An Investigation of Approximate Nearest Neighbor Techniques: Assumptions and Efficiency

Andrew Wagenmaker
University of Washington
ajwagen@cs.washington.edu

Emil Azadian
University of Washington
emilaz@cs.washington.edu

Romain Camilleri
University of Washington
camilr@cs.washington.edu

1 Introduction

With the extensive growth of available datasets of different kinds, retrieving information quickly, efficiently and accurately is an important and well-studied topic. A special case of such information retrieval is approximate nearest neighbor search, where the goal is to find similar datapoints in a dataset for a given query. An algorithm aimed at solving this problem will have to overcome several challenges arising with the potentially large amount of high-dimensional data. First, memory space is limited and needs to be intelligently used. Second, standard approaches to finding nearest neighbors are subject to the curse of dimensionality, having a runtime that is dependent on the dimension of the input and size of the dataset. For extremely large amounts of data, these approaches will take too much computation time. In short, a good algorithm for this problem should (1) use memory efficiently (2) be fast, all the while (3) yielding accurate results. A final piece of the puzzle is the ability of a method to adapt to the underlying distribution and efficiently query in a data-dependent fashion.

In our project, we investigate three different approaches for querying a dataset and compare their performance over several different metrics. In particular, we are interested in the ability of each method to adapt to the underlying data distribution. In doing so, we attempt to identify the settings in which standard euclidean Locality Sensitive Hashing (LSH) [DIIM04] breaks down. As for the two additional approaches, we compare LSH to Product Quantization (PQ) as described in [JDS11] and Spectral Hashing (SH) [WTF09]—both of which are data-driven methods. A more in-depth description of these algorithms can be found in Section 3. Note that a comparison among these methods is important because one wants to achieve a good approximation of the nearest neighbors independent of the underlying distribution of the data, especially when applied to high-dimensional real-world datasets, where insight into the distribution might be hard to obtain.

1.1 Hypothesis

As discussed in Section 3.1.1, LSH is agnostic to the underlying distribution of the data on which it performs nearest neighbor search. We thus hypothesize that LSH will produce vastly different results for different types of distributions. We further conjecture that the two data-dependent approaches to ANN are more robust to data distribution types. In summary, the goals of our project are to (a) carefully quantify distributions and real world tasks where LSH breaks down and (b) analyze other ANN methods that are able to overcome the shortcomings of LSH in these cases.

In order to test our hypothesis, we run extensive experiments for each method on both real and synthetic data. The experiments on synthetic data, described in Section 4, allow us to accurately quantify the shortcomings of each method for the specific distributions of data. For our experiments on real data, outlined in Section 5, we test each approach on a subset of ImageNet [DDS$^+$09] and a

corpus of texts from Wikipedia. In each case, we design specific real-world tasks that allow us to identify the shortcomings of each method.

2 Methodology

As mentioned above, to efficiently and accurately retrieve data points, we plan to run and compare several different approaches—euclidean LSH [DIIM04], Product Quantization (PQ) [JDS11], and Spectral Hashing (SH) [WTF09]. At a high level, PQ seeks to quantize the feature vectors in an intelligent way in order to more efficiently search the dataset, taking into account the underlying distribution when generating hash functions. SH, on the other hand, uses a spectral based algorithm and binary quantization to take into account the distribution and efficiently search the data.

Implementations of all methods are in Python. For LSH, we wrote an implememtation ourselves, following the algorithm outlined in [DIIM04]. For Product Quantization, we used the nanopq Python implementation[1], and for Spectral Hashing the following implementation[2]. Due to the variance in the implementation efficiency, it is difficult to compare the performance of the different methods against each other in terms of absolute runtime. Instead, we will focus on the relative change by which the runtime increases when sweeping over different paramaters, e.g. database size. It is worth mentioning that a highly optimized implementation of LSH and PQ exists in the FAISS library developed by Facebook[3]. However, the version of LSH implemented in FAISS is not the standard Euclidean LSH proposed in [DIIM04] and, for both their LSH and PQ implementations, the library does not allow for the low-level control we needed to perform our experiments. For the aforementioned reasons, and to make fair comparisons between approaches, we use a pure Python implementation of each method.

