Bayesian Optimization on Fast Nearest Neighbor Approximation

Anonymous Author(s)

Affiliation

Address

email

Abstract

Computing nearest neighbor matches in high-dimensional space is a computationally expensive task. It constitutes a major component of many computer vision problems. Linear search is the only possible method to solve these high dimensional problems exactly. Approximate algorithms can significantly improve speedup at cost of little loss of accuracy. But these approximate algorithms vary based on the type of dataset and hence choosing optimum parameters for these approximate algorithms is the contribution of Fastest Approximate Nearest Neighbor (FLANN) algorithm. This algorithm uses an optimization technique to choose a parameter set for an algorithm among a set of algorithms that is the best possible approximator for given precision and memory/speed preference for the data. The proposed optimization technique improves the query time with improved precision over current FLANN implementation. The results show that bayesian optimization can reduce the number of queries required to reach the optimal parameters. The parameters thus selected were more close to optimum solution than the original implementation. Our implementation is also available as open source demo which we intend to further integrate in the FLANN library.

1 Introduction

In computer vision algorithms, computing closest matches to high-dimensional vectors is commonly required. Computing best matches for local image features in large datasets, clustering local features to visual-words using k-means or alike clustering algorithms are some examples of nearest neighbor search techniques. Searching nearest neighbors is also of major importance in applied areas of machine learning, data compression and bioinformatics.

Nearest neighbor problem can be simply described as given a set of points \( P \) in a \( X \) vector space, we want to preprocess these points in such a way that when we query any point in the vector space \( X \), its nearest neighbor is computed efficiently. For high-dimensional spaces, there are no algorithms for nearest neighbor search that are more efficient than linear search. For applications where linear search is costly, approximate nearest neighbor search can be computed instead that are faster than exact linear search, while still providing near optimal precision.

There are several techniques to approximate nearest neighbor search. Fastest Approximate Nearest Neighbor (FLANN) [1] algorithm attempts to make a systematic approach to guide the choice among algorithms and set their parameters. These parameters change significantly varying with the properties of the datasets, such as dimensionality, correlations, clustering characteristics and size. In FLANN approach, hierarchical k-means tree and randomized kd-tree algorithms were found to be the most prominent approximators. FLANN authors propose to use Nelder-Mead downhill simplex method for parameter optimization but their results do not show any evidence of optimization on parameter selection. Meanwhile, because they discretize parameters into few sample values, the op-
timum results are likely to be sub-optimal. FLANN automatically chooses one of the two algorithms and their respective parameters for given dataset and precision. The choice of algorithm is based on linear search over the entire possible discrete values of the parameters.

We introduce an optimization approach to automatically select optimal parameters using bayesian optimization technique. The previous method use few possible discrete values of parameters to get the optimum parameters. Using bayesian optimization [2], we could explore all possible integer values of the parameters with fewer sampling to obtain the optimum bayesian estimate of the parameters. The novelty of our work lies in the underlying desire to produce more optimum results with fewer sample evaluations.

2 Background and Related Work

2.1 Previous Research

Kd-tree [3] is the most widely used algorithm for nearest neighbor search in low dimensional space but is limited by the curse of dimensionality. Approximate matching kd-tree algorithms have been proposed to get approximate nearest neighbors [4][5]. Silpa-Anan and Hartley [6] proposed multiple randomized kd-trees to speed up approximate nearest neighbor search. K-means clustering algorithm [7] is another technique for nearest neighbor matching. Recently published variations of k-means clustering [8][9] can do efficient clustering and matching of features in large datasets as well. One of the two algorithms obtained best performance on several test datasets and desired precision in an experiment conducted in [1]. These algorithms were either the hierarchical k-means tree or multiple randomized kd-trees. In Section III, we will discuss the parameters for these algorithms and the optimization technique.

2.2 Feature Descriptors

For any object in an image, a set of interesting points can be extracted to describe features of the object. The features are such that they can be extracted from the training image under varying image scale, noise and illumination. These features when match correspond to matching objects in images which can be useful in applications like object categorization, 3D reconstruction, motion tracking and segmentation, robot localization. In our experiments, we use dataset comprising of 100K scale-invariant feature transform (SIFT) [10] features. The results will compare the performance of bayesian optimization against the author’s original optimization technique.

