FEATURE HASHING

The purpose of this note is to present some basic techniques to extract numerical data from raw text data which may serve as feature (or attribute) values in ML algorithms.

1. Bags for feature extraction

Let $s = [t_1, \ldots, t_N]$ be list of tokens from a set T. T could be for instance a set of words from a (finite) alphabet and s be derived from a text.

Let $T_s = \{t_1, \ldots, t_N\}$ be the set of all tokens appearing in s. Then let $B(s) : T \to \mathbb{N}_0$ be defined by

$$B(s)(t) = \#\{j \mid t_j = t\}.$$

Note that $B(s)(t) > 0$ if and only if $t \in T_s$. $B(s)$ is called the **bag** of tokens[1] of s.

We can interpret s as a sample, the elements of T as features and $B(s)(t)$ as the (numerical) value of the feature t for the sample s.

Example. Bags are often encoded as dictionaries. Consider the text
`John likes to watch movies. Mary likes movies too.`
and the token set
$T = \{"Anna","cinema","John","likes","to","watch","movies","Mary","likes","movies","too"\}$.
Then:

- $s = ["John","likes","to","watch","movies","Mary","likes","movies","too"]$
- $B(s) = \{"Anna" : 0,"cinema" : 0,"John" : 1,"likes" : 2,"to" : 1,"watch" : 1,"movies" : 2,"Mary" : 1,"too" : 1\}$

Algorithm 1: Bag of tokens

Input: A set T of tokens, a list s of tokens
Output: The bag of tokens of s
Initialize B as the dictionary with keys in T and value 0 for each key t.
for $t \in s$ **do**
  $B[t] \leftarrow B[t] + 1$
end
return B

2. Feature hashing

Let s be a sample (e.g. document, email, website, exe file), $\mathcal{F}$ a set of features (e.g. strings/-words) and $v_f(s)$ be the (real, non-negative) value of the feature f for the sample s.

[1] or Bag of Words (BoW), in particular when the tokens are actually words

If $\mathcal{F}$ is very large, the corresponding feature vector $[v_f(s)]_{f\in\mathcal{F}}$ could be simply too large for numerical calculations. In this case one can use the so-called **hashing trick**. Let

$$H : \mathcal{F} \to X$$

be a hash function with $m = \#X$ small. Then we use X as the new feature set where we define the value of a feature x for a sample s by

$$\hat{v}_x(s) = \sum_{f:H(f)=x} v_f(s),$$

so that the feature vector $[\hat{v}_x(s)]_{x\in X}$ has only m entries.

In pratical implementations one takes $X = \{0, ..., m-1\}$ (e.g. $m = 1024$), and

$$H = \mathcal{H} \mod m,$$

where $\mathcal{H}$ is a standard n-bit hash function with values between 0 and $2^n - 1$ such that $2^n - 1 > m$ (e.g. $n = 32$). We will assume that $\mathcal{F}$ is contained in the domain of definition of $\mathcal{H}$.

The following algorithms apply the hashing trick to bags and calculate the feature vector $[v_x]_{0\le x<m}$ with $v_x = \#\{j \mid H(t_j) = x\}$ for a given list $s = [t_1, ..., t_N]$.

Algorithm 2: Feature extraction

Input: A list s of tokens.
Output: A list of length m of feature values.
Initialize $feature\_values$ as a list of length m.
for $x \leftarrow 0$ **to** $m - 1$ **do**
  $feature\_values[x] \leftarrow 0$
  for $t \in s$ **do**
    if $H(t) = x$ **then**
      $feature\_values[x] \leftarrow feature\_values[x] + 1$
    end
  end
end
return $feature\_values$

Algorithm 3: Implementation of Algorithm 2 in Python

```
import numpy as np
def extract_features(list_of_tokens):
  hash_buckets = [H(w) for w in list_of_tokens]
  buckets, counts = np.unique(hash_buckets, return_counts=True)
  feature_values = np.zeros(m)
  for bucket, count in zip(buckets, counts):
    feature_values[bucket] = count
  return feature_values
```