Support Vector Machines for Text Classification

Bart Hamers

1 Introduction

The task of text categorization is the classification of natural text (or hypertext) documents into a fixed number of predefined categories based on their content. This problem arises in a number of different areas including email filtering, web searching, office automation, sorting documents by topic, and classification of new agency stories. Since a document can be assigned to more than one category this is not a multi-classification problem, but can be viewed as a series of binary classification problems, one for each category.

One of the standard representations of text for the purpose of information retrieval (IR) provides an ideal feature mapping for constructing a Mercer kernel. Hence, in this case the use of prior knowledge developed in another field inspires the kernel design; we will see in the following sections that this is a pattern repeated in many case studies. Indeed the kernels somehow incorporate a similarity measure between instances, and it is reasonable to assume that experts working in the specific application domain have already identified valid similarity measures, particular in areas such as information retrieval and generative models.

2 Bag-of-Words approach

The first step in text categorization is to transform documents, which typically are strings of characters, into a representation suitable for the learning algorithm and the classification task. Information Retrieval research suggests that word stems work well as representation units and that their ordering in a document is of minor importance for many tasks. Most frequently used is

the Porter stemming algorithm ([1]). This leads to an attribute–value representation of text. Each distinct word w_i corresponds to a feature, with the number of times word w_i occurs in the document d as its value. To avoid unnecessarily large feature vectors, typically words are considered as features only if they occur in the training data at least 3 times. (This prevents misspelled words and words used rarely from appearing in the dictionary). Also, "stop–words" (like "and", "or", etc.) are not considered because they contain no information. This representation, also called the vector space model, encodes the document in a k-dimensional space where each component represents a corresponding word, neglecting the grammatical structure of the text.

Now lets go into more detail. In the explanation below, w refers to a word, $\mathbf{x}$ is a feature vector that is composed of the various words from a dictionary formed by analyzing the documents. There is one feature vector $\mathbf{x}$ per document d. There are various alternatives and enhancements in constructing the $\mathbf{x}$ vectors. We consider some of them:

- *TF (Term Frequency)*: The i'th component of the feature vector $\mathbf{x}$ is the number of times that a word appears in that document. In our case, a word is a feature only if it occurs in three or more documents. Sometimes the feature vector is normalized to unit length.

- *TF-IDF* uses the above TF multiplied with the IDF (inverse document frequency). The document frequency (DF) is the number of times that word occurs in all the documents (excluding words that occur in less than three documents). The inverse document frequency (IDF) is defined as:

$$IDF\left(w_i\right) = \log\left(\frac{|D|}{DF(w_i)}\right) \tag{1}$$

where $|D|$ is the number of documents. Typically, the feature vector of the TF-IDF entries is normalized to unit length.

- Binary representation which indicates whether a particular word occurs in a particular document.

[1]For more information and software about the Porter stemming algorithm see: http://www.tartarus.org/˜martin/PorterStemmer/

These representation schemes leads to very high–dimensional feature spaces containing 10000 dimensions and more. Once you have this vector space model of your documents, you already can do mathematical analysis of our data by using simple linear algebra techniques. A very nice introduction is given by M. Berry. [1]. Our approach however will focus more on AI techniques and the use of kernel methods.

3 Kernel methods and text

Why Should SVMs Work Well for Text Categorization?[4]

To find out what methods are promising for learning text classifiers, we should find out more about the properties of text.

- *High dimensional input space:* When learning text classifiers, one has to deal with very many (more than 10000) features. Since SVMs use overfitting protection, which does not necessarily depend on the number of features, they have the potential to handle these large feature spaces.

- *Few irrelevant features*: One way to avoid these high dimensional input spaces is to assume that most of the features are irrelevant. Feature selection tries to determine these irrelevant features. Many have noted the need for feature selection to make the use of conventional learning methods possible, to improve generalization accuracy, and to avoid "overfitting". Unfortunately, in text categorization there are only very few irrelevant features. A classifier using only those "worst" features has a performance much better than random. Since it seems unlikely that all those features are completely redundant, this leads to the conjecture that a good classifier should combine many features (learn a "dense" concept) and that aggressive feature selection may result in a loss of information.