The primary metric we use to measure the performance of each method is the recall@k metric. For some set of data points queried, the recall@k corresponds to the ratio of queries for which the actual closest point was among the k closest matches returned. Hence, evaluating this method requires computing the ground truth top k nearest neighbors of each query. We do this via a linear search. In all cases, we use the Euclidean norm to measure distances between points.

3 Detailed Algorithm Descriptions

We briefly describe the approximate nearest neighbor methods we investigated in this project. These approaches fall into roughly two categories, both relevant for our project. When performing ANN search, Locality Sensitive Hashing is completely agnostic to the distribution of the data. In contrast, both Spectral Hashing and Product Quantization take into account the data distribution in their ANN search and, as such, we believe that they may significantly outperform LSH on certain tasks.

3.1 Locality Sensitive Hashing [DIIM04]

As a baseline, we investigated the Locality Sensitive Hashing method discussed in class. Since we are interested in the Euclidean distance between points, we created our hash functions by randomly generating hyperplanes, projecting data points onto them, and binning the points on this 1D subspace. Formally, following [DIIM04], we generated vectors $a \sim \mathcal{N}(0, I)$ and $b \sim \mathcal{N}(0, 1)$. For every data point x, we then projected it onto the hyperplane induced by a, b using the formula:

$$\left\lfloor \frac{a^\top x + b}{r} \right\rfloor$$

where here r is a scaling factor that quantifies the size of the bins. As is shown in [DIIM04], hash functions generated in this way are locality preserving for the Euclidean distance.

After generating these hash functions, we grouped them together using the approach outlined in class. Namely, we define L bins of k functions each. Two data points x, y are then denoted as "similar" if, for any bin, for all k hash functions in that bin both x and y hash to the same value. For any query, once a set of similar points is identified, linear search is then run over these points to determine which is the closest to the query.

[1] `https://github.com/matsui528/nanopq`

[2] `https://github.com/wanji/sh`

[3] `https://github.com/facebookresearch/faiss`

3.1.1 Lack of Data Sensitivity

Importantly, one must remark that LSH does not exploit the distribution of the feature vectors. Recall that LSH methods hash the points of a dataset by projecting them onto hyper-planes that are drawn from random variables independent of the distribution of the data. Figure 1 illustrates how different the mapping of our data among those bins will be when the distribution of the data is ignored. Note that for a uniform distribution, the data is in expectation evenly spread out across the bins, leading to fast retrieval but a potentially higher error rate. For the Gaussian distribution, very few bins share the entirety of the data points among them, leading to high search times, but also a lower error rate. Our upcoming results will confirm this intuition.

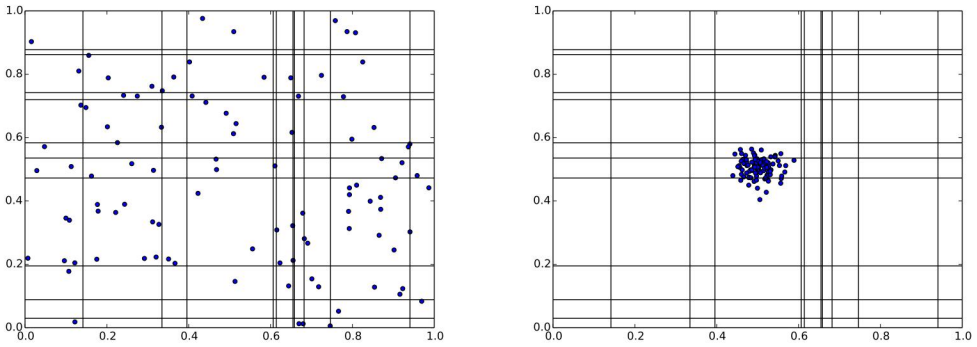


Figure 1: Illustrations of the hashing process of 2-dimensional datapoints that are drawn from a uniform distribution (left) and a normal distribution (right). Notice that for the latter, (a) most bins are empty, which is a waste of memory space and (b) a lot of points end up in the same bin as the query, which drastically penalizes the search time.

