} \right] \right) && (89)\\
&\geq E_q[\log p(x, z, \beta)] - E_q[\log q(z, \beta)] && (90)\\
&= L(q) && (91)
\end{aligned}$$

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:

$$\begin{aligned} \min KL(q(z,\beta)||p(z,\beta|x)) &= E_q[\log q(z,\beta)] - E_q[\log p(z,\beta|x)] && (92)\\ &= E_q[\log q(z,\beta)] - E_q[\log p(x,z,\beta)] + \log p(x) && (93)\\ &= -L(q) + \text{const} && (94) \end{aligned}$$

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:

$$q(z,\beta) = q(\beta|\lambda)\prod_{n=1}^{N}\prod_{j=1}^{J} q(z_{nj}|\phi_{nj}) \tag{95}$$

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}:

$$\begin{aligned} q(\beta|\lambda) &= h(\beta)\exp\{\lambda^T t(\beta) - a_g(\lambda)\} && (96)\\ q(z_{nj}|\phi_{nj}) &= h(z_{nj})\exp\{\phi_{nj}^T t(z_{nj} - a_l(\phi_{nj}))\} && (97) \end{aligned}$$

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

$$-E_q[\log q(z,\beta)] = -E_\lambda[\log q(\beta)] - \sum_{n=1}^{N}\sum_{j=1}^{J} E_{\phi_{nj}}[\log q(z_{nj})] \tag{98}$$

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:

$$\begin{aligned} L(\lambda) &= E_q[\log p(x,z,\beta)] - E_q[\log q(\beta)] + \text{const} && (99)\\ &= E_q[\log p(\beta|x,z)] - E_q[\log q(\beta)] + \text{const} && (100) \end{aligned}$$

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 [9]:

$$\nabla_\lambda L = \nabla^2_\lambda a_g(\lambda)(E_q[\eta_g(x,z,\alpha)] - \lambda) \tag{101}$$

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:

$$\nabla_{\phi_{nj}} L = \nabla^2_{\phi_{nj}} a_l(\phi_{nj})(E_q[\eta_l(x_n, z_{n,-j}, \beta)] - \phi_{nj}) \tag{102}$$

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

Algorithm 3 Coordinate Ascent SVI [9]

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

local parameter and the global parameter. The full algorithm is described in Algorithm 3 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 [17]. 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.

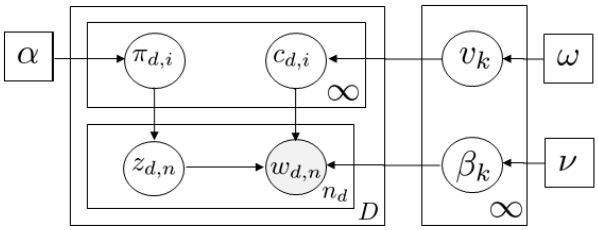

Figure 23: HDP graphical model for SVI inference.

Figure 23 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:

$$q(\beta, \nu, z, \pi) = \Big(\prod_{k=1}^{K} q(\beta_k|\lambda_k) q(\nu_k|a_k) \Big) \Big(\prod_{d=1}^{D} \prod_{i=1}^{T} q(c_{di}|\zeta_{di}) q(\pi_{di}|\gamma_{di}) \prod_{n=1}^{N} q(z_{dn}|\phi_{dn}) \Big) \tag{103}$$

The Stochastic Variational Inference (SVI) algorithm is summarized in Algorithm 4.

Algorithm 4 Stochastic Variational Inference for HDP [9]

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 4 are computed as follows:

$$E[\log \beta_{kv}] = \Psi(\lambda_{kv}) - \Psi(\sum_{v'} \lambda_{kv'}) \tag{104}$$

$$E[\log \sigma_k(V)] = \Psi(a_k) - \Psi(a_k + b_k) + \sum_{l=1}^{k-1} [\Psi(b_l) - \Psi(a_l + b_l)] \tag{105}$$

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 24 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.

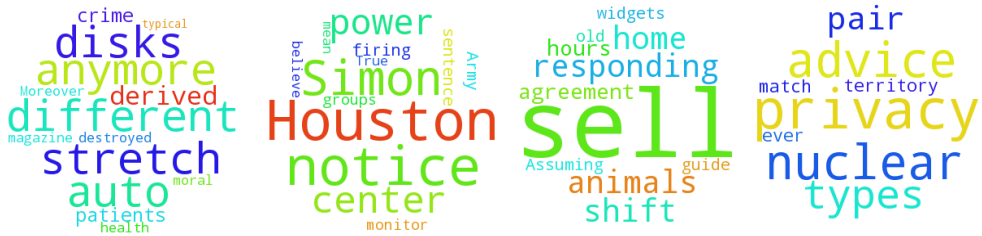

Figure 24: Sample of HDP topics inferred using online variational bayes algorithm on 20newsgroups dataset.

3 Optimization

3.1 Simulated Annealing

Simulated annealing 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:

$$p(x) \propto \exp\{-f(x)/T\} \tag{106}$$

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:

$$x' = x_k + \epsilon_k, \quad \text{where } \epsilon_k \sim N(0, \Sigma) \tag{107}$$

Having proposed a new state, we compute α as in Algorithm 5. Thus, if a new state has

Algorithm 5 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.

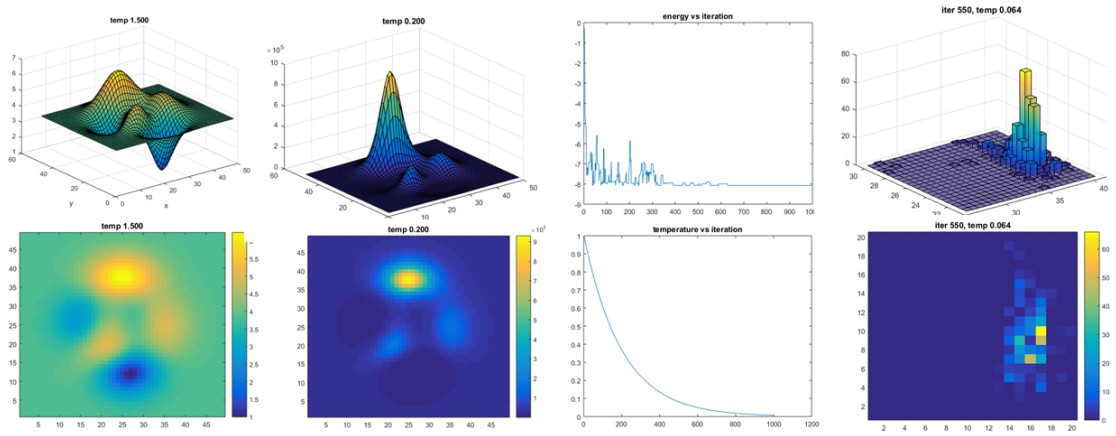

Figure 25: Simulated Annealing

Figure 25 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).