- *Document vectors are sparse*: For each document, the corresponding document vector contains only few entries which are not zero. SVMs are well suited for problems with dense concepts and sparse instances.

- Most text categorization problems are linearly separable. The idea of SVMs is to find such linear (or polynomial, RBF, etc.) separators.

These arguments give theoretical evidence that SVMs should perform well for text categorization.

4 Design of appropriate kernels

4.1 Kernel based on bag-of-words

This vector space model can now be used to design a kernel for classifying our documents. An often used technique is see the document representation vector as $\phi(x)$. Where this $\phi(x)$ can be found by the Term Frequency, TF-IDF or binary representation. The function

$$K(x, y) = \langle \phi(x) \cdot \phi(y) \rangle \tag{2}$$

is clearly a valid kernel, since it is the inner product in an explicitly constructed feature space. Its kernel matrix is therefor always positive semi-definite, and Support Vector Machines can be used with $K(x, y)$ for discrimination. Notice also that this kernel is similar to the cosine distance measure which has proven its use in Information Retrieval. Given the extreme sparseness and the high dimensionality of the vector $\phi(x)$, it is reasonable to assume that the points the feature are going to be easy to separate, and standard hard margin SVM will be sufficient. However other kernels like the Polynomial or RBF may also be used if necessary. Furthermore, it is possible to devise fast techniques for computing sparse inner products.

4.2 String or Sequence kernels

The assumption behind the popular bag-of-words representation is that the relative position of words or phrases has little importance in most Information Retrieval (IR) tasks. Documents are just represented by word frequencies (and possibly additional weighting and normalization), and the loss of the information regarding the word positions is more than compensated for by the use of powerful algorithms working in vector space.

Although some methods inject positional information using phrases or multi-word units or local co-occurrence statistics, the underlying techniques are based on a vector space representation.

String kernels (or *sequence kernels*) are one of the first significant departures from the vector-space representation in this domain. String kernels [5] are similarity measures between documents seen as *sequences* of symbols (e.g., possible characters) over an alphabet. In general, similarity is assessed

by the number of (possibly non-contiguous) matching subsequences shared by two sequences. The more subsequences they have in common, the more similar they are. An important part is that such subsequence do not to be contiguous, and the degree of contiguity of one such subsequence in a document determines how much weight it will have in the comparison. Non contiguous occurrences are penalized according to the number of gaps they contain.

Formally, let Σ be a finite alphabet, and $s = s_1 s_2 ... s_{|s|}$ a sequence of length $|s|$ over Σ (i.e. $s_i \in S; 1 \leq i \leq |s|$). Let $i = [i_1, i_2, ..., i_n]$, with $1 \leq i_1 < i_2 < ... < i_n \leq |s|$, be a subset of the indices in s: we will indicate as $s[\mathbf{i}] \in \Sigma^n$ the subsequence $s_{i1} s_{i2} ... s_{in}$. Σ^n is the set of all finite sequences of length n. Note that $s[\mathbf{i}]$ does not necessarily form a contiguous subsequence of s. For example, if s is the sequence $CART$ and $\mathbf{i} = [2; 4]$, then $s[\mathbf{i}]$ is AT. Let us write $l(\mathbf{i})$ the length spanned by $s[\mathbf{i}]$ in s, that is: $l(\mathbf{i}) = i_n - i_1 + 1$. The set of all sequences will be denoted by Σ^*

$$\Sigma^* = \bigcup_{n=1}^{\infty} \Sigma^n. \tag{3}$$

We now define a feature space $F_n = \mathbb{R}^{\Sigma^n}$. The feature mapping ϕ for a sequence s is given by defining the u coordinate $\phi_u(s)$ for each $u \in \Sigma^n$ subsequence of s

$$\phi_u(s) = \sum_{\mathbf{i}: u = s[\mathbf{i}]} \lambda^{l(\mathbf{i})} \tag{4}$$

where $\lambda \in]0, 1]$ is a decay factor used to penalize non-contiguous subsequences.

