Variational Inference

Vishal Rana

August 13, 2019

1 Introduction

The problem of approximating difficult to compute probability densities is central to many problems in statistics. This problem is especially relevant to Bayesian computations, where non-conjugacy of priors and likelihoods leads to intractable posterior distributions.

In this project, we have studied Variational Inference as a method for approximating probability densities. It is an alternative strategy to using MCMC methods, especially in cases with large datasets or models where simple MCMC algorithms would be slow. VI works by transforming the inference problem into an optimization problem. The main idea behind it is to define a set of approximating distributions and then finding the member that minimizes the Kullback-Leibler (KL) divergence to the true posterior distribution [1].

$$q^*(\mathbf{z}) = \underset{q(\mathbf{z})\in\mathcal{Q}}{arg\ min}\, \mathrm{KL}(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})) \tag{1}$$

First we will cover the basic ideas behind VI and then work through two examples, one being Bayesian mixture of Gaussians and the other being binary image denoising using the Ising model.

2 Variational Inference

We want to estimate the conditional density of latent variables given the observed variables. The problem can be formally stated as follows. Let $\mathbf{x} = x_{1:n}$ be set of observed variables and $\mathbf{z} = z_{1:n}$ be set of latent variables. Our goal is to estimate $p(\mathbf{z}|\mathbf{x})$, i.e. the conditional density of latent variables given the observed variables.

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{z},\mathbf{x})}{p(\mathbf{x})} \tag{2}$$

The denominator of the conditional expectation expression is called *evidence* and is what makes the problem intractable. To compute $p(\mathbf{x})$ we need to find the following integral, which is not available in a closed form for most models and takes exponential time to compute numerically.

$$p(\mathbf{x}) = \int p(\mathbf{z},\mathbf{x})d\mathbf{z} \tag{3}$$

In VI, as mentioned before, we define a family of distributions over the latent variables $q(\mathbf{z}) \in \mathcal{Q}$ and then find the $q^*(\mathbf{z})$ that is closest to the required conditional. This amounts to solving equation 1. However this is still not solvable because we need to find $logp(\mathbf{x})$.

$$\mathrm{KL}(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})) = \mathbb{E}[log(q(\mathbf{z}))] - \mathbb{E}[log(p(\mathbf{z}|\mathbf{x}))] \tag{4}$$

2.1 Expected Lower Bound (ELBO)

Expanding the conditional,

$$\mathrm{KL}(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})) = \mathbb{E}[log(q(\mathbf{z}))] - \mathbb{E}[log(p(\mathbf{z},\mathbf{x}))] + log(p(\mathbf{x})) \tag{5}$$

Notice that $log(p(\mathbf{x}))$ is a constant with respect to $q(\mathbf{z})$. Since, the KL divergence is difficult to optimize over, we define an alternative objective function which differs from it by a constant.

$$ELBO(q) = \mathbb{E}[log(p(\mathbf{z},\mathbf{x}))] - \mathbb{E}[log(q(\mathbf{z}))] = -\mathrm{KL}(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})) + log(p(\mathbf{x})) \tag{6}$$

This shows that minimizing KL is equivalent to maximizing the ELBO. It is called ELBO because it lower bounds the log evidence $p(\mathbf{x})$ as shown by the following equation,

$$log(p(\mathbf{x})) = \mathrm{KL}(q(\mathbf{z}), p(\mathbf{z}|\mathbf{x})) + ELBO(q) \geq ELBO(q) \tag{7}$$

The inequality comes from the fact that $\mathrm{KL}(.) \geq 0$. This has lead to ELBO being used as a criteria for model selection which works well in practice. Further, we can re-write the ELBO as,

$$ELBO(q) = \mathbb{E}[log(p(\mathbf{x}|\mathbf{z}))] - \mathrm{KL}(q(\mathbf{z})||p(\mathbf{z})) \tag{8}$$

The first term corresponds to expected log-likelihood and promotes densities that give higher weights to latent variable configurations that better explain data. The second term encourages the variational densities to be closer to the priors. This is a fundamental trade-off in many inference problems.

2.2 Mean-field Variational Family

Next step in formulating the problem completely is to specify a family of distributions $\mathcal{Q}$ over which we will perform the optimization. The complexity of the family will determine the difficulty of optimization.

To make the optimization easier, the mean-field approximation is a popular assumtion. What it says is that the latent variables are mutually independent of each other and governed by a distinct factor in the mean-field variational density.

$$q(\mathbf{z}) = \prod_{j=1}^{m} q_j(z_j) \tag{9}$$

Various latent variables can take parametric forms appropriate to the particular random variables. The accuracy of VI can be improved by relaxing the mean-field assumption like in

structured variational inference or mixture-based variational inference. But these methods generally involve a more difficult optimization problem.