3.2 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 [16].

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: $f(x) \sim \mathrm{GP}(m(x), \kappa(x, x'))$. 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:

$$a_{PI}(x; \{x_n, y_n\}, \theta) = \Phi(\gamma(x)), \quad \gamma(x) = \frac{f(x_{best}) - \mu(x; \{x_n, y_n\}, \theta)}{\sigma(x; \{x_n, y_n\}, \theta)} \tag{108}$$

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

$$a_{EI}(x; \{x_n, y_n\}, \theta) = \sigma(x; \{x_n, y_n\}, \theta)(\gamma(x)\Phi(\gamma(x)) + N(\gamma(x); 0, 1)) \tag{109}$$

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:

$$a_{UCB}(x; \{x_n, y_n\}, \theta) = \mu(x; \{x_n, y_n\}, \theta) - \kappa\sigma(x; \{x_n, y_n\}, \theta) \tag{110}$$

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:

$$K_{M52}(x, x') = \theta_0\left(1 + \sqrt{5r^2(x, x')} + \frac{5}{3}r^2(x, x')\right)\exp\{-\sqrt{5r^2(x, x')}\} \tag{111}$$

$$r^2(x, x') = \sum_{d=1}^{D}(x_d - x'_d)^2/\theta_d^2 \tag{112}$$

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 6.

Algorithm 6 Bayesian Optimization

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

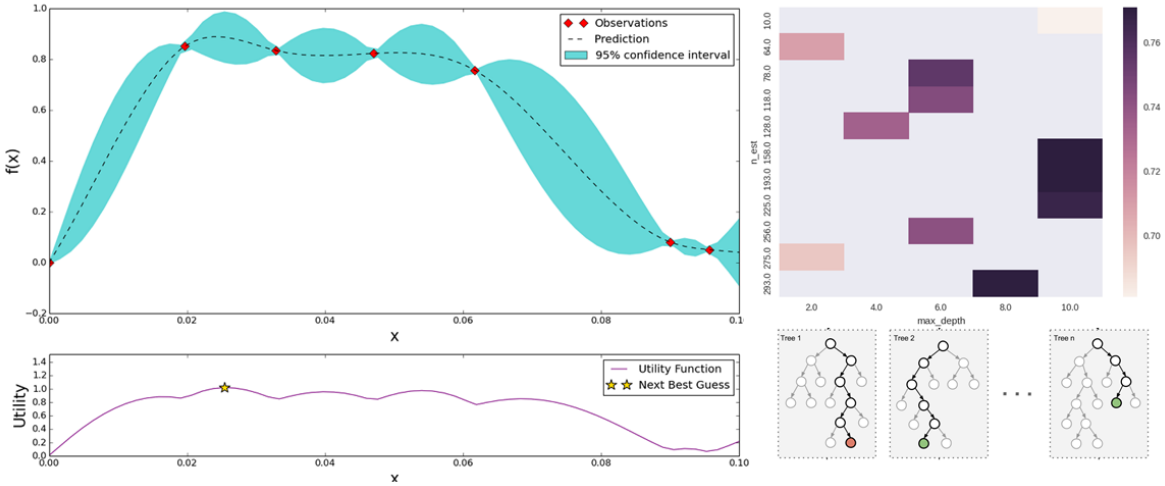

Figure 26: Bayesian Optimization of SVM and Random Forest parameters

Figure 26 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.

3.3 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 [15].

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.

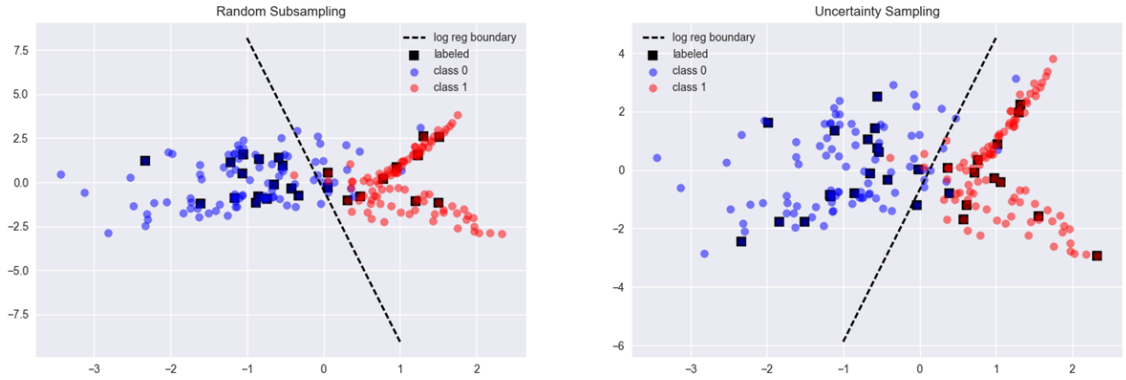

Figure 27: Random Subsampling vs Uncertainty Sampling

Figure 27 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.

3.3.1 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:

$$x^*_{LC} = \arg\max_x 1 - P_\theta(\hat{y}|x) \tag{113}$$

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:

$$x^*_M = \arg\min_x P_\theta(\hat{y}_1|x) - P_\theta(\hat{y}_2|x) \tag{114}$$

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:

$$x_H^* = \arg\max_x -\sum_i P_\theta(y_i|x) \log P_\theta(y_i|x) \tag{115}$$

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 [15]: the vote entropy and KL divergence. The vote entropy is defined as follows:

$$x_{VE}^* = \arg\max_x -\sum_i \frac{V(y_i)}{C} \log \frac{V(y_i)}{C} \tag{116}$$