3.2 Spectral Hashing [WTF09]

Noticing that locality sensitive hashing methods such as [DIIM04] require an overwhelming number of bits to perform semantic hashing, that is, to efficiently code a large dataset, this work introduces the novel algorithm "spectral hashing" that computes the code of the points of a dataset in a data dependent way. For our purposes, it is particularly relevant that this method takes into account the distribution of the data.

The authors first tackle the challenge of formalizing the requirements for a good code when performing semantic hashing for a given dataset. The corresponding optimization problem:

$$\begin{aligned} \min_{\{y_i\}_{i=1}^n} \quad & \sum_{i,j} W_{ij} \|y_i - y_j\|^2 \\ \text{s.t.} \quad & y_i \in \{-1, 1\}^k \\ & \sum_i y_i = 1 \\ & \frac{1}{n} \sum_i y_i y_i^T = I \end{aligned} \tag{1}$$

where $\{y_i\}_{i=1}^n$ are the codes of an n points dataset of affinity matrix W_{ij}, is shown to be equivalent to an NP hard problem (balanced graph partitioning), relaxing the problem enables one to compute an approximation of the optimal code for a given dataset. Interestingly, the solutions of the relaxed problem are the k eigenvectors of $\text{diag}(W\mathbf{1}) - W$ with minimal eigenvalue.

With this in view, one can now efficiently compute a code only for the points already in the dataset (by computing these eigenvectors). Assuming an underlying distribution for the datapoints, the authors show that the code of the query points can be then computed using the k eigenfunctions of the weighted Laplacian operator. This allows one to design the spectral hashing algorithm:

1. Find the principal components of the data using PCA.
2. Compute the k smallest single-dimension analytical eigenfunctions of Lp using a rectangular approximation along every PCA direction.
3. Threshold the analytical eigenfunctions at zero, to obtain binary codes.

3.3 Product Quantization With Nearest Neighbor Search [JDS11]

This paper by Jegou et al. tries to overcome the issue of scarce memory while still yielding good search results. It is based on the quantization method, prominently put forward in [GN06]. Crucially, as with Spectral Hashing, Product Quantization attempts to take into account the distribution of the data to generate an efficient method of querying. In particular, in the training process it performs a k-means clustering step on the training data and generates hash functions based on these clusters—in contrast to LSH which simply generates hash functions randomly.

Quantization works as follows. Given a D-dimensional vector $x \in \mathbb{R}^D$, a quantizer is defined as mapping $q : \mathbb{R}^D \to \mathcal{C} = \{c_i : i \in \mathcal{I}\} \subset \mathbb{R}^D$, where $\mathcal{I} = \{1, \ldots, k\}$. So the quantizer q maps to a finite set of points $c_i \in \mathbb{R}^D$ called centroids. The set of all centroids $\mathcal{C}$ is called a codebook. Note that we can also define the set of all points mapped onto a given centroid c_i. This is the Voroni cell $\mathcal{V}_i := \{x \in \mathbb{R}^D : q(x) = c_i\}$. However, classic algorithms utilizing quantization for nearest-neighbor search, e.g. Lloyd's algorithm [Llo82], demand an infeasible amount of centroids for obtaining adequate results and are therefore not practical for large datasets. This paper circumvents the problem by using several low-dimensional quantizer on subvectors of the input vector $x \in \mathbb{R}^D$. First, x is divided into $m \in \mathbb{N}$ subvectors u_i of size $D^* \in \mathbb{N}$, where $mD^* = D$. Subsequently, a codebook C_i and an associated quantizer q_i are generated for each subvector u_i (in practice this is done through k-means clustering of the training data):

$$\underbrace{x_1, \ldots, x_{D^*}}_{u_1(x)}, \ldots, \underbrace{x_{(m-1)D^*+1}, \ldots, x_D}_{u_m(x)} \to q_1(u_1(x)), \ldots, q_m(u_m(x))$$

Assuming that each codebook has the same amount of k^* centroids, we only need to save mk^* centroids to encode the codebook $\mathcal{C} = \mathcal{C}_1 \times \ldots \times \mathcal{C}_m$. Having this quantization, they propose to calculate distances as follows. For two database vectors x and y, we can approximate their distance by the distance of the centroids of their subvectors:

$$d(x, y) \approx d((q(x), q(y)) = \left(\sum_j d(q_j(u_j(x)), q_j(u_j(y)))^2 \right)^{\frac{1}{2}}$$

