Course Project, CS698X, IIT Kanpur
Mentor: Piyush Rai

Scaling Up GPs and Learning Kernels from Data for GP

DS Abhishek	Harshit Nyati	Saumya Shah
160218	160282	160633

1 Motivation

The Gaussian Processes(GPs) is a very popular method for Bayesian non-linear Regression and Classification. Due to the non-parametric nature of GPs there are computational problems at both training($O(N^3)$) and test times($O(N^2)$) and also it requires $O(N^2)$ storage for a data with N inputs. So, it becomes a huge challenge for applying GPs to very large datasets. There are a different approaches to tackle this problem and in this project we studied and surveyed one such approach - Inducing Point Methods in detail. We tried various approximations possible via Inducing points methods and compared across these methods.
Another topic that we studied is learning kernels from data for Gaussian Processes to give better predictive performance as compared to other well known kernels. Using some standard kernels like the Squared Exponential Kernel (RBF kernel) allows use to learn very limited patterns in the data. We would like to study ways in which this can be resolved. Specifically, we are interested in studying kernels whose Fourier Transform is taken as a Gaussian Mixture.

2 Literature Review

2.1 Inducing Point Methods

2.1.1 Work done by Seeger et al.

In order to scale down the computational complexity sparse approximations have been made to full GPs. The idea behind this is to select/learn a subset of points ($M << N$) which are then used for further computations. The original work in [1] discusses one such method in which points(which were named as 'active set' in this method) are chosen as subset from original data points. This subset is used to approximate the kernel matrix and compute the Gaussian Process. This method lacked a reliable way of learning the kernel hyperparameters in parallel with the active set selection.

2.1.2 SPGP - Snelson et al.

The problem faced by Seegar et al. is that their method lacked a reliable way parallely learning both kernel hyperparameters and to choose which points to be included in the subset('active set') for GP approximation. This problem was tackled in [2] by suitably learning active set such that they need not belong to the input data. These points are called 'pseudo inputs'. The pseudo-inputs and kernel hyperparameters are learned by a smooth gradient optimization of marginal likelihood.

2.1.3 Work by Quinonero-Candela et al.

The work in [3] analysed various existing SPGP techniques and presented a new unifying view to gaussian sparse approximations. Earlier others papers had interpretation of GPs as- 'approximate inference with the exact prior'. Their work gave a new way to look into this approximation. They started off with posterior distribution(for regression) and

analysed its form(keeping the likelihood same for all approximations) , and found out what was the 'effective prior' each approximation was using. Thus they reinterpreted the existing techniques for sparse approximations as - exact inference with an approximated prior. The advantage of this approach was that now the approximations can directly be interpreted from the 'effective prior' and hence it becomes easier to understand how do various approximations affect the posterior. Since this approach of 'approx prior' was applied on all existing SPGP approximations, this unifying view helped compare various approximations and see how are they similar or different from each other. It made easier to study,analyse,compare and find relations between various SPGP approximations like SoD, DTC, FITC. Normal GP is written in a new way: Assuming $\mathbf{f}_*$ and $\mathbf{f}$ are conditionally independent given $\mathbf{u}$ (inducing variables). $\mathbf{u}$ are like latent variables in this view, and has same distribution as '$\mathbf{f}$' evaluated at inducing point inputs($X_\mathbf{u}$).

$$p(\mathbf{u}) = \mathcal{N}(0, \mathbf{K_{uu}})$$

'u' are called inducing variables because it induces dependencies between f_* and f, means information between f_* and f can only be transferred through 'u'. The approximation on joint prior is such that:

$$p(\mathbf{f}_*, \mathbf{f}) = q(\mathbf{f}_*, \mathbf{f}) = \int q(\mathbf{f}_*|\mathbf{u})q(\mathbf{f}|\mathbf{u})p(\mathbf{u})d\mathbf{u}$$

Different proposed SPGP approximations have different assumptions for $q(\mathbf{f}_*|\mathbf{u})$ and $q(\mathbf{f}|\mathbf{u})$. To compare with original GP, we can write these conditions for original GPs as well. These are:

$$\text{Training Conditional: } p(\mathbf{f}|\mathbf{u}) = \mathcal{N}(K_{\mathbf{f},\mathbf{u}}K_{\mathbf{u},\mathbf{u}^{-1}}\mathbf{u}, K_{\mathbf{f},\mathbf{f}} - \mathbf{Q}_{\mathbf{f},\mathbf{f}})$$

$$\text{Test Conditional: } p(\mathbf{f}_*|\mathbf{u}) = \mathcal{N}(K_{*,\mathbf{u}}K_{\mathbf{u},\mathbf{u}^{-1}}\mathbf{u}, K_{*,*} - \mathbf{Q}_{*,*})$$

$$\mathbf{Q}_{\mathbf{a},\mathbf{b}} = K_{\mathbf{a},\mathbf{u}}K_{\mathbf{u},\mathbf{u}^{-1}}K_{\mathbf{u},\mathbf{b}}$$

Exact likelihood is used across all approximations as discussed earlier.

$$p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{f}, \sigma^2\mathbf{I})$$

Now we will describe how these training and test conditionals change in different algorithms.