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:

$$x_{KL}^* = \arg\max_x \frac{1}{C} \sum_{c=1}^{C} KL(P_{\theta^{(c)}} || P_C) \tag{117}$$

$$KL(P_{\theta^{(c)}} || P_C) = \sum_i P_{\theta^{(c)}}(y_i|x) \log \frac{P_{\theta^{(c)}}(y_i|x)}{P_C(y_i|x)} \tag{118}$$

$$P_C(y_i|x) = \frac{1}{C} \sum_{c=1}^{C} P_{\theta^{(c)}}(y_i|x) \tag{119}$$

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

$$\begin{aligned} \text{MSE} &= E\Big[(\hat{\theta}-\theta^*)^2\Big] = E\Big[\big[(\hat{\theta}-\bar{\theta})+(\bar{\theta}-\theta^*)\big]^2\Big] && (120)\\ &= E\Big[(\hat{\theta}-\bar{\theta})^2\Big] + 2(\bar{\theta}-\theta^*)E\Big[\hat{\theta}-\bar{\theta}\Big] + (\bar{\theta}-\theta^*)^2 && (121)\\ &= E\Big[(\hat{\theta}-\bar{\theta})^2\Big] + (\bar{\theta}-\theta^*)^2 && (122)\\ &= \text{VAR}[\hat{\theta}] + \text{bias}^2(\hat{\theta}) && (123) \end{aligned}$$

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

$$\text{VAR}(\hat{\theta}) \geq \frac{1}{-E\big[\frac{\partial^2 \log p(x|\theta)}{\partial \theta^2}\big]} = \frac{1}{I(\theta)} \tag{124}$$

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:

$$[I(\theta)]_{ij} = -E\left[\frac{\partial^2 \log p(x|\theta)}{\partial \theta_i \partial \theta_j}\right] \tag{125}$$

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

Figure 28 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

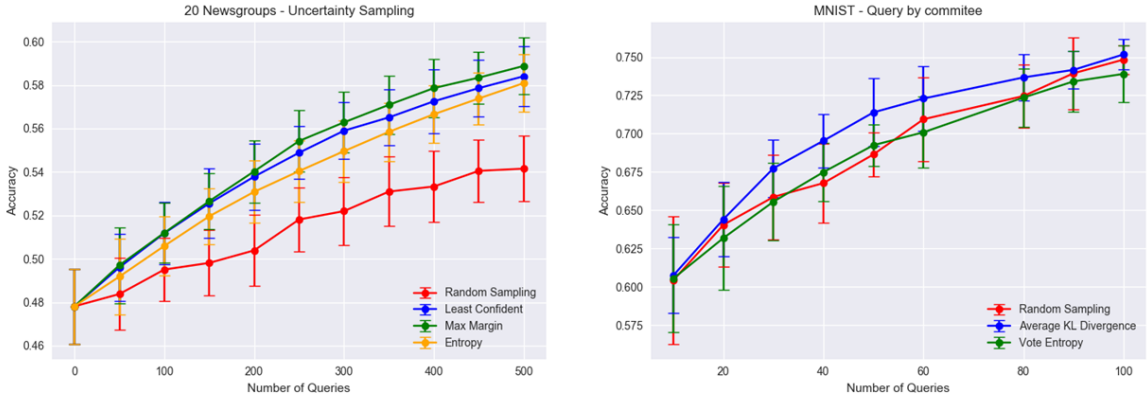

Figure 28: Active Learning: Uncertainty Sampling and Query By Committee.

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.

3.4 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

$$F(A) + F(B) \geq F(A \cup B) - F(A \cap B) \tag{126}$$

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

$$F(A \cup \{s\}) - F(A) \geq F(B \cup \{s\}) - F(B) \tag{127}$$

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:

$$f(e|A) \geq f(e|B) \tag{128}$$

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 3.2 *(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.*

$$f(A) = \sum_{i=1}^{n} \alpha_i f_i(A) \tag{129}$$

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

$$f_i(j|A) \geq f_i(j|B) \Rightarrow \sum_{i=1}^{n} \alpha_i f_i(j|A) \geq \sum_{i=1}^{n} \alpha_i f_i(j|B) \Rightarrow f(j|A) \geq f(j|B) \tag{130}$$

Theorem 3.3 *(Entropy is submodular). Let V be the index set of a set of random variables, then the following function is submodular:*

$$f(A) = H(X_A) = -\sum_{X_A} p(X_A) \log p(X_A) \tag{131}$$

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

$$\begin{aligned} F(A \cup e) - F(A) &= H(X_A, X_e) - H(X_A) = H(X_e|X_A) && (132)\\ &\geq H(X_e|X_A, X_B) = H(X_e|X_B) = H(X_B, X_e) - H(X_B) && (133)\\ &= F(B \cup e) - F(B) && (134) \end{aligned}$$

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

Theorem 3.4 *(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:

$$\begin{aligned} I(Y_j; Y_{B \setminus A}|Y_A) &\geq 0 && (135)\\ H(Y_j|Y_A) - H(Y_j|Y_{(B \setminus A) \cup A}) &\geq 0 && (136)\\ H(Y_j|Y_A) &\geq H(Y_j|Y_B) && (137) \end{aligned}$$

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:

$$\begin{aligned} H(Y_j|Y_A) - H(Y_j|X) &\geq H(Y_j|Y_B) - H(Y_j|X) && (138)\\ H(Y_j|Y_A) - H(Y_j|X, Y_A) &\geq H(Y_j|Y_B) - H(Y_j|X, Y_B) && (139)\\ I(Y_j; X|Y_A) &\geq I(Y_j; X|Y_B) && (140)\\ I(X; Y_j|Y_A) &\geq I(X; Y_j|Y_B) && (141) \end{aligned}$$

3.4.1 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:

$$\max_{S \subseteq V} f(S) \tag{142}$$

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 7 that starts with an empty set S_0 and at iteration i, adds the element maximizing the increment function:

$$S_i = S_{i-1} \cup \{\arg\max_e f(e|S_{i-1})\} \tag{143}$$

Algorithm 7 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 [13] proves that that the greedy algorithm provides a good approximation to the optimal solution of the NP-hard optimization problem.

Theorem 3.5 *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:*

$$f(S_l) \geq \left(1 - e^{-l/k}\right) \max_{S:|S| \leq k} f(S) \tag{144}$$

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 $i < l$:

$$\begin{aligned}
f(S^*) &\leq f(S^* \cup S_i) && (145)\\
&= f(S_i) + \sum_{j=1}^{k} f(v_j^* | S_i \cup \{v_1^*, ..., v_{j-1}^*\}) && (146)\\
&\leq f(S_i) + \sum_{v \in S^*} f(v|S_i) && (147)\\
&\leq f(S_i) + \sum_{v \in S^*} (f(S_{i+1}) - f(S_i)) && (148)\\
&\leq f(S_i) + k(f(S_{i+1}) - f(S_i)) && (149)
\end{aligned}$$

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,

$$f(S^*) - f(S_i) \leq k(f(S_{i+1}) - f(S_i)) \tag{150}$$

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:

$$\delta_{i+1} \leq \left(1 - \frac{1}{k}\right)\delta_i \tag{151}$$

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

$$\delta_l \leq \left(1 - \frac{1}{k}\right)^l \delta_0 \leq e^{-l/k} f(S^*) \tag{152}$$

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:

$$f(S_l) \geq (1 - e^{-l/k}) f(S^*) \tag{153}$$

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!

