Unsupervised Discretization Using Kernel Density Estimation

Marenglen Biba, Floriana Esposito, Stefano Ferilli, Nicola Di Mauro, Teresa M.A Basile
Department of Computer Science, University of Bari
Via Orabona 4, 70125 Bari, Italy
{biba,esposito,ferilli,ndm,basile}@di.uniba.it

Abstract

Discretization, defined as a set of cuts over domains of attributes, represents an important preprocessing task for numeric data analysis. Some Machine Learning algorithms require a discrete feature space but in real-world applications continuous attributes must be handled. To deal with this problem many supervised discretization methods have been proposed but little has been done to synthesize unsupervised discretization methods to be used in domains where no class information is available. Furthermore, existing methods such as (equal-width or equal-frequency) binning, are not well-principled, raising therefore the need for more sophisticated methods for the unsupervised discretization of continuous features. This paper presents a novel unsupervised discretization method that uses non-parametric density estimators to automatically adapt sub-interval dimensions to the data. The proposed algorithm searches for the next two sub-intervals to produce, evaluating the best cut-point on the basis of the density induced in the sub-intervals by the current cut and the density given by a kernel density estimator for each sub-interval. It uses cross-validated log-likelihood to select the maximal number of intervals. The new proposed method is compared to equal-width and equal-frequency discretization methods through experiments on well known benchmarking data.

1 Introduction

Data format is an important issue in Machine Learning (ML) because different types of data make relevant difference in learning tasks. While there can be infinitely many values for a continuous attribute, the number of discrete values is often small or finite. When learning, e.g., classification trees/rules, the data type has an important impact on the decision tree induction. As reported in [Dougherty et al.,1995], discretization makes learning more accurate and faster. In general, the decision trees and rules learned using discrete features are more compact and more accurate than those induced using continuous ones. In addition to the advantages of discrete values over continuous ones, the point is that many learning algorithms can only handle discrete attributes, thus good discretization methods are a key issue for them since they can significantly affect the learning outcome. There are different axes by which discretization methods can be classified, according to the different directions followed by the implementation of discretization techniques due to different needs: global vs. local, splitting (top-down) vs. merging (bottom-up), direct vs. incremental and supervised vs. unsupervised.

Local methods, as exemplified by C4.5, discretize in a localized region of the instance space. (i.e. a subset of instances). On the other side, global methods use the entire instance space [Chmielevski and Grzymala-Busse, 1994].

Splitting methods start with an empty list of cutpoints and, while splitting the intervals in a top-down fashion, produce progressively the cut-points that make up the discretization. On the contrary, merging methods start with all the possible cutpoints and, at each step of the discretization refinement, eliminate cut-points by merging intervals.

Direct methods divide the initial interval in n sub-intervals simultaneously (i.e., equal-width and equal-frequency), thus they need as a further input from the user the number of intervals to produce. Incremental methods [Cerquides and Mantaras, 1997] start with a simple discretization step and progressively improve the discretization, hence needing an additional criterion to stop the process.

Supervised discretization considers class information while unsupervised discretization does not. Equal-width and equal-frequency binning are simple techniques that perform unsupervised discretization without exploiting any class information. In these methods, continuous intervals are split into sub-intervals and it is up to the user specifying the width (range of values to include in a sub-interval) or frequency (number of instances in each sub-interval). These simple methods may not lead to good results when the continuous values are not compliant with the uniform distribution. Additionally, since outliers are not handled, they can produce results with low accuracy in the presence of skew data. Usually, to deal with these problems, class information has been used in supervised methods, but when no such information is available the only option is exploiting unsupervised methods. While there exist many supervised methods in literature, not much work has been done for synthe-

sizing unsupervised methods. This could be due to the fact that discretization has been commonly associated with the classification task. Therefore, work on supervised methods is strongly motivated in those learning tasks where no class information is available. In particular, in many domains, learning algorithms deal only with discrete values. Among these learning settings, in many cases no class information can be exploited and unsupervised discretization methods such as simple binning are used.

The work presented in this paper proposes a top-down, global, direct and unsupervised method for discretization. It exploits density estimation methods to select the cut-points during the discretization process. The number of cutpoints is computed by cross-validating the log-likelihood. We consider as candidate cutpoints those that fall between two instances of the attribute to be discretized. The space of all the possible cut-points to evaluate could grow for large datasets that have continuous attributes with many instances with different values among them. For this reason we developed and implemented an efficient algorithm of complexity *Nlog(N)* where *N* is number of instances.

The paper is organized as follows. In Section 2 we describe non-parametric density estimators, a special case of which is the kernel density estimator. In Section 3 we present the discretization algorithm, while in Section 4 we report experiments carried out on classical datasets of the UCI repository. Section 5 concludes the paper and outlines future work.

2 Non-parametric density estimation

Since data may be available under various distributions, it is not always straightforward to construct density functions from some given data. In parametric density estimation, an important assumption is made: available data has a density function that belongs to a known family of distributions, such as the normal distribution or the Gaussian one, having their own parameters for mean and variance. What a parametric method does is finding the values of these parameters that best fit the data. However, data may be complex and assumptions about the distributions that are forced upon the data may lead to models that do not fit well the data. In these cases, where making assumptions is difficult, non-parametric density functions are preferred.

Simple binning (histograms) is one of the most well-known non-parametric density methods. It consists in assigning the same value of the density function $\boldsymbol{f}$ to every instance that falls in the interval $[C - h/2, C + h/2)$, where C is the origin of the bin and h is the binwidth. The value of such a function is defined as follows (symbol # stands for 'number of'):

$$\boldsymbol{f} = \frac{1}{n * h} \#\{\text{instances that fall in} \left[C - \frac{h}{2}, C + \frac{h}{2} \right)\}$$

Once fixed the origin C of a bin, for every instance that falls in the interval centered in C and of width h, a block of size 1 by the bin width is placed over the interval (Figure 1). Here, it is important to note that, if one wants to get the density value in x, every other point in the same bin, contributes equally to the density in x, no matter how close or far away from x these points are.

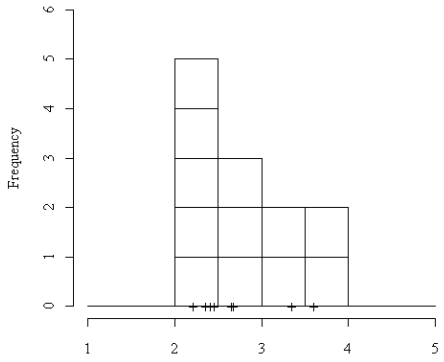

Figure 1. Simple binning places a block in every sub-interval for every instance x that falls in it

This is rather restricting because it does not give a real mirror of the data. In principle, points closer to x should be weighted more than other points that are far from it. The first step in doing this is eliminating the dependence on bin origins fixed a-priori and place the bin origins centered at every point x. Thus the following pseudo-formula:

$$\frac{1}{n * \text{binwidth}} \#\{\text{instances that fall in a bin } \underline{\text{containing}} \text{ x}\}$$

should be transformed in the following one:

$$\frac{1}{n * \text{binwidth}} \#\{\text{instances that fall in a bin } \underline{\text{around}} \text{ x}\}$$

The subtle but important difference in constructing binning density with the second formula, permits to place the bin around x and the calculation of the density is performed not in a bin containing x and depending from the origin C, but in a bin whose center is upon x. The bin center on x, allows successively to assign different weights to the other points in the same bin in terms of impact upon the density in x depending on the distance from x. If we consider intervals of width h centered on x, then the density function in x is given by the formula:

$$\boldsymbol{f} = \frac{1}{2hn} \#\{\text{instances that fall in } [\text{x} - h, \text{x} + h]\}$$

In this case, when constructing the density function, a box of width h is placed for every point that falls in the interval centered in x. These boxes (the dashed ones in Figure 2) are then added up, yielding the density function of Figure 2.
This provides a way for giving a more accurate view of what the density of the data is, called box kernel density

estimate. However, the weights of the points that fall in the same bin as x have not been changed yet.

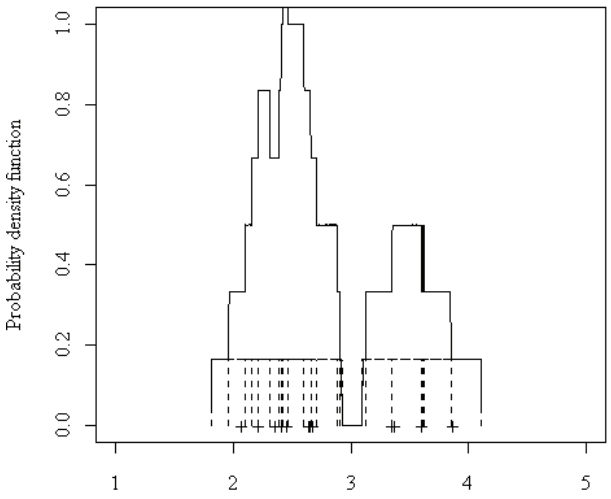

Figure 2. Placing a box for every instance in the interval around x and adding them up.

In order to do this, the kernel density function is introduced:

$$\boldsymbol{p} = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right)$$

where K is a weighting function. What this function does is providing a smart way of estimating the density in x, by counting the frequency of other points X_i in the same bin as x and weighting them differently depending on their distance from x. Contributions to the density value of $\boldsymbol{f}$ in x from points X_i vary, since those that are closer to x are weighted more than points that are further away. This property is fulfilled by many functions, that are called kernel functions. A kernel function K is usually a probability density functions that integrates to 1 and takes positive values in its domain. What is important for the density estimation does not reside in the kernel function itself (Gaussian, Epanechnikov or quadratic could be used) but in the bandwidth selection [Silverman 1986]. We will motivate our choice for the bandwidth (the value h in the case of kernel functions) selection problem in the next section where we introduce the problem of cutting intervals based on the density induced by the cut and the density given by the above kernel density estimation.

3 Where and what to cut

The aim of discretization is always to produce sub-intervals whose induced density over the instances best fits the available data. The first problem to be solved is where to cut. While most supervised top-down discretization method cut exactly at the points in the main interval to discretize that represent instances of the data, we decided to cut in the middle points between instance values. The advantage is that this cutting strategy avoids the need of deciding whether the point at which the cut is performed is to be included in the left or in the right sub-interval.

The second question is which (sub-)interval should be cut/split next among those produced at a given step of the discretization process. Such a choice must be driven by the objective of capturing the significant changes of density in different separated bins. Our proposal is to evaluate all the possible cut-points in all the sub-intervals, by assigning to each of them a score according to a method whose meaning is as follows. Given a single interval to split, any of its cut-points produces two bins and thus induces upon the initial interval two densities, computed using the simple binning density estimation formula. Such a formula, as shown in the previous section, assigns the same density value of the function $\boldsymbol{f}$ to every instance in the bin and ignores the distance from x of the other instances of the bin when computing the density in x. Every sub-interval produced has an averaged binned density (the binned density in each point) that is different from the density estimated with the kernel function. The less this difference is, the more the sub-interval fits the data well, i.e. the better this binning is, and hence there is no reason to split it. On the contrary, the idea underlying our discretization algorithm is that, when splitting, one must search for the next two worst sub-intervals to produce, where "worst" means that the density shown by each of the sub-intervals is much different than it would be if the distances among points in the intervals and a weighting function were considered. The identified worst sub-intervals are just those to be split to produce other intervals, because they do not fit the data well. In this way intervals whose density differs much from the real data situation are eliminated, and replaced by other sub-intervals. In order to achieve the density computed by the kernel density function we should reproduce a splitting of the main interval such as that in Figure 2.

An obvious question that arises is: when a given sub-interval is not to be cut anymore? Indeed, searching for the worst sub-intervals, there are always good candidates to be split. This is true, but on the other hand at each step of the algorithms we can split only one sub-intervals in other two. Thus if there are more than one sub-interval (this is the case after the first split) to be split, the scoring function of the cut-points allows to choose the sub-interval to split.

3.1 The scoring function for the cutpoints

At each step of the discretization process, we must choose from different sub-intervals to split. In every sub-interval we identify as candidate cut-points all the middle points between the instances. For each of the candidate cut-points T we compute a score as follows:

$$\text{Score(T)} = \sum_{i=1}^{k}(p(x_i) - f(x_i)) + \sum_{i=k+1}^{n}(p(x_i) - f(x_i))$$

where $i = 1,..,k$ refers to the instances that fall into the left sub-interval and $i = k+1,..,n$ to the instances that fall into the right bin. The density functions p and f are respectively the kernel density function and the simple binning density function. These functions are computed as follows:

$$f(x_i) = \frac{m}{w * N}$$

where m is the number of instances that fall in the (left or right) bin, w is the binwidth and N is the number of instances in the interval that is being split. The kernel density estimator is given by the formula:

$$p(x_i) = \frac{1}{hN} \sum_{j=1}^{N} K\left(\frac{x_i - X_j}{h} \right)$$

where h is the bandwidth and K is a kernel function. In this framework for discretization, it still remains to be clarified how the bandwidth of the kernel density estimator is chosen. Although there are several ways to do it, as reported in [Silverman 1986], in fact in this context we are not interested in the density computed by a classic kernel density estimator that considers globally the entire set of available instances. The classic way a kernel density estimation works considers N as the total number of instances in the initial interval and chooses h as the smoothing parameter. The choice of h is not easy and various techniques have been investigated to find an optimal h. Our proposal, in this context, is to adapt the classic kernel density estimator by taking h equal to the binwidth w, specified as follows. Indeed, as can be seen from the formula of $p(x_i)$, instances that are more distant than h from x_i, contribute with weight equal to zero to the density of x_i. Hence, if a sub-interval (bin) under consideration has binwidth h, only the instances that fall in it will contribute, depending on their distance from x_i, to the density in x_i. As we are interested in knowing how the current binned density (induced by the candidate cut-point and computed by f with binwidth w) differs from the density in the same bin but computed weighting the contributions of X_j to the density in x_i on the basis of the distance $x_i - X_j$, it is useless to consider, for the function p, a bandwidth greater than w.

3.2 The discretization algorithm

Once a scoring function has been synthesized, we explain how the discretization algorithm works. Figure 3 shows the algorithm in pseudo language. It starts with an empty list of cut-points (that can be implemented as a priority queue in order to maintain, at each step, the cut-points ordered after their value according to the scoring function) and another priority queue that contains the sub-intervals generated thus far. Let us see it through an example. Suppose the initial interval to be discretized is the one in Figure 4 (frequencies of the instances are not shown).

```
Discretize(Interval)
Begin
   PotentialCutpoints = ComputeCutPoints(Interval);
   PriorityQueueIntervals.Add(Interval);
While stopping criteria is not met do
   If PriorityQueueCPs is empty
         Foreach cutpoint CP in PotentialCutpoints do
         scoreCP = ComputeScoringFunction(CP,Interval);
         PriorityQueueCPs.Add(CP,scoreCP);
         End for
   Else
   BestCP = PriorityQueue.GetBest();
   CurrentInterval = PriorityQueueIntervals.GetBest();
   NewIntervals = Split(CurrentInterval,BestCP);
   LeftInterval = NewIntervals.GetLeftInterval();
   RightInterval = NewIntervals.GetRightInterval();
   PotentialLeftCPs  = ComputeCutPoints(LeftInterval);
   PotentialRightCPs =ComputeCutPoints(RightInterval);
         Foreach cutpoint CP in PotentialLeftCPs
   scoreCP = ComputeScoringFunction(CP,LeftInterval);
   PriorityQueueCPs.Add(CP,scoreCP);
   PriorityQueueIntervals.Add(LeftInterval,scoreCP);
         End For
         // the same foreach cycle for PotentialRightCPs
End while
End
```

Figure 3. The discretization algorithm in pseudo language

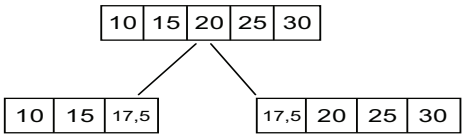

Figure 4. The first cut

The candidate cut-points are placed in the middle of adjacent instances: 12.5, 17.5, 22.5, 27.5; the sub-intervals produced by cut-point 12.5 are [10 , 12.5] and [12.5 , 30], and similarly for all the other cut-points. Now, suppose that, computing the scoring function for each cut-point, the greatest value (indicating the cut-point that produces the next two worst sub-intervals) is reached by the cut-point 17.5. Then the sub-intervals are: [10 , 17.5] and [17.5 , 30] and the list of candidate cut-points becomes <12.5, 16.25, 18.75, 22.5, 27.5>. Suppose the scoring function evaluates as follows: Score(12.5) = 40, Score(16.25) = 22, Score(18.75) = 11, Score(22.5) = 51, Score(27.5) = 28. The algorithm selects 22.5 as the best cut-point and splits the corresponding interval as shown in Figure 5.

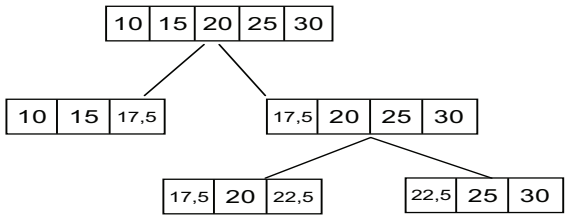

Figure 5. The second cut

This second cut produces two new sub-intervals and hence the current discretization is made up of three sub-intervals: [10 , 17.5], [17.5 , 22.5], [22.5 , 30], with candidate cut-points <12.5, 18.75, 23.75, 27,75>. Suppose values of the scoring function are as follows: Score(12.5) = 40, Score(18.75) = 20, Score(23.75) = 35, Score(27,5) = 48. The best cut-point 27.5 suggests the third cut and the discretization becomes [10 , 17.5], [17.5 , 22.5], [22.5 , 27.5], [27.5 , 30]. Thus, the algorithm refines those sub-intervals that show worst fit to the data. A note is worth: in some cases it might happen that a split is performed even if one of the two sub-intervals (which could be the left or the right one) it produces shows such a good fit, compared to the other sub-intervals, that it is not split in the future. This is not strange, since the scoring function evaluates the overall fit of the two sub-intervals. This is the case of the first cut in the present example: the cut-point 17.5 has been chosen, where the left sub-interval [10 , 17.5] shows good fit to the data in terms of density while the right one [17.5 , 30] shows bad fit. In this case the interval [10 , 17.5] will not be cut before the interval [17.5 , 30] and perhaps will remain untouched till the end of the discretization algorithm. The algorithm will stop cutting when the stopping criterion (the maximal number of cut-points, computed by a procedure explained in the next paragraph) is met.

3.3 Stopping criteria and complexity

The definition of a stopping criterion is fundamental, to prevent the algorithm from continuing to cut until each bin contains a single instance. Even without reaching such an extreme situation, the risk of running into overfitting the model is real, because, as usual in the literature, we use log-likelihood to evaluate the density estimators, the simple binning and the kernel density estimate. As a solution, instead of requiring a specific number of intervals (that could be too rigid and not based on valid assumptions), we propose the use of cross-validation to provide an unbiased estimation of how the model fits the real distribution. For the experiments performed the 10-fold cross-validation was used. For each fold the algorithm computes the stopping criterion as follows: Supposing there are $N - 1$ candidate cut-points, for each of them the cross-validated log-likelihood is computed. In order to optimize performance, at each step a structure maintains the sub-intervals in the current discretization and the corresponding splitting values, so that only the new values for the interval to be split have to be computed at each step. Thus the algorithm that computes the log-likelihood for the N – 1 cut-points is performed 10 times overall. The number of cut-points that shows the maximum value of the averaged log-likelihood on the test folds is chosen as the best. The log-likelihood on the test data is given by the following formula:

$$\text{Log-likelihood} = \sum_{j=1}^{n} n_{j-test} \log \frac{n_{j-train}}{w * N}$$

where $n_{j\text{-}train}$ is the number of training instances in bin j, $n_{j\text{-}test}$ is the number of test instances that fall in bin j, N is the total number of instances and w is the width of bin j.

As regards the kernel density estimator complexity, from the formula of p, it can be deduced that the complexity for evaluating the kernel density in N points is N^2. For univariate data, the complexity problem has been solved by the algorithms proposed in [GreenGard and Strain, 1991] and [Yang et al 2003] which compute the kernel density estimate in $O(N+N)$ instead of $O(N^2)$. In our context we deal only with univariate data because only single continuous attributes have to processed, and thus for N instances, the theoretical complexity of the algorithm is $O(N\log N)$.

4 Experiments

In order to assess the validity and performance of the proposed discretization algorithm, we have performed experiments on several datasets taken from the UCI repository and classically used in the literature to evaluate discretization algorithms in the past. Specifically, the dataset used are: autos, bupa, wine, ionosphere, ecoli, sonar, glass, heart, hepatitis, arrhythmia, anneal, cylinder, and auto-mpg. These datasets contain a large set of numeric attributes of various types, from which 200 continuous attributes were extracted at random and used to test the discretization algorithm.

