**CS246: Mining Massive Data Sets** **Winter 2019**

# Problem Set 1

Please read the homework submission policies at http://cs246.stanford.edu.

## 1 Spark (25 pts)

Write a Spark program that implements a simple "People You Might Know" social network friendship recommendation algorithm. The key idea is that if two people have a lot of mutual friends, then the system should recommend that they connect with each other.

**Data:**

- Associated data file is soc-LiveJournal1Adj.txt in `q1/data`.
- The file contains the adjacency list and has multiple lines in the following format:

  <User><TAB><Friends>

  Here, <User> is a unique integer ID corresponding to a unique user and <Friends> is a comma separated list of unique IDs corresponding to the friends of the user with the unique ID <User>. Note that the friendships are mutual (i.e., edges are undirected): if $A$ is friend with $B$ then $B$ is also friend with $A$. The data provided is consistent with that rule as there is an explicit entry for each side of each edge.

**Algorithm:** Let us use a simple algorithm such that, for each user $U$, the algorithm recommends $N = 10$ users who are not already friends with $U$, but have the most number of mutual friends in common with $U$.

**Output:**

- The output should contain one line per user in the following format:

  <User><TAB><Recommendations>

  where <User> is a unique ID corresponding to a user and <Recommendations> is a comma separated list of unique IDs corresponding to the algorithm's recommendation of people that <User> might know, ordered in decreasing number of mutual friends.
- Even if a user has less than 10 second-degree friends, output all of them in decreasing order of the number of mutual friends. If a user has no friends, you can provide an empty list of recommendations. If there are recommended users with the same number of mutual friends, then output those user IDs in numerically ascending order.

**Pipeline sketch:** Please provide a description of how you used Spark to solve this problem. Don't write more than 3 to 4 sentences for this: we only want a very high-level description of your strategy to tackle this problem.

**Tips:**

- Before submitting a complete application to Spark, you may use the Shell to go line by line, checking the outputs of each step. Command `.take(X)` should be helpful, if you want to check the first `X` elements in the RDD.
- For sanity check, your top 10 recommendations for **user ID 11** should be: `27552,7785,27573,27574,27589,27590,27600,27617,27620,27667`.
- The default memory assigned to the Spark runtime may not be enough to process this data file, depending on how you write your algorithm. If your Spark job fails with a message starting as:

```
17/12/28 10:50:35 INFO DAGScheduler: Job 0 failed: sortByKey at FriendsRecomScala.scala:45, took 519.084974 s
Exception in thread "main" org.apache.spark.SparkException:
Job aborted due to stage failure: Task 0 in stage 2.0 failed 1 times, most recent failure:
Lost task 0.0 in stage 2.0 (TID 4, localhost, executor driver)
```

  then you'll very likely need to increase the memory assigned to the Spark runtime. If you are running in stand-alone mode (i.e. you did not setup a Spark cluster), use `--driver-memory 8G` to set the runtime memory to 8GB. If you are running on a Spark cluster, use `--executor-memory 8G` to set the memory to 8GB.

## What to submit

(1) Upload your code on snap website.

(2) Include in your writeup a short paragraph sketching your spark pipeline.

(3) Include in your writeup the recommendations for the users with following user IDs: 924, 8941, 8942, 9019, 9020, 9021, 9022, 9990, 9992, 9993.

★ **SOLUTION:** Recommendations for the 10 users:
924 439,2409,6995,11860,15416,43748,45881
8941 8943,8944,8940
8942 8939,8940,8943,8944
9019 9022,317,9023
9020 9021,9016,9017,9022,317,9023
9021 9020,9016,9017,9022,317,9023
9022 9019,9020,9021,317,9016,9017,9023
9990 13134,13478,13877,34299,34485,34642,37941
9992 9987,9989,35667,9991
9993 9991,13134,13478,13877,34299,34485,34642,37941

# 2 Association Rules (30 pts)

Association Rules are frequently used for Market Basket Analysis (MBA) by retailers to understand the purchase behavior of their customers. This information can be then used for many different purposes such as cross-selling and up-selling of products, sales promotions, loyalty programs, store design, discount plans and many others.

**Evaluation of item sets:** Once you have found the frequent itemsets of a dataset, you need to choose a subset of them as your recommendations. Commonly used metrics for measuring significance and interest for selecting rules for recommendations are:

1. **Confidence** (denoted as $\text{conf}(A \to B)$): *Confidence* is defined as the probability of occurrence of $B$ in the basket if the basket already contains $A$:

   $$\text{conf}(A \to B) = \Pr(B|A),$$

   where $\Pr(B|A)$ is the conditional probability of finding item set $B$ given that item set $A$ is present.

2. **Lift** (denoted as $\text{lift}(A \to B)$): *Lift* measures how much more "$A$ and $B$ occur together" than "what would be expected if $A$ and $B$ were statistically independent":

   $$\text{lift}(A \to B) = \frac{\text{conf}(A \to B)}{S(B)},$$

   where $S(B) = \frac{\text{Support}(B)}{N}$ and $N =$ total number of transactions (baskets).

3. **Conviction** (denoted as $\text{conv}(A \to B)$): *Conviction* compares the "probability that $A$ appears without $B$ if they were independent" with the "actual frequency of the appearance of $A$ without $B$":

   $$\text{conv}(A \to B) = \frac{1 - S(B)}{1 - \text{conf}(A \to B)}.$$

### (a) [3pts]

A drawback of using *confidence* is that it ignores $\Pr(B)$. Why is this a drawback? Explain why *lift* and *conviction* do not suffer from this drawback.

★ **SOLUTION:** It ignores $\Pr(B)$. In some cases, it may lead to incorrect rules. For e.g.: occurrence of $B$ may be unrelated to $A$ (e.g. $A$ and $B$ are independent, such that $\text{conf}(A \to B) = \Pr(B|A) = \Pr(B)$) but $B$ has high support, so that $A \to B$ is identified as a valid rule.

By observing the formulas, we can see that $\text{conv}$ and $\text{lift}$ take $\Pr(B)$ into account.

### (b) [3pts]

A measure is *symmetrical* if measure($A \to B$) = measure($B \to A$). Which of the measures presented here are symmetrical? For each measure, please provide either a proof that the measure is symmetrical, or a counterexample that shows the measure is not symmetrical.

★ **SOLUTION:** Lift is symmetrical. Confidence and conviction are not since confidence and conviction are directional but lift is not.

1. lift($A \to B$) = lift($B \to A$) = $\frac{\Pr(A,B)}{\Pr(A)\Pr(B)}$
2. conf($A \to B$) = $\Pr(B|A)$ and conf($B \to A$) = $\Pr(A|B)$. $\Pr(A|B)$ and $\Pr(B|A)$ might be different.
3. conv is based on conf and is directional.

**Example:**
If we have baskets AB, AC, AD, then S(A) = 3/3, S(B) = 1/3, and $\Pr(A, B) = 1/3$.
Then conf($A \to B$)=$\Pr(B|A) \neq \Pr(A|B)$ = conf($B \to A$) since: $\frac{1/3}{3/3} \neq \frac{1/3}{1/3}$.

Similarly, conv($A \to B$)=$\frac{1-S(B)}{1-conf(A \to B)} \neq \frac{1-S(A)}{1-conf(B \to A)}$= conv($B \to A$) since: $\frac{1-1/3}{1-1/2} = 4/3 \neq \frac{1-2/3}{1-1} =$ inf.

### (c) [4pts]

*Perfect implications* are rules that hold 100% of the time (or equivalently, the associated conditional probability is 1). A measure is *desirable* if it reaches its maximum achievable value for all perfect implications. This makes it easy to identify the best rules. Which of the above measures have this property? You may ignore 0/0 but not other infinity cases. Also you may find it easy to explain by an example.

★ **SOLUTION:** Conviction and confidence are desirable while lift is not. If $B$ occurs every time $A$ occurs then

1. conf($A \to B$) = 1
2. conv($A \to B$) → infinity
3. lift($A \to B$) depends on the value of $\Pr(B)$ and may differ as $B$ might occur in baskets which do not have $A$.

**Example:**
If we have baskets AB, AB, CD, EF, then $\Pr(B|A) = 1$, $S(B) = 1/2$, $\Pr(D|C) = 1$, and $S(D) = 1/4$.
Then lift($A \to B$) = $\frac{1}{1/2} = 2$ and lift($C \to D$) = $\frac{1}{1/4} = 4$. Although both rules are 100% rules, they have different lift scores.