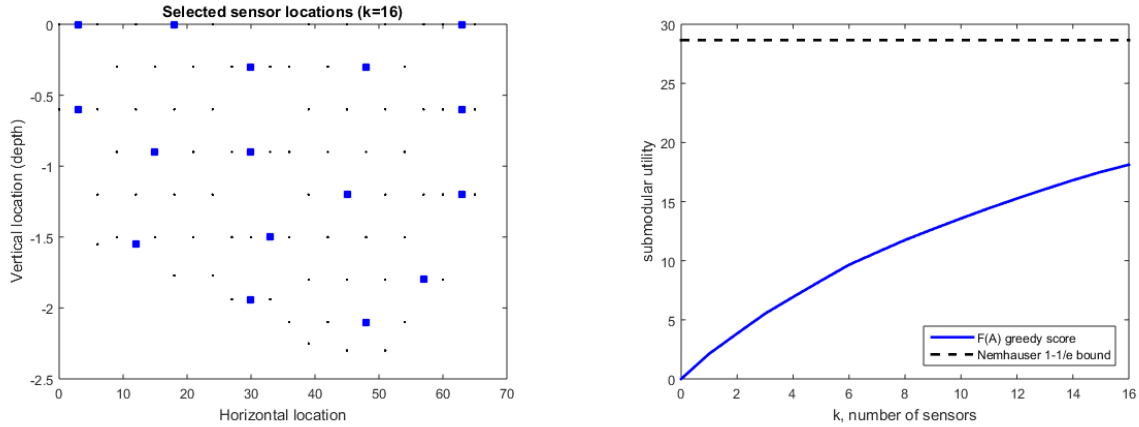

Figure 29: Submodular function optimization subject to cardinality constraint.

Figure 29 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:

$$\max_S f(S) s.t. \sum_{v \in S} c(v) \leq B \tag{154}$$

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:

$$S_{i+1} = S_i \cup \left\{ \arg \max_{v \in V \setminus S_i : c(v) \leq B - c(S_i)} \frac{f(v|S_i)}{c(v)} \right\} \tag{155}$$

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:

$$\max\{f(S_{uc}), f(S_{cb})\} \geq \frac{1 - 1/e}{2} \text{OPT} \tag{156}$$

The submodular maximization algorithm with a budget constraint also known as Cost-Effective Lazy Forward-selection (CELF) Algorithm is summarized in Algorithm 8.

Algorithm 8 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**

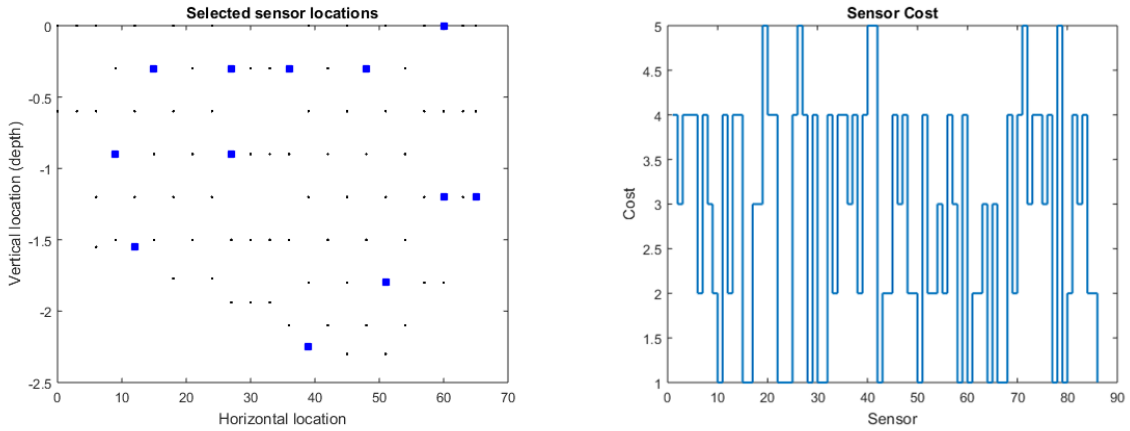

Figure 30: Submodular function optimization subject to budget constraint.

4 Misc

4.1 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 [3]:

$$z_t \sim P(z_t|z_{t-1}) \tag{157}$$
$$x_t = A(z_t)x_{t-1} + B(z_t)w_t + F(z_t)u_t \tag{158}$$
$$y_t = C(z_t)x_t + D(z_t)v_t + G(z_t)u_t \tag{159}$$

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:

$$p(x_t|z_t, x_{t-1}) \sim N(A(z_t)x_{t-1} + F(z_t)u_t, B(z_t)B(z_t)^T) \tag{160}$$
$$p(y_t|x_t, z_t) \sim N(C(z_t)x_t + G(z_t)u_t, D(z_t)D(z_t)^T) \tag{161}$$

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:

$$p(x_{1:t}, z_{1:t}|y_{1:t}) = p(x_{1:t}|y_{1:t}, z_{1:t})p(z_{1:t}|y_{1:t}) \tag{162}$$

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:

$$\begin{aligned} p(z_{1:t}|y_{1:t}) &= \frac{p(y_t|z_{1:t}, y_{1:t-1})p(z_{1:t}|y_{1:t-1})}{p(y_t|y_{1:t-1})} && (163) \\ &= \frac{p(y_t|z_t)p(z_t|z_{1:t-1}, y_{1:t-1})p(z_{1:t-1}|y_{1:t-1})}{p(y_t|y_{1:t-1})} && (164) \\ &\propto p(y_t|z_t)p(z_t|z_{t-1})p(z_{1:t-1}|y_{1:t-1}) && (165) \end{aligned}$$

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:

$$p(z_{1:t}|y_{1:t}) \approx \sum_{s=1}^{S} \hat{w}_t^s \delta_{z_{1:t}^s}(z_{1:t}) \tag{166}$$

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:

$$w_t^s \propto \frac{p(z_{1:t}^s|y_{1:t})}{q(z_{1:t}^s|y_{1:t})} \propto \frac{p(y_t|x_t, z_t)p(x_t, z_t|x_{t-1}, z_{t-1})}{q(x_t, z_t|x_{t-1}, z_{t-1})} \tag{167}$$

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

$$q(x_t, z_t|x_{t-1}, z_{t-1}) = p(x_t, z_t|x_{t-1}, z_{t-1}) = p(x_t|x_{t-1}, z_{t-1})p(z_t|z_{t-1}) \tag{168}$$

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 9.

