Dealing with Massive Data January 24, 2011

Lecture 1: Duplicate Detection

1 Duplicate Detection

We are given a set of n webpages, would like to find all of the duplicates.

There is an easy algorithm if we want to find all of the exact duplicates. For every page W, compute a unique fingerprint (hash) $h(W)$. It is easy to check whether two fingerprints are exactly the same. The running time is $O(n)$.

However, in practice, pages are rarely exact duplicates of each other. In fact, most of the time they are near duplicates:

- Webpages with different time stamps
- Identical content, but with different fonts, or layouts.
- Copies of Amazon reviews, wikipedia, etc. interspersed with advertisements.

Observe that the hashing methodology above will not find near duplicates. How do we find them?

1.1 Similarity Metric

To begin, let's define a reasonable similarity metric. Consider each web page as a set of words (the bag of words model). Let the metric of similarity be the fraction of words the two pages share. For example if the first page is *the black cat ate a mouse* and the second is *the black dog chased the cat*. Then the two pages share three words {the, black, cat} out of the total lexicon of eight words {the, black, cat, ate, a, mouse, dog, chased}. The proposed metric would give a similarity of $3/8$ to these two pages.

We can now make this more formal:

Definition 1 *For any two sets V and W the* Jaccard *similarity of the two sets is defined as:*

$$J(V, W) = \frac{|V \cap W|}{|V \cup W|}.$$

This metric allows us to have a sliding scale of similarity. Note that if $V = W$, that is the two pages are identical, then $J(V, W) = 1$. On the other hand, if $V \cap W = 0$, that is the two pages share no words in common, then $J(V, W) = 0$.

This is a purely syntactic metric – it does not capture any of the semantic meaning of the words. Two pages *The dog chased the cat* and *The cat chased the dog* would have a similarity of 1. One way to deal with this is to expand the representation of the webpage by a set by including all of the bigrams (consecutive pairs of words that appear). In this case, the set representation of the first sentence would be {The dog, dog chased, chased the, the cat}, and the second as {The cat, cat chased, chased the, the dog}, with the resulting Jaccard similarity of $3/5$.

1.2 Finding near duplicates

Now that we have a good notion of similarity, we can try to find all pages that have Jaccard similarity at least τ, for some threshold τ. One obvious solution is to check all possible pairs. Assume that it takes $O(k)$ time to compute the Jaccard similarity between two pages. Then this algorithm has a total running time of $O(n^2k)$.

Suppose that we are dealing with a set of a billion (10^9) pages. And the Jaccard similarity takes a microsecond (10^{-6} seconds) to compute. Even if we are given 1000 machine, the total running time of the algorithm is: $\binom{10^9}{2} \cdot 10^{-6}/10^3 \approx 5 \times 10^8$ seconds, or almost 16 years. The trouble with the above algorithm is that we compare all pairs, even those that are unlikely to be near duplicate. To speed it up, we will first have a filter for finding candidate duplicates.

Hashing for near duplicate detection For two sets V and W, consider the set of elements in $V \cup W$. Let $x \in V \cup W$ be an element chosen uniformly at random. The probability that $x \in V$ and $x \in W$ (i.e. the element is present in both sets) is:

$$Prob[x \in V \wedge x \in W] = \frac{|V \cap W|}{|V \cup W|} = J(V, W).$$

Therefore the following algorithm would give us a pretty good estimate of the similarity of two sets V and W:

Algorithm 1 Similarity(V,W)

```
counter ← 0
for i = 1 to 100 do
    Pick a random element x ∈ V ∪ W
    if x ∈ V ∧ x ∈ W then
        counter ← counter + 1
return counter/100
```

In order to make near duplicate detection efficient, we need one more idea. Let $g(\cdot)$ be a hash function from the words to $[0, 1, \ldots, M]$ for some large M. For a set W, denote by $H(W) = \min_{w \in W} g(w)$. That is $H(W)$ is the minimum value obtained when hashing all of the words with g. The function H has a special property:

Lemma 1 *Let V, and W be two sets and the function H as above. Then*

$$Prob[H(V) = H(W)] = \frac{|V \cap W|}{|V \cup W|} = J(V, W).$$

Given the Lemma, we can try to summarize a set V using its hash value. For example, let $g_1, g_2, \ldots, g_k$ be k independent hash functions g. These give rise to k independent hash functions $H_1, H_2, \ldots, H_k$. For a set V, denote by $S(V)$ its summary vector: $S(V) = \langle H_1(V), H_2, (V), \ldots, H_k(V) \rangle$. The above Lemma says that we can approximately compute the expected similarity of two documents by looking just at their summary vectors.

Denote by $d(S(V), S(W))$ the number of dimensions where V and W are identical. Then:

Lemma 2 *For two sets V and W, let $S(V)$ and $S(W)$ be two summary vectors as defined above. Then*

$$\mathbb{E}[d(S(V), S(W))] = J(V, W).$$

This suggests the following algorithm for duplicate detection. For every webpage W, compute the summary vector $S(W)$. For two pages V and W, if none of the coordinates match, then it is very unlikely that the two documents are near duplicates (see next section for a more precise version of this statement). One way to use this information is to first sort all of the vectors by their 1st coordinate and only compare those pages that have a match. Then resort the vectors by their 2nd coordinate, and compare only those pages that have a match, etc. In practice this leads to a significant improvement.

1.3 Estimating the Errors

A question not answered by the above analysis is how large we should set k, the number of independent hash functions we are using. Intuitively, a higher value of k increases the accuracy of the estimation, but at the cost of larger summary vectors.

To analyze the problem, let X_i be the random variable which is 1 if $H_i(V) = H_i(W)$, and 0 otherwise. We saw that $Prob[X_i = 1] = J(V, W)$. Suppose we have k different hash functions what is the probability that we underestimate the true Jaccard similarity by more than 10% ? This is equivalent to a at most $J(V, W) \cdot 0.9k$ values of X_i being set to 1:

$$\sum_{j=0}^{J(V,W)\cdot 0.9k} \binom{k}{J(V,W)\cdot 0.9k} J(V,W)^j (1 - J(V,W))^{k-j}.$$

While the above formula is correct, it is hard to deal with. In fact, it is hard to figure out what the correct value of k should be to make the failure probability small (e.g. 1%). To answer this question, we turn to the following bounds on the formula above.

Theorem 1 (Chernoff Bound) *Let $X_1, X_2, \ldots, X_n$ be independent 0/1 random variables, with $Prob[X_i = 1] = p_i$. Denote by $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathbb{E}[X] = \sum_{i=1}^{n} p_i$. Then:*

$$Pr[X > (1+\delta)\mu] \leq \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)$$

and

$$Pr[X < (1-\delta)\mu] \leq e^{-\frac{\mu\delta^2}{2}}.$$

Consider the example above. We have $p_i = J(V, W)$, $\mu = k \cdot J(V, W)$ and $\delta = 0.1$. Then the second bound in the Theorem states that:

$$Pr[X < 0.9k \cdot J(V,W)] \leq \exp\left(-\frac{k \cdot J(V,W)}{0.02}\right).$$

In particular, taking $k \approx 100$ leads to a small probability of error. More generally, one can verify that to achieve a probability of error of δ one needs to use $k = O(\log \frac{1}{\delta})$.