2.1.3.1 SoD Approximation

It can be seen as the most basic approximation possible. In this approximation, only a subset of data is used to train the GP. Note that the GP model is same as original GP, the only difference is now only a subset of M points($M << N$) are used for training model. It is like cheating with the model and giving it only limited data to train. Since only a subset of training data (SoD) is used, hence it is not expected to be much useful. It is mostly used as a baseline against which we can compare other better sparse approximation. Since only M data points are used, $O(M^3)$ is time complexity.

2.1.3.2 DTC Approximation

This is the new view for Seeger et al.'s original work in [1]. A deterministic training conditional and the exact test conditional is used here.

$$\text{Training Conditional: } q_{DTC}(\mathbf{f}|\mathbf{u}) = \mathcal{N}(K_{\mathbf{f},\mathbf{u}}K_{\mathbf{u},\mathbf{u}^{-1}}\mathbf{u}, 0)$$

$$\text{Test Conditional: } q_{DTC}(\mathbf{f}_*|\mathbf{u}) = p(\mathbf{f}_*|\mathbf{u})$$

Note that no variance in training conditional means it is deterministic.

2.1.3.3 FITC Approximation

This is the new view for Snelson et al.'s original work in [2]. It proposes a better prior approximation and has a richer covariance function for training conditional . Training conditional is not deterministic and has a diagonal covariance now. Test conditional is still the same as before and is exact. Variance of prior of training conditional chosen in such a way that independent assumption on to the training conditional distribution of f given u is satisfied.

$$\text{Training Conditional: } q_{FITC}(\mathbf{f}|\mathbf{u}) = \prod_{n=1}^{N} p(\mathbf{f}_i|\mathbf{u}) = \mathcal{N}(K_{\mathbf{f},\mathbf{u}}K_{\mathbf{u},\mathbf{u}^{-1}}\mathbf{u}, diag[K_{\mathbf{f},\mathbf{f}} - \mathbf{Q}_{\mathbf{f},\mathbf{f}}])$$

Test Conditional: $q_{FITC}(\mathbf{f}_*|\mathbf{u}) = p(\mathbf{f}_*|\mathbf{u})$

Other than mentioned above, various different approximations are unified in [3] like PITC(Partially Independent Training Conditional) and SoR(Subset of Regressors Approximation). We focused our work on the described approximations(SoD, FITC and DTC) only to find out key differences and similarities between them.

2.1.4 VFE (Variational Free Energy) Approximation

The methods covered in [3] deals with approximations in such a way that it turns inducing variables 'u' into additional kernel hyperparameters. So it can lead to overfitting at times after optimisation.
To deal with this issue, Michalis Titsias in [4] formulated inducing inputs as variational parameters.This work constructed a Variational approximation to find the posterior. Pseudo input points Z were learned alongside kernel hyperparameters by maximising ELBO. KL Divergence is minimised between true posterior and approximated posterior. Here,in the equation given below, m, Σ and Z are variational parameters which needs to be adjusted. We can give an approximation to the posterior process as:

$$q(f(.)) = GP(\mu(.), \sigma(.,.))$$

$$\mu(.) = k(.,Z)k(Z,Z)m^{-1}$$

$$\sigma(.,.) = k(.,.) - k(.,Z)[k(Z,Z)^{-1} - k(Z,Z)^{-1}\Sigma k(Z,Z)^{-1}]k(Z,.)$$

Time complexity of DTC, FITC, VFE is $O(NM^2)$ which is significantly less than $O(N^3)$ for original GP when $M << N$.

2.2 Inducing Point Methods on High Dimensional Data

2.2.1 Problems Faced

GPs generally scale well with dimensions typically linear in D. One limitation of the SPGP is that this optimization space becomes impractically big for high dimensional data sets. In standard GP the number of parameters was small - $|\theta|$. In SPGP additionally learn the M inducing points as well leading to the number of parameters to be learnt equal to MD + $|\theta|$. In cases where the number of dimensions of the input data is high i.e D is high the number of parameters is very high. The parameter M is decided by the user and can be used as a tradeoff between accuracy and computation complexity, but the dimension of the input D cannot be decided by the user. As suggested by the authors in [2], one way in which to deal with high D is to learn a low dimensional projection and then apply SPGP to the data.

2.3 Gaussian Process Kernels for Pattern Discovery and Extrapolation

2.3.1 Fourier Transform of the RBF Kernel

The work done in [5] describes a method to learn kernels by analysing their Fourier Transforms. Assuming stationary kernels, the kernel function and its spectral density are Fourier duals of each other. So, the Fourier Transform can represent the expressiveness of the kernel function. The Fourier Transform of the popular RBF kernel is a Gaussian distribution centered at the origin. This is a very specific case of a function in the frequency domain. A more generalised kernel function called the Spectral Mixture (SM) kernel has its Fourier Transform as a Mixture of Gaussians.