Algorithm 9 Particle Filter Algorithm [3]

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. (160)
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

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

$$\hat{p}_N(x) = \sum_{k=1}^{N} \frac{1}{N} \delta(x - x_k^*) = \sum_{i=1}^{N} \frac{n_i}{N} \delta(x - x_i) \tag{169}$$

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:

$$\mu_{t|t-1}^i = A(z_t^i)\mu_{t-1|t-1}^i + F(z_t^i)u_t \tag{170}$$
$$\Sigma_{t|t-1}^i = A(z_t^i)\Sigma_{t-1|t-1}^i A(z_t^i)^T + D(z_t^i)D(z_t^i)^T \tag{171}$$
$$S_t^i = C(z_t^i)\Sigma_{t|t-1}^i C(z_t^i)^T + D(z_t^i)D(z_t^i)^T \tag{172}$$
$$y_{t|t-1}^i = C(z_t^i)\mu_{t|t-1}^i + G(z_t^i)u_t \tag{173}$$
$$\mu_{t|t}^i = \mu_{t|t-1}^i + \Sigma_{t|t-1}^i C(z_t^i)^T S_t^{-1}(y_t - y_{t|t-1}^i) \tag{174}$$
$$\Sigma_{t|t}^i = \Sigma_{t|t-1}^i - \Sigma_{t|t-1}^i C(z_t^i)^T S_t^{-1} C(z_t^i)\Sigma_{t|t-1}^i \tag{175}$$

The RBPF takes slightly longer to compute but results in more accurate predictions. Figure 31 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

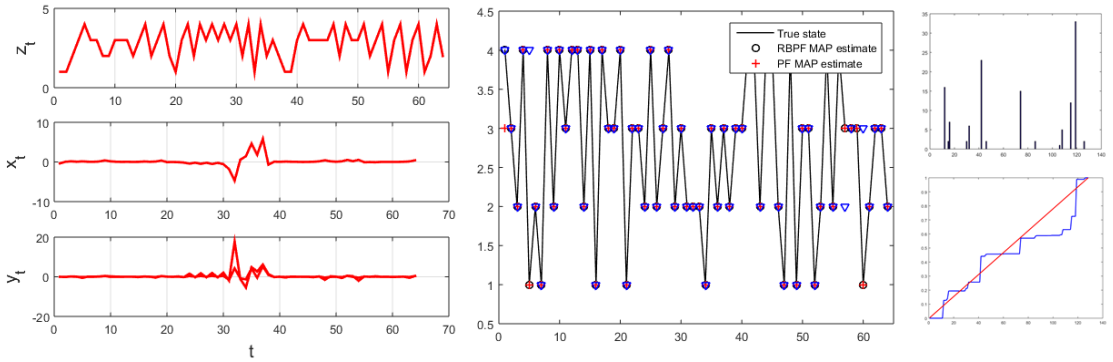

Figure 31: Particle Filter inference using PF and RBPF applied to Switching Linear Dynamic System.

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.

4.2 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.

4.2.1 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:

$$H(X) = -\sum_x p(x)\log p(x) = -E[\log p(x)] \tag{176}$$

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

$$H(X) = -p \log p - (1 - p) \log(1 - p) \tag{177}$$

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:

$$H(\lambda p_1 + (1 - \lambda)p_2) \geq \lambda H(p_1) + (1 - \lambda)H(p_2) \tag{178}$$

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:

$$\begin{aligned} H(X) &= \sum_{x \in \mathcal{X}} p(x) \log \frac{1}{p(x)} && (179) \\ &\leq \log \sum_{x \in \mathcal{X}} p(x) \times \frac{1}{p(x)} && (180) \\ &= \log \sum_{x \in \mathcal{X}} 1 && (181) \\ &= \log |\mathcal{X}| && (182) \end{aligned}$$

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:

$$H(X, Y) = -\sum_x \sum_y p(x, y) \log p(x, y) = -E[\log p(x, y)] \tag{183}$$

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:

$$\begin{aligned} H(X|Y) &= \sum_y p(y) H(X|Y = y) = -\sum_y p(y) \sum_x p(x|y) \log p(x|y) && (184) \\ &= -\sum_x \sum_y p(x, y) \log p(x|y) = -E[\log p(x|y)] && (185) \end{aligned}$$

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:

$$H(X, Y) = H(X) + H(Y|X) = H(Y) + H(X|Y) \tag{186}$$

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 $X_1, ..., X_N$:

$$H(X_1, ..., X_N) = \sum_{i=2}^{N} H(X_i|X_1, ..., X_{i-1}) + H(X_1) \leq \sum_{i=1}^{N} H(X_i) \tag{187}$$

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

$$\begin{aligned} h(X_1, ..., X_N) &= \int p(x) \log \frac{1}{p(x)} dx & (188) \\ &= \int p(x) \left[\frac{1}{2} \log(2\pi)^d |\Sigma| + \frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu) \right] dx & (189) \\ &= \frac{1}{2} \log(2\pi)^d |\Sigma| + \frac{1}{2} E[(x-\mu)^T \Sigma^{-1}(x-\mu)] & (190) \\ &= \frac{1}{2} \log(2\pi)^d |\Sigma| + \frac{1}{2} Tr\{\Sigma^{-1}\Sigma\} & (191) \\ &= \frac{1}{2} \log(2\pi)^d |\Sigma| + \frac{1}{2} d \log e = \frac{1}{2} \log \left[(2\pi e)^d |\Sigma| \right] & (192) \end{aligned}$$

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

Theorem 4.1 *(AEP) If X_1,X_2,...,X_n are iid $\sim p(x)$ then*

$$-\frac{1}{n} \log p(X_1, X_2, ..., X_n) \to H(X) \quad \text{in probability} \tag{193}$$

Proof.

$$-\frac{1}{n} \log p(X_1, X_2, ..., X_n) = -\frac{1}{n} \sum_i \log p(X_i) \to -E[\log p(X)] = H(X) \tag{194}$$

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.