2.3 Coordinate Ascent Mean-field Variational Inference CAVI

Now the optimization problem is completely specified with ELBO and variational family and we can use optimization algorithms to solve this problem. One of the popular ones is the coordinate ascent. It iteratively optimizes each factor of the variational family while holding the other factors constant. To derive the update equations for each factor, we first re-write the ELBO as a function of the j-th variational parameter, separating the factors that depend on j-th factor from the ones constant with respect to it.

$$ELBO(q_j) = \mathbb{E}_j[\mathbb{E}_{-j}[log(p(z_j, \mathbf{z}_{-j}, \mathbf{x}))]] - \mathbb{E}_j[log(p(z_j))] + \text{constant} \tag{10}$$

This equation is equivalent to negative KL divergence between $q_j(z_j)$ and $q_j^*(z_j)$ upto an additive constant, where $q_j^*(z_j)$ is defined as,

$$q_j^*(z_j) \propto \exp(\mathbb{E}_{-j}[log(p(z_j, \mathbf{z}_{-j}, \mathbf{x}))]) \tag{11}$$

Thus, $q_j = q_j^*$ is the optimal solution to the problem. Since, q_j^* does not involve expectation with respect to j-th variational factor, it is a valid coordinate update equation.

Algorithm 1 Coordinate Ascent Variational Inference

1: **Input** A model $p(\mathbf{x}, \mathbf{z})$ and data set $\mathbf{x}$
2: **Output** A variational density $q(\mathbf{z}) = \prod_{j=1}^{m} q_j(z_j)$
3: **Initialize** Variational factors $q_j(z_j)$
4: **while** the ELBO has not converged **do**
5: **for** $j \in \{1, 2, \cdots, m\}$ **do**
6: Set $q_j^*(z_j) \propto \exp(\mathbb{E}_{-j}[log(p(z_j, \mathbf{z}_{-j}, \mathbf{x}))])$
7: **end for**
8: Compute $ELBO(q) = \mathbb{E}[log(p(\mathbf{z}, \mathbf{x}))] - \mathbb{E}[log(q(\mathbf{z}))]$
9: **end while**
10: **Return q**$(\mathbf{z})$

We will use the above set-up to solve some simple problems.

3 Bayesian Mixture of Gaussians

Consider a mixture of K unit-variance univariate Gaussians, with K corresponding means $\mu = \{\mu_1, \mu_2, \cdots, \mu_K\}$ being drawn independently from prior $p(\mu_k)$, assumed to be Gaussian $\mathcal{N}(0, \sigma^2)$, where σ is a hyper-parameter. [1,5] An observation x_i is generated by first generating a categorical variable c_i indicating the membership of the observation and then generating x_i from the corresponding Gaussian $\mathcal{N}(c_i^T \mu, 1)$. Here, c_i are represented as one-hot vectors.

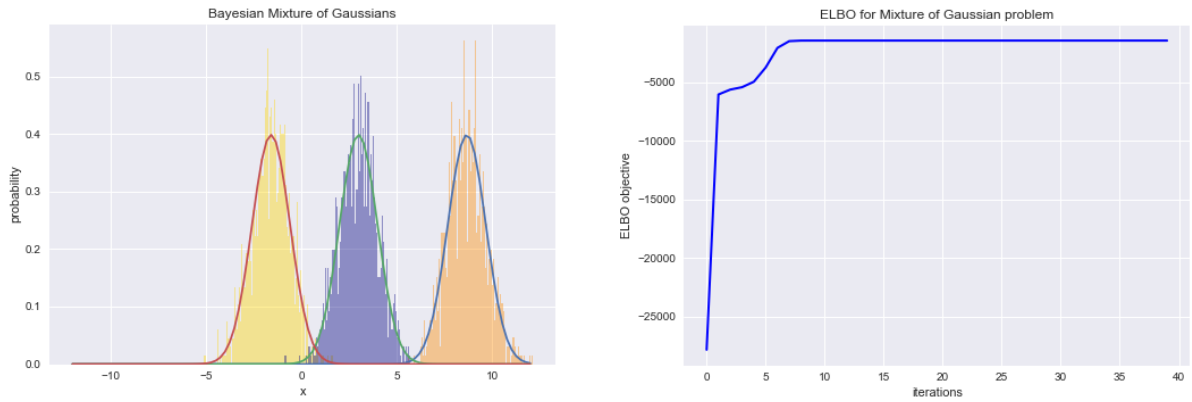

Figure 1: CAVI for Bayesian mixture of Gaussians

The full model is given as,

$$\mu_k \sim \mathcal{N}(0, \sigma^2) \qquad k = 1, \cdots, K \tag{12}$$

$$c_i \sim \text{categorical}(1/K, \cdots, 1/K) \qquad i = 1, \cdots, N \tag{13}$$

$$x_i | c_i, \mu \sim \mathcal{N}(c_i^T \mu, 1) \qquad i = 1, \cdots, N \tag{14}$$