2.3.2 Learning the SM kernel

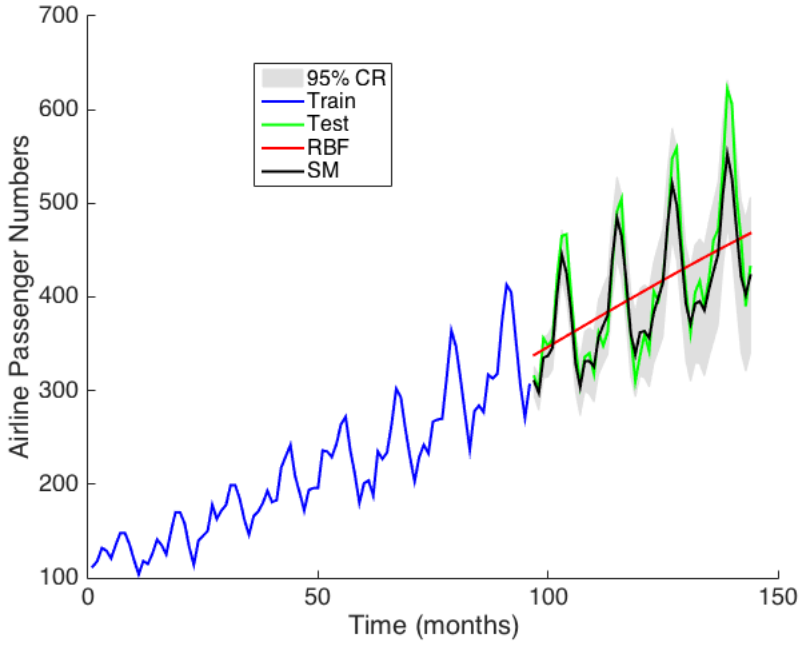
Each Gaussian in the Mixture of Gaussians distribution helps in incorporating the contribution of different frequencies. This learned SM kernel provides a better estimation of the real life density functions in which several frequencies influence the variation in data. Choosing any Gaussian mixture gives us closed form expressions for inference with Gaussian processes. The assumption of Gaussian noise is taken. This allows us to integrate out the unknown function from the likelihood in a closed form, thus giving us the marginal likelihood $p(\mathbf{y}|\theta, \{x_n\}_{n=1}^N)$. Here, $\mathbf{y}$ and $\{x_n\}_{n=1}^N$ are the data points and θ are the hyperparameters. The marginal likelihood is maximized with respect to θ to learn the hyperparameters of the SM kernel.

2.4 Performance of the SM Kernel

The Airline Passenger Data is used in [5] to demonstrate the effectiveness of the SM Kernel as compared to the popular RBF kernel. This dataset consists of airline passenger numbers, from 1949 to 1961. The results are plotted in the graph shown below. The first 96 monthly measurements, shown in blue, are used for the purpose of training. The test data consists of the next 48 measurements, shown in green. Prediction of the 48 measurements using the RBF Kernel (red) and the SM Kernel (black) is shown in the graph.

In the below figure, we can see that the airline data has mainly 2 trends: short term fluctuating trend (high frequency) and a long term increasing trend (low frequency). These two trends represent two inherent frequencies present in the data. If we use an RBF kernel, as shown by the red line, then we see that it is able to forecast the long term increasing trend. However, it fails to detect the short term fluctuating trend and treats them as noise. This makes intuitive sense because if we look at the frequency domain, then we can see that the Fourier Transform of the RBF kernel is a Gaussian. This only has a single peak. So, it will only be able to learn one of the two main frequencies present in the data. Since the peak is centred at the origin, it will learn the lower of the two frequencies i.e. the long term rising trend as seen from the red line.

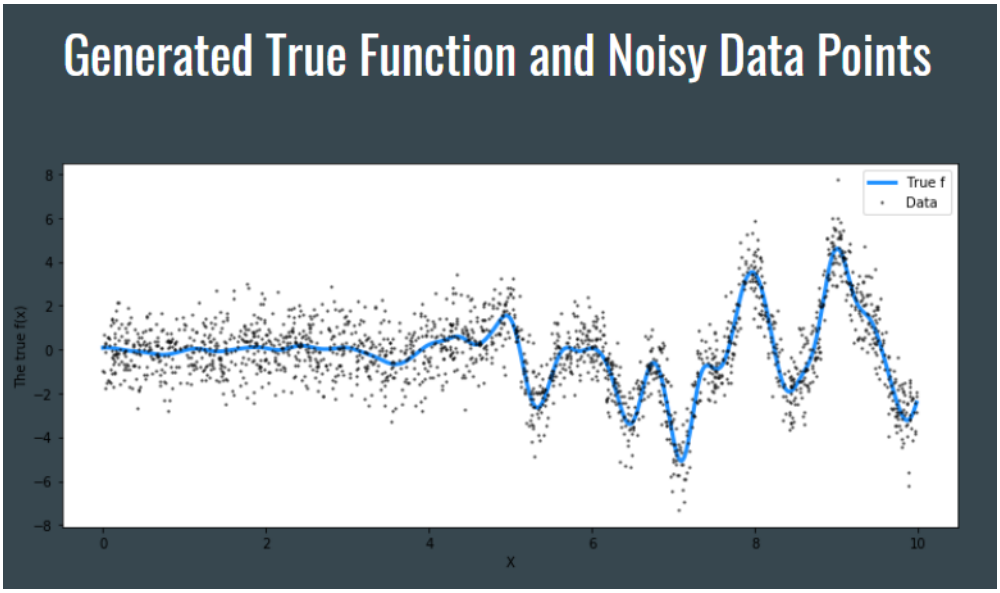
Now, if we consider the SM kernel, we can intuitively expect it to learn both of the two frequencies present in the data. This is because in the frequency domain, its Fourier Transform is a Mixture of Gaussians which has several peaks. So, it is capable of learning several different frequencies present in the data. Thus, it can reliably learn both the long term and the short term trends present in the data, which can be seen from the black line. This learning of different frequencies is further reinforced by looking at the Fourier Transform of the learned SM kernel, which has peaks close to the origin (accounting for the long term increasing trend) and also at higher values of the frequency (accounting for the short term fluctuating trend). Moreover, it is able to reasonably forecast the passenger numbers a long time after the last training data point is available.