These features measure the number of occurrences of subsequences in the sequence s weighting them according to their length. Hence, the inner product of the feature vectors for two strings s and t give a sum over all common subsequences weighted according to their frequency of occurrence and lengths. The *sequence kernel* of two strings s and $t$over Σ is defined as:

$$K_n(s, t) = \sum_{u \in \Sigma^n} \langle \phi_u(s) \cdot \phi_u(t) \rangle \tag{5}$$

$$= \sum_{u \in \Sigma^n} \sum_{\mathbf{i}: u = s[\mathbf{i}]} \sum_{\mathbf{j}: u = t[\mathbf{j}]} \lambda^{l(\mathbf{i}) + l(\mathbf{j})} \tag{6}$$

A direct computation of all the terms under the nested sum becomes impractical even for small values of n. In addition, in the spirit of the kernel trick, we wish to calculate $K_n(s,t))$ directly rather than perform an explicit expansion in feature space. This can be done using a recursive formulation proposed by [5], which leads to a more efficient dynamic-programming-based implementation. Let

$$K'_i(s,t) = \sum_{u \in \Sigma^i} \sum_{\mathbf{i}: u=s[\mathbf{i}]} \sum_{\mathbf{j}: u=t[\mathbf{j}]} \lambda^{|s|+|t|-i_1-j_1+2}, \quad i = 1, ..., n-1, \tag{7}$$

that is counting the length to the end of the strings s and t instead of just $l(\mathbf{i})$ and $l(\mathbf{j})$. We can now define a recursive computation for K'_i and hence compute K_n,

$$K'_0(s,t) = 1, \text{ for all } s,t, \tag{8}$$

$$K'_i(s,t) = 0, \text{ if } \min(|s|,|t|) < i, \tag{9}$$

$$K_i(s,t) = 0, \text{ if } \min(|s|,|t|) < i, \tag{10}$$

$$K'_i(sx,t) = \lambda K'_i(s,t) + \sum_{j: t_j = x} K'_{i-1}(s, t[1:j-1]) \lambda^{|t|-j+2} \quad i = 1, ..., n-1, \tag{11}$$

$$K_n(sx,t) = K_n(s,t) + \sum_{j: t_j = x} K'_{n-1}(s, t[1:j-1]) \lambda^2 \quad i = 1, ..., n-1$$

where $t[i:j]$ is the subsequence $t_i ... t_j$ of t.

The time required to compute the kernel according to this formulation is $O(n|t||s|)$. In some situations, it may be convenient to perform a linear combination of sequence kernels with different values of n. Using the recursive formulation, it turns out that computing all kernel values for subsequences of lengths up to n is not significantly more costly than computing the kernel for n only. In order to keep the kernel values comparable for different values of n and to be independent from the length of the strings , it may be advisable to consider the normalized version:

$$\hat{K}(s,t) = \frac{K(s,t)}{\sqrt{K(t,t) K(s,s)}} \tag{12}$$

This is a generalization of the "cosine normalization" which is widespread in IR and corresponds to the mapping $\hat{\phi}(s) = \frac{\phi(s)}{\|\phi(s)\|_2}$.

Recent it was proven that the use of these sequence kernels with sequences of words instead of characters. This approach has several advantages, in particular it is more efficient computationally and it ties in closely with standard linguistic pre-processing techniques.

5 Advances in text mining with kernels

Since kernel methods are one of the most popular techniques used in text-mining, this area is also the test area for many new advances in machine learning. In this section we will shortly mention some of them.

- One-Class classification: In many applications, like search engines, we don't have access to two classes. So for training this task, one only has one class of subjects at its disposal. For example, websites we are interested in. This problem is called *one-class classification.* It was recently studied for document classification by Manevitz et al.[6]

- Transductive Inference: In text classification problem, often the setup is to find relevant documents in a database based on training set. So, the test set is already known in advance. This information contained in this database of test examples, can be used during training of the classifier, without knowing the labels of the test set. It is theorertically proven that the SVM trained by this principle has a better generalization bound than a normal SVM. This was also experimentally verified by Joachims[4].