The joint density of observed and latent variables is given as,

$$p(\mu, \mathbf{c}, \mathbf{x}) = p(\mu) \prod_{i=1}^{N} p(c_i) p(x_i | c_i, \mu) \tag{15}$$

The evidence can be written as,

$$p(\mathbf{x}) = \int p(\mu) \prod_{i=1}^{N} \sum_{c_i} p(c_i) p(x_i | c_i, \mu) d\mu \tag{16}$$

Even if we distribute the integral over the product and use the conjugacy properties between Gaussian prior and likelihood, the computation still remains an exponential in K.

The mean-field variational family is specified as,

$$q(\mu, \mathbf{c}) = \prod_{k=1}^{K} q(\mu_k; m_k, s_k^2) \prod_{i=1}^{N} q(c_i; \phi_i) \tag{17}$$

The factor $q(\mu_k; m_k, s_k^2)$ is a Gaussian on k-th mixture component's mean, with its mean and variance m_k and s_k^2 respectively, while $q(c_i; \phi_i)$ is a distribution on i-th observation's mixture assignment. It is taken as a multinomial distribution in this example.

The ELBO after substituting the above expressions can be simplified to,

$$ELBO \propto \sum_j -\mathbb{E}_q[\frac{\mu_j}{2\sigma^2}] + \sum_i \sum_j \mathbb{E}_q[c_{ij}] \mathbb{E}_q[-\frac{(x_i - \mu_j)^2}{2}] - \sum_i \sum_j \mathbb{E}_q[log \phi_{i,j}] + \sum_j \frac{log s_j^2}{2} \tag{18}$$

The update equations for various parameters are given as,

$$\phi_{ij}^{*} \propto \exp\{-\frac{m_j^2 + s_j^2}{2} + x_i m_j\} \tag{19}$$

$$m_j^{*} = \frac{\sum_i \phi_{ij} x_i}{\sigma^{-2} + \sum_i \phi_{ij}} \tag{20}$$

$$(s_j^2)^{*} = \frac{1}{\sigma^{-2} + \sum_i \phi_{ij}} \tag{21}$$

We can use the above update equations in the CAVI algorithm. We implemeted the algorithm in Python for K=3. This is a very simple problem and the ELBO converges within first 10 iterations. The plots in 1 show the histogram of observed data with the resulting Guassians superimposed on the plot.

Although the CAVI algorithm is prone to getting stuck in local minimas depending on the initialization, we do not face such a problem for the simple 1D case implemented here and the algorithm converges to a global optimum.