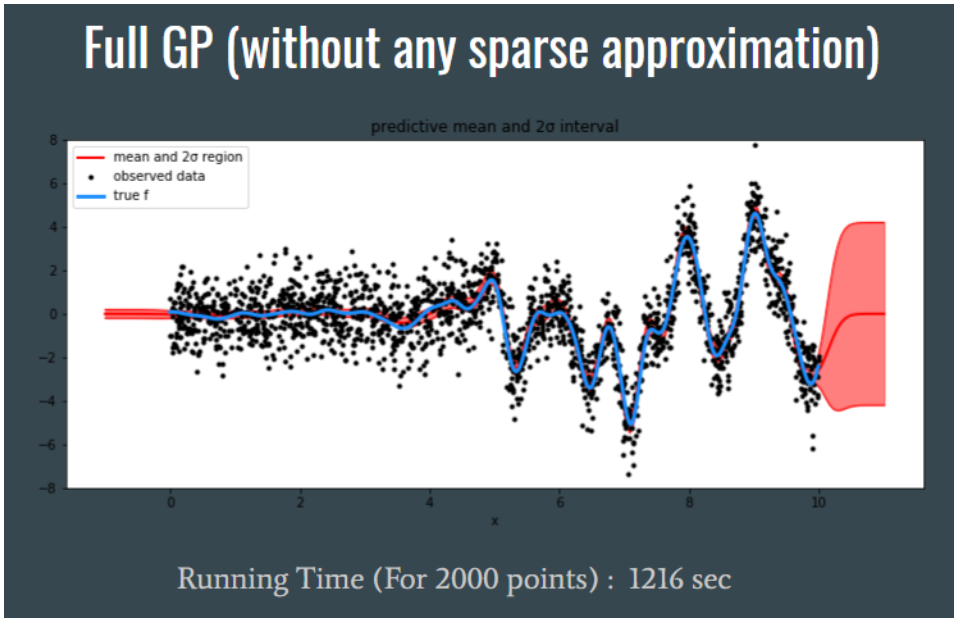
3 Experimental Results

We used PyMC3 library to generate our own true function f(x)(The function we want to find by GP) . We tried various combinations of true mean and covariance functions and got different true functions f(x). On top of it, random white

was added and we generated noisy data points 'y'. These were used as test data points. The following image shows this:

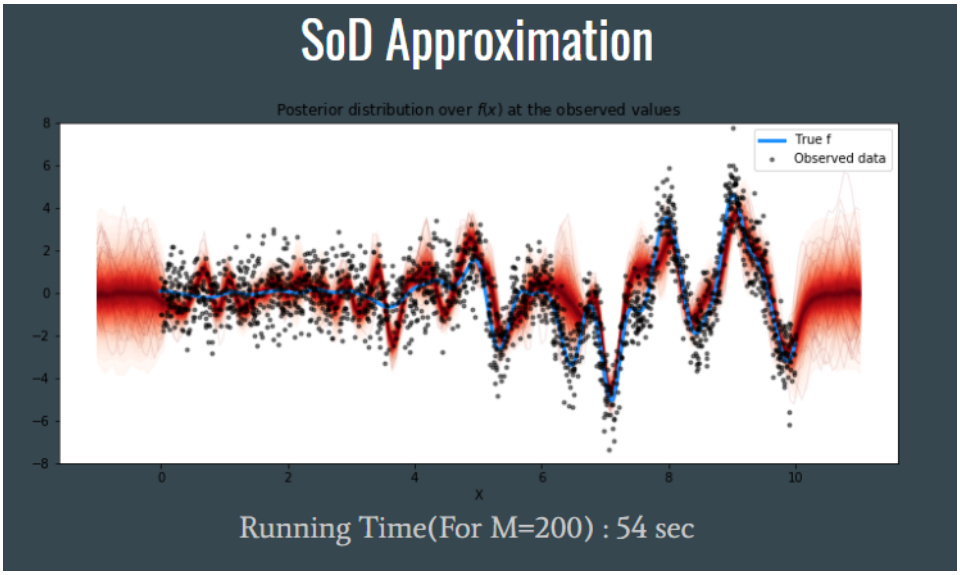


We first trained our model using original GP to observe the computation time and see how does full GP fits. Here N=2000 points were taken. Computation time was found to be 1216 sec.



We then used SoD sparse approximation by choosing M=200 data points only to train full GP. The computation time was significantly less but the accuracy loss was also significant as only limited data was used for training. This can be seen in the following figure:

Predicted function's mean is shown 'solid red' curve and 1000 random samples of predicted f_* are shown using different red curves.