4.2.2 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:

$$KL(p||q) = \sum_x p(x) \log \frac{p(x)}{q(x)} \tag{195}$$

we can re-write it as:

$$KL(p||q) = \sum_x p(x) \log p(x) - \sum_x p(x) \log q(x) = -H(p) + H(p, q) \tag{196}$$

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

Proof.

$$\begin{aligned} -KL(p||q) &= -\sum_x p(x) \log \frac{p(x)}{q(x)} = \sum_x p(x) \log \frac{q(x)}{p(x)} & (197) \\ &\leq \log \sum_x p(x) \frac{q(x)}{p(x)} = \log \sum_x q(x) & (198) \\ &\leq \log 1 = 0 & (199) \end{aligned}$$

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

$$f\Big(\sum_{i=1}^{n}\lambda_i x_i\Big) \leq \sum_{i=1}^{n}\lambda_i f(x_i) \tag{200}$$

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:

$$\begin{aligned} 0 &\leq KL(p||u) = \sum_x p(x)\log\frac{p(x)}{u(x)} && (201)\\ &= \sum_x p(x)\log p(x) - \sum_x p(x)\log u(x) && (202)\\ &= -H(X) + \log|\mathcal{X}| && (203)\end{aligned}$$

Thus, $H(X) \leq \log|\mathcal{X}|$ with equality iff $p(x) = u(x)$.

Theorem 4.3 *$KL(p||q)$ is convex in the pair (p, q), i.e.*

$$KL(\lambda p_1 + (1-\lambda)p_2||\lambda q_1 + (1-\lambda)q_2) \leq \lambda KL(p_1||q_1) + (1-\lambda)KL(p_2||q_2) \tag{204}$$

Proof. Expanding the inequality above, we want to show that

$$\sum_x(\lambda p_1(x) + (1-\lambda)p_2(x))\log\frac{\lambda p_1(x) + (1-\lambda)p_2(x)}{\lambda q_1(x) + (1-\lambda)q_2(x)} \leq \tag{205}$$

$$\lambda\sum_x p_1(x)\log\frac{p_1(x)}{q_1(x)} + (1-\lambda)\sum_x p_2(x)\log\frac{p_2(x)}{q_2(x)} \tag{206}$$

Then for a fixed x, it suffices to show that

$$\sum_{i=1,2}\lambda_i p_i\log\frac{\lambda_i p_i}{\lambda_i q_i} \geq \Big(\sum_{i=1,2}\lambda_i p_i\Big)\Big(\log\frac{\sum_i\lambda_i p_i}{\sum_i\lambda_i q_i}\Big) \tag{207}$$

where $\lambda_1 = \lambda$ and $\lambda_2 = 1-\lambda$, which is exactly the log-sum inequality.

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:

$$(\mathrm{I}-\mathrm{projection}): \quad q(x) = \arg\min_{q\in Q} KL(q||p) \tag{208}$$

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:

$$(\mathrm{M}-\mathrm{projection}): \quad q(x) = \arg\min_{q\in Q} KL(p||q) \tag{209}$$

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:

$$KL(q||p) \geq KL(q||p^*) + KL(p^*||p) \tag{210}$$

The inequality can be interpreted as information-geometric version of Pythagoras' triangle inequality theorem, where KL divergernce 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:

$$\begin{aligned}
KL(p||q) &= \int p(x) \log \frac{p(x)}{q(x)} = \int [\log p(x) - \log q(x)] p(x) dx \\
&= \int \left[\frac{1}{2} \log \frac{|\Sigma_2|}{|\Sigma_1|} - \frac{1}{2}(x-\mu_1)^T \Sigma_1^{-1}(x-\mu_1) + \frac{1}{2}(x-\mu_2)^T \Sigma_2^{-1}(x-\mu_2) \right] p(x) dx \\
&= \frac{1}{2} \log \frac{|\Sigma_2|}{|\Sigma_1|} - \frac{1}{2} Tr\{E[(x-\mu_1)(x-\mu_1)^T \Sigma_1^{-1}]\} + \frac{1}{2} E[(x-\mu_2)^T \Sigma_2^{-1}(x-\mu_2)] \\
&= \frac{1}{2} \log \frac{|\Sigma_2|}{|\Sigma_1|} - \frac{1}{2} Tr\{I_d\} + \frac{1}{2}(\mu_1-\mu_2)^T \Sigma_2^{-1}(\mu_1-\mu_2) + \frac{1}{2} Tr\{\Sigma_2^{-1}\Sigma_1\} \\
&= \frac{1}{2} \left[\log \frac{|\Sigma_2|}{|\Sigma_1|} - d + Tr(\Sigma_2^{-1}\Sigma_1) + (\mu_2-\mu_1)^T \Sigma_2^{-1}(\mu_2-\mu_1) \right]
\end{aligned} \tag{211}$$

4.2.3 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:

$$I(X;Y) = KL(p(x,y)||p(x)p(y)) = \sum_x \sum_y p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \tag{212}$$

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:

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) = H(X) + H(Y) - H(X,Y) \tag{213}$$

The relationship between entropy and MI is captured in a Venn diagram in Figure 32. Note that entropy can be viewed as self-information $H(X) = I(X; X)$. The MI is the expected value of pointwise mutual information:

$$PMI(x,y) = \log \frac{p(x,y)}{p(x)p(y)} = \log \frac{p(x|y)}{p(x)} = \log \frac{p(y|x)}{p(y)} \tag{214}$$

This can be interpreted as the amount we learn from updating the prior $p(x)$ into the posterior $p(y|x)$

Theorem 4.4 *(Data Processing Inequality). Let X, Y, Z form the following Markov chain: $X \to Y \to Z$, then the following inequality holds:*

$$I(X;Y) \geq I(X;Z) \tag{215}$$

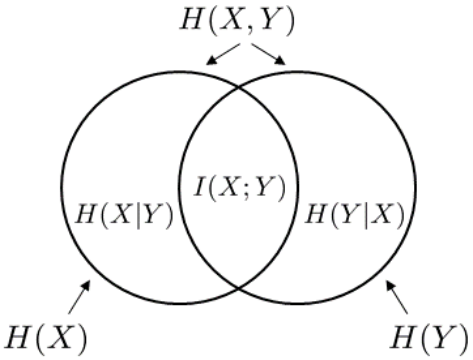

Figure 32: Relationship between entropy and mutual information

Proof. By the chain rule, we can expand mutual information in two different ways:

$$I(X;Y,Z) = I(X;Z) + I(X;Y|Z) = I(X;Y) + I(X;Z|Y) \tag{216}$$

Since X is conditionally independent of Z given Y, we have $I(X;Z|Y) = 0$ and therefore

$$I(X;Y) = I(X;Z) + I(X;Y|Z) \tag{217}$$

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:

$$I(\theta;X) = I(\theta;T(X)) \tag{218}$$

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

$$h(X,Y) = \frac{1}{2}\log\left[(2\pi e)^2|\Sigma|\right] = \frac{1}{2}\log(2\pi e)^2\sigma^4(1-\rho^2) \tag{219}$$

Therefore,

$$I(X;Y) = h(X) + h(Y) - h(X,Y) = -\frac{1}{2}\log(1-\rho^2) \tag{220}$$

4.3 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 [7].