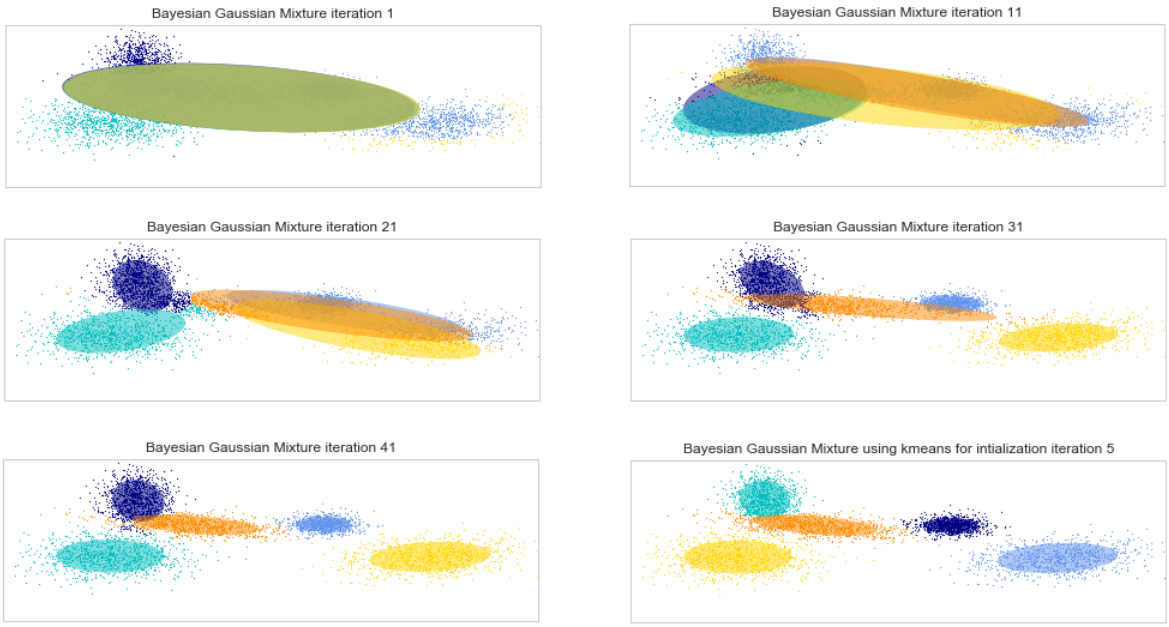

Figure 2: Bivariate Mixture of Gaussians

The above analysis can be extended to multi-variate Gaussian case [2]. We used Python library Scikit-learn [4] to experiment with mixture of bi-variate Gaussians. The effect of initialization on convergence is more pronounced here. If we initialize the parameters randomly, the algorithm converges after about 40 iterations. Instead if we use k-means to find a good initialization, the algorithm gives good results withing first few iterations.

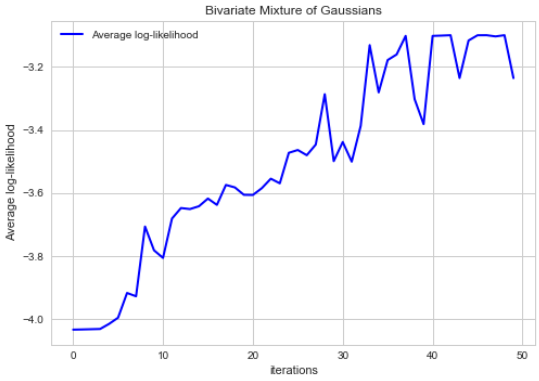

Figure 3: Average log-likelihood with random initialization

We have given scatter plots with superimposed Gaussians learnt by the algorithm after different number of iterations in 2 and average log-likelihood per sample in 3.

4 Mean-field for the Ising Model

We will now consider a more interesting example of binary image denoising based on the Ising model. Ising model is an example of Markov Random Field and has its roots in statistical Physics [3, 6]. According to this model, the image is considered as an array of nodes, where each node is a pixel and can take one of two values, +1 (white) or -1 (black). The state of each node depends on that of the neighbouring states through an interaction potential. This leads to a smoothing effect, where each node tries to be in the same state as its neighbouring nodes.

In the de-noising problem, we have noisy observations corresponding to a true underlying pixel state distribution. Given these noisy observations, the goal is to obtain the true states of the pixels. Following similar notations, we have $\{x_i\}$ representing the observed data (noisy) and $\{z_i\}$ representing the latent variables (true pixel values). We can write the joint distribution as $p(\mathbf{z}, \mathbf{x}) = p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$, where

$$p(\mathbf{z}) = \frac{1}{Z_0}\exp(-E_0(\mathbf{z})) \tag{22}$$

$$E_0(\mathbf{z}) = \sum_{i=1}^{N} \sum_{(i,j)\in E} W_{ij} z_i z_j \tag{23}$$

E is a graph with edges between neighbouring nodes. Likelihood can be written as

$$p(\mathbf{x}|\mathbf{z}) = \prod_{i=1}^{N} p(x_i|z_i) = \sum_{i=1}^{N} \exp(-L_i(x_i)) \tag{24}$$

The function L can take different forms. In our simulations we have used a Gaussian likelihood. Thus, the posterior can be written as,

$$p(\mathbf{z}|\mathbf{x}) = \frac{1}{Z}\exp(-E(\mathbf{z})) \tag{25}$$

$$E(\mathbf{z}) = E_0(\mathbf{z}) - \sum_{i=1}^{N} L_i(z_i) \tag{26}$$

Using the mean-field assumption, we write out the variational family in fully factored form as below,

$$q(\mathbf{z}) = \prod_{i=1}^{N} q(z_i; \mu_i) \tag{27}$$

where μ_i is the mean corresponding to i-th pixel. The $q(z_i; \mu_i)$ that minimizes the KL divergence can be shown to be [3],

$$q(z_i; \mu_i) \propto \exp(z_i \sum_{(i,j)\in E} W_{ij}\mu_j + \sum_{i=1}^{N} L_i(z_i)) \tag{28}$$

