Bayesian Batch Approach to Active Learning

Piyush Gaurav
20104442

Shashwat Vaibhav
20111056

Jay Vora
20111023

Muskan Rathore
20111036

1 Problem description/motivation

Traditionally, designers tend to consider the entire data set while inferring a machine learning model. However, when the data sets are too large to assess, it is rational to consider only a subset of data points that can capture the entire data set's property, rather than going for the entire data set. In such cases, *Active Learning* comes to the rescue. We are planning to take up a Bayesian approach to overcome such a situation.

In our approach, we aim to select points that maximally reduce uncertainty in posterior over model parameters. In Active Learning, traditionally, we use sequential greedy methods, wherein we keep selecting the best data point and updating the model until exhaustion. This method was used in MAXENT (by maximizing predictive entropy), and BALD (by maximizing decrease in expected entropy) approaches of Active Learning. However, selecting a single point in each iteration has almost negligible impact on the training model and was not able to avoid correlated queries. As a solution, we come up with this approach. Rather than selecting a single point, we will select a batch of points in each iteration and avoid correlated queries by making a sparse approximation of data points.

2 Literature review and description of prior work on the problem

Interpreting a massive amount of data possesses inherent limitations for the modern data analyst. Training a model is not just computationally expensive and slow but also heavily unreliable in such a setting. The field of Bayesian Active Learning approximates complete data manifold using a lesser yet statistically representative version of the data [1, 2]. In order to speed up the learning curve, the machine selects data points for annotation (annotators can be human). Active Learning has been approached broadly in two ways, namely Non Probabilistic and Probabilistic.

Non Probabilistic approaches are primarily model specific for e.g. [3] for linear regression, [6] for k-nearest neighbor. Probabilistic approaches mainly perform Bayesian Optimization to minimize a loss function [6, 7]. As the data size grows, these optimization problems become computationally intensive. [8, 9] adopts a strategy to online update models as the data arrives. [10] provides a principled, scalable, and model-independent approach to Bayesian Active Learning, focusing on Bayesian Coresets [11, 12] but avoids updating the model after every data point. [13, 14] explores the idea of pseudo coresets and uses a variational inference method to derive a Bayesian posterior to ensure scalability and robustness even for high dimensional data. Since these algorithms create pseudo data sets, they tend to offer high security to sensitive data.

The algorithm in our referred paper [10] while selecting batch of data points gives a time complexity of $\mathcal{O}(|P^2|)$ initially, where $|P|$ is pool data size. This restricts our approach to moderately-sized pool sets. We address both of these issues using random feature projections in our algorithm (discussed below), which scales our algorithm to pool set size $|P|$. Batches can be efficiently constructed in $\mathcal{O}(|P|J)$ time. In results, we get to see that this enables us to scale our algorithm up to pool sets comprising hundreds of thousands of examples.

Our referred research paper's [10] works for any Machine Leaning model with a tractable likelihood. This approach uses the model to predict labels of selected data points and update the model accordingly, though not necessarily update after every data point.

A similar approach was used in the paper on "Active Learning for Convolutional Neural Networks: A coreset approach" [15] which formulates Active Learning as a coreset selection problem. It constructs batches by solving the k-center problem, thus attempting to minimize the maximum distance to one of the k queried data points. Though this approach is suitable for large pool sets but since it is heavily dependent on heavily structured feature space, our referred approach [10] still has the edge over it.

3 Procedure

Consider an initial set of labelled data $\mathcal{D}_O = \{\mathbf{x_n}, \mathbf{y_n}\}_{\mathbf{n=1}}^{\mathbf{N}}$. This data is utilized to learn the posterior distribution $(p(\theta|\mathcal{D}_O)$ of a discriminative model parameter (θ). In the Active Learning scenario, the learner also has access to an unlabelled data pool $(\mathcal{X}_p = \{\mathbf{x_m}\}_{\mathbf{m=1}}^{\mathbf{M}})$. The learner can choose specific data from this pool and request for its label $(\mathcal{Y}_p = \{\mathbf{y_m}\}_{\mathbf{m=1}}^{\mathbf{M}})$ from an oracle or expert.

