Problem 1 (Let a Hundred Flowers Bloom). Consider the following data stream consisting of nonnegative data

\[ X = (1, 5, 9, 7, 12, 5, 6, 14, 13, 8). \]

We want to use a Bloom filter \( B \) with \( k = 7 \) bits and 2 hash functions

\[
\begin{align*}
    h_1(x) &= x \mod 7, \\
    h_2(x) &= x^2 \mod 7.
\end{align*}
\]

(a) Compute \( B \) step by step.

(b) According to \( B \), is the element 2 included in our stream? How about 10?

(c) Find a nonnegative number which is not included in the stream but which would result in a false positive. Give a reason why this number works.

(d) Based on the properties that the hash functions used should have would you say that the choice of \( h_1 \) and \( h_2 \) is a good one? If so, why? If not, why not?

Solution.

(a) The bloom filter after each step will become as follows:

<table>
<thead>
<tr>
<th>Step</th>
<th>( x )</th>
<th>( h_1(x) )</th>
<th>( h_2(x) )</th>
<th>Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0 1 0 0 0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>0 1 0 0 1 1 0</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>2</td>
<td>4</td>
<td>0 1 1 0 1 1 0</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1 1 1 0 1 1 0</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>5</td>
<td>4</td>
<td>1 1 1 0 1 1 0</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>1 1 1 0 1 1 0</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td>1 1 1 0 1 1 1</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>1 1 1 0 1 1 1</td>
</tr>
<tr>
<td>8</td>
<td>13</td>
<td>6</td>
<td>1</td>
<td>1 1 1 0 1 1 1</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1 1 1 0 1 1 1</td>
</tr>
</tbody>
</table>

(b) In this case for \( x = 2 \) we get \( h_1(x) = 2 \) and \( h_2(x) = 4 \), and since those bits are both set in the bloomfilter (i.e., \( B[2] = B[4] = 1 \)) it seems that 2 is included in the filter, which makes it a false positive.

For \( x = 10 \) we get \( h_1(10) = 3 \) and \( h_2(10) = 2 \), but since \( B[3] = 0 \) we can safely say that 10 was never in the stream.

(c) Here we can make use of basic properties of modulo arithmetic. One particularly useful one is:

\[
(ab \mod \kappa) = ((a \mod \kappa)(b \mod \kappa) \mod \kappa),
\]
or, in our case, for a \( b = 8 \) we have

\[
(8a \mod 7) = ((a \mod 7) \cdot (8 \mod 7) \mod 7) = ((a \mod 7) \cdot 1 \mod 7) = (a \mod 7)
\]

Using the same identity, we see that multiplication with 8 is likewise not affecting the second hash; therefore, every number \( x \cdot 8^n \) for any positive \( n \) would result in a false positive. This is, of course, by no means an exhaustive list.

(d) Although in practice the hash functions are deterministic in nature (be it due to the use of pseudorandom generators or use of deterministic hashes), in the algorithm analysis it is useful to think of them as random variables of the input \( x \). From this perspective, we require that the hash functions are ideally:

(a) uniformly distributed within the space \( 0, \ldots, k - 1 \), where \( k = 7 \) is the number of bits.

Here \( h_1(x) \) is perfectly uniformly distributed, whereas \( h_2(x) \) is not, as it only produces outputs \( 0, 1, 2, 4 \) and has zero Probability of outputting \( 3, 5, 6 \).

(b) statistically independent, that is \( P(h_1(x), h_2(x)) = P(h_1(x)) P(h_2(x)) \).

Clearly, these functions are not independent, as we can compute \( h_2(x) = h_1(x)^2 \mod 7 \).

Thus the choice of \( h_1 \) and \( h_2 \) is not a good one.

**Problem 2** (Singular). Let the data

\[
X = \begin{pmatrix}
-10 & 1 \\
0 & -2 \\
10 & 1
\end{pmatrix}
\]

be given, assuming column vectors (i.e., rows and columns correspond to instances and their attributes, respectively).

(a) Compute the mean \( \mu \) and covariance matrix \( \Sigma \).

(b) Compute the SVD of \( X \).

(c) Perform PCA on \( X \).

(d) What is the “intrinsic” dimensionality of this dataset?

**Solution.**

(a) The mean vector is \( \mu = [0, 0] \), where \( m_j = \sum_{i=1}^{3} X_{ij} \). The covariance matrix is

\[
C^* = E[(X - \mu)^T(X - \mu)] = E[X^TX] - \mu^T \mu;
\]

this can be estimated by the empirical covariance matrix \( C = \frac{1}{n-1}X^TX \), which in turn gives

\[
C = \frac{1}{2} \begin{pmatrix}
-10 & 0 & 10 \\
1 & -2 & 1 \\
0 & 10 & 1
\end{pmatrix} \begin{pmatrix}
-10 & 1 \\
0 & -2 \\
10 & 1
\end{pmatrix} - \begin{pmatrix}
0 \\
0 \\
0
\end{pmatrix} \begin{pmatrix}
0 \\
0 \\
3
\end{pmatrix}.
\]