Let $m_i = \sum_{(i,j)\in E} W_{ij}\mu_j$, $L_i^+ = L(+1)$ and $L_i^- = L(-1)$. The approximate posterior can be written as,

$$q_i(z_i = 1) = \frac{\exp(m_i + L_i^+)}{\exp(m_i + L_i^+) + \exp(m_i + L_i^-)} = \frac{1}{1 + \exp(-2m_i + (L_i^+ - L_i^-))} = sigm(2a_i) \tag{29}$$

$$a_i = -m_i + 0.5(L_i^+ - L_i^-) \tag{30}$$

Similarly, $q_i(z_i = -1) = sigm(-2a_i)$. Now, we can write the update equation for μ_i which can be used in the CAVI algorithm.

$$\mu_i = \mathbb{E}_{q_i}[z_i] = q_i(z_i = 1)(+1) + q_i(z_i = -1)(-1) = tanh(a_i) \tag{31}$$

This can be turned into a fixed point update as follows,

$$\mu_i^t = tanh(\sum_{(i,j)\in E} W_{ij}\mu_j^{t-1} + 0.5(L_i^+ - L_i^-)) \tag{32}$$

Practically, adding damping to the updates gives better results. So the final update equation becomes,

$$\mu_i^t = (1 - \lambda)\mu^{t-1} + \lambda tanh(\sum_{(i,j)\in E} W_{ij}\mu_j^{t-1} + 0.5(L_i^+ - L_i^-)) \tag{33}$$

For the simulation, we took a bmp image, converted it to grayscale, binarized it and then added Gaussian noise to obtain the noisy observed image. This makes the Gaussian

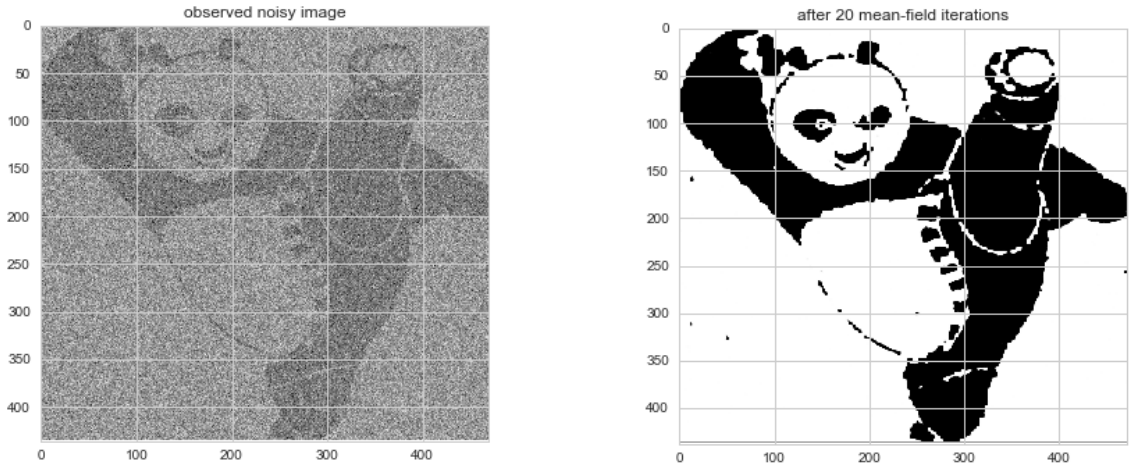

Figure 4: Observed noisy image and Image after 20 Mean-field iterations

distribution an appropriate choice for likelihood function. Substituting the appropriate distributions, we obtain the value of a_i which we have used in the simulation. Remaining updates are same as the equations given above.

$$a_i = \sum_{(i,j)\in E} W_{ij}\mu_j + 0.5\left(\frac{\mathcal{N}(z_i = +1, \sigma^2)}{\mathcal{N}(z_i = -1, \sigma^2)}\right) \tag{34}$$

Here, σ^2 represents the noise level in the observed image. We can observe that our model can restore the image to reasonable accuracy, even with simplifying mean-field assumptions. The restoration, however, is not very good. The Ising model inherently seems to wash out fine lines and details.