Note that the distances between the centroids can be saved beforehand in a lookup-table. For a new query point x, we approximate the distance to a database point y as:

$$d(x, y) \approx d(x, q(y)) = \left(\sum_j d((u_j(x)), q_j(u_j(y)))^2 \right)^{\frac{1}{2}} \tag{2}$$

Since this approach still requires an exhaustive search over all y in the database, the authors introduced the combination of a coarse quantizer $q_c(x)$, giving a rough estimation of location, with a finer product quantizer on the residual $r(x) = x - q_c(x)$. In other words, a vector x is then approximated by $q_c(x) + q_p(r(x))$. The workflow to obtain nearest neighbors then is as follows:

1. Indexing: First, quantize x to $q_c(x)$, compute the residual $r(x)$ and quantize $r(x)$ using product quantization. Then, add an entry to the Voroni cell pertaining to $q_c(x)$, V_{c_i} consisting of a vector identifier and $q_p(r(x))$.
2. Searching: For a query y, quantize y using q_c. Compute the distances to all points in V_{c_i} using Equation 2. Return K approximate nearest neighbors based on these estimated distances.

4 Results on Synthetic Data

We conjecture that LSH is independent of the underlying distribution of the dataset, whereas PQ and SH are sensitive to the underlying distribution. In this section, we will use two main approaches to validate this claim. We first precise how we conducted our experiments.

4.1 Dataset Description

We generate two different general distribution types on which we benchmark the methods we investigate. The first dataset is drawn from a Gaussian distribution with mean zero, the second from a uniform distribution. To make both comparable, we set the variance of the uniform distribution to be the same as the variance Gaussian distribution. That is, we let the uniform distribution denoted here P have boundaries $[-\sqrt{3}\sigma, \sqrt{3}\sigma]$, such that $\text{Var}(P) = \frac{1}{12}(\sqrt{3}\sigma + \sqrt{3}\sigma)^2 = \sigma^2$, where σ is the standard deviation of the Gaussian distribution. Throughout the upcoming case studies, we fix the dimension of each sample to be $d = 64$ and use recall@5. In each experiment, the parameters that we do not specify are chosen to be the same across the three methods (LSH, PQ, SH) and the two distribution types. To reduce the stochasticity of our experiments, we repeat each of them ten times by changing the seed when generating the datasets and averaging the time and performance for each method.

4.2 Runtime vs. Recall for LSH

Vanilla LSH as presented in class and [DIIM04] has two hyperparameters that aid in eliminating false positives (AND functions) and false negatives (OR functions). When having only a limited amount of memory, one has to trade off between the two, visualized in Figure 2. Since we will not be able to compare across the methods for each possible combination of AND and OR functions as well as the fact that Figure 2 suggests a linear trade-off between runtime and recall, we fix the number of functions in a way that allows LSH to maximize the recall from here on. We will find that even under this concession, LSH has difficulties dealing with different distributions.

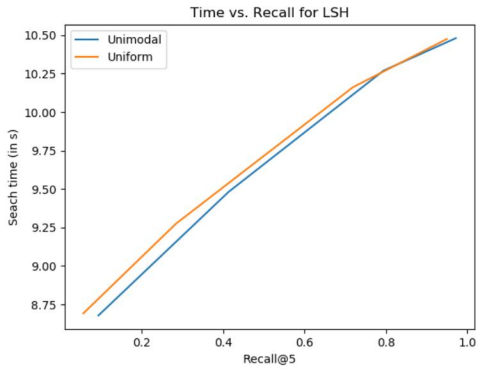
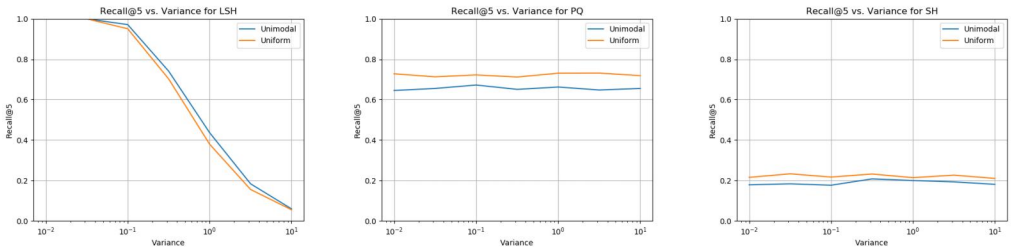


Figure 2: Time vs Recall for LSH.