**Application in product recommendations:** The action or practice of selling additional products or services to existing customers is called *cross-selling*. Giving product recommendation is one of the examples of cross-selling that are frequently used by online retailers. One simple method to give product recommendations is to recommend products that are frequently browsed together by the customers.

Suppose we want to recommend new products to the customer based on the products they have already browsed online. Write a program using the *A-priori* algorithm to find products which are frequently browsed together. Fix the support to $s$ =100 (*i.e.* product pairs need to occur together at least 100 times to be considered frequent) and find itemsets of size 2 and 3.

Use the online browsing behavior dataset from `browsing.txt` in `q2/data`. Each line represents a browsing session of a customer. On each line, each string of 8 characters represents the ID of an item browsed during that session. The items are separated by spaces.

Note: for parts (d) and (e), the writeup will require a specific rule ordering but the program need not sort the output. We are not giving partial credits to coding when results are wrong. However, two sanity checks are provided and they should be helpful when you progress: (1) there are 647 frequent items after 1st pass ($|L_1| = 647$), (2) the top 5 pairs you should produce in part (d) all have confidence scores greater than 0.985. See detailed instructions below.

### (d) [10pts]

Identify pairs of items $(X, Y)$ such that the support of $\{X, Y\}$ is at least 100. For all such pairs, compute the *confidence* scores of the corresponding association rules: $X \Rightarrow Y, Y \Rightarrow X$. Sort the rules in decreasing order of *confidence* scores and list the top 5 rules in the writeup. Break ties, if any, by lexicographically increasing order on the left hand side of the rule.

★ **SOLUTION:** Top 15 pairs by confidence (only need top 5):
DAI93865⇒FRO40251 1.0
GRO85051⇒FRO40251 0.999176276771
GRO38636⇒FRO40251 0.990654205607
ELE12951⇒FRO40251 0.990566037736
DAI88079⇒FRO40251 0.986725663717
FRO92469⇒FRO40251 0.983510011779
DAI43868⇒SNA82528 0.972972972973
DAI23334⇒DAI62779 0.954545454545
ELE92920⇒DAI62779 0.732664995823
DAI53152⇒FRO40251 0.717948717949
SNA18336⇒DAI62779 0.713681241185
ELE55848⇒GRO32086 0.709459459459
GRO89004⇒ELE25077 0.698051948052
GRO81647⇒GRO73461 0.677551020408

DAI37288⇒ELE32164 0.646408839779

**(e) [10pts]**

Identify item triples $(X, Y, Z)$ such that the support of $\{X, Y, Z\}$ is at least 100. For all such triples, compute the *confidence* scores of the corresponding association rules: $(X, Y) \Rightarrow Z$, $(X, Z) \Rightarrow Y$, $(Y, Z) \Rightarrow X$. Sort the rules in decreasing order of *confidence* scores and list the top 5 rules in the writeup. Order the left-hand-side pair lexicographically and break ties, if any, by lexicographical order of the first then the second item in the pair.

★ **SOLUTION:** Top 15 triples by confidence (only need top 5):
DAI23334,ELE92920⇒DAI62779 1.0
DAI31081,GRO85051⇒FRO40251 1.0
DAI55911,GRO85051⇒FRO40251 1.0
DAI62779,DAI88079⇒FRO40251 1.0
DAI75645,GRO85051⇒FRO40251 1.0
ELE17451,GRO85051⇒FRO40251 1.0
ELE20847,FRO92469⇒FRO40251 1.0
ELE20847,GRO85051⇒FRO40251 1.0
ELE26917,GRO85051⇒FRO40251 1.0
FRO53271,GRO85051⇒FRO40251 1.0
GRO21487,GRO85051⇒FRO40251 1.0
GRO38814,GRO85051⇒FRO40251 1.0
GRO73461,GRO85051⇒FRO40251 1.0
GRO85051,SNA45677⇒FRO40251 1.0
GRO85051,SNA80324⇒FRO40251 1.0

## What to submit

Upload all the code on snap and include the following in your writeup:

(i) Explanation for 2(a).

(ii) Proofs and/or counterexamples for 2(b).