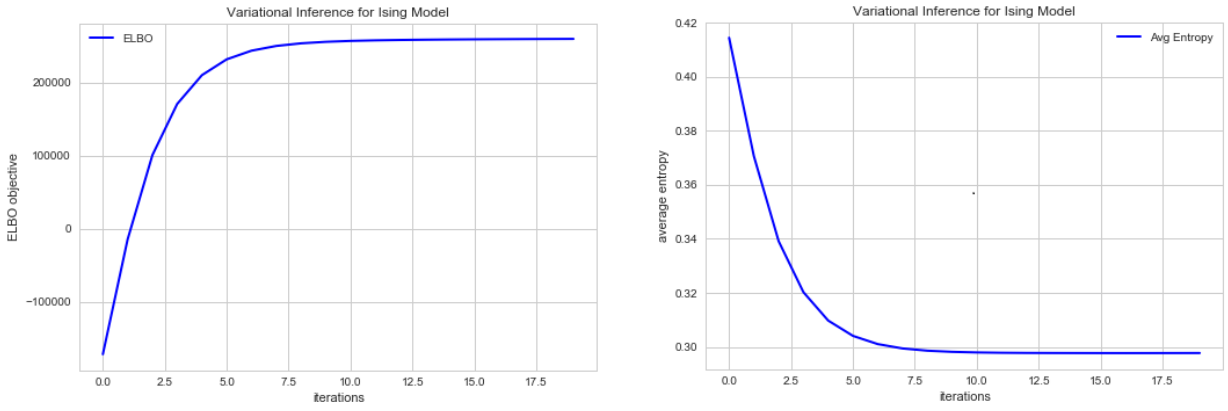

Figure 5: ELBO and Average Entropy

We have given the plots for ELBO and the average entropy with respect to iterations. We can observe that the ELBO stabilizes after around 10 iterations. Further, the plot of entropy shows how the initial randomness in the noisy image decreases as pixel values converge to their means.

5 Conclusions

Variational Inference is a very useful deterministic approximate inference tool that can be used as an alternative to MCMC methods and is especially suited for problems with large datasets or complicated model, which makes the posteriors intractable. We studied two very simple examples which illustrate how VI works.

VI has found wide application in areas like Computational Biology for analyzing genetic data, Computational Neuroscience, Natrual Laguage Processing and Speech Recognition, Computer Vision and Robotics, Reinforcement Learning, Economics and so on. The simple Mean-field VI has been modified to improve it's performance. Stochastic VI allows one to use a sub-sample of the data points to compute the ELBO and update the parameters, which speeds up the algorithm and also allows to adapt the algorithm for online setting. There have been efforts in direction of capturing the inter-dependencies between various latent variables which are lost by making the mean field assumption. These include structured VI and mixture based VI. Monte-Carlo approximations to ELBO and its derivatives can also be used to handle richer forms of $q(\mathbf{z})$, likelihoods and priors.

Thus, Variational Inference is a versatile algorithm that can be modified to meet different problem requirements and allows one to decide on the trade-off between richness of model and difficulty in solving it. It is an active area of research and provides opportunity for a lot of both practical and theoretical studies.

References

[1] David M. Bleia, Alp Kucukelbirb, and Jon D. McAuliffec, *Variational Inference: A Review for Statisticians*, Journal of the American Statistical Association, 2017.

[2] Christopher M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

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

[4] Pedregosa et al., *Scikit-learn: Machine Learning in Python*, JMLR 12, 2011.

[5] *Variational Inference*, available at `https://zhiyzuo.github.io/VI/`.

[6] *Variational Inference: Ising Model*, available at `https://towardsdatascience.com/variational-inference-ising-model-6820d3d13f6a`.