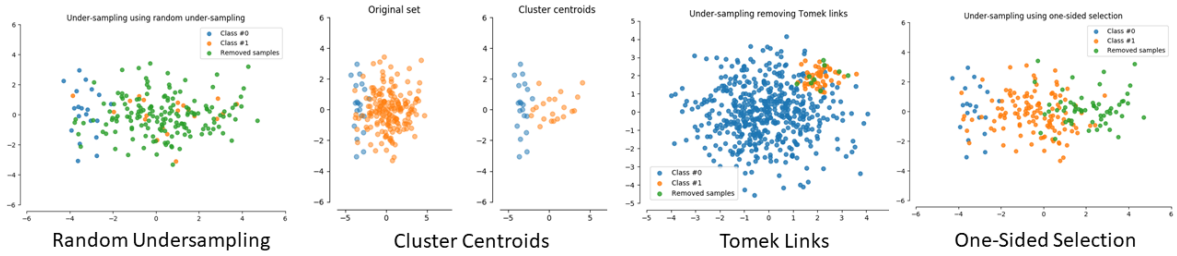

Figure 33: Four undersampling strategies: random, cluster centroids, Tomek links, one-sided selection.

Undersampling strategies. Undersampling methods remove data from the majority class of the original dataset as shown in Figure 33. 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.

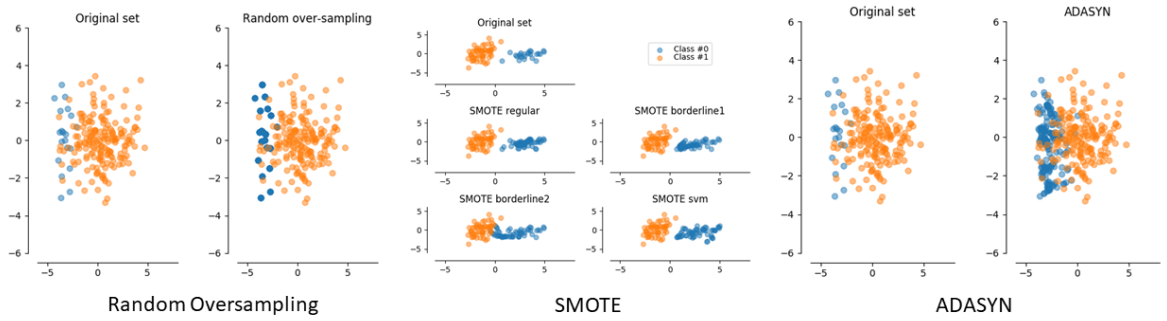

Figure 34: Three oversampling strategies: random, SMOTE, ADASYN.

Oversampling strategies. Oversampling methods append data to the minority class of the original dataset as shown in Figure 34. 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.

4.4 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.

4.4.1 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:

$$f(x) = \frac{1}{M} \sum_{m=1}^{M} f_m(x) \tag{221}$$

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 35 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

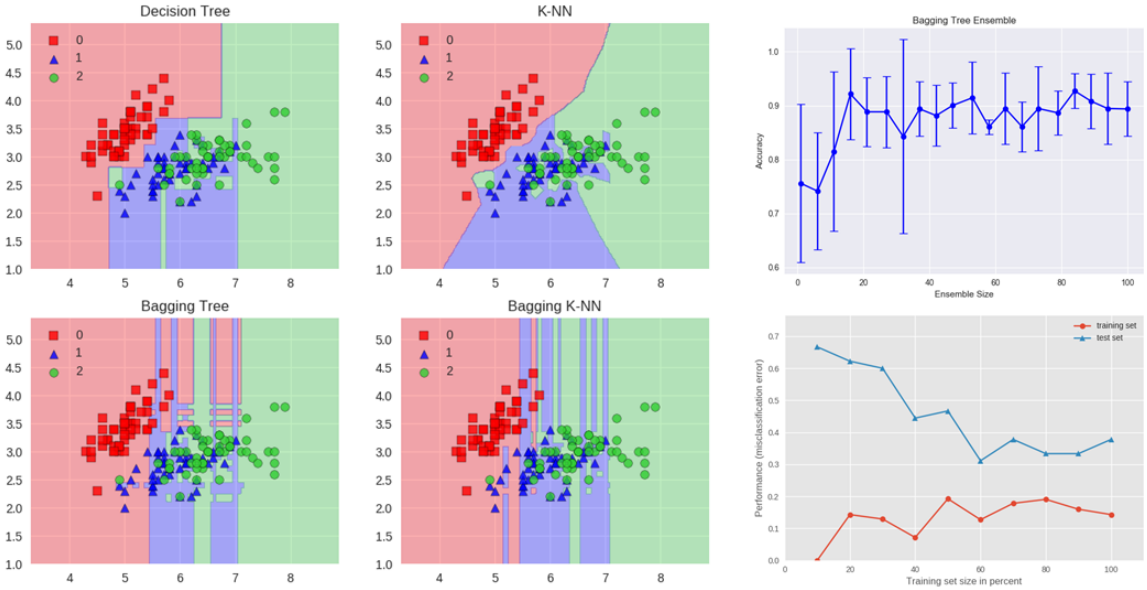

Figure 35: Bagging Ensemble with decision tree and k-NN as base estimator.

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.

4.4.2 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 sequency on a weighted version of the data.

Algorithm 10 describes the most widely used form of boosting algorithm called **AdaBoost** which stands for adaptive boosting [5].

Algorithm 10 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.

