Document Features for Text Processing

Stephen Vardeman

Analytics Iowa LLC

ISU Statistics and IMSE

Generalities

An important application of both supervised and unsupervised learning methods is that of text processing. The object is to quantify structure and commonalities in text documents. Patterns in characters and character strings and words are used to characterize documents, group them into clusters, and classify them into types..

Suppose that N documents in a collection (or corpus) are under study. One needs to define "features" for these, or at least some kind of "kernel" functions for computing the inner products required for producing principal components in an implicit feature space (and subsequently clustering or deriving classifiers, and so on).

Word counts

If one treats documents as simply sets of words (ignoring spaces and punctuation and order of words) one simple set of features for documents $\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_N$ is a set of *counts of word frequencies*. For a set of p words appearing in at least one document, one might take

$$x_{ij} = \text{the number of occurrences of word } j \text{ in document } i$$

and operate on an $N \times p$ data matrix $\mathbf{X}$. These raw counts x_{ij} are often transformed before processing. One popular idea is the use of a `"tf-idf"` (term frequency-inverse document frequency) weighting of elements of $\mathbf{X}$. This replaces x_{ij} with

$$t_{ij} = x_{ij} \ln \frac{N}{\sum_{i=1}^{N} I\left[x_{ij} > 0\right]}$$

or variants thereof. (This up-weights non-zero counts of words that occur in few documents. The logarithm prevents this up-weighting from overwhelming all other aspects of the counts.)

Word counts cont.

One might also decide that document length is not of primary interest and determine to normalize vectors $\mathbf{x}_i$ (or $\mathbf{t}_i$) in one way or another. That is, one might begin with values

$$\frac{x_{ij}}{\sum_{j=1}^{p} x_{ij}} \quad \text{or} \quad \frac{x_{ij}}{\|\mathbf{x}_i\|}$$

rather than values x_{ij}. Of course, if the documents in the corpus all have roughly the same length this normalization changes nothing.

Processing methods that are based only on variants of the word counts x_{ij} are usually said to be based on the "**Bag-of-Words**." They ignore potentially important word order. (The instructions "turn right then left" and "turn left then right" are obviously quite different instructions.) One could then instead consider ordered pairs or n-tuples of words.

Considering order

So, with some "alphabet" $\mathcal{A}$ (that might consist of English words, Roman letters, amino acids in protein sequencing, base pairs in DNA sequencing, etc.) consider strings of elements of the alphabet, say

$$\mathbf{s} = b_1 b_2 \cdots b_{|\mathbf{s}|} \quad \text{where each } b_i \in \mathcal{A}$$

A document (... or protein sequence ... or DNA sequence) might be idealized as such a string of elements of $\mathcal{A}$. An n-gram in this context is simply a string of n elements of $\mathcal{A}$, say

$$\mathbf{u} = b_1 b_2 \cdots b_n \quad \text{where each } b_i \in \mathcal{A}$$

Frequencies of unigrams (1-grams) in documents are (depending upon the alphabet) bag-of-words statistics for words or letters or amino acids, etc.

Considering order cont.

Use of counts of occurrences of all possible n-grams in documents is often be problematic, because unless $|\mathcal{A}|$ and n are fairly small, $p = |\mathcal{A}|^n$ will be huge and $\mathbf{X}$ huge and sparse (for ordinary N and $|\mathbf{s}|$). And in many contexts, sequence/order structure is not so "local" as to be effectively expressed by only frequencies of n-grams for small n.

One idea that seems to be currently popular is to define a set of interesting strings, say $\mathcal{U} = \{\mathbf{u}_i\}_{i=1}^{p}$ and look for their occurrence anywhere in a document, with the understanding that they may be realized as substrings of longer strings. That is, when looking for string $\mathbf{u}$ (of length n) in a document $\mathbf{s}$, one counts every different substring of $\mathbf{s}$ (say $\mathbf{s}' = s_{i_1} s_{i_2} \cdots s_{i_n}$) for which

$$\mathbf{s}' = \mathbf{u}$$

But those substrings of $\mathbf{s}$ matching $\mathbf{u}$ are discounted according to length.