Codes for various examples in the report are reproduced below.

```
# CAVI for 1D Bayesian Mixture of Gaussians

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab

np.random.seed(7)
itr_n = 40

N = 3          #number of components
sigma2 = 1     #hyperparameter, variance of the prior on means

M = 1000    #number of points per component
mu_arr = np.random.choice(np.arange(-10, 10, 2),N) + np.random.random(N)
    #Generating the means for data

#Generating the data
X = np.random.normal(loc=mu_arr[0], scale=1, size=M)
for i, mu in enumerate(mu_arr[1:]):
    X = np.append(X, np.random.normal(loc=mu, scale=1, size=M))

#declaring and initalizing the parameters and the ELBO
ELBO = np.zeros(itr_n)
phi = np.reshape(np.zeros((M*N)*(N)),(M*N,N))
mu_x = np.random.randint(int(X.min()), high=int(X.max()), size=N).astype(float)
    + X.max()*np.random.random(N)
sigma_x = np.ones(N) * np.random.random(N)

for j in range(itr_n):

    #updating the vales of parameters
    t1 = np.outer(X, mu_x)
    t2 = -(0.5*mu_x**2 + 0.5*sigma_x)
    phi = np.exp(t1 + t2[np.newaxis, :])
    phi = phi / phi.sum(1)[:, np.newaxis]

    mu_x = (phi*X[:, np.newaxis]).sum(0) * (1/sigma2 + phi.sum(0))**(-1)
    sigma_x = (1/sigma2 + phi.sum(0))**(-1)

    #calculating the ELBO
    t1 = (np.log(sigma_x) - mu_x/sigma2)*0.5
    t1 = t1.sum()
    t2 = -0.5*np.add.outer(X**2, mu_x**2)
    t2 += np.outer(X, mu_x)
    t2 -= np.log(phi)
    t2 *= phi
    t2 = t2.sum()
```

```
    ELBO[j] = t1+t2

plt.figure()
plt.title('Bayesian Mixture of Gaussians')
plt.xlabel('x'); plt.ylabel('probability')
plt.hist(X[:M], normed=True, bins=100,color='gold',alpha=0.4)
plt.hist(X[M+1:2*M], normed=True, bins=100,color='darkorange',alpha=0.4)
plt.hist(X[2*M+1:3*M], normed=True, bins=100,color='navy',alpha=0.4)
plt.plot(np.linspace(-12, 12, 100),mlab.normpdf(np.linspace(-12, 12, 100),
    mu_x[0], 1))
plt.plot(np.linspace(-12, 12, 100),mlab.normpdf(np.linspace(-12, 12, 100),
    mu_x[1], 1))
plt.plot(np.linspace(-12, 12, 100),mlab.normpdf(np.linspace(-12, 12, 100),
    mu_x[2], 1))
#plt.savefig('F:/Courses/Computational Statistics/gmm_vi_1d.png')

plt.figure()
plt.title('ELBO for Mixture of Gaussian problem')
plt.xlabel('iterations'); plt.ylabel('ELBO objective')
plt.plot(ELBO, color='b', lw=2.0, label='ELBO')
#plt.savefig('F:/Courses/Computational Statistics/gmm_vi_1d_elbo.png')
```

```
#VI for Bi-variate Gaussian

import itertools
import numpy as np
from scipy import linalg
import matplotlib.pyplot as plt
import matplotlib as mpl
from sklearn import mixture

#plot_results function has been used from the documentation and examples of
    scikit learn
color_iter = itertools.cycle(['navy', 'c', 'cornflowerblue', 'gold',
                              'darkorange'])
def plot_results(X, Y_, means, covariances, index, title):
    splot = plt.subplot(2, 1, 1 + index)
    for i, (mean, covar, color) in enumerate(zip(
            means, covariances, color_iter)):
        v, w = linalg.eigh(covar)
        v = 2. * np.sqrt(2.) * np.sqrt(v)
        u = w[0] / linalg.norm(w[0])
        # as the DP will not use every component it has access to
        # unless it needs it, we shouldn't plot the redundant
        # components.
        if not np.any(Y_ == i):
```

```
            continue
        plt.scatter(X[Y_ == i, 0], X[Y_ == i, 1], .8, color=color)

        # Plot an ellipse to show the Gaussian component
        angle = np.arctan(u[1] / u[0])
        angle = 180. * angle / np.pi # convert to degrees
        ell = mpl.patches.Ellipse(mean, v[0], v[1], 180. + angle, color=color)
        ell.set_clip_box(splot.bbox)
        ell.set_alpha(0.5)
        splot.add_artist(ell)

    plt.xlim(-5., 5.)
    plt.ylim(-5., 5.)
    plt.xticks(())
    plt.yticks(())
    plt.title(title)


# Number of samples per component
n_samples = 500
np.random.seed(1)

#Generating some synthetic data
num_components = 5
mu_arr = np.array([[-2.5,2.5],[-1.5,1],[3,-1],[1,1],[-3,-1]])
n_sample = 1000
rho_arr =
    np.array([[[0.1,0],[0,0.7]],[[0.7,-0.2],[-0.2,0.2]],[[0.6,0.1],[0.1,0.4]],[[0.1,0],[0,0.1]]
X = np.random.multivariate_normal(mu_arr[0,:],rho_arr[0,:,:], n_sample)
for i in range(num_components-1):
    X = np.append(X,np.random.multivariate_normal(mu_arr[i+1,:],rho_arr[i+1,:,:],
        n_sample),axis=0)


#average log-likelihood of the data
log_like = np.zeros(45)