The key idea here is to construct (iteratively) a batch $\mathcal{D}^{'}$ such that it's log posterior $(p(\theta|\mathcal{D}_O \cup \mathcal{D}^{'}))$ best approximates the complete data log posterior $(p(\theta|\mathcal{D}_O \cup \mathcal{D}_p)$, $\mathcal{D}_p$ being the Data pool for AL). Complete Data log posterior can be represented as:

$$log(p(\theta|\mathcal{D}_O \cup (\mathcal{X}_P, \mathcal{Y}_P))) = log(p(\theta|\mathcal{D}_O)) + log(p(\mathcal{Y}_P|\mathcal{X}_P, \theta)) + log(p(\mathcal{Y}_P|\mathcal{X}_P, \mathcal{D}_O))$$

Now, since $\mathcal{Y}_P$ is not accessible unless it is queried, the Expectation of the above expression is taken with respect to the Posterior Predictive Distribution of $\mathcal{Y}_P$.

$$\mathop{\mathbb{E}}_{\mathcal{Y}_P} logp(\theta|\mathcal{D}_O \cup (\mathcal{X}_P, \mathcal{Y}_P)) = \mathop{\mathbb{E}}_{\mathcal{Y}_P} logp(\theta|\mathcal{D}_O) + \mathop{\mathbb{E}}_{\mathcal{Y}_P} logp(\mathcal{Y}_P|\mathcal{X}_P, \theta) - \mathop{\mathbb{E}}_{\mathcal{Y}_P} logp(\mathcal{Y}_P|\mathcal{X}_P, \mathcal{D}_O) \quad (1)$$

$$\mathop{\mathbb{E}}_{\mathcal{Y}_P} log(p(\theta|\mathcal{D}_O \cup (\mathcal{X}_P, \mathcal{Y}_P))) = log(p(\theta|\mathcal{D}_O)) + \mathop{\mathbb{E}}_{\mathcal{Y}_P} log(p(\mathcal{Y}_P|\mathcal{X}_P, \theta)) - \mathbb{H}[log(p(\mathcal{Y}_P|\mathcal{X}_P, \mathcal{D}_O))] \quad (2)$$

$$\mathop{\mathbb{E}}_{\mathcal{Y}_P} log(p(\theta|\mathcal{D}_O \cup (\mathcal{X}_P, \mathcal{Y}_P))) = log(p(\theta|\mathcal{D}_O)) + \sum_{m=1}^{M} \left(\underbrace{\mathop{\mathbb{E}}_{y_m} log(p(y_m|x_m, \theta)) - \mathbb{H}[log(p(y_m|x_m, \mathcal{D}_O))]}_{\mathcal{L}_m(\theta)} \right) \quad (3)$$

In equation (3), the first term on the right can be neglected as it depends only on $\mathcal{D}_O$. Hence the batch can be chosen such that the complete data log posterior is best approximated by $\sum_m \mathcal{L}_m(\theta)$. The problem can now be viewed as Bayesian batch construction as a sparse approximation of complete data log posterior where $\mathcal{L}(w) = \sum_m w_m \mathcal{L}_m(\theta)$ can be viewed as vector in function space such that $w_m \in \{0, 1\}$. Subsequently, the optimization problem can be defined as:

$$\begin{aligned} w^* = &\underset{w}{minimize} \; \|\mathcal{L} - \mathcal{L}(w)\|^2 \\ &subject\,to \;\; w_m \in \{0,1\} \, \forall \, m \end{aligned} \tag{4}$$

As the optimization problem in equation (4) is intractable, it is approximately solved by Active Bayesian coreset with Frank Wolfe Optimization [13]. The binary weight constraint is replaced to be non-negative, and the cardinality constraint is changed to a polytope constraint. Hence the renewed optimization problem is as follows:

$$\begin{aligned} &\underset{w}{minimize} \, (\mathbf{1} - \mathbf{w})^T \mathbf{K} (\mathbf{1} - \mathbf{w}) \\ &subject\,to \;\; w_m \geq 0 \;\, \forall \, m, \;\; \sum_m w_m \sigma_m = \sigma \end{aligned} \tag{5}$$

where $\sigma_m = \|\mathcal{L}_m\|$, $\sigma = \sum_m \sigma_m$ and $\mathbf{K}$ is a Kernel matrix with $\mathbf{K}_{mn} = <\mathcal{L}_m, \mathcal{L}_n>$