4.3 Comparison Under Memory Constraint

Here, we fix the memory used by all three methods to be 64 bits and the number of samples to be n=1000. As can be seen in Figure 7a, an increase in the variance of our data leads to significantly worse results for LSH. At the same time, both PQ and SH are very consistent across different variances. These results are in support of our hypothesis that LSH is less robust to differences in the distribution. Interestingly, although SH performs significantly worse than PQ, it still outperforms LSH for variances greater than 2 for the limited amount of memory provided. In terms of time, the performance of LSH does not change much albeit the significantly worse results. PQ handles an increased variance very well, where the constant recall is accompanied by a relatively stable runtime. Spectral Hashing requires a longer time to maintain its results.

Next, we investigate the quality of the three methods under an increasing database size and fixed variance. For reasons of clarity, we show plots only for a variance of 0.1. Note that for LSH, we do not expect a large change in recall for an increase in the database size since the hyperplanes drawn are completely independent of the underlying distribution, so new datapoints do not affect where the old points were hashed to. For PQ on the other hand, more datapoints under fixed memory means that the clusters found in the k-means step will shift according to the data, potentially leading to a



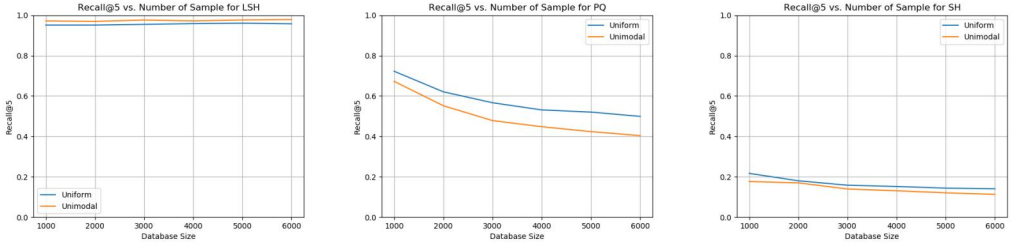
(a) Variance vs Recall LSH (b) Variance vs Recall PQ (c) Variance vs Recall SH

Figure 3: Recall as a function of variance

(a) Variance vs Time Increase Factor LSH (b) Variance vs Time Increase Factor PQ (c) Variance vs Time Increase Factor SH

Figure 4: Time Increase Factor as a function of variance

higher coarseness and thus lower recall. Similarly, we expect SH to perform worse since calculating the Principal Dimensions becomes more difficult with a higher number of samples. Figure 5 confirms this assumption. Interestingly however, LSH pays a lot for maintaining its accuracy: The runtime increases 23-fold from n=1000 to n=6000, whereas PQ and SH both only see an increase by 10-15.



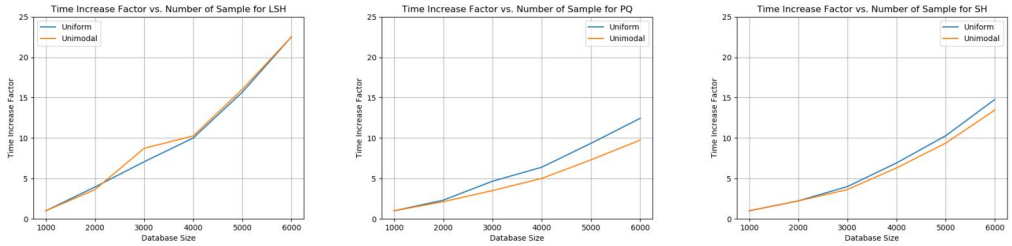
(a) Database size vs Recall LSH (b) Database size vs Recall PQ (c) Database size vs Recall SH

Figure 5: Recall as a function of the database size

So far, we have seen how an increasing variance strongly influences the result of LSH under a memory constraint, whereas the same was not observed with the other two methods. However, both uniform and gaussian distributions seem to produce almost the same result for LSH. The upcoming section will give more insight into why and when this fact holds.