for i in range(45):

    #VI with random itialization
    #This function uses Dirichlet prior
    dpgmm2 = mixture.BayesianGaussianMixture(n_components=5, max_iter =
        i+1,init_params = 'random',
                                        covariance_type='full').fit(X)
    if(i%10 == 0):
        plot_results(X, dpgmm2.predict(X), dpgmm2.means_, dpgmm2.covariances_, 1,
             'Bayesian Mixture of Gaussians')
        plt.show()
```

```
    log_like[i] = (dpgmm2.score(X[:,:]))

plt.plot(log_like, color='b', lw=2.0, label='Average log-likelihood')
plt.title('Bivariate Mixture of Gaussians')
plt.xlabel('iterations'); plt.ylabel('Average log-likelihood')
plt.legend(loc='upper left')
plt.figure()
#plt.savefig('F:/Courses/Computational Statistics/log_like.png')

#VI with kmeans for itialization
dpgmm = mixture.BayesianGaussianMixture(n_components=5, max_iter =
    10,init_params = 'kmeans',
                                        covariance_type='full').fit(X)
plot_results(X, dpgmm.predict(X), dpgmm.means_, dpgmm.covariances_, 1,
             'Bayesian Gaussian Mixture with kmeans for intialization')
plt.show()
```

```
#Image de-noising using Ising model

import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from PIL import Image
from scipy.special import expit as sigmoid
from scipy.stats import multivariate_normal

np.random.seed(0)
sns.set_style('whitegrid')

#data generating method adopted from ref [6] in report
#loading the image
#please modify the loaction of the file depending on where you have it saved
data = Image.open('F:/Courses/Computational Statistics/kung.bmp')
img_tmp = np.double(data)
img = img_tmp[:,:,0]
#binarizing the image
img_mean = np.mean(img)
img_binary = -1*(img>img_mean) + +1*(img<img_mean)
[M, N] = img_binary.shape

#mean-field parameters
sigma = 2  #noise level
y = img_binary + sigma*np.random.randn(M, N) #y_i ~ N(x_i; sigma^2);
J = 1 #coupling strength (w_ij) assumed constant
rate = 0.5 #update smoothing rate, lambda
max_iter = 20
ELBO = np.zeros(max_iter)
```

```
Hx_mean = np.zeros(max_iter)

#generate plots
plt.figure()
plt.imshow(y)
plt.title("observed noisy image")
#plt.savefig('F:/Courses/Computational Statistics/ising_vi_observed_image.png')

#Mean-Field VI
print ("running mean-field variational inference...")

logp1 = np.reshape(multivariate_normal.logpdf(y.flatten(), mean=+1,
    cov=sigma**2), (M, N))
logm1 = np.reshape(multivariate_normal.logpdf(y.flatten(), mean=-1,
    cov=sigma**2), (M, N))
logodds = logp1 - logm1

#initialize
mu = np.reshape(np.zeros((M+2)*(N+2)),(M+2,N+2))
mu[1:M+1,1:N+1] = 2*sigmoid(logodds)-1 #mu_init

a = mu[1:M+1,1:N+1] + 0.5 * logodds
qxp1 = sigmoid(+2*a) #q_i(x_i=+1)
qxm1 = sigmoid(-2*a) #q_i(x_i=-1)

for i in (range(max_iter)):
    muNew = mu

    #sum of neighbouring states
    m_sum = np.reshape(np.zeros((M)*(N)),(M,N))

    for ix in range(1,N+1):
        for iy in range(1,M+1):
            #calculating sum of neighbouring states
            m_sum[iy-1,ix-1] = J*(mu[iy,ix-1] + mu[iy,ix+1] + mu[iy-1,ix] +
                mu[iy+1,ix])

            #updating means
            muNew[iy,ix] = (1-rate)*muNew[iy,ix] + rate*np.tanh(m_sum[iy-1,ix-1]
                + 0.5*logodds[iy-1,ix-1])

    mu = muNew

    a = mu[1:M+1,1:N+1] + 0.5 * logodds
    qxp1 = sigmoid(+2*a) #q_i(x_i=+1)
    qxm1 = sigmoid(-2*a) #q_i(x_i=-1)
    Hx = -qxm1*np.log(qxm1+1e-10) - qxp1*np.log(qxp1+1e-10) #entropy
```

```
    ELBO[i] = ELBO[i] + np.sum(m_sum*(qxp1 - qxm1)) +np.sum(qxp1*logp1 +
        qxm1*logm1) + np.sum(Hx)
    Hx_mean[i] = np.mean(Hx)

plt.figure()
plt.imshow(mu)
plt.title("after %d mean-field iterations" %max_iter)
#plt.savefig('F:/Courses/Computational Statistics/ising_vi_denoised_image1.png')

plt.figure()
plt.plot(ELBO, color='b', lw=2.0, label='ELBO')
plt.title('Variational Inference for Ising Model')
plt.xlabel('iterations'); plt.ylabel('ELBO objective')
plt.legend(loc='upper left')
#plt.savefig('F:/Courses/Computational Statistics/ising_vi_elbo1.png')

plt.figure()
plt.plot(Hx_mean, color='b', lw=2.0, label='Avg Entropy')
plt.title('Variational Inference for Ising Model')
plt.xlabel('iterations'); plt.ylabel('average entropy')
plt.legend(loc="upper right")
#plt.savefig('F:/Courses/Computational Statistics/ising_vi_avg_entropy1.png')
```

Figure 6: Image used to obtain the binary noisy image data for Ising model