(b) Here we must find \( U \Sigma V^T = X \in \mathbb{R}^{3 \times 2} \) with orthonormal \( U, V \), i.e., \( U^TU = I_3 \) and \( V^TV = I_2 \), where \( I_n \) is the identity matrix with \( n \times n \) elements.
We first notice that \(X^TX = U\Sigma\Sigma^TU^T = U\Sigma^T\Sigma U = C\), so that we can compute \(U, \Sigma^T\Sigma\) from the eigenvalue decomposition of \(C\). Since \(C\) is diagonal, its eigenvalue decomposition is trivial with \(C\) and \(I_2\) coinciding with the eigenvalue and eigenvector matrices, respectively. From this we find

\[ V = I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \Sigma = \sqrt{(n-1)C} = \begin{pmatrix} 10\sqrt{2} & 0 \\ 0 & \sqrt{6} \end{pmatrix}. \]

Letting \(\Sigma'\) be the first two rows of \(\Sigma\), we can now compute \(U\) as

\[ U = (\Sigma'^{-1}UX)^T = \left(\frac{\sqrt{2}}{\sqrt{6}} \begin{pmatrix} 0 \\ \frac{\sqrt{6}}{\sqrt{2}} \end{pmatrix} \right)^T \left(\begin{pmatrix} \frac{\sqrt{2}}{\sqrt{6}} \\ \frac{\sqrt{6}}{\sqrt{2}} \end{pmatrix} \right)^T = \begin{pmatrix} -\frac{\sqrt{2}}{\sqrt{6}} \\ \frac{\sqrt{2}}{\sqrt{6}} \end{pmatrix} \]

Since \(U\) is rank deficient, to complete the orthonormal basis we need to compute a basis to the left null space of \(U\) (c.f., row echelon form). This is beyond the scope of this exercise. In any case, the third vector in this case is: \(v_3 = \frac{\sqrt{3}}{\sqrt{6}} (1 \quad 1 \quad 1)^T\), so the full \(U\) is

\[ \begin{pmatrix} 1 \quad 0 \quad \frac{1}{\sqrt{6}} \\ 0 \quad \frac{2\sqrt{6}}{\sqrt{2}} \quad 2\sqrt{3} \\ 3\sqrt{2} \quad \frac{\sqrt{6}}{2} \quad 2\sqrt{3} \end{pmatrix} \]

(c) The PCA decomposition is a linear transform that maps the original instances, each containing multiple entries of possibly correlated attributes, into a new space of attribute combinations such that a) these combinations are now un-correlated and b) ordered in decreasing variance. This space coincides with the eigenspace of the correlation matrix \(C\) of Eq. (2.1). However, since the data are centered (i.e., \(\mu = 0\)), this matrix coincides with \((n-1)C = X^TX = V\Sigma\Sigma^TU^T\Sigma^TU\Sigma^T\), and we can thus directly “read” the eigenvalue decomposition of the correlation matrix \(C\) from the SVD decomposition of \(X\), as

\[ SAS^T = C \Rightarrow \begin{cases} S = V \\ \Lambda = \frac{\Sigma^T \Sigma}{n-1} \end{cases}. \]

Since in the SVD decomposition, above, we arranged the vectors of \(V\) in decreasing order of variance (singular values), the first and second principal

- **directions** are \(v_1 = (1 \quad 0)^T\) and \(v_2 = (0 \quad 1)^T\),
- **loadings** are \(v_1\sqrt{\lambda_1} = (10\sqrt{2} \quad 0)^T\) and \(v_2\sqrt{\lambda_2} = (0 \quad \sqrt{6})^T\) and
- **components** are \(u_1 = 1/2 \begin{pmatrix} -\sqrt{2} \\ 0 \quad \sqrt{2} \end{pmatrix}^T\) and \(u_2 = 1/6 \begin{pmatrix} \sqrt{6} \quad -2\sqrt{6} \quad \sqrt{6} \end{pmatrix}^T\).

(d) Since the eigenvalues of \(C\) are 200 \(\gg\) 6 we can say that the intrinsic dimensionality is just 1.

**Problem 3 (Lossy).** Consider the following stream

\[ S = (1, 1, 2, 1, 3, 2, 1, 1, 2, 3, 3, 3) \]

and split it into windows of size \(w = 1/\epsilon = 3\)

(a) Perform the lossy counting algorithm.

(b) Is the result of the algorithm independent of the order of the data in the stream \(S\)? If so, why? If not, can you give a counterexample?

(c) What significance does the parameter \(\epsilon\) have beyond the role of its inverse \(w\)?
(d) Can the count even of frequent items be underestimated?

**Solution.**

<table>
<thead>
<tr>
<th>step</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>1,1,2</td>
<td>1,3,2</td>
<td>1,1,2</td>
<td>3,3,3</td>
</tr>
<tr>
<td>1</td>
<td>+2 = 2 → 1</td>
<td>+1 = 2 → 1</td>
<td>+2 = 3 → 2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>+1 = 1 → 0</td>
<td>+1 = 1 → 0</td>
<td>+1 = 1 → 0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>+1 = 1 → 0</td>
<td>0</td>
<td>+3 = 3 → 2</td>
</tr>
</tbody>
</table>

(b) In this example, 3 would obtain a count of 2. But it’s easy to see that if we put the last 3 entries at the very front, the count for 3 would drop to 0 by the end of the algorithm.

(c) $\epsilon$ is by how much we underestimate the frequency of each element. It therefore serves as a baseline frequency that we want items to have at a minimum for us to consider them further, and elements with frequency less than $\epsilon$ are going to be dropped.

(d) Since there are $\epsilon \cdot n$ windows into which we partition the stream and for each window we subtract 1 from the count of elements we kept so far we can underestimate the count of even frequent elements by that much.