- Active Learning: In many supervised learning tasks, labeling instances to create a training set is time–consuming and costly; thus, finding ways to minimize the number of labeled instances is beneficial. Usually, the training set is chosen to be a random sampling of instances. However, in many cases active learning can be employed. Here, the learner can actively choose the training data. It is hoped that allowing the learner this extra flexibility will reduce the learner's need for large quantities of labeled data. Pool–based active learning has been recently introduced. The learner has access to a pool of unlabeled data and can request the true class label for a certain number of instances in the pool. In many domains this is reasonable since a large quantity of unlabeled data is readily available. The main issue with active learning is finding a way

to choose good requests or queries from the pool. Active learning with SVMs can significantly reduce the need for labeled training instances[7].

6 Application: Spam Categorization

Spam[3] is defined as an e-mail message that is unwanted basically it is the electronic version of junk mail that is delivered by the postal service. One of the reasons for the proliferation of spam is that bulk e-mail is very cheap to send and although it is possible to build filters that reject e-mail if it is from a known spammer, it is easy to obtain alternative sending addresses. Good online sources and information about spam include http://spam.abuse.net, http://www.cauce.org, and http://www.junke-mail.org. There have been various attempts to use learning machines that classify e-mail.

Solutions to the proliferation of spam are either technical or regulatory. Technical solutions include filtering based on sender address or header content. The problem with filtering is that sometimes a valid message may be blocked. Thus, it is not our intention to automatically reject e-mail that is classified as spam. Rather, the following scenario is envisioned: in the training mode, users will mark their e-mail as either spam or nonspam. After a finite number of examples are collected, the learning machine will be trained and the performance on new examples predicted. The user can then invoke the e-mail classifier immediately or wait until the number of examples is enough such that performance is acceptable. After the training mode is complete, new e-mail will be classified as spam or nonspam. In one presentation mode, a set of new email messages is presented in a manner consistent with the time of delivery and the spam messages color-coded. It is then up to the user to either read the e-mail or trash the email.

An alternative presentation mode is to deliver e-mail to the user in decreasing order of probability that the e-mail is nonspam. That is, e-mail with high probability (according to the classifier) of being nonspam is at the top of the list. In either of these modes, the filter does not reject any messages, only indicates whether the message has a high priority of being spam. It is highly desirable that if the user decides that e-mail messages be rank-ordered by degree of confidence that the rank ordering be reliable. By reliable, we mean that the user can either start at the top of the list of e-mail messages and be fairly confident that they represent nonspam messages or start at the bottom of the list and be confident that the messages are spam. It is only

near the middle of the list (low confidence) that it is reasonable to the user that a few nonspam or spam messages may be misclassified. Therefore, it is important that our learning algorithm not only classify the messages correctly but that a measure of confidence is associated with that classification so that the message can be rank ordered.

References

[1] M.W. Berry, Z. Drmac, E.R. Jessup. Matrices, Vector Spaces, and Information Retrieval, SIAM Review, vol 41, No 2, pp 335-362, 1999.

[2] N. Cancedda, E. Gaussier, C. Goutte, JM. Renders. Word-Sequence Kernels, Journal of Machines Learning Research 3, pp 1059-1082, 2003.

[3] H. Drucker, D. Wu, V. Vapnik. Support Vector Machines for Spam Categorization, In Advances in Neural Information Processing Systems 9, pp 155-161, Cambridge, MA, 1997.

[4] T. Joachims. *Learning to Classify Text using Support Vector Machines*, Kluwer, 2002.

[5] H. Lohdi, C. Saunders, J. Shawe-Taylor, and C. Watkins. Text classification using string kernels, Journal of Machines Learning Research 2, pp 419-444, 2002.

[6] L.M. Manevitz, M. Yousef. One-Class SVMs for Document Classification, Journal of Machines Learning Research 2, pp 139-154, 2002.

[7] S. Tong, D. Koller. Support Vector Machine Active Learning with Applications to Text Classification, In Proceedings of the Seventeenth International Conference on Machine Learning, 2000.