Algorithm:

```
1. Draw parameter ($\theta$) samples from Posterior Distribution, $\{\theta_j\}_{j=1}^{J} \sim p(\theta|\mathbf{X}, \mathbf{y})$
2. Compute $\mathcal{L}_n = \frac{1}{\sqrt{J}} [\mathcal{L}_n(\theta_1), \ldots, \mathcal{L}_n(\theta_J)]^T$ for all n
3. sub-function ACS-FW ($b, \{\mathcal{L}_n\}_{n=1}^{N}, (.)^T(.)$)
   (a) Compute $\sigma_n = \sqrt{(\mathcal{L}_n)^T(\mathcal{L}_n)}$ for all n.
   (b) Compute $\sigma = \sum_n \sigma_n$
   (c) Initialize $\mathbf{w} = \mathbf{0}$
   (d) for $t \in 1, \ldots, b$ do
       i. Compute
          $f = \underset{n \in N}{argmax} \left[ (\mathcal{L} - \mathcal{L}(\mathbf{w}))^T (\frac{1}{\sigma_n} \mathcal{L}_n) \right]$
       ii. Compute
          $\gamma = \frac{\left[ (\frac{\sigma}{\sigma_f} \mathcal{L}_f - \mathcal{L}(\mathbf{w}))^T (\mathcal{L} - \mathcal{L}(\mathbf{w}) \right]}{\left[ (\frac{\sigma}{\sigma_f} \mathcal{L}_f - \mathcal{L}(\mathbf{w}))^T (\frac{\sigma}{\sigma_f} \mathcal{L}_f - \mathcal{L}(\mathbf{w})) \right]}$
       iii. Update corresponding $w$
          $w = (1 - \gamma) w + \gamma \frac{\sigma}{\sigma_f}$
   (e) return $\mathbf{w}$
```

The above algorithm firstly projects the data to the J-dimensional projection of the original data to make the algorithm compatible with large pooled non-linear models. Furthermore, it uses weighted Euclidean inner product instead of weighted Fisher inner product. The latter depends on the number of model parameters based on gradient(which makes it challenging for complex models).

Whereas the sub-function ACS-FW firstly computes the norms and initializes the weights to zero. Furthermore, then at each iteration, the algorithm greedily selects the vector $\mathcal{L}_f$ most aligned with residual error $\mathcal{L} - \mathcal{L}(w)$. The weights are then updated using the line search along with f^{th} vertex of the polytope, which by construction is the f^{th} coordinate unit vector, which will correspond to adding at most one data point to the batch in every iteration. Gradually, when we select more and more batches, we notice that the batches will become smaller and smaller as the algorithm allows us to re-select data points from previous iterations.

4 Active Learning Implementation

The active learning implementation includes preparing the data, for which we have generated a 5-featured 1-D Data for classification(logistic regression). Moreover, splitting the data into Train, Pool & Test datasets, followed by training the logistic regression model using the train data and testing it. Then, compute the acquisition function AL pool data and extract the most valuable data based on the acquisition function. Moreover, we update the training data by appending the selected batch of data points and training the model(Logistic Regression) again, followed by evaluating the updated model performance on the test data. Moreover, if the data points acquired have not passed the budget, go through the procedure again from computing the acquisition function for pool data.

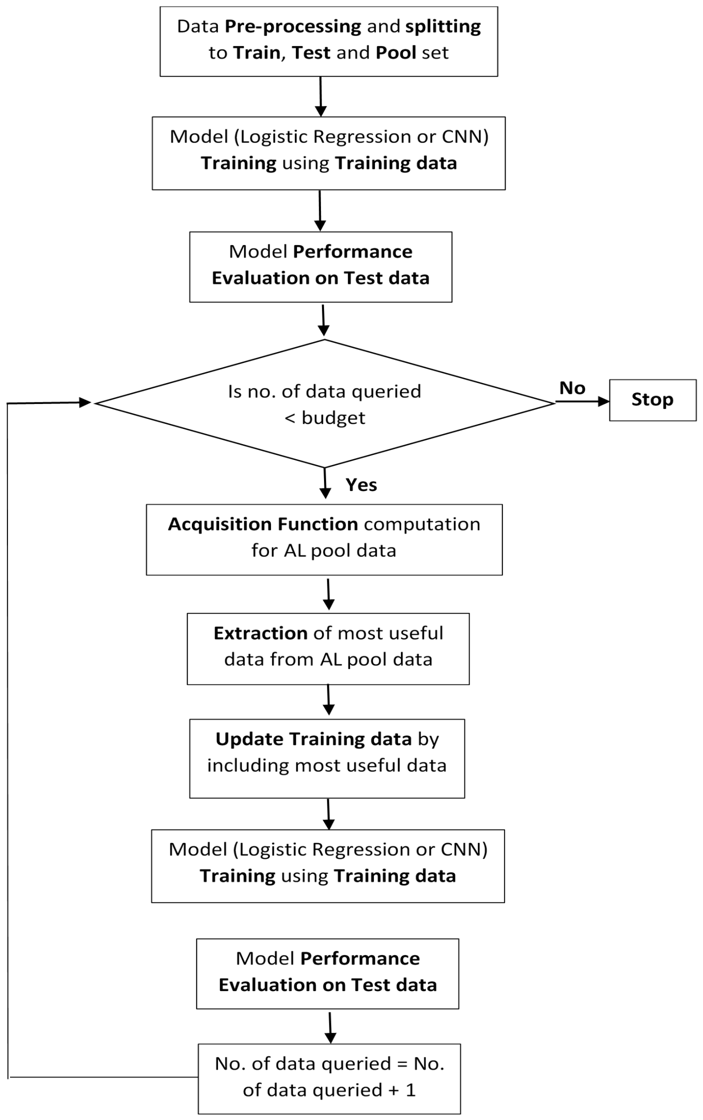
Data Pre-processing and splitting to Train, Test and Pool set
Model (Logistic Regression or CNN) Training using Training data
Model Performance Evaluation on Test data
Is no. of data queried < budget
No
Stop
Yes
Acquisition Function computation for AL pool data
Extraction of most useful data from AL pool data
Update Training data by including most useful data
Model (Logistic Regression or CNN) Training using Training data
Model Performance Evaluation on Test data
No. of data queried = No. of data queried + 1

Figure 1: Overall Active Learning Approach

The complete project is implemented on Jupyter Notebook in Python programming Language with NumPy as the primary library.[1] Models were trained using Logistic Regression (from scikit-learn). The Acquisition function requires the computation of the posterior distribution of model parameters (of Multi-class Logistic Regression). Since the Logistic Regression model has non-conjugate Likelihood and Prior w.r.t. parameters, the posterior distribution of model parameters can be computed either approximately (Laplace Approximation) or via Monte-Carlo Sampling methods. In this work, the Probabilistic programming language "PyStan" was utilized to derive samples (through Monte Carlo Sampling) of the posterior distribution of samples. These samples were utilized to project $\mathcal{L}_n$ for each θ. The expectation in the second term in R.H.S. of (3) is taken over the classes for each θ

5 Experimental results

Considering the high computational intensity of the Monte-Carlo methods, 5-dimensional artificial data is generated for classification. The data is divided into three parts: Training Data, AL Pool set data, and Test data. The classification accuracy vs data samples was plotted to demonstrate the AL process via Random selection and Active Bayesian Coreset methods. Active Bayesian Coreset method improves the accuracy faster than random selection and keeps the curve smoother due to batch selection, as seen in Figures (2) & (3).

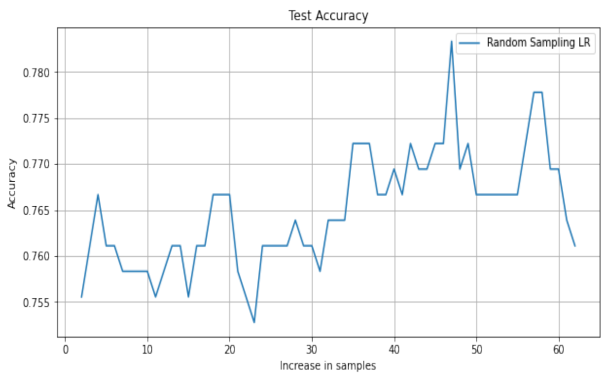

Figure 2: Random Sampling selection of data in AL setting

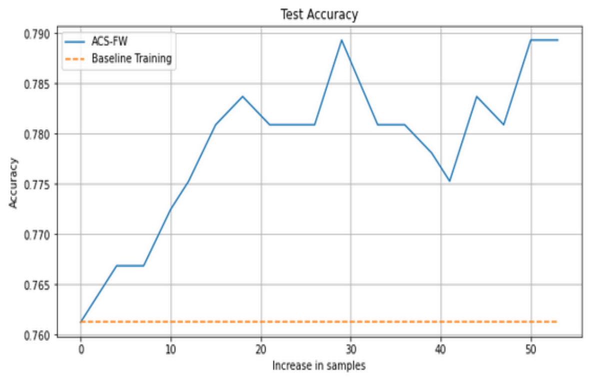

Figure 3: Selection of data using Active Bayesian Coreset in AL setting

We also attempted the algorithm on standard MNIST Data, but it became impossible to conduct the complete experiment due to limited computational resources.