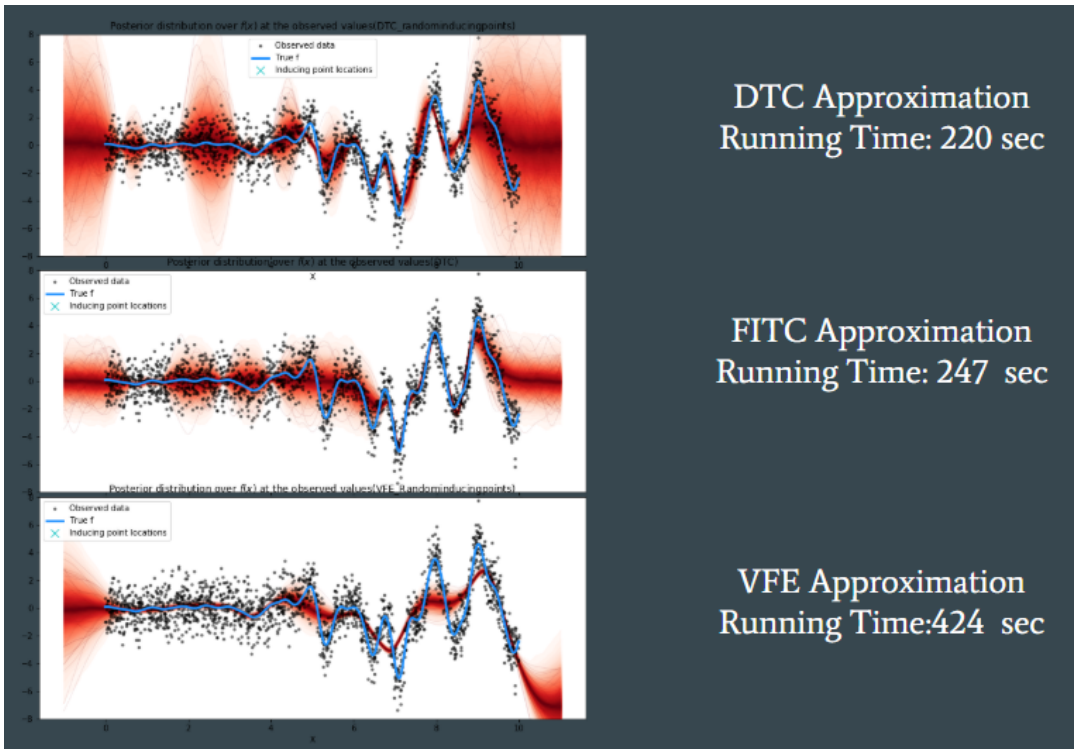
3.1 Choosing Inducing Points

We needed to make a choice that where to place the inducing points. We tried and tested various methods of Inducing points selection. :

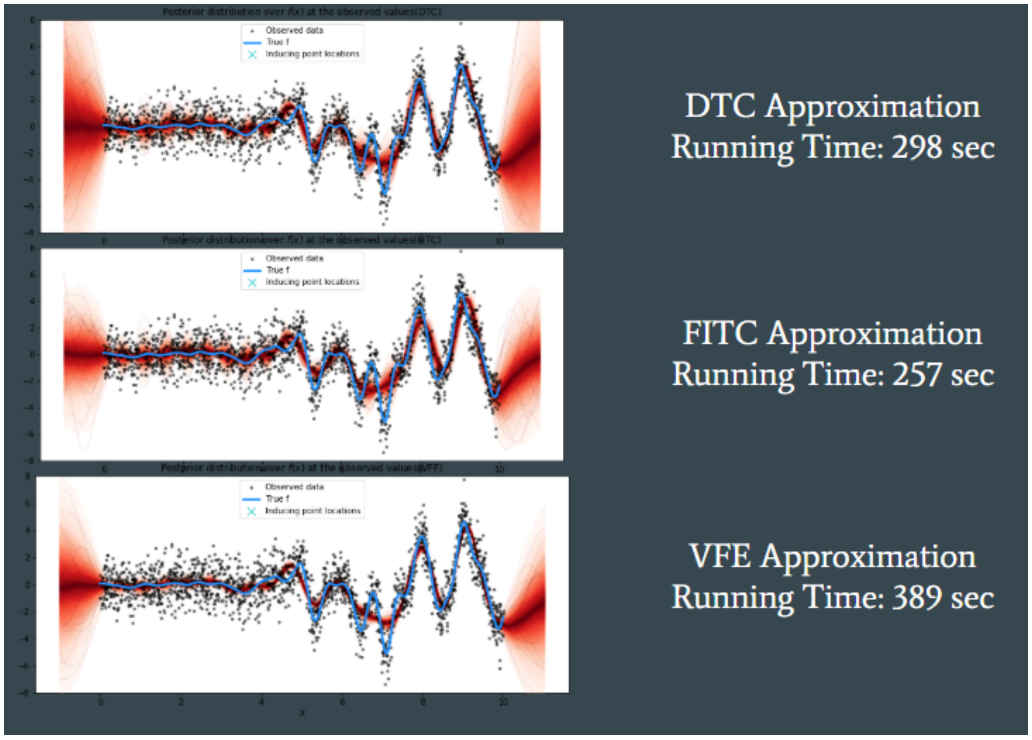
- Randomly choose inducing points as subset from the data.
- K-means optimised inducing points selection.
- Learning Inducing points by maximising log marginal likelihood.