2.3 Bayesian Optimization

Bayesian optimization is a powerful technique to find the extrema of objective functions that are expensive to evaluate. This technique is useful when the close form expression for the objective function is not available but values can be sampled from the objective function. The efficiency of bayesian optimization lies in the ability of bayesian optimization to incorporate prior belief to direct the sampling and to trade off exploitation and exploration in the search space. The posterior captures the updated belief using the prior and the sampled observations to generate new belief of unknown objective function. It allows for an elegant way by which informative priors can describe attributes of objective functions like smoothness, variance and location of extrema.

Generally, optimization problems assume the objective function to be convex. Instead, we assume the objective function to be a black box function. Objective function evaluation is strictly extracting the value of the objective function for a given value of input. However, based on the FLANN implementation, we define bounds on the parameters search space.

Bayesian optimization procedure, as described in Algorithm 1 [2], constitutes of two components: the posterior distribution of the objective function and the acquisition function. As we get more observations, \( D_{1:t} = \{ x_{1:t}, y_{1:t} \} \), a prior distribution \( P(f) \) is combined with these observations \( P(D_{1:t} | f) \) to produce the posterior distribution: \( P(f | D_{1:t}) \propto P(D_{1:t} | f) P(f) \). The posterior captures the updated beliefs about the unknown objective function. \( P(f) \) is modeled as gaussian process priors in bayesian optimization.
for $t = 1, 2, \ldots$ do
    Find $x_t$ by optimizing the acquisition function over the GP: $x_t = \arg\max_x u(x|D_{1:t-1})$.
    Sample the objective function: $y_t = f(x_t) + \epsilon_t$.
    Augment the data $D_{1:t} = \{D_{1:t-1}, (x_t, y_t)\}$ and update the GP.
end

Algorithm 1: Bayesian Optimization Algorithm

2.4 Gaussian Process Priors

Gaussian process (GP) is an extension of multivariate gaussian distribution to larger dimension where any finite combination of dimensions will be a gaussian distribution. As gaussian distribution over a random variable can be specified by mean and covariance, a GP can be completely specified by its mean function, $m$ and covariance function, $k$:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x'))$$

We assume the prior mean to be zero function $m(x) = 0$. We use squared exponential kernel to define the covariance function, $k$:

$$k(x_i, x_j) = \exp \left( -\frac{1}{2} ||x_i - x_j||^2 \right)$$

Here, the function approaches value 1 if the values are close together and 0 for values further apart. This also shows that two points that are close together have similar value and have large influence on each other. Two points that are far from each other are more likely to have different values and have less influence on each other. To sample from the prior, we choose $\{x_{1:t}\}$ and sample values of the function to get pairs $\{x_{1:t}, f_{1:t}\}$, where $f_{1:t} = f(x_{1:t})$. The function values are drawn from normal distribution $\mathcal{N}(0, \mathbf{K})$, where $\mathbf{K}$, kernel matrix is

$$\mathbf{K} = \begin{bmatrix} k(x_1, x_1) & \cdots & k(x_1, x_t) \\ \vdots & \ddots & \vdots \\ k(x_t, x_1) & \cdots & k(x_t, x_t) \end{bmatrix}$$

For optimization, given a pair of previous observations $\{x_{1:t}, f_{1:t}\}$, we want to decide new point $x_{t+1}$ to be considered next. If $f_{t+1} = f(x_{t+1})$, by using properties of GPs, $f_{t+1}$ and $f_{t+1}$ are jointly gaussian.

$$\begin{bmatrix} f_{t+1} \\ f_{t+1} \end{bmatrix} \sim \mathcal{N}(0, \begin{bmatrix} \mathbf{K} & \mathbf{k} \\ \mathbf{k}^T & k(x_{t+1}, x_{t+1}) \end{bmatrix})$$

where

$$\mathbf{k} = [k(x_{t+1}, x_1), k(x_{t+1}, x_2), \ldots, k(x_{t+1}, x_t)]$$

Now the predictive distribution is

$$P(f_{t+1}|D_{1:t}, x_{t+1}) = \mathcal{N}(\mu_t(x_{t+1}), \sigma^2_t(x_{t+1}))$$

where

$$\mu_t(x_{t+1}) = \mathbf{k}^T \mathbf{K}^{-1} f_{1:t}$$

$$\sigma^2_t(x_{t+1}) = k(x_{t+1}, x_{t+1}) - \mathbf{k}^T \mathbf{K}^{-1} \mathbf{k}$$

The choice of next sampling point $x_{t+1}$ depends on the acquisition function. The role of the acquisition function is to guide the search algorithm towards the optimum. Acquisition functions tend to balance the tradeoff between exploiting locations nearby with high value of objective function or exploring locations with high variance to minimize uncertainty in keeping track of global optimum. Maximizing the acquisition function is used to select the next location to evaluate the objective function. We use GP-Upper Confidence Bound (GP-UCB) as acquisition function in our case. The parameters to weight the variance was empirically selected as 0.1.

$$\text{GP-UCB}(x) = \mu(x) + \sqrt{\nu \sigma(x)}$$

The readers are suggested to refer [2] for detailed description of bayesian optimization.
3 Bayesian Optimization on FLANN

Optimal algorithm selection for fast approximate nearest neighbor search depends highly on the structure of the dataset and desired search precision. Algorithms also have their own set of parameters to influence the search performance. The parameters for hierarchical k-means trees are branching factor and number of iterations while the parameter for randomized kd-trees is the number of kd-trees. In [1], the authors consider algorithms as a parameter of a generic nearest neighbor search algorithm to present optimization problem as a problem of optimizing the parameters. The cost function was generated using combination of search time, build time and tree memory overhead. The relative importance of these factors was controlled by setting build-time weight, \( w_b \) and a memory weight, \( w_m \). The overall cost was given by

\[
J_1 = \frac{s + w_b b}{(s + w_b b)_{opt}} + w_m m
\]  

where \( s \) represents the search time for the number of vectors in the sample dataset, \( b \) represents the tree build time and \( m = m_t/m_d \) represents the ratio of memory used for the tree, \( m_t \) and memory used by the algorithm to store the data, \( m_d \). The build-time weight, \( w_b \) controls the relative importance of build-time over search time. When \( w_b = 0 \), we want the algorithm to ignore tree build time and maximize search speed. When \( w_b = 1 \), we want the algorithm to give equal importance to both tree build time and search time. The denominator \( (s + w_b b)_{opt} \) is the optimal search and build-time if memory usage is not a factor. This computation was feasible in the approach discussed in [1] because the set of parameters were fixed. The authors chose \( \{1, 4, 8, 16, 32\} \) as number of random kd-trees, \( \{16, 32, 64, 128, 256\} \) as the branching factor for the k-means tree and \( \{1, 5, 10, 15\} \) as the number of k-means iteration.

In Bayesian optimization, we consider the entire range of possible values of parameters. So, we choose number of random kd-trees in the closed interval \([1,32]\), branching factor for k-means tree in the closed interval \([16,256]\) and number of k-means iteration in the closed interval \([1,15]\). We want to avoid computing \( (s + w_b b)_{opt} \) as it requires to compute the cost of the function for each possible value of the parameter. This approach defeats the purpose of bayesian optimization. So, we assume that the denominator term can be neglected and still obtain the cost function of similar nature. Our desire is to find the location of the maximum cost, we choose this new cost function as

\[
J_2 = s + w_b b + w_m m
\]

In Section IV, we show that our optimization cost function is close approximation of cost function of the FLANN algorithm.

4 Experiment

4.1 Cost Function Synergy

First, we prove that our proposed cost function has similar nature to the FLANN cost function. The results of cost functions generated using two randomly selected weights for build time and memory

\[
J_2 = s + w_b b + w_m m
\]

Figure 1: Optimization cost function of FLANN \((J_1)\) and proposed algorithm \((J_2)\).
Figure 2: (a) Kd-tree and (b) K-means with build time weight $w_b = 0$ and memory weight $w_m = 1$, precision = 60%.

cost for kd-trees is shown in Fig. 1. The costs $J_1$ (shown in red) are the costs normalized with the optimum values while the costs $J_2$ (shown in blue) are unnormalized costs. The nature of the curves for unnormalized and normalized costs show similar trend. By visual inspection, we agree to the argument that optimizing the unnormalized cost would return same set of parameters as normalized cost function. Similar trend would be observed in the two-dimensional case of hierarchical k-means clustering as well.

4.2 Bayesian Optimization

To compute the cost function for bayesian optimization, we extracted tree build time, search time for a query and memory used. Existing FLANN library did not provide access to these parameters. FLANNs source files were hacked to extract these values. Python bindings for hacked source file were created to extract the values in python environment. Queries were made to FLANN library to return the cost parameters for given algorithm, required precision and weights to build time and memory cost. The cost parameters were used to compute the cost that was to be optimized in the bayesian optimization framework. Unlike the authors approach to treat different algorithms as parameters, we separated two optimization problems of kd-trees and k-means clustering. The formulation of bayesian optimization of kd-trees took number of kd-trees as input based on which the cost of nearest neighbor approximation would be computed. This is one-dimensional Bayesian optimization problem. For k-means clustering, the parameters were in two-dimensional space namely i) branching factor and ii) number of iterations. In this case, we implemented a two-dimensional bayesian optimization. To find the best parameter, we choose the best of the two algorithms. Since computing cost for a set of parameters is computationally expensive operation, we precomputed the costs for each value of number of kd-trees for kd-tree optimization. To optimize our bayesian optimization, we created a lookup table comprising all the costs for all possible values of the parameter for the kd-tree.

In case of hierarchical k-means, we could not compute the cost for all possible values of the parameter because they are computationally expensive in higher dimensions. Hence, we propose to sample the cost for different parameters at coarse resolution. And later, we performed linear interpolation to compute cost for all possible values in the parameter space of the k-means.
Figure 3: (a) Kd-tree and (b) K-means with build time weight $w_b = 0$ and memory weight $w_m = 1$, precision = 90%.

Now, let's look at the results obtained using bayesian optimization when $w_b = 0$ and $w_m = 1$ for precision of 60% and 90% respectively. For 60% accuracy, FLANN autotuning algorithm selected k-means with 32 branching factor and 10 iterations. Whereas, for 90% accuracy, FLANN autotuning algorithm selected k-means with 128 branching and 15 iterations. In Fig. 2(a), we observe the negative of cost (which we want to maximize), is maximum for kd-tree of size 1. But value of cost function of the optimum kd-tree parameter is less than that of maximum of kmeans with 10 branching and approximately 32 iterations as shown in Fig. 2(b). It is to be noted that the objective functions that we are attempting to maximize can be monotonically increasing or decreasing at times. In those cases, optimization algorithm would fall into either corner of the plots. We also observe that, number of iterations in k-means did not have significant change in the cost function. Hence, the parameter selection for number of iterations in k-means is fuzzy. However, we got the near optimal approximation of the parameters by sampling fewer points that the original autotuning algorithm. Hence, our approach is computationally more efficient as well as more optimal to the original paper.

In Fig. 3(a), the kd-trees cost function is gradually decreasing as the number of kd-trees increases. Hence, the optimum parameter to choose would be the lowest possible value of kd-tree which is 1. For k-means however, we see that the cost function does not change quite as much in the direction of number of iterations. The number of branching was approximately 128 which matches with the autotuned parameters from FLANN. And, once again we obtain better results that autotuned FLANN by sampling fewer points than the original autotuning algorithm. Our approach is computationally more efficient as well as more optimal to the original paper.

5 Conclusion

The approach described in this paper, bayesian optimization for automatic algorithm configuration allows user to achieve high performance in approximate nearest neighbor matching. Simple tweaks have been made to the current FLANN library to make the algorithm compatible for bayesian optimization. The user only has to provide desired precision and importance weights of minimizing memory or build time rather than just search time. Currently, we optimize two algorithms separately...
to compute the optimum set of parameters for individual algorithm. We finally combine those costs to see which algorithm would provide more optimal results. In our experiments, we have found that usually the cost function decreases with increasing size and desired precision. But, these changes are monotonic within the parameter space which makes the optimization problem challenging. We also observed that changing the number of iterations of k-means did not produce any significant change in the cost function and hence had little contribution to choosing parameters for k-means. The other observations that we make are coherent with the original paper. We also observed that, either of the two algorithms could outperform the other depending on the dataset and desired precision. One of the methods was the hierarchical k-means tree with priority search order and the other was multiple randomized kd-trees. Using bayesian optimization can increase the speed up significantly and also provide more optimal solution than the traditional FLANN implementation.

Integrating our add-on to the FLANN library would be the next step of this project. Based on this research, we would like to suggest future recommendations to extend this work. Currently, we are performing optimization on two algorithms separately. Another possible option would be to run both optimization processes in parallel and use the best cost to sample the next parameter for both algorithm. Other potential improvement in future could be to develop an acquisition function to sample across algorithms in multidimensional parameter space would be useful addition to this work. Another drawback to this work was the nature of the cost function. The cost functions were either monotonically increasing or decreasing and invariant with certain features at times. These cost functions did not provide a good distribution of cost along certain axes such as number of iterations of k-means algorithm. The optimization can yield better results for more suitably designed cost functions.

Acknowledgments

I acknowledge the original work by authors Marius Muja and David Lowe on FLANN. I thank Nando De Freitas for his suggestions and guidance throughout the course project. I also thank anonymous reviewers for their efforts in gathering thoughtful feedback to this work.

References