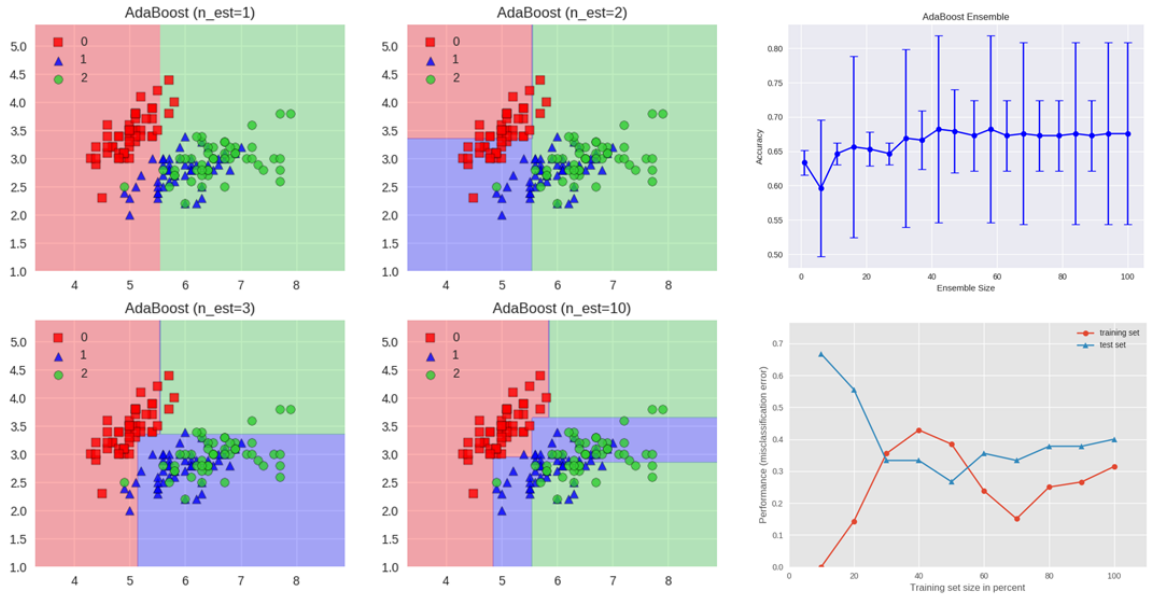

Figure 36: Boosting Ensemble with decision tree as base estimator.

The AdaBoost algorithm is illustrated in Figure 36. 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:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x) \tag{222}$$

At each stage the decision tree $h_m(x)$ is chosen to minimize a loss function L given the current model $F_{m-1}(x)$:

$$F_m(x) = F_{m-1}(x) + \arg\min_h \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + h(x_i)) \tag{223}$$

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}:

$$F_m(x) = F_{m-1}(x) + \gamma_m \sum_{i=1}^{n} \nabla_F L(y_i, F_{m-1}(x_i)) \tag{224}$$

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.

4.4.3 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 11 summarizes stacking.

Algorithm 11 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 37. It consists of k-NN, Random Forest and Naive Bayes base classifiers whose predictions are combined by Lostic Regression as a meta-classifier. We can see the blending of decision boundaries achieved by the stacking classifier.

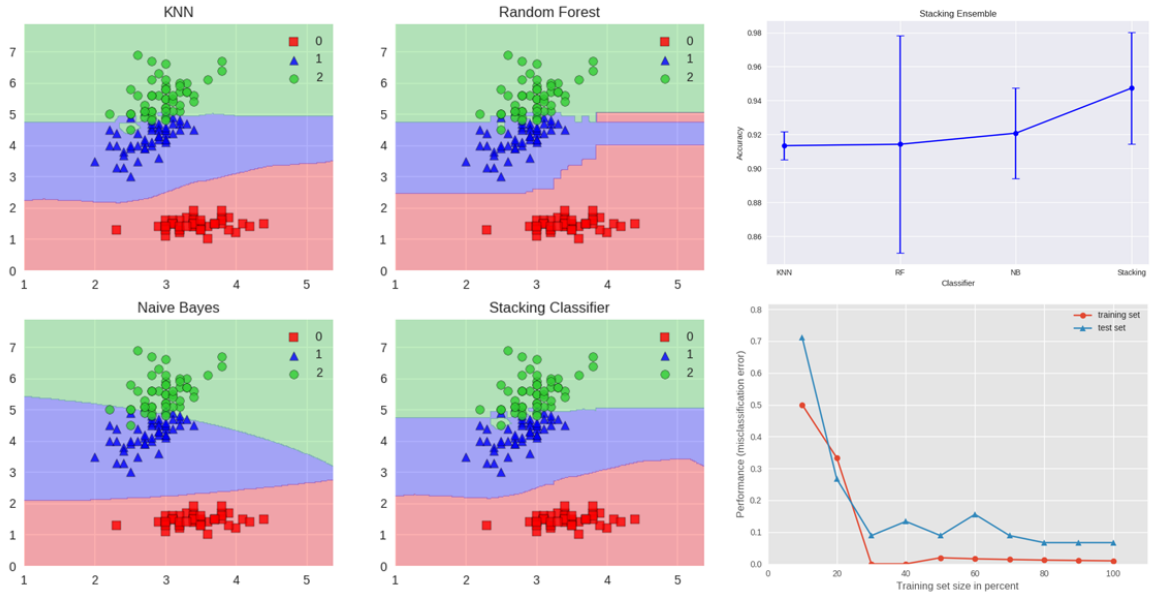

Figure 37: Stacking Ensemble with Logistic Regression as meta-classifier.

The figure also shows that stacking achieves higher accuracy than individual classifiers and based on learning curves, it shows no signs of overfitting.

References

[1] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. In *J. Mach. Learn. Res.*, pages 993–1022, 2003.

[2] 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.

[3] N. de Freitas. Rao-blackwellized particle filtering for fault diagnosis. In *IEEE Aerospace Conference*, 2002.

[4] 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.

[5] 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.

[6] T. L. Griffiths and Z. Ghahramani. The indian buffet process: An introduction and review. In *J. Mach. Learn. Res.*, pages 1185–1224, 2011.

[7] H. He and E. A. Garcia. Learning from imbalanced data. In *IEEE Trans. on Knowl. and Data Eng.*, pages 1263–1284, 2009.

[8] 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.

[9] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. In *Journal of Machine Learning Research*, pages 1303–1347, 2013.

[10] 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.

[11] S. N. MacEachern. Dependent nonparametric processes. In *Proceedings of the Bayesian Statistical Science Section*, 1999.

[12] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[13] 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.

[14] J. Sethuraman. A constructive definition of Dirichlet priors. In *Statistica Sinica*, pages 639–650, 1994.

[15] B. Settles. Active learning literature survey. In *Computer Sciences Technical Report*, pages 1–67, 2009.

[16] 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.

[17] Y. Teh, M. Jordan, M. J. Beal, and D. M. Blei. Hierarchical Dirichlet Processes. In *Journal of the American Statistical Association (JASA)*, 2005.

[18] Y. W. Teh. Dirichlet processes. In *Encyclopedia of Machine Learning*. Springer, 2010.

[19] 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.

[20] 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.