In all these, we have used number of inducing points as 20. We tested DTC, FITC and VFE on all these choice of inducing points. We have compiled our plots obtained on a single figure for easy comparison between DTC, FITC and VFE. Alongside them, the time taken is also written.

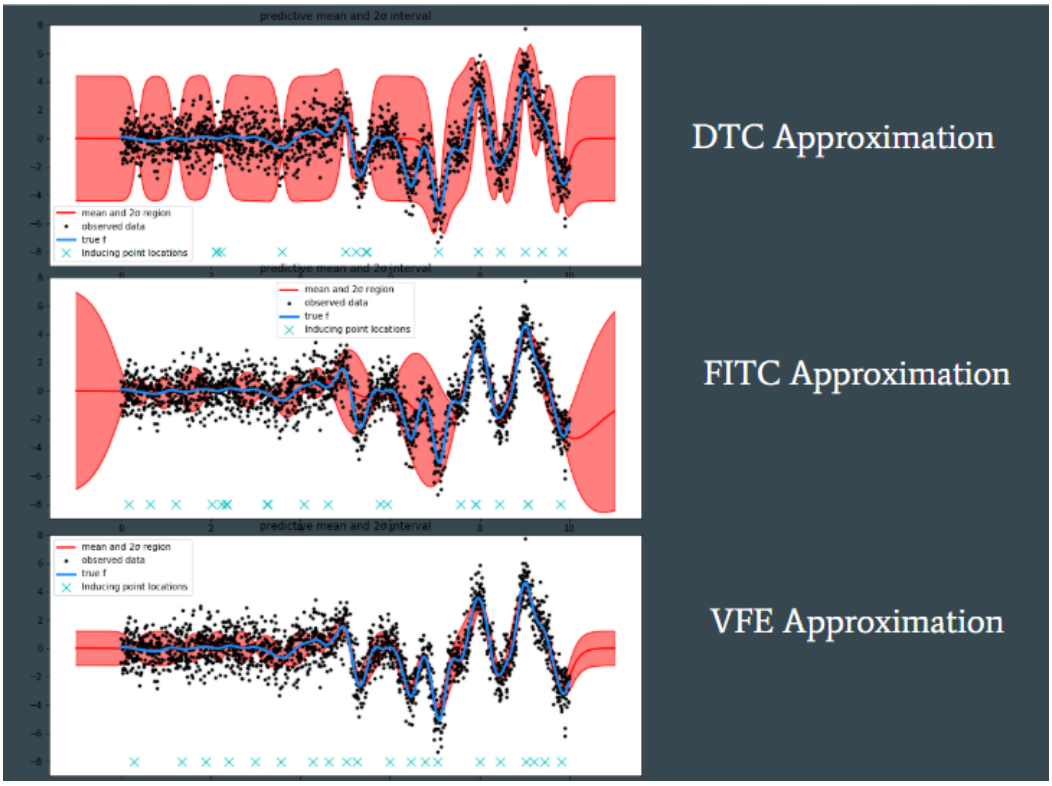
3.1.1 Random Initialisation of Inducing Points



3.1.2 K-Means Inducing Points



3.1.3 Learning Inducing Points



3.2 Conclusions for above plots (DTC, FITC and VFE Approximations)

- All 3(DTC, VFE and FITC) took significantly less time than full GP giving a reasonable fit, thus showing that reduction of time complexity is high although the loss in accuracy is low. Although the fitting was not as good as full GP but its reasonable enough. There is a trade-off between computation time and accuracy.
- The running time for code was found to be more in VFE approximation and almost similar in DTC and FITC approximation.
- K-means based learning of inducing points gave much better fitting than random initialisation based learning of inducing points.
- The DTC Approximation gives high variance as compared to other methods, this is due to the $K_{*,*} - Q_{*,*}$ term that appears in predictive distribution $q_{DTC}(f_*|y)$.
- FITC Approximation generally overfits, as the inducing points are the parameters of a very large model, and optimising so many parameters leads to overfitting.
- We performed this experiment on different number of inducing points and observed that on increasing the number of inducing points, VFE always predicts better but FITC deteriorates sometimes.
- DTC lacked a reliable way of learning the kernel hyperparameters in parallel with the active set selection. This happens because re-selecting the active set causes non-smooth fluctuations in the marginal likelihood and its gradients, meaning that they cannot get smooth convergence. FITC and VFE perform much better prediction while learning hyperparameters as can be seen from the plots above.