(iii) Explanation for 2(c).

(iv) Top 5 rules with confidence scores [2(d)].

(v) Top 5 rules with confidence scores [2(e)].

# 3 Locality-Sensitive Hashing (15 pts)

When simulating a random permutation of rows, as described in **Sect. 3.3.5** of MMDS, we could save time if we restricted our attention to a randomly chosen $k$ of the $n$ rows, rather than hashing all $n$ row numbers. The downside of doing so is that, if none of the $k$ rows contains a 1 in a certain column, then the result of the minhashing is "don't know". In other words, we get no row number as the minhash value. It would be a mistake to assume that two columns that both minhash to "don't know" are likely to be similar. However, if the probability of getting "don't know" as a minhash value is small, we can tolerate the situation and simply ignore such minhash values when computing the fraction of minhashes in which two columns agree.

In part (a) we determine an upper bound on the probability of getting "don't know" as the minhash value when considering only a $k$-subset of the $n$ rows, and in part (b) we use this bound to determine an appropriate choice for $k$, given our tolerance for this probability.

### (a) [5pts]

Suppose a column has $m$ 1's and therefore $n - m$ 0's, and we randomly choose k rows to consider when computing the minhash. Prove that the probability of getting "don't know" as the minhash value for this column is at most $(\frac{n-k}{n})^m$.

★ **SOLUTION:** The number of columns with $m$ 1's out of $n$ is $\binom{n}{m}$. The number of these columns that have no 1 in one of the $k$ selected rows is $\binom{n-k}{m}$. The probability of no 1 in the chosen $k$ rows is therefore the latter divided by the former. If we expand the binomial coefficients in terms of factorials, we get $\frac{(n-k)!m!(n-m)!}{m!(n-k-m)!n!}$. The $m!$s cancel, and when we reorganize we can write this expression as $(\frac{n-k}{n})(\frac{n-k-1}{n-1})\cdots(\frac{n-k-m+1}{n-m+1})$. Each of the $m$ factors is at most $(\frac{n-k}{n})$. Thus, their product is at most $(\frac{n-k}{n})^m$.

### (b) [5pts]

Suppose we want the probability of "don't know" to be at most $e^{-10}$. Assuming $n$ and $m$ are both very large (but $n$ is much larger than $m$ or $k$), give a simple approximation to the smallest value of $k$ that will ensure this probability is at most $e^{-10}$. Your expression should be a function of $n$ and $m$. Hints: (1) You can use $(\frac{n-k}{n})^m$ as the exact value of the probability of "don't know." (2) Remember that for large $x$, $(1 - \frac{1}{x})^x \approx 1/e$.

★ **SOLUTION:** We want $(\frac{n-k}{n})^m \leq e^{-10}$. Equivalently, $(1 - \frac{k}{n})^m < e^{-10}$. If we multiply and divide the exponent by $n/k$, we see this condition is equivalent to $\left((1 - \frac{k}{n})^{n/k}\right)^{mk/n} \leq e^{-10}$. Since we assume $k << n$, we can approximate $(1 - \frac{k}{n})^{n/k}$ by $1/e$. That makes the desired

condition $e^{-mk/n} \leq e^{-10}$. Thus, the first exponent must be less than or equal to the second; i.e., $-mk/n \leq -10$, or $mk/n \geq 10$. That, in turn means $k \geq 10n/m$, so the correct answer, the lower bound on $k$, is $10n/m$.

### (c) [5pts]

Note: Part (c) should be considered separate from the previous two parts, in that we are no longer restricting our attention to a randomly chosen subset of the rows.

When minhashing, one might expect that we could estimate the Jaccard similarity without using all possible permutations of rows. For example, we could only allow cyclic permutations, i.e. start at a randomly chosen row $r$, which becomes the first in the order, followed by rows $r+1$, $r+2$, and so on, down to the last row, and then continuing with the first row, second row, and so on, down to row $r-1$. There are only $n$ such permutations if there are $n$ rows. However, these permutations are not sufficient to estimate the Jaccard similarity correctly.

Give an example of two columns such that the probability (over cyclic permutations only) that their minhash values agree is not the same as their Jaccard similarity. In your answer, please provide (a) an example of a matrix with two columns (let the two columns correspond to sets denoted by $S1$ and $S2$), (b) the Jaccard similarity of $S1$ and $S2$, and (c) the probability that a random cyclic permutation yields the same minhash value for both $S1$ and $S2$.

★ **SOLUTION:** The two columns (sets) are $[0,1,0]^T$ and $[0,1,1]^T$. Jaccard similarity = 0.5. But if the cycle starts at either of the first two rows, the minhash values are the same, while if the cycle starts at the last row, then the minhash values differ. Thus, the probability of the minhash values agreeing is 2/3, when only cyclic permutations are allowed.

## What to submit

Include the following in your writeup:

(i) Proof for 3(a)

(ii) Derivation and final answer for 3(b)

(iii) Example for 3(c) including the three requested items

# 4 LSH for Approximate Near Neighbor Search (30 pts)

In this problem, we study the application of LSH to the problem of finding approximate near neighbors.

Assume we have a dataset $\mathcal{A}$ of $n$ points in a metric space with distance metric $d(\cdot, \cdot)$. Let $c$ be a constant greater than 1. Then, the $(c, \lambda)$-Approximate Near Neighbor (ANN) problem is defined as follows: Given a query point $z$, assuming that there is a point $x$ in the dataset with $d(x, z) \leq \lambda$, return a point $x'$ from the dataset with $d(x', z) \leq c\lambda$ (this point is called a $(c, \lambda)$-ANN). The parameter $c$ therefore represents the maximum approximation factor allowed in the problem.

Let us consider a LSH family $\mathcal{H}$ of hash functions that is $(\lambda, c\lambda, p_1, p_2)$-sensitive for the distance measure $d(\cdot, \cdot)$. Let[1] $\mathcal{G} = \mathcal{H}^k = \{g = (h_1, \ldots, h_k) | h_i \in \mathcal{H}, \ \forall \ 1 \leq i \leq k\}$, where $k = \log_{1/p_2}(n)$.

Let us consider the following procedure:

1. Select $L = n^\rho$ random members $g_1, \ldots, g_L$ of $\mathcal{G}$, where $\rho = \frac{\log(1/p_1)}{\log(1/p_2)}$.
2. Hash all the data points as well as the query point using all $g_i$ $(1 \leq i \leq L)$.
3. Retrieve at most[2] $3L$ data points (chosen uniformly at random) from the set of $L$ buckets to which the query point hashes.
4. Among the points selected in phase 3, report the one that is the closest to the query point as a $(c, \lambda)$-ANN.

The goal of the first part of this problem is to show that this procedure leads to a correct answer with constant probability.

### (a) [5 pts]

Let $W_j = \{x \in \mathcal{A} | g_j(x) = g_j(z)\}$ $(1 \leq j \leq L)$ be the set of data points $x$ mapping to the same value as the query point $z$ by the hash function $g_j$. Define $T = \{x \in \mathcal{A} | d(x, z) > c\lambda\}$. Prove:

$$\Pr\left[\sum_{j=1}^{L} |T \cap W_j| \geqslant 3L\right] \leqslant \frac{1}{3}.$$

*(Hint: Markov's Inequality.)*

★ **SOLUTION:** For each $1 \leq j \leq L$, and each data point $x \in T$, $\Pr[x \in T \bigcap W_j] \leq p_2^k = 1/n$, and hence, $\mathrm{E}[|T \bigcap W_j|] \leq 1$. Therefore, by linearity of expectation, $\mathrm{E}[\sum_{j=1}^{L} |T \bigcap W_j|] \leq L$. Then, an application of Markov's inequality gives the desired probability bound.

[1] The equality $\mathcal{G} = \mathcal{H}^k$ is saying that every function of $\mathcal{G}$ is an AND-construction of $k$ functions of $\mathcal{H}$, so $g(x) = g(y)$ only if $h_i(x) = h_i(y)$ for every $h_i$ underlying $g$.

[2] If there are fewer than $3L$ data points hashing to the same buckets as the query point, just take all of them.

**(b) [5 pts]**

Let $x^* \in \mathcal{A}$ be a point such that $d(x^*, z) \leq \lambda$. Prove:

$$\Pr\left[\forall\ 1 \leq j \leq L,\ g_j(x^*) \neq g_j(z)\right] < \frac{1}{\mathrm{e}}.$$

★ **SOLUTION:** Since $d(x^*, z) \leq \lambda$, for any $1 \leq j \leq L$, we have $\Pr[g_j(x^*) = g_j(z)] \geq p_1^k$, and hence $\Pr[g_j(x^*) \neq g_j(z)] \leq 1 - p_1^k = 1 - 1/L$, where the last equality is by $p_1^k = p_1^{\log_{1/p_2}(n)} = n^{-\frac{\log(1/p_1)}{\log(1/p_2)}} = n^{-\rho} = L^{-1}$. Then, by independence of $g_j$'s, we have: $\Pr\left[\forall\ 1 \leq j \leq L,\ g_j(x^*) \neq g_j(z)\right] \leq (1 - 1/L)^L \leq 1/\mathrm{e}$.

**(c) [5 pts]**

Conclude that with probability greater than some fixed constant the reported point is an actual $(c, \lambda)$-ANN.

★ **SOLUTION:** Let's denote by $U$ the set of ANN points, *i.e.* $U = \{x \in \mathcal{A};\ d(x, z) \leqslant c\lambda\}$. Note that $x^* \in U$. There are two ways a reported point is not a $(c, \lambda)$-ANN:

- None of the ANN points are hashed to the same buckets as $z$, *i.e.* $\forall\ 1 \leq j \leq L,\ W_j \cap U = \emptyset$. Let's denote by $E$ the event "none of the ANN points are hashed to the same buckets as $z$". Since $x^* \in U$, we have, using question (b):

$$\Pr[E] \leqslant \Pr\left[x^* \notin \bigcup_{j=1}^{L} W_j\right] = \Pr\left[\forall 1 \leq j \leq L,\ g_j(x^*) \neq g_j(z)\right] < \frac{1}{\mathrm{e}}.$$

- Or there is at least one $(c, \lambda)$-ANN point that is hashed to one of the buckets where $z$ is hashed, but there are more than $3L$ points at distance greater than $c\lambda$ in the union of those buckets. (If there are less than $3L$ points at distance greater than $c\lambda$ in $\bigcup_{j=1}^{L} W_j$, the algorithm will necessarily return a $(c, \lambda)$-ANN.) Let's denote by $F$ the event "there are more than $3L$ points at distance greater than $c\lambda$ from $z$ in the union of the buckets where $z$ is hashed". In that case, we can apply question (a)[3] and we know that the event $F$ appears with a probability lesser than $\frac{1}{3}$.

[3]Note that the contrapositive of the event "$\sum_{j=1}^{L} |T \cap W_j| \geqslant 3L$" just says that there are less than 3L non-ANN points in $\bigcup_{j=1}^{L} W_j$ (let's denote by $m$ this number of non-ANN points in this union). It doesn't say that there exists a $m + 1^{\text{th}}$ point in this union (note that its existence would ensure that we can find a $(c, \lambda)$-ANN). This is why question (a) is not enough to answer question (c) in the case where there are less than $3L$ points in $\bigcup_{j=1}^{L} W_j$, and that we also need question (b).

So, if we name $\bar{p}$ the probability that the point returned by the algorithm is not a $(c, \lambda)$-ANN, we have $\bar{p} = \Pr[E \cup F] \leq \Pr[E] + \Pr[F] < \frac{1}{3} + \frac{1}{\mathrm{e}}$ (by the union bound), and so the algorithm always reports an actual $(c, \lambda)$-ANN with a probability greater than $1 - \frac{1}{3} - \frac{1}{\mathrm{e}}$.

Note: We haven't shown that events $E$ and $F$ are independent, so it would be incorrect to say $\Pr[E \cap F] = \Pr[E] \cdot \Pr[F]$. It is possible to define events similar to $E$ and $F$ and prove that they are independent, and doing so would give a probability of correctness which is greater than the one we got with a union bound.

### (d) [15 pts]

A dataset of images,[4] `patches.csv`, is provided in `q4/data`.

Each row in this dataset is a $20 \times 20$ image patch represented as a 400-dimensional vector. We will use the $L_1$ distance metric on $\mathbb{R}^{400}$ to define similarity of images. We would like to compare the performance of LSH-based approximate near neighbor search with that of linear search.[5] You should use the code provided with the dataset for this task.

The included starter code in `lsh.py` marks all locations where you need to contribute code with `TODO`s. In particular, you will need to use the functions `lsh_setup` and `lsh_search` and implement your own linear search. The default parameters $L = 10, k = 24$ to `lsh_setup` work for this exercise, but feel free to use other parameter values as long as you explain the reason behind your parameter choice.

- For each of the image patches in columns $100, 200, 300, \ldots, 1000$, find the top 3 near neighbors[6] (excluding the original patch itself) using both LSH and linear search. What is the average search time for LSH? What about for linear search?
- Assuming $\{z_j | \ 1 \leq j \leq 10\}$ to be the set of image patches considered (*i.e.*, $z_j$ is the image patch in column $100j$), $\{x_{ij}\}_{i=1}^{3}$ to be the approximate near neighbors of $z_j$ found using LSH, and $\{x_{ij}^*\}_{i=1}^{3}$ to be the (true) top 3 near neighbors of $z_j$ found using linear search, compute the following error measure:

$$error = \frac{1}{10} \sum_{j=1}^{10} \frac{\sum_{i=1}^{3} d(x_{ij}, z_j)}{\sum_{i=1}^{3} d(x_{ij}^*, z_j)}$$

  Plot the error value as a function of $L$ (for $L = 10, 12, 14, \ldots, 20$, with $k = 24$). Similarly, plot the error value as a function of $k$ (for $k = 16, 18, 20, 22, 24$ with $L = 10$). Briefly comment on the two plots (one sentence per plot would be sufficient).

[4] Dataset and code adopted from Brown University's Greg Shakhnarovich

[5] By linear search we mean comparing the query point $z$ directly with every database point $x$.

[6] Sometimes, the function `lsh_search` may return less than 3 nearest neighbors. You can use a `while` loop to check that `lsh_search` returns enough results, or you can manually run the program multiple times until it returns the correct number of neighbors.

- Finally, plot the top 10 near neighbors found[7] using the two methods (using the default $L = 10, k = 24$ or your alternative choice of parameter values for LSH) for the image patch in column 100, together with the image patch itself. You may find the function `plot` useful.
  How do they compare visually?

## What to submit

(i) Include the proof for 4(a) in your writeup.

(ii) Include the proof for 4(b) in your writeup.

(iii) Include the reasoning for why the reported point is an actual $(c, \lambda)$-ANN in your writeup [4(c)].

(iv) Include the following in your writeup for 4(d):

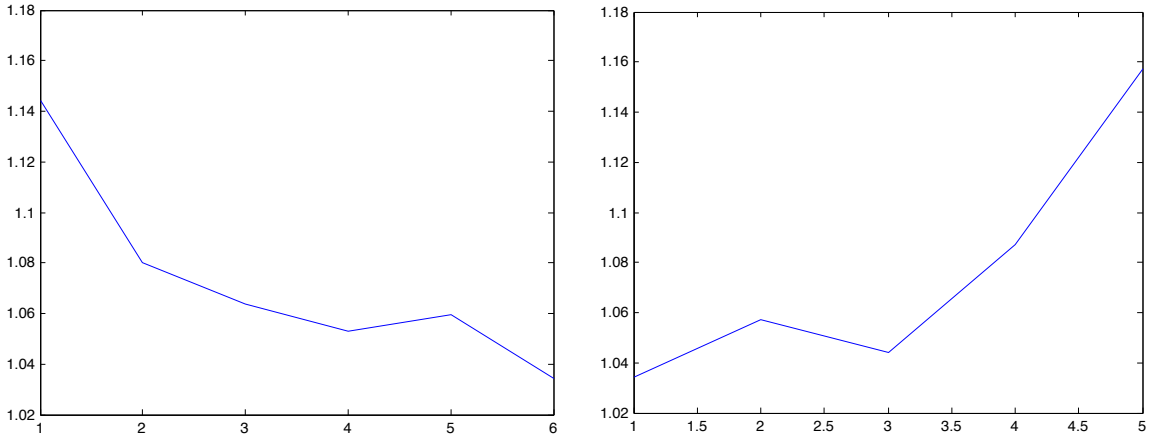
- Average search time for LSH and linear search.
- Plots for error value vs. $L$ and error value vs. $K$, and brief comments for each plot
- Plot of 10 nearest neighbors found by the two methods (also include the original image) and brief visual comparison

(v) Upload the code for 4(d) on snap.

★ **SOLUTION:** The exact running times, error values, and retrieved near neighbors may vary based on the exact implementation details and also due to the randomness of the LSH data structure. But, the basic points are that LSH is significantly faster than linear search (up $10\times$ or even higher speedup can be very well expected in this problem), provides comparable error (*i.e.* the computed error value is of course bigger than 1, but still quite close to 1), and provides comparable visual quality as well. Note: In the case that you find LSH not that much faster than linear search, there may be implementation-specific quirks. It may still be faster if you amortize the time it takes to hash the data.

The following figures give an idea of what you could get. Notice how the error growing with $k$ translates to the results: the images in Figure 4 are much closer to the linear search results and to the original image than in Figure 3.

[7] Same remark, you may sometimes have less that 10 nearest neighbors in your results; you can use the same hacks to bypass this problem.




(a) The error functions of $L$.

(b) The error functions of $k$.

Figure 1: Errors functions of $k$ and $l$.

Figure 2: Original picture.

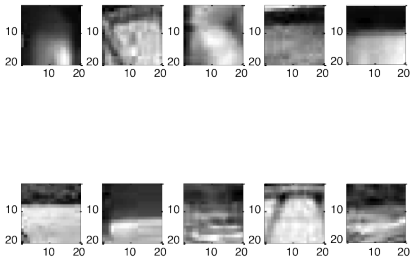


Figure 3: The 10 nearest neighbors with the LSH method. $L = 10$, $k = 24$.

| | |
|---|---|
| Average linear search time | 4.8706634 |
| Average LSH search time | 2.1910855 |

Table 1: Part 1. Note that your search times may have varied depending on things like whether you included the LSH setup time as part of the runtime; in our solution code we only hashed the data once, and divided that preprocessing time by 10, and added it to the average query time.

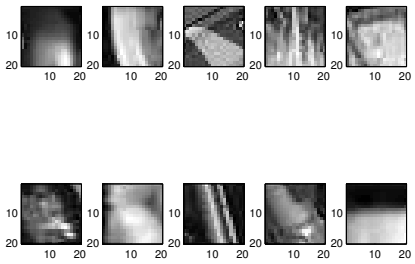


Figure 4: The 10 nearest neighbors with the LSH method. $L = 10$, $k = 10$.

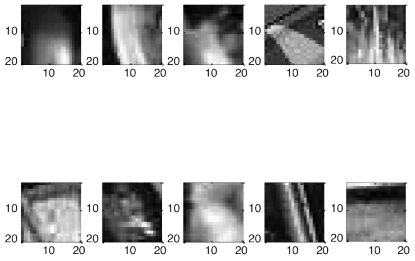


Figure 5: The 10 nearest neighbors with the LS method.

| k | L | error |
|---|---|---|
| 24 | 10 | 1.0207009015989 |
| 24 | 12 | 1.0568577195142024 |
| 24 | 14 | 0.979260973530206 |
| 24 | 16 | 1.061661200657818 |
| 24 | 18 | 1.0517631261733165 |
| 24 | 20 | 1.0442864335995619 |
| 16 | 10 | 1.0255919223388357 |
| 18 | 10 | 1.0491323879236516 |
| 20 | 10 | 1.1015141271997577 |
| 22 | 10 | 1.0797637531960969 |
| 24 | 10 | 1.084848827950971 |

Table 2: Part 2. Your graphs may have varied depending on randomness. Note that it is possible for the ratio to be less than 1, if the number of candidates returned by LSH search is less than 3.

Figure 6: The original image

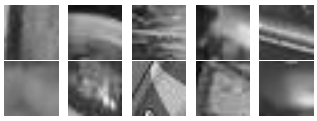


Figure 7: The 10 nearest neighbors from linear search

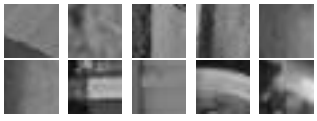


Figure 8: The 10 nearest neighbors from LSH search