String features

For some $\lambda > 0$ (the choice $\lambda = .5$ seems pretty common) give matching substring $\mathbf{s}' = s_{i_1} s_{i_2} \cdots s_{i_n}$ weight

$$\lambda^{i_n - i_1 + 1} = \lambda^{|\mathbf{s}'|}$$

Document i (represented by string $\mathbf{s}_i$) gets value of feature j

$$x_{ij} = \sum_{s_{il_1} s_{il_2} \cdots s_{il_n} = \mathbf{u}_j} \lambda^{l_n - l_1 + 1}$$

It further seems common to normalize the rows of $\mathbf{X}$ by the usual Euclidean norm, producing in place of x_{ij} the value

$$\frac{x_{ij}}{\|\mathbf{x}_i\|}$$

Inner products for string feature vectors

These features or normalized features are attractive, but potentially computationally prohibitive, particularly since the "interesting set" of strings $\mathcal{U}$ is often taken to be $\mathcal{A}^n$. One doesn't want to have to compute all these features directly and then operate with the very large matrix $\mathbf{X}$. But $\mathbf{X}\mathbf{X}'$ is required to find principal components of the features (or to define SVM classifiers or any other classifiers or clustering algorithms based on principal components). So if there is a way to efficiently compute or approximate inner products for rows of $\mathbf{X}$ defined by the unnormalized features, namely

$$\langle \mathbf{x}_i, \mathbf{x}_{i'} \rangle = \sum_{\mathbf{u} \in \mathcal{A}^n} \left(\sum_{s_{il_1} s_{il_2} \cdots s_{il_n} = \mathbf{u}} \lambda^{l_n - l_1 + 1} \right) \left(\sum_{s_{i'm_1} s_{i'm_2} \cdots s_{i'm_n} = \mathbf{u}} \lambda^{m_n - m_1 + 1} \right)$$

it might be possible to employ this idea.

Inner products for string features

And if the inner products $\langle \mathbf{x}_i, \mathbf{x}_{i'} \rangle$ can be computed efficiently, then so can the inner products

$$\left\langle \frac{1}{\|\mathbf{x}_i\|}\mathbf{x}_i, \frac{1}{\|\mathbf{x}_{i'}\|}\mathbf{x}_{i'} \right\rangle = \frac{\langle \mathbf{x}_i, \mathbf{x}_{i'} \rangle}{\sqrt{\langle \mathbf{x}_i, \mathbf{x}_i \rangle \langle \mathbf{x}_{i'}, \mathbf{x}_{i'} \rangle}}$$

needed to employ $\mathbf{X}\mathbf{X}'$ for the normalized features.

String kernels

It is common to call the function of documents $\mathbf{s}$ and $\mathbf{t}$ defined by

$$K(\mathbf{s},\mathbf{t}) = \sum_{\mathbf{u}\in\mathcal{A}^n} \sum_{s_{l_1}s_{l_2}\cdots s_{l_n}=\mathbf{u}} \sum_{t_{m_1}t_{m_2}\cdots t_{m_n}=\mathbf{u}} \lambda^{l_n-l_1+m_n-m_1+2}$$

the String Subsequence Kernel and then call the matrix $\mathbf{XX}' = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle) = (K(\mathbf{s}_i, \mathbf{s}_j))$ the Gram matrix for that "kernel."

The good news is that there are fairly simple recursive methods for computing $K(\mathbf{s},\mathbf{t})$ exactly in $\mathcal{O}(n|\mathbf{s}||\mathbf{t}|)$ time and that there are approximations that are even faster (see the 2002 *Journal of Machine Learning Research* paper of Lodhi *et al.*). That makes the implicit use of the features or normalized features possible in many text processing problems.