3.3 SPGP on High Dimensional Data

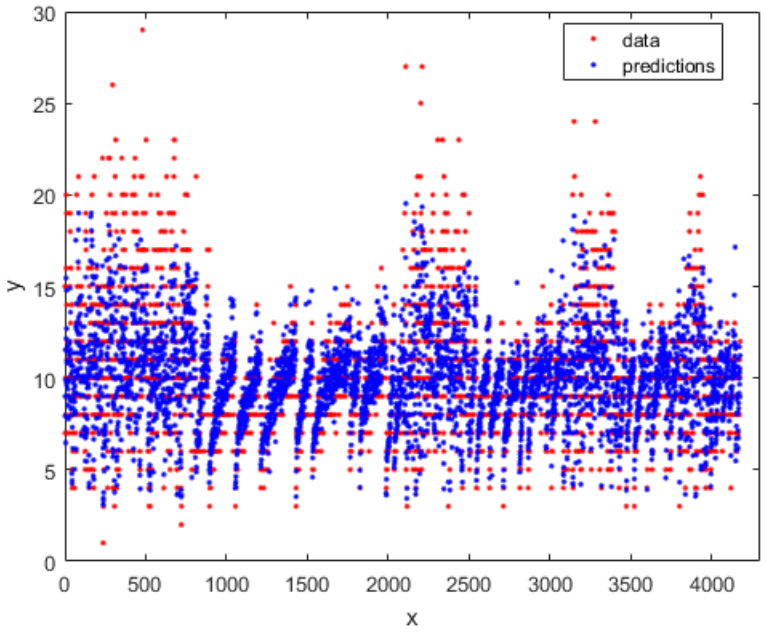


Figure 1: Caption

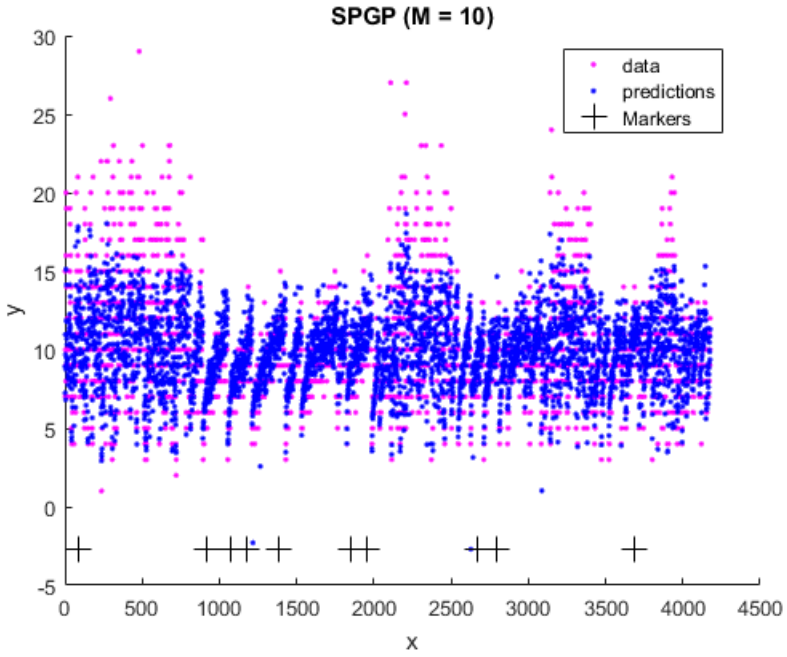


Figure 2: Caption

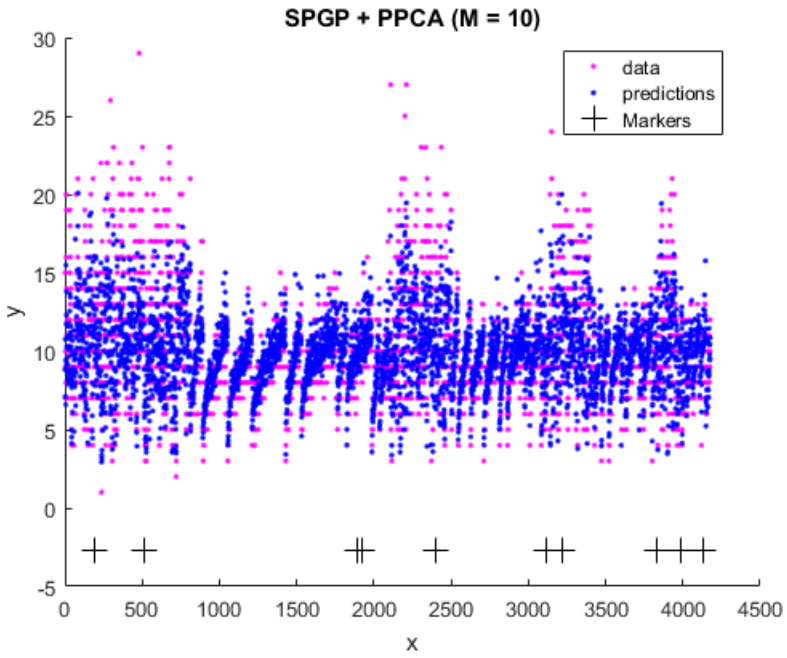


Figure 3: Caption

Model Applied	Mean Squared Error	Total Run Time(in seconds)
GP	4.0068	736.1861
SPGP (M = 10)	4.1540	20.9407
SPGP + PPCA (M = 10)	4.2730	15.5070
SPGP (M = 50)	4.5809	37.3516
SPGP + PPCA (M = 50)	4.6121	21.9410
SPGP (M = 100)	4.7567	56.8748
SPGP+ PPCA (M = 100)	4.6929	33.1260

Table 1: Mean Squared Errors and Runtimes of different models

3.3.1 Implementation

SPGP can be used to high dimensional data sets by reducing the input space to a lower dimensional space. To do this dimensionality reduction we choose a simple model - PPCA. Wedo not expect better results in terms of accuracy but we expect significant reduction in total run time. We apply SPGP as it is to the dataset without any dimensionality Reduction, then we apply SPGP to a reduced input space obtained by PPCA and compare how they perform. We have implemented on the abalone dataset.