[1]Codes available at `https://github.com/piyushkg/CS698X_Project.git`

Since the dimensional of an MNIST image (is 28 x 28 leading to a vector of 784 x 1) is exceptionally high for successive computation. An attempt was made to reduce the dimension to 12 x 1 using Principle component analysis. Additionally, the MNIST dataset consists of massive 60000 images, and creating a large pool set of that order was infeasible to work with. Hence it became compulsory to reject the majority of data to show the efficacy of the algorithm.

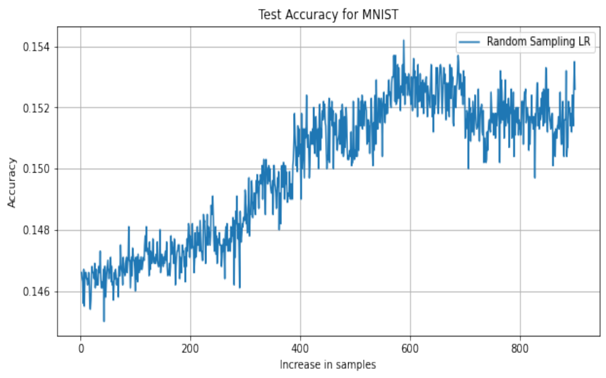

Figure 4: Random Selection of MNIST data in AL setting

Despite the above steps to lessen the computational intensity, we could carry out only the random selection of data to perform active learning. The application of the Active Bayesian Coreset method on MNIST remained infeasible (in terms of heavy computation) due to the implementation of the Monte Carlo method on high dimensional data and a vast pool set. Given this, although the code on the Active Bayesian Coreset method for MNIST was also developed, the conclusive plots could not be demonstrated.

6 Possible future work

In the survey/implementation, we noticed that the proposed algorithm was very much computationally intensive if we use some high-dimensional features for data points. As a result, the possible future work includes reducing the dimensionality of the input features to make the algorithm less computationally intensive by using (Probabilistic) PCA or using CNN to reduce the dimensionality of input features. Specifically, when working with images, CNN would work better to extract the feature from the images. Another future work includes using Deep Frank-Wolfe Optimisation or Block Frank-Wolfe Optimisation instead of vanilla Frank-Wolfe Optimisation for getting better results.

Acknowledgments

We are thankful to Dr. Piyush Rai for guiding us in the project and allowing us to work on such a great topic of “Active Learning”.

References

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

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

[3] Kai Yu, Jinbo Bi, and Volker Tresp. Active learning via transductive experimental design. In International Conference on Machine Learning, pages 1081–1088, 2006.

[4] Steven CH Hoi, Rong Jin, Jianke Zhu, and Michael R Lyu. Batch mode active learning and its application to medical image classification. In International Conference on Machine Learning, pages 417–424, 2006.

[5] Yuhong Guo and Dale Schuurmans. Discriminative batch mode active learning. In Advances in Neural Information Processing Systems, pages 593–600, 2008.

[6] Clément Chevalier and David Ginsbourger. Fast computation of the multi-points expected improvement with applications in batch selection. In International Conference on Learning and Intelligent Optimization, pages 59–69, 2013.

[7] Amar Shah and Zoubin Ghahramani. Parallel predictive entropy search for batch global optimization of expensive objective functions. In Advances in Neural Information Processing Systems, pages 3330–3338, 2015.

[8] Javad Azimi, Alan Fern, and Xiaoli Z Fern. Batch Bayesian optimization via simulation matching. In Advances in Neural Information Processing Systems, pages 109–117, 2010.

[9] Javier González, Zhenwen Dai, Philipp Hennig, and Neil Lawrence. Batch Bayesian optimization via local penalization. In Artificial Intelligence and Statistics, pages 648–657, 2016.

[10] Robert Pinsler, Jonathan Gordon, Eric Nalisnick, José Miguel Hernández-Lobato, Bayesian Batch Active Learning as Sparse Subset Approximation.

[11] Jonathan Huggins, Trevor Campbell, and Tamara Broderick. Coresets for scalable Bayesian 2016. logistic regression. In Advances in Neural Information Processing Systems, pages 4080–4088,

[12] Trevor Campbell and Tamara Broderick. Automated scalable Bayesian inference via Hilbert coresets. The Journal of Machine Learning Research, 20(1):551–588, 2019.

[13] D Manousakas, C Mascolo - β-Cores: Robust Large-Scale Bayesian Data Summarization in the Presence of Outliers arXiv preprint arXiv:2008.13600, 2020

[14] Dionysis Manousakas, Zuheng Xu, Cecilia Mascolo, Trevor Campbell Bayesian Pseudocoresets Advances in Neural Information Processing Systems, 2020

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