Term based semantic clusters for very short text classification

Citation for published version (APA):

Document license:
CC BY

Document status and date:
Published: 08/11/2019

Publisher Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Term Based Semantic Clusters for Very Short Text Classification

Jasper Paalman\textsuperscript{1}, Shantanu Mullick\textsuperscript{2}, 
Kalliopi Zervanou\textsuperscript{2}, and Yingqian Zhang\textsuperscript{2}

\textsuperscript{1} Jheronimus Academy of Data Science 
\texttt{j.v.paalman@tilburguniversity.edu}
\textsuperscript{2} School of Industrial Engineering, Eindhoven University of Technology 
\texttt{\{s.mullick, k.zervanou, yqzhang\}@tue.nl}

Extended Abstract

Very short texts, such as tweets and invoices, present challenges in classification. Such texts abound in ellipsis, grammatical errors, misspellings, and semantic variation. Although term occurrences are strong indicators of content, in very short texts, sparsity makes it difficult to capture enough content for a semantic classifier. A solution calls for a method that not only considers term occurrence, but also handles sparseness well. In this work, we introduce such an approach for the classification of short invoice descriptions, in such a way that each class reflects a different group of products or services. The developed algorithm is called Term Based Semantic Clusters (TBSeC).

TBSeC attempts to exploit the information about semantically related words offered by pre-trained word embeddings which acts as a query expansion technique for our invoice descriptions. The contribution of this paper lies in (i) combining the advantages of word embeddings with conventional term extraction techniques (ii) applying our method in an application domain not previously investigated, namely invoice text, which is characterised by specialised terminology and very short, elliptical and/or ungrammatical text, in Dutch, a language that is morphologically richer than English and therefore posing an additional challenge in statistical approaches.

Our proposed method, TBSeC, consists of two stages. In the first stage we use class-specific textual information to build semantic concept clusters. Concept clusters are vector representations of strongly related terms that are distinctive for a certain class. In the second stage, we compute cluster similarity scores on generated concept clusters for a given description, thereby forming a semantic feature space. This serves as a ranking function that can be used in both unsupervised and supervised learning tasks (in this work we use Support Vector Machines as a supervised learning algorithm).

Concept clustering starts by extracting distinctive terms, particular to a class (representing invoice expense types). Distinctive terms are extracted based on...
normalized term frequency. For a given class, word embeddings belonging to
found terms are used to form numerous clusters. Hence, for each class multiple
clusters are created and each cluster can be seen as a group of terms that are
closely related. Each cluster is transformed into a concept cluster vector by
taking the respective word embeddings of terms and computing the mean over
each dimension. For semantic cluster similarity we use an adapted form of a
similarity measure from related work, where we combine idf scoring with word
by word cosine similarity to calculate a weighted similarity score.

In our experimental setup, we use an invoice dataset of 1.5 million instances
requiring classification into 111 distinct classes. Invoice descriptions on average
contain 2.80 terms, with a standard deviation of 1.55. FastText word embeddings
pretrained on Dutch Wikipedia are used for sentence embedding construction
and for use in semantic similarity computations. A supervised machine learning
approach using a Support Vector Machine is used to perform the final classifi-
cation using the similarity measure values as features. We compare the perfor-
mance of TