Fast Training of a Graph Boosting for Large-Scale Text Classification

Hiyori Yoshikawa(✉) and Tomoya Iwakura

Fujitsu Laboratories Ltd., Kawasaki, Japan
{y.hiyori,iwakura.tomoya}@jp.fujitsu.com

Abstract. This paper proposes a fast training method for graph classification based on a boosting algorithm and its application to sentimental analysis with input texts represented by graphs. Graph format is very suitable for representing texts structured with Natural Language Processing techniques such as morphological analysis, Named Entity Recognition, and parsing. A number of classification methods which represent texts as graphs have been proposed so far. However, many of them limit candidate features in advance because of quite large size of feature space. Instead of limiting search space in advance, we propose two approximation methods for learning of graph-based rules in a boosting. Experimental results on a sentimental analysis dataset show that our method contributes to improved training speed. In addition, the graph representation-based classification method exploits rich structural information of texts, which is impossible to be detected when using other simpler input formats, and shows higher accuracy.

Keywords: Text classification · Feature engineering · Graph boosting

1 Introduction

Text classification is a fundamental task in Natural Language Processing (NLP) and has applications to a wide variety of tasks including spam filtering, sentimental analysis, topic classification, and profile estimation. While bag-of-words are widely used as features for classification, a number of researches show that using richer structure of texts results in better performance (Kudo and Matsumoto 2004; Gee and Cook 2005; Matsumoto et al. 2005; Arora et al. 2010; Jiang et al. 2010; Iwakura 2013). In other words, features incorporating additional information about texts such as word dependencies, part of speech (POS) tags, and named entity types have potential to be key features for classification.

A remarkable approach to classification using rich information of texts is to represent the texts as graphs. Compared to the other formats such as bag-of-words, n-grams and trees, graph format has strong power of expression enough to incorporate almost any kinds of characteristics related to words or texts at the same time. More precisely, the other formats listed above can be interpreted as special cases of graphs: bag-of-words correspond to vertexes and n-grams

R. Booth and M.-L. Zhang (Eds.): PRICAI 2016, LNAI 9810, pp. 638–650, 2016.
DOI: 10.1007/978-3-319-42911-3_53

to paths, and trees are graphs in themselves. For a sentimental analysis task, for example, a key feature might be a combination of word order, dependency, and sentiment polarity of each word. Classification methods with graph based features have potential to achieve higher performance especially in such cases that key features might be combinations of different kinds of characteristics. As we refer to in Sect. 5, there are a number of works which use graph representation of texts for text classification. Most of these existing methods convert inputs into subgraph-based feature vectors and then apply a classification algorithm for the vectors such as perceptron (Frank 1958) or Support Vector Machines (SVMs) (Boser et al. 1992). Since the number of potential subgraphs tend to be quite large and it is practically impossible to consider all of them, such methods usually select a part of features in advance using a frequent pattern mining algorithm such as gSpan (Yan and Han 2002). However, infrequent features are sometimes important. Another approach is to deal with the problem as a graph classification problem. One of the most popular graph classification algorithms is perceptron or SVMs with graph kernels (Kashima et al. 2003), which works without previous selection of subgraph features. Although such methods achieve considerably high performance, they have some disadvantages. First, in learning and classification it sometimes requires the calculation of graph kernels for a large number of pairs of graphs. Second, it is difficult to see which subgraphs have strong effect because features do not appear explicitly.

In this paper, we use a graph boosting algorithm originally proposed by (Kudo et al. 2004) for text classification. This boosting method learns subgraph based decision stumps as weak classifiers, and finally constructs a classifier as a linear combination of the stumps. The calculation time for classification does not depend on the size of training dataset but the size of rules, and rules are represented explicitly by subgraphs that constitutes the classifier. In addition, as Kudo et al. (2004) point out, the boosting based method can reflect slight difference of structures of features, while kernel based methods are not good at distinguishing features which have similar structures. It would be an important property for text classification, since the difference of a single word may result in opposite meaning of the whole sentence. A problem is that the graph boosting method requires much learning time, despite using pruning methods suggested in the original paper. We propose two approximation methods to improve training speed of the graph boosting: one is to divide subgraph features into some buckets in order to limit search space of rules, and the other is to expand the search space dynamically according to weak classifiers chosen in previous steps. Experimental results show that our approximation makes it possible to improve the classification accuracy much faster than the original algorithm.

The rest of the paper is organized as follows. In Sect. 2 we define the problem setting, and refer to the graph boosting method. In Sect. 3 we show two approximation methods to calculate weak classifiers efficiently. Section 4 shows experimental results, Sect. 5 discusses the relation to related works, and Sect. 6 concludes this paper.

2 Preliminary

2.1 Problem Setting

In this paper, we focus on binary text classification problems. We are given a set of texts $T = (t_1, t_2, \ldots, t_N)$, each of which is associated with a class label $y_i \in \{\pm 1\}$ $(i = 1, \ldots, N)$. Generally, the class labels are defined based on particular characteristics of the texts such as topics, sentiment, or profiles of writers. The task is to induce a classifier which assigns labels to new texts.

We solve this problem as a graph classification problem by representing the input texts as graphs with NLP techniques to extract syntactic and semantic structure of the original texts. Then the problem reduces to the task to induce a classifier which assigns labels to graphs made from new texts.

2.2 Boosting Based Graph Classification

Our algorithm is based on the graph based classification method by (Kudo et al. 2004). As a boosting method we adopt an improved AdaBoost proposed by (Schapire and Singer 1999), since it showed higher accuracy than the boosting algorithm used in (Kudo et al. 2004). We call the algorithm *Boost-K*. Here we summarize the idea of the general boosting method and Boost-K.

Boosting is one of the well-known meta-algorithms for ensemble learning. Boosting sequentially learns $K(> 0)$ *weak classifiers* and finally constructs a classifier as the linear combination of the weak classifiers. Let h_j be the weak classifier obtained at the jth iteration $(j = 1, 2, \ldots, K)$. Then we eventually obtain the final classifier consisting of the weak classifiers as:

$$f(x) = \mathrm{sgn}\left(\sum_{j=1}^{K} h_j(x)\right), \tag{1}$$

At each iteration in a typical boosting algorithm, a weak classifier is trained to minimize the current weighted error rate. When the classifier is updated by a weak classifier, the weight is recalculated so that misclassified examples have larger weight and correctly classified ones have smaller weight. In this way, the classifier is efficiently trained focusing on the misclassified examples at previous steps.

Boost-K classifies graphs based on their subgraphs. The weak classifiers are decision stumps each of which reflects existence of a particular subgraph in a graph. For a subgraph g and a real number α (confidence value), the subgraph-based decision stump is defined as:

$$h_{\langle g,\alpha\rangle}(x) := \begin{cases} \alpha \text{ if } g \subseteq x \\ 0 \text{ otherwise} \end{cases}, \tag{2}$$

where $g \subseteq x$ means the graph g is the subgraph of the graph x.[1] At each iteration j, the boosting algorithm chooses a weak classifier $h_j = h_{\langle g_j, \alpha_j \rangle}$ with:

$$\langle g_j, \alpha_j \rangle = \arg\min_{\langle g, \alpha \rangle} \sum_{i=1}^{N} d_i^j \exp(-y_i h_{\langle g, \alpha \rangle}(x_i)), \tag{3}$$

where d_i^j is the weight for the input graph x_i at the current iteration j.[2] The right hand side is minimized for a particular g by choosing:

$$\alpha = \frac{1}{2} \log \left(\frac{D_{j,+1}(g)}{D_{j,-1}(g)} \right), \tag{4}$$

where $D_{j,*}(g) := \sum_{i=1}^{N} d_i^j I(g \subseteq x_i \wedge y_i = *)$ $(* \in \{\pm 1\})$ with the indicator function $I(\cdot)$. As shown in (Iwakura and Okamoto 2008), we can minimize (3) by maximizing the following *gain function*, or *gain* simply:

$$gain_j(g) := \left| \sqrt{D_{j,+1}(g)} - \sqrt{D_{j,-1}(g)} \right|. \tag{5}$$

At every step, the algorithm choose a weak classifier which maximizes (5) and then the weight $\boldsymbol{d} = (d_1, d_2, \ldots, d_N)$ is updated by:

$$d_i^{j+1} = d_i^j \exp(-y_i h_{\langle g_j, \alpha_j \rangle}(x_i)) \tag{6}$$

and then normalized to satisfy $\sum_{i=1}^{N} d_i = 1$.

2.3 Efficient Calculation of Weak Classifiers

At each step in the above boosting algorithm, we need to find a weak classifier that maximizes the gain function (5). Generally, the number of possible subgraphs is so large that it is practically impossible to calculate gains for all subgraph features. Thus we need some efficient ways to find the most appropriate subgraph feature. Boost-K addresses this problem based on the following two ideas, both of which do not affect the result of learning.

The first idea, which is by (Kudo et al. 2004), is to search subgraphs on a canonical search space based on *gSpan* algorithm (Yan and Han 2002). gSpan is an efficient method to enumerate subgraphs which appear in a given graph set frequently. The key idea is to retain subgraphs by *DFS codes*. A DFS code is constructed by running depth-first search (DFS) in a search space called *DFS Code Tree*. A node of the DFS Code Tree corresponds to a 5-tuple $(i, j, l_i, l_{i,j}, l_j)$ which represents an edge of a subgraph. Here i and j are the vertex indexes of endpoints of e, and l_i, l_j, and $l_{i,j}$ are labels of the vertices i and j, and the edge $\{i, j\}$, respectively. By running depth-first search in the DFS Code Tree,

[1] In (Kudo et al. 2004), a weak classifier is defined to return $-\alpha$ if $g \not\subseteq x$. Considering the results of preliminary experiments, we decided to use the above definition instead.

[2] We may omit the iteration index j when no confusion can arise.

we obtain a DFS code as a sequence of the tuples. Since DFS codes have a lexicographic order, we can use the *minimum DFS code* as the 'canonical' code of a graph. When we find that the current DFS code is not minimum, we can 'prune' the search space to avoid the redundant search. In this way, we can efficiently enumerate all subgraphs.

The second idea, which is also rooted in (Kudo et al. 2004), is to use an upper bound of gain functions. The following is a key observation:

$$\{i : g' \subseteq x_i, y_i = y\} \subseteq \{i : g \subseteq x_i, y_i = y\} \quad (\forall g' \supseteq g). \tag{7}$$

That is, a graph g' appears in a graph x_i only if its subgraph g appears in x_i. Then the following is directly derived from the definition (5): for every graph g' which contains g, the gain $gain(g')$ is bounded by:

$$u(g) := \max\left(\sqrt{D_{j,+1}(g)}, \sqrt{D_{j,-1}(g)}\right). \tag{8}$$

In the depth-first traversal of a DFS Code Tree, the graphs are referred to starting from a single edge graph[3], and then larger graphs are referred to as the search reaches deeper levels. Using the above observation, we can avoid redundant searches for larger graphs: we can prune the search space when we find that the upper bound does not exceed the current maximal gain.

3 Approximation Methods for More Efficient Learning

Despite the processes described above, it still takes much time to search for subgraph features in a large graph set. In addition to the above methods, we adopt two other approximation methods for further efficient learning.

3.1 Dividing Features into Buckets

The first method is to divide features into some buckets, whose idea comes from (Iwakura and Okamoto 2008). They show that distribution of feature into hundreds of buckets results in almost the same or sometimes higher accuracy. We adopt *F-dist*-like distribution, that is, distribution of features in ascending order based on their frequencies. This distribution keeps average frequencies in each bucket roughly the same. Since it is practically impossible to enumerate all possible subgraph features, we modify the method to adapt to our situation as follows.

1. Count frequency of each vertex label in the graph set.
2. Sort vertex labels according to their frequency.
3. Put the vertex labels into $b(> 0)$ buckets in order of their frequency.

[3] With a slight modification, we can start searches from single node graphs so that the result may contain single node feature graphs.

4. In the jth iteration we search only for the subgraphs whose start point (id 0 in the DFS Code) has the label in the $(j\%b)$th bucket, where $(j\%b)$ means the remainder of j divided by b.

In this way, one can expect that the total frequency of the feature subgraphs searched in each iteration become roughly the same.

When applying this method, using only the minimum DFS codes results in excessive limitation of the search space, since the search space depends on the first vertex in DFS codes. Thus we omit minimum DFS code tests when applying this approximation. Note that a subgraph feature can appear in more than two buckets. The experimental results in Sect. 4 show that the method reduce the calculation time even though it omits minimum DFS code tests.

3.2 Smaller Rule Priority

The search space of weak classifiers expand explosively with size of subgraphs to search. Usually, however, only a small fraction of subgraph features are significant. In order to avoid unnecessary search, we limit the search space based on the following hypothesis: when a large subgraph feature is important, some of its subgraphs are also important. To realize this idea, we propose to apply the idea of (Freund 1999) to the graph boosting algorithm. (Freund 1999) learns alternating decision tree by boosting. Starting from a set of the simplest rules, the algorithm extends the search space according to the result of each step of boosting. We apply the learning method to our situation of learning subgraph based decision stumps as follows:

1. Initialize $\mathcal{H}$ as the set of all single node graphs.
2. For each jth iteration, do the following:
   (a) Search for the best weak classifier $h_{\langle g_j, \alpha_j \rangle}$ with $g_j \in \mathcal{H}$.
   (b) Update $\mathcal{H}$ by $\mathcal{H} \leftarrow \mathcal{H} \cup \{g' \mid g_j \subseteq g', |g'|_{\mathrm{E}} = |g_j|_{\mathrm{E}} + 1\}$, where $|x|_{\mathrm{E}}$ indicates the number of edges in x.

That is, the algorithm searches for only subgraph features which contains g_j and larger than g_j by one edge for subgraphs g_j chosen in previous iterations.

4 Experiments

To evaluate the proposed method, we used the Amazon review data created by (Blitzer et al. 2007). This dataset contains customer review texts for products available at Amazon. Table 1 shows the product categories we used and the size of each dataset. For each category, we picked positive (4.0 or more score) reviews and negative (2.0 or less score) reviews and learned binary classifiers that distinguish between positive and negative reviews. The construction of graphs from input texts is described in Sect. 4.1. In addition to the above approximation, we also limit the search space to the subgraph features whose size (number of edges) are no more than $ms \in \{0, 1, 2, ...\}$, where $ms = 0$ means the search space is limited to single node subgraph features. We implemented the algorithm by C++.

Table 1. Used categories in Amazon review data. '#Train' and '#Test' mean the number of training and test data.

Category	#Train		#Test	
	positive	negative	positive	negative
video	27489	5074	3054	563
electronics	16165	4544	1796	504
kitchen-housewares	14164	3708	1573	411
toys-games	9522	2312	1057	256
apparel	7111	1216	790	135
camera-photo	5679	990	630	109

4.1 Construction of Input Graphs

Here we describe how we construct graph features from input texts. We construct a graph from each input text, and finally obtain a set of graphs whose number equals that of input texts. From now we call a graph corresponding to a text a *feature graph*.

For a word w, let p_w be the POS tag and n_w be the named entity type of w. Let us write an input text as $t = w_1 w_2 \dots w_l$, where w_i $(1 \leq i \leq l)$ is the i-th word of the text. The set of vertices of a feature graph consists of some of the following vertices:

- v_w^i: the vertex corresponding to the surface form of the word w_i $(i = 1, \dots, l)$,
- v_p^i: the vertex corresponding to the POS of w_i $(i = 1, \dots, l)$,
- v_n^i: the vertex corresponding to the named entity type of w_i $(i \in \{1, \dots, l\})$, which appears only if w_i is a part of a named entity,
- v_{S}: the vertex representing the start of the input text, and
- v_{T}: the vertex representing the end of the input text,

whose labels correspond to w_i, p_{w_i}, n_{w_i}, [S] and [T], respectively. Every edge corresponds to order or dependency of words. Edges with the label ORDER are between v_*^i and v_{**}^{i+1} $(*, ** \in \{w, p, n\})$ if such nodes exist $(i = 1, \dots, l-1)$, v_{S} and v_*^1, and v_*^l and v_{T} $(* \in \{w, p, n\})$. In addition, each pair of vertices v_w^i and v_w^j which has a dependency relation has an edge with a label corresponding to the kind of relation. For graph construction, we used SENNA (Collobert et al. 2011; Collobert 2011)[4]. In this paper, we call a graph which has all the above vertexes and edges 'type A', a graph which has only the information of words and order 'type B', a graph which has only the information of words and dependencies 'type D', and a graph which has the information of words, order and dependencies 'type BD'. Figure 1 shows an example of a feature graph

[4] To convert the output of SENNA into tree format, we used Penn2Malt 0.2 (http://stp.lingfil.uu.se/~nivre/research/Penn2Malt.html) with the following options: head rules in (http://stp.lingfil.uu.se/~nivre/research/headrules.txt), deprel 1, and punctuation 1.

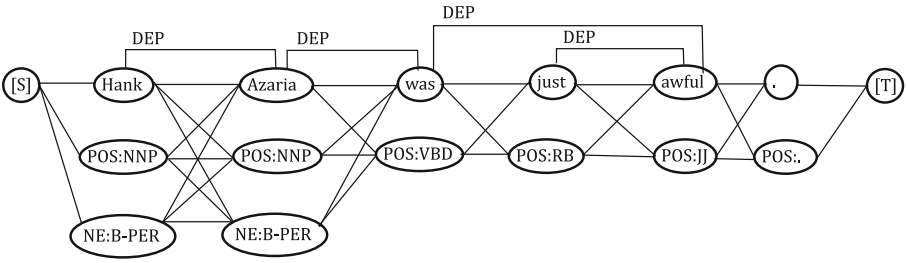

Fig. 1. An example of a feature graph corresponding to a sentence "Hank Azaria was just awful".

(type A) made from a sentence in a negative review in 'video' category, "Hank Azaria was just awful". Words in the vertices and on the edges indicate the labels of the vertices and edges. The edges on which no word appears have the label ORDER.

We also constructed other kinds of graphs: 'type Bs'. These graphs contain information about sentimental polarity of words. We append polarity to the words according to SentiWordNet 3.0 (Baccianella et al. 2010). We simply append 'Sent:p', 'Sent:n', or 'Sent:pn' labels to a word if it has non-zero positive, negative, or both score in SentiWordNet 3.0. The polarity of words is expressed as corresponding vertices: if a word w_i has positive, negative, or both polarity, we append a corresponding node v^i_{P}, v^i_{N}, or v^i_{PN} with the label 'Sent:p', 'Sent:n', or 'Sent:pn' respectively and connect these nodes and other nodes v^j_w ($j \in \{1, \ldots, l\}$) according to word order. 'Type Bs' graphs are constructed by adding polarity nodes to 'type B' graphs. Note that 'type Bs' graphs do not contain dependency edges.

Table 2 shows the average number of vertices and edges of training graphs in each category.

4.2 Results

Calculation Time and Accuracy. We conducted a preliminary experiment to evaluate the effect of the proposed approximation. We used 'camera-photo' category from Amazon review dataset. The input is a graph set of type A and we set $ms = 3$. We compared the calculation time and accuracy (F-measure for the 'negative' label) for 4 types of algorithms. The algorithm 'orig' is the original graph boosting method with no approximation, 'B1000' uses only the former approximation with bucket size 1,000, 'S' uses the latter approximation, and 'B1000S' uses both. The result in Fig. 2 shows that the two approximation methods contribute to improvement of accuracy in much shorter time than the original algorithm.

Table 2. The average number of vertices and edges of training graphs in each category. Ave(V) and Ave(E) mean the average number of vertices and edges of the graphs, respectively.

	type B		type D		type BD	
Category	Ave(V)	Ave(E)	Ave(V)	Ave(E)	Ave(V)	Ave(E)
video	184.78	183.78	184.78	137.49	184.78	321.27
electronics	122.49	121.49	122.49	96.18	122.49	217.68
kitchen-housewares	104.02	103.02	104.02	81.17	104.02	184.19
toys-games	106.27	105.27	106.27	79.32	106.27	184.60
apparel	70.54	69.54	70.54	53.69	70.54	123.24
camera-photo	148.34	147.34	148.34	116.60	148.34	263.94
	type A		type Bs			
	Ave(V)	Ave(E)	Ave(V)	Ave(E)		
video	379.51	920.62	245.70	305.63		
electronics	246.51	593.17	166.11	208.73		
kitchen-housewares	207.88	497.07	142.86	180.70		
toys-games	212.78	506.06	142.64	178.00		
apparel	140.44	333.57	97.36	123.18		
camera-photo	298.66	718.98	202.20	255.06		

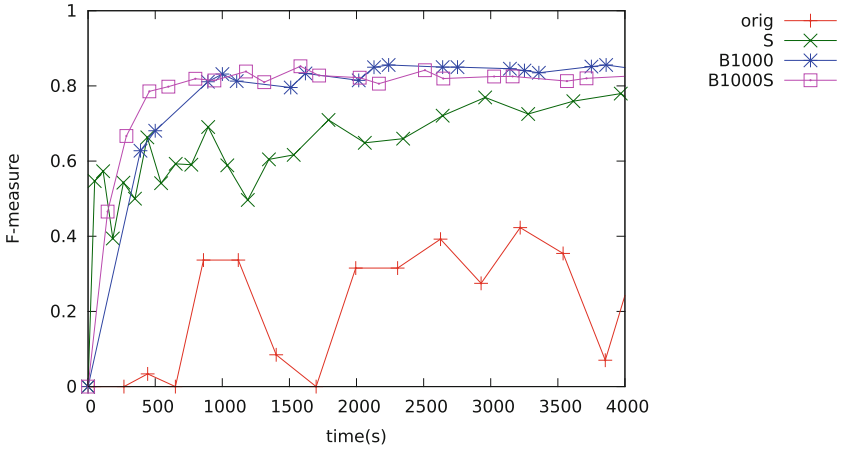

Fig. 2. Effect of the approximation methods

Accuracy Obtained from Different Graph Types. Table 3 shows the results of boosting algorithms measured by F-measure for the 'negative' label. All classifiers are trained through 10,000 iterations with the two approximation methods. We again devided the features into 1,000 buckets. The results show that the graph inputs which include structural information (type BD, type A, and type Bs) performs better than inputs which do not in the most categories. This indicates that some subgraph features with structural information contribute considerably to improved accuracy. Especially, in the most cases the graphs of

Table 3. F-measures for the 'negative' label. The underlined score indicates the best score of the category. 'Ranking average' means the averaged ranking about F-measures for each graph type. '#Best' or '#Worst' indicates the number of categories each type of graph achieved the best or the worst scores, respectively.

(single vertices, strings or trees) Category	type B, $ms = 0$ (bag-of-words)	type B, $ms = 3$ ($n(\leq 4)$-grams)	type A, $ms = 0$	type D, $ms = 3$
video	0.882	0.886	0.864	0.878
electronics	0.795	0.824	0.793	0.809
kitchen-housewares	0.755	0.789	0.803	0.806
toys-games	0.746	0.731	0.737	0.741
apparel	0.694	0.776	0.725	0.732
camera-photo	0.847	0.822	0.804	0.814
Average	0.787	0.804	0.787	0.798
Ranking average	4.67	3.17	5.83	4.5
#Best	1	1	0	0
#Worst	2	0	3	0
(general graphs)	type BD, $ms = 3$	type A, $ms = 3$	type Bs, $ms = 3$	SVM with gSpan, type A, $ms = 3$
video	0.881	0.885	0.892	0.753
electronics	0.823	0.822	0.820	0.765
kitchen-housewares	0.768	0.837	0.816	0.739
toys-games	0.724	0.757	0.770	0.683
apparel	0.724	0.771	0.778	0.680
camera-photo	0.810	0.825	0.816	0.746
Average	0.788	0.816	0.815	0.728
Ranking average	5.33	2.33	2.17	-
#Best	0	1	3	-
#Worst	1	0	0	-

type Bs performs better than others. We emphasize that such a kind of representation of texts is not possible by strings or trees but by graphs. Figure 3 shows the examples of extracted features.

Comparing to SVM with Graph Mining. The last column of the Table 3 shows the results of L_1-regularized L_1-loss SVM implemented in Classias (Okazaki 2009). The feature vectors for SVM are frequent subgraphs in the training datasets. To make the vectors, we conducted gSpan with minimum support 0.01. The input graphs are type A with $ms = 3$. The displayed results are the best ones among different coefficients $c \in \{0.01, 0.05, 0.1, 0.5, 1, 5, 10\}$ for L_1-regularization. The total training time (including gSpan and SVM) for the category 'camera-photo' is 1,036.3 s, while the boosting with the two approximation almost converges within 1,000 s (See Fig. 2). The fact that all the results

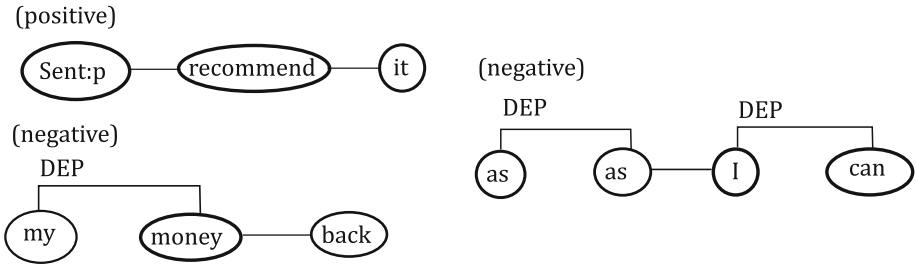

Fig. 3. Examples of extracted features.

of SVM are below those of boosting suggests that feature selection with frequent subgraph mining scrapes off not only unneccesary features but also significant ones.

5 Related Works

There is a number of researches which use graph based features for text classification. Matsumoto et al. (2005) combine word sub-sequences and dependency sub-trees for sentiment classification. Jiang et al. (2010) represent texts as graphs by combining word sequences and syntactic trees. Arora et al. (2010) introduce graph based features representing several linguistic annotations for sentiment classification. This method uses not only unigrams and subgraphs but also newly defined features constructed by combining original features. Because of the large feature space, all of these methods choose features by some mining methods in advance and apply vector based learning algorithms such as perceptron and SVMs. Our method need not mine features previously, since the boosting algorithm find significant subgraph features from the whole search space automatically. The proposed method also has an advantage of easy parameter tuning. The parameters are only number of iteration, maximal size of subgraphs and bucket size, while the mining based algorithms need to decide both parameters for mining such as minimum frequency and for learning algorithms.

A boosting based classification of semi-structured text has already been proposed by (Kudo and Matsumoto 2004). However, the method is applicable only to tree formats, while our algorithm runs on any graph sets. It enables us to make use of rich structure of text more flexibly.

Recent improvement of graph boosting algorithms includes attacks on extended problem settings (Pan et al. 2015a; Wu et al. 2015) and on imbalanced data in the real world (Pan et al. 2015b) and use of additional information (Fei and Huan 2014; Pan et al. 2016). Pan et al. (2015a, 2015b, 2016) and Wu et al. (2015) take similar approaches in that they explore subgraph-based weak classifiers using gSpan and pruning by gain upper bound as we referred to in Sect. 2.3. In addition, Pan et al. (2015a) solves linear programming to minimize the risk function like gBoost (Saigo et al. 2009) and accelerates this step for

large scale graphs. These methods can easily be combined with our approximation methods, since our approximation method modifies only the selection step of the discriminative subgraph features.

6 Conclusion

In this paper, we proposed a graph boosting based text classification and efficient approximation methods for the calculation of weak classifiers. The experimental results show that our algorithm extracts significant subgraph features efficiently. Our algorithm helps guess what kinds of information are significant for classifiers. In the case of Amazon review data, the information of sentimental tags seems to be important. It is possible to add any other kinds of nodes or edges to the input graph. It is also a future work to combine our methods with some other text classification methods including the recently-proposed ones we refer to in Sect. 5.

References

Arora, S., Mayfield, E., Rosé, C.P., Nyberg, E.: Sentiment classification using automatically extracted subgraph features. In: Proceedings of the NAACL HLT 2010 Workshop on Computational Approaches to Analysis and Generation of Emotion in Text, pp. 131–139 (2010)

Baccianella, S., Esuli, A., Sebastiani, F.: SentiWordNet 3.0: an enhanced lexical resource for sentiment analysis and opinion mining. In: Proceedings of Seventh International Conference on Language Resources and Evaluation, pp. 2200–2204 (2010)

Blitzer, J., Dredze, M., Pereira, F.: Biographies, bollywood, boom-boxes and blenders: domain adaptation for sentiment classification. In: Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, pp. 440–447 (2007)

Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. In: Proceedings of the Fifth Annual ACM Conference on Computational Learning Theory, pp. 144–152 (1992)

Collobert, R.: Deep learning for efficient discriminative parsing. In: International Conference on Artificial Intelligence and Statistics (2011)

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. J. Mach. Learn. Res. **12**, 2493–2537 (2011)

Fei, H., Huan, J.: Structured sparse boosting for graph classification. ACM Trans. Knowl. Discov. Data **9**, 1–22 (2014)

Frank, R.: The perceptron: A probabilistic model for information storage and organization in the brain. Psycholog. Rev. **65**, 386–408 (1958)

Freund, Y.: The alternating decision tree algorithm. In: Proceedings of the Sixteenth International Conference on Machine Learning, pp. 124–133 (1999)

Gee, K.R., Cook, D.J.: Text classification using graph-encoded linguistic elements. In: Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference, pp. 487–492 (2005)

Iwakura, T.: A boosting-based algorithm for classification of semi-structured text using frequency of substructures. In: Proceedings of 9th International Conference on Recent Advances in Natural Language Processing, pp. 319–326 (2013)

Iwakura, T., Okamoto, S.: A fast boosting-based learner for feature-rich tagging and chunking. In: Proceedings of Twelfth Conference on Computational Natural Language Learning, pp. 17–24 (2008)

Jiang, C., Coenen, F., Sanderson, R., Zito, M.: Text classification using graph mining-based feature extraction. Knowl-Bas. Syst. **23**, 302–308 (2010)

Kashima, H., Tsuda, K., Inokuchi, A.: Marginalized kernels between labeled graphs. In: Proceedings of the Twentieth International Conference on Machine Learning, pp. 321–328 (2003)

Kudo, T., Maeda, E., Matsumoto, Y.: An application of boosting to graph classification. Adv. Neural Inf. Process. Syst. **17**, 729–736 (2004)

Kudo, T., Matsumoto, Y.: A boosting algorithm for classification of semi-structured text. In: Proceedings of 9th Conference on Empirical Methods in Natural Language Processing, pp. 301–308 (2004)

Matsumoto, S., Takamura, H., Okumura, M.: Sentiment classification using word sub-sequences and dependency sub-trees. In: Ho, T.B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS, vol. 3518, pp. 301–311. Springer, Heidelberg (2005)

Okazaki, N.: Classias: a collection of machine-learning algorithms for classification (2009). http://www.chokkan.org/software/classias/

Pan, S., Wu, J., Zhu, X.: CogBoost: boosting for fast cost-sensitive graph classification. IEEE Trans. Knowl. Data Eng. **27**, 2933–2946 (2015)

Pan, S., Wu, J., Zhu, X., Long, G., Zhang, C.: Boosting for graph classification with universum. Knowl. Inf. Syst. **47**, 1–25 (2016)

Pan, S., Wu, J., Zhu, X., Zhang, C.: Graph ensemble boosting for imbalanced noisy graph stream classification. IEEE Trans. Cybern. **45**, 940–954 (2015)

Saigo, H., Nowozin, S., Kadowaki, T., Kudo, T., Tsuda, K.: gBoost: a mathematical programming approach to graph classification and regression. Mach. Learn. **75**, 69–89 (2009)

Schapire, R.E., Singer, Y.: Improved boosting algorithms using confidence-rated predictions. Mach. Learn. **37**, 297–336 (1999)

Wu, J., Pan, S., Zhu, X., Cai, Z.: Boosting for multi-graph classification. IEEE Trans. Cybern. **45**, 430–443 (2015)

Yan, X., Han, J.: gSpan: graph-based substructure pattern mining. In: Proceedings of 2002 IEEE International Conference on Data Mining, pp. 721–724 (2002)