Dataset Description: We have implemented on the abalone dataset[1]. This data has 4177 instances. Each instance has 8 values(8-dimensional input). The task here is to predicting the number of rings(equivalently age) of a abalone from physical measurements like sex, length, diameter, height,etc. . Though D here is not very large it is large enough to draw the conclusions.

[1]https://archive.ics.uci.edu/ml/datasets/abalone

3.4 Conclusions for above plots and Table(SPGP+ Dimensionality reduction)

The figures 1, 2 and 3 shows on the y-axis the predicted age(blue) and the actual age(red/magenta) of each instance(each abalone). As we observe from the table, when we use a relatively low number of inducing points(M=10), the number of parameters are less and hence the runtime differnce between SPGP and SPGP+PPCA cannot be observed clearly. Hence we tried for higher number of inducing points(M=100), in this case we observe that the runtime for SPGP after dimensionality reduction is almost halved as compared to traditional SPGP. Observe that in each case, although the dimensionality reduction did not produce better performance than the standard SPGP (compared using MSE), we see that we are able to reduce the dimensions from 8 to 3 with only a slight loss in accuracy. Thus, we conclude that in cases where we have input data of very high dimensions we can use this method. The implementation of SPGP was obtained from here.

4 Tools/Softwares Used

4.1 Implementation of DTC, FITC and VFE Approximations

Our code for implementation of various approximations and full GP as discussed in section 2.1 can be found in this link- https://github.com/saumyagshah/scalingGPs. We used PyMC3 package for implementation of these methods. It is a Python package used for implementation of probabilistic machine learning problems. We extensively used Jupyter notebook for implementing our code and used its interactive environment to efficiently debug the code.

4.2 Implementation on High Dimensional Data

We implemented the SPGP on the high dimensional on MATLAB for this we used the Gaussian Process Regression(GPR) library. We also used the SPGP code provided by Snelson on his website[2].

5 Things We Learned

We learnt about different methods to learn non-linear functions using GPs on very large datasets. This project enlightened us about the vast areas of GPs and its applications on large datasets using inducing points. This project involved lot of literature review. This inculcated in us the right way to read a paper and analyse it. In the implementation part, we faced problems producing our own data and also fitting the many sparse GP Models available in PyMC3 package to our generated data. In order to solve this, we had to go through the few relevant source codes of PyMC3 GP functions which helped us understand the intricacies involved in implementing GPs (Sparse Models).
We also learned that for input data with very high dimensions, the sparse model can still be applied by reducing the dimensions with very little loss in accuracy but significant gain in computation time. We faced certain problem during the implementation to this part, such as applying our model to a different high dimension data and had to modify the code accordingly to suit our needs.
From the SM kernel, we learnt how the Fourier Transform, a concept we had learnt in signal processing, can be used to obtain a simple kernel that can generalise well to real world data and can even outperform the popular RBF kernel.

6 Conclusions

- Conclusions drawn for DTC, FITC and VFE approximations and SPGP+PPCA approximation are already written in section 3.2 and section 3.4 respectively just below graphs for the convenience of the reader.
- The performance of the various methods described above varies on different datasets as some models may perform better on some datasets as they may show some properties which are particularly suitable to that dataset.
- We conclude that in cases where we have input data of very high dimensions we can apply these methods by first applying dimensionality reduction.

[2]http://www.gatsby.ucl.ac.uk/ snelson/

- The SM kernel is capable of learning different trends and patterns present in the data and is capable of learning them over long ranges.

7 Future Work

We can study methods other than inducing points which perform sparse approximation on GPs. There might be better ways to apply these models combined with dimensionality reduction(here we have used PPCA, there may exist better ways to perform this task in this case) to high dimensional datasets.

The SM kernel can be tried out on the Abelone dataset to compare its results with the other methods used. This will require us to carefully set the hyperparameters. This is because the authors have mentioned that we will require some good thought into setting the hyperparameters to try out this kernel on another dataset.

We can also study approaches that combine Deep Learning based methods with Gaussian Processes.

References

[1] Matthias Seeger, Christopher Williams, and Neil Lawrence. Fast forward selection to speed up sparse gaussian process regression. In *Artificial Intelligence and Statistics 9*, number EPFL-CONF-161318, 2003.

[2] Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Advances in neural information processing systems*, pages 1257–1264, 2006.

[3] Joaquin Quiñonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6(Dec):1939–1959, 2005.

[4] Michalis Titsias. Variational learning of inducing variables in sparse gaussian processes. In *Artificial Intelligence and Statistics*, pages 567–574, 2009.

[5] Andrew Wilson and Ryan Adams. Gaussian process kernels for pattern discovery and extrapolation. In *International Conference on Machine Learning*, pages 1067–1075, 2013.