4.4 Comparison Over Increasing Memory

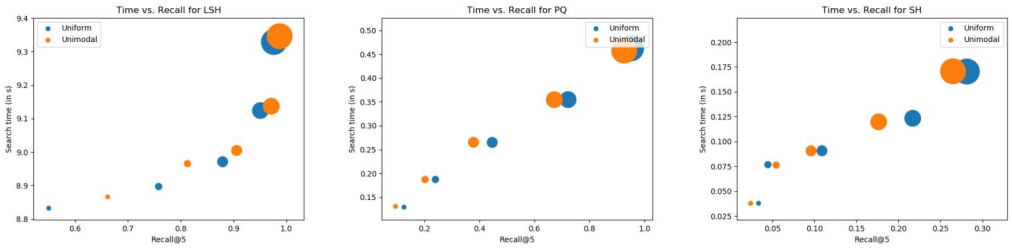
This section contains an analysis of how an increase in memory affects both runtime and recall. We use a memory range of $[8, 16, 32, 64, 128]$ bits, fix the variance to be 0.1 and $n = 1000$. Several insights can be taken from Figure 7a. First, for a lower memory, the recall highly depends on the distribution type, with a uniform distribution performing significantly worse than the uniform one. This effect vanishes with higher memory, which is in accordance with the results found in Section 4.3, where we fixed the memory to be 64 bits. The same effect does not occur with the other two



(a) Database size vs Time Increase Factor LSH (b) Database size vs Time Increase Factor PQ (c) Database size vs Time Increase Factor SH

Figure 6: Time increase factor as a function of the database size

ANN methods, providing further evidence towards the claim that SH and PQ are robust against the underlying distribution of the data, whereas LSH is not. Second, for increasing memory, recall and search time seem to increase in a linear relationship for SH and PQ, whereas LSH follows more of an exponential trend.



(a) Time vs Recall LSH (b) Time vs Recall PQ (c) Time vs Recall SH

Figure 7: Time as a function of recall, for a Variance of 0.1.

5 Results on Real Data

In addition to the results on synthetic data, we wished to test our hypothesis on real data as well. To this end, we investigated the performance of the various approximate nearest neighbor approaches on two real datasets—a portion of ImageNet and a portion of Wikipedia. We briefly outline below our methodology for each approach.

5.1 Dataset Descriptions

5.1.1 ImageNet

The entire ImageNet dataset checks in at 14 million images—much larger than what we needed to validate our hypothesis. To obtain a more manageable dataset size, we randomly selected 14,400 of ImageNet images to use as our training set and 1600 for our test set[4] in the uniform distribution case. By selecting them randomly, we conjecture that the distributions under which they are drawn are close to uniform distribution—an ideal case for LSH. Next, we selected two specific categories—Cars and Berries—and randomly chose 7,200 images from each category to use as our training set, and 800 from each to use as our test set. We hypothesize that by drawing the images from two specific classes, the data will look bimodal. Each image is represented by a vector of length 1000 corresponding to the SIFT features of that image.

[4]In practice, due to time constraints, for all experiments on real data we only used 50 test data points to compute recall on.