In order to evaluate the discretization carried out by the proposed algorithm with respect to other algorithms in the literature, we compared it to three other methods: equal-width with fixed number of bins (we use 10 for the experiments), equal-frequency with fixed number of bins (we use 10 for the experiments), equal-width cross-validated for the number of bins. The comparison was made along the log-likelihood on the test data using a 10-fold cross-validation methodology. The results on the test folds were compared through a paired t-test as regards cross-validated log-likelihood. Table 1 presents the results of the t-test based on cross-validated log-likelihood with a risk level $\alpha = 0.05$. It shows the number of continuous attributes whose discretization through our method was significantly better, equal or significantly worst compared to the other methods.

	Our method significanlty more accurate	Equal	Our method significanlty less accurate
EqualWidth 10 bins	71 (35,5%)	126	3 (1,5%)
EqualFreq 10 bins	79 (39,5%)	119	2 (1,0%)
EqualWidth Cross-Validated	54 (27,0%)	136	10 (5,0%)

Table 1. Results of paired t-test based on cross-validated log-likelihood on 10 folds.

It is clear that, even if in the majority of cases the new algorithm shows no difference in performance with respect to

the others, there is an outstanding percentage of cases (at least 27%) in which it behaves better, while the opposite holds only in very rare cases. Among the datasets there can be found many cases of continuous attributes whose interval of values contain many occurrences of the same value. This characteristic had an impact on the results of the equal frequency method that often, in such cases, was not able to produce a valid model that could fit the data. This is natural, since this method creates the bins based on the number of instances that fall in it. For example if the total number of instances is 200 and the bins to generate are 10, then the number of instances that must fall in a bin is 20. Thus, if among the instances there is one that has 30 occurrences, then the equal frequency method is not able to build a good model because it cannot compute the density of the bin that contains only the occurrences of the single instance. This would be even more problematic in case of cross-validation, which is the reason why no comparison with the Equal Frequency Cross-Validation method was carried out.

An important note can be made concerning (very) discontinuous data, on which our method performs better than the others. This is due to the ability of the proposed algorithm to catch the changes in density in separated bins. Thus very high densities in the intervals (for example large number of instances in a small region) are "isolated" in bins different from those which "host" low densities. Although it is not straightforward to handle very discontinuous distributions, the method we have proposed achieves good results when trying to produce bins that can fit these kind of distributions.

5 Conclusions and future work

Discretization represents an important preprocessing task for numeric data analysis. So far many supervised discretization methods have been proposed but little has been done to synthesize unsupervised methods. This paper presents a novel unsupervised discretization method that exploits a kernel density estimator for choosing the intervals to be split and cross-validated log-likelihood to select the maximal number of intervals. The new proposed method is compared to equal-width and equal-frequency discretization methods through experiments on well known benchmarking data. Preliminary results are promising and show that kernel density estimation methods are good for developing sophisticated discretization methods. Further work and experiments are needed to fine-tune the discretization method to deal with those cases where the other methods show better accuracy.

As future application we plan to use the proposed discretization algorithm in a learning task that requires discretization and where no class information is always available. One such context could be Inductive Logic Programming, where objects whose class is not known, are often described by continuous attributes. This investigation will aim at assessing the quality of the learning task and how this is affected by the discretizaton of the continuous attributes.

References

[Cerquides and Mantaras, 1997] Cerquides. J and Mantaras R.L. Proposal and empirical comparison of a parallelizable distance-based discretization method. In *KDD97. Third International Conference on Knowledge Discovery and Data Mining*, pp. 139 - 142.

[Chmielevski and Grzymala-Busse, 1994] Chmielevski, M.R and Grzymala-Busse,J.W. Global discretization of continuous attributes on preprocessing for machine learning. In *Third International Workshop on Rough Sets and Soft Computing*, pp. 294-301, 1994.

[Dougherty et al..,1995] Dougherty.J.,Kohavi,R., and Sahami,M. Supervised and unsupervised discretization discretization of continuous features. In *Proc. Twelfth International Conference on Machine Learning,* Los Altos, CA:Morgan Kaufman,pp 194-202, 1995.

[Grengard and Strain 1991] Greengard, L. and Strain, J.The fast Gauss Transform. *SIAM Journal of Scientific and statistical computing*. 12, 1, 79-94.

[Silverman 1986] Silverman, B.W. Density estimation for statistics and data analysis. Chapman and Hall, London, 1986.

[Yang et al 2003] Yang, C., Duraiswami, R., and Gumerov, N. 2003. Improved fast Gauss transform. *Tech. Rep.CS-TR-4495*, Dept. of Computer Science, University of Maryland, College Park.