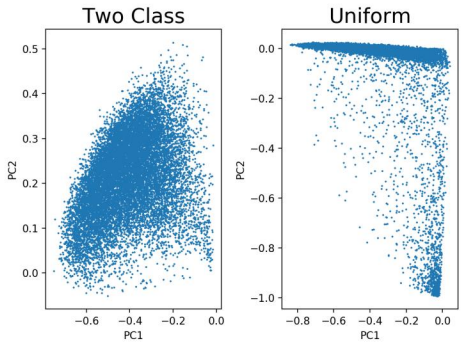


Figure 8: ImageNet data projected onto first two principal components

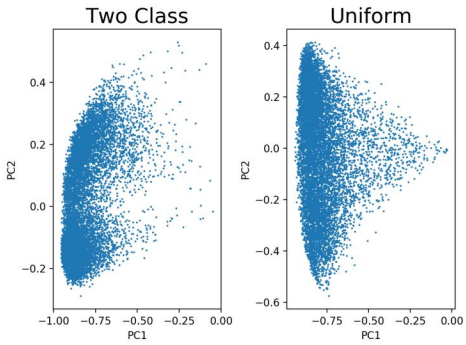


Figure 9: Wikipedia data projected onto first two principal components

5.1.2 Wikipedia

To test our approach in a different setting, we also ran each ANN approach on a selection of Wikipedia articles. Each article is encoded as a bag-of-words vector counting the frequency of specific words in the document. As with ImageNet, to test the performance of each approach on different distributions, we first create a set of 10,000 train and 1,000 test articles drawn uniformly at random from the set of all Wikipedia articles. We then simulate a bimodal case, where we draw 5,000 articles related to the topic of "Computers" and 5,000 related to the topic of "Football" as well as 1,000 texts from the same breakdown as test set. To generate the bag-of-words encoding, for each corpus, we iterate over all documents in the corpus, determine the 3,000 most frequently used words of length greater than 3, and encode each article as a vector based on the frequency with which these 3,000 words appear in the article.

For each test set, we obtain the true nearest neighbors for each query by running a linear search. To make the outcome of our ANN experiments more robust, we averaged the results over 20 trials for every experiment, each time using a different random seed when generating the hash functions.

5.2 Distribution of Real Data

While we hypothesize that the real data we test on has either a roughly uniform distribution or a roughly bimodal distribution—based on the process we used to select the data points—it is difficult to know whether this is actually the case. To get a better handle on how the data is truly distributed, we calculated the first two principal components for each dataset and projected the data onto the subspace spanned by them. The results for ImageNet are shown in Figure 8 and Wikipedia in Figure 9.

For ImageNet, the plots offer little insight into the true distribution of the data—the case we hypothesized to have a bimodal distribution seems to instead be roughly Gaussian and the case we hypothesized to be uniform seems to have a large cluster of outliers separated from the rest of the data. However, for the Wikipedia data, the two class case exhibits a clear bimodal distribution and the uniform case seems to be, at the very least, unimodal.

We note that simply observing the projections onto the first two principal components is not sufficient for robustly determining the true distribution of the data—they simply offer insight into the distributions. For a cleaner insight into how different distributions affect the methods examined, we refer the reader to the analysis of synthetic data in Section 4.

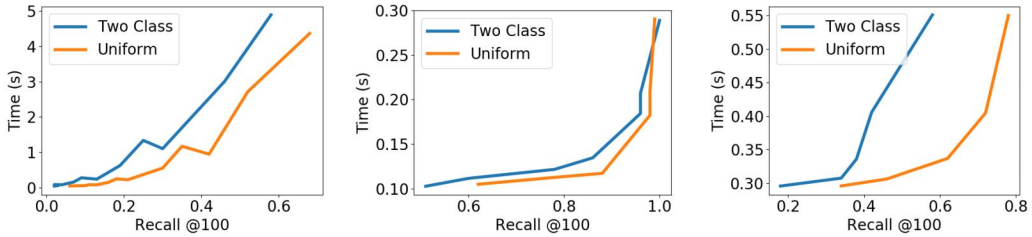
5.3 Comparison Metrics

To compare the performance of each method, we plot the recall against the runtime to find an approximate nearest neighbor. This explicitly excludes the training time. By varying the parameters (in particular, the memory used by each approach), each method can be tuned to yield nearly perfect recall at the expense of computation time and memory. Thus, the metric more important than pure recall is how the recall and runtime relate to each other. For each method, we plot two curves on

the same set of axes—one corresponding to the uniform task and one to the two class task. We are primarily interested in how the performance of each method varies with the distribution.

5.4 Performance of Approximate Nearest Neighbor Methods on ImageNet

The results running on ImageNet are given in figures 10a, 10b, and 10c. We observe that for both LSH and Spectral Hashing, there is a noticeable performance drop when running on the two class task versus the uniform task—for both methods, it takes a significantly longer runtime to achieve the same level of accuracy. For Product Quantization, uniform seems to do marginally better though the difference is much less significant than what we observe for the other methods.

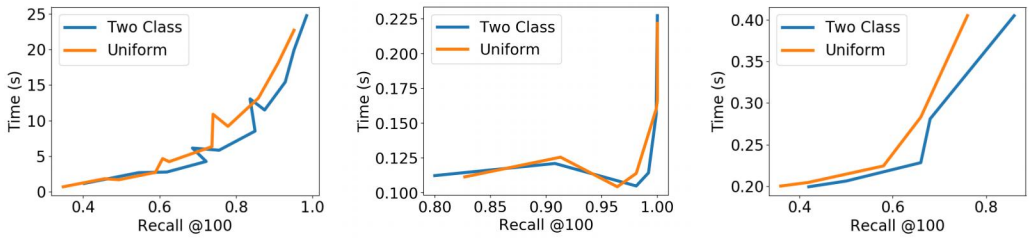


(a) Euclidean LSH (b) Product Quantization (c) Spectral Hashing

Figure 10: Performance of ANN Methods on ImageNet Data

5.5 Performance of Approximate Nearest Neighbor Methods on Wikipedia

The results running on Wikipeda are given in figures 10a, 10b, and 10c. We observe that the performance on the two class versus uniform data for Wikipedia is opposite to that of ImageNet—both Spectral Hashing and LSH now perform better on the two class case than on uniform. For Product Quantization, there is essentially no difference between performance on the uniform or two class case.



(a) Euclidean LSH (b) Product Quantization (c) Spectral Hashing

Figure 11: Performance of ANN Methods on Wikipedia Data

5.6 Discussion

We hypothesized that—due to the way the data was chosen—LSH would perform better on the uniform case than the two class case. However, while we did observe this to be the case for ImageNet, the opposite was true for Wikipedia. This motivates us to conclude that, for some real world tasks—such as attempting to distinguish between only two image classes—the performance of LSH does decrease significantly when compared against its performance on more uniform data. At the same time however, in other cases LSH seems to do better for a bimodal distribution than a uniform one. The variance in performance we observe can be seen as further validation for the hypothesis that the performance of LSH can depend significantly on how the data is distributed—it is certainly not agnostic to the underlying distribution and can perform significantly worse in the same modality if the data is distributed in a certain way.

Somewhat surprisingly, we observe that Spectral Hashing seems to also exhibit this property—its performance can vary significantly based on how the data is distributed. As a more sophisticated approach, we had hoped that SH would overcome some of the shortcomings of LSH but this does not seem to hold true in these cases.

Finally, we observe that PQ seems to perform in a way that is roughly agnostic to the distribution of the data. While small variations exist in the performance on each distribution, these are relatively minor compared to the variations of LSH and SH—the overall performance is nearly the same for PQ on the uniform and two class tasks. This is what we hoped to see. Due to the way PQ adaptively hashes the data based on its distribution, we would expect the performance to be roughly constant across distributions which is essentially what we observe.

It is worth noting again that, while we can estimate certain properties of the data distribution, it is difficult to determine how it is truly distributed. As such, these results should be interpreted less as definitive conclusions (as we obtained in Section 4) and more as practical observations related to the performance of each method on certain real-world tasks.

6 Discussion and Future Work

In this project, we verified that the performance of LSH is highly dependent on the distribution of the data. Our experiments illustrated that due to the inability of LSH to take into account the distribution of the data, it exhibits severe shortcomings that limit its performance. In contrast, other approaches such as PQ, which do take into account the data distribution, are very robust to different tasks and exhibit almost no change in performance when the distribution changes. We observe these effects both on synthetic data, as well as real-world problems, such as multi-class classification. In summary, this data verifies our hypothesis presented in the introduction.

7 Contributions

Emil Azadian: Experiments on synthetic data, writing poster and report.*

Romain Camilleri: Experiments on synthetic data, writing poster and report.*

Andrew Wagenmaker: Experiments on real data, writing poster and report.*

*All group members contributed equally.

References

[DDS+09] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[DIIM04] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM, 2004.

[GN06] R. M. Gray and D. L. Neuhoff. Quantization. *IEEE Trans. Inf. Theor.*, 44(6):2325–2383, September 2006.

[JDS11] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2011.

[Llo82] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982.

[WTF09] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *Advances in neural information processing systems*, pages 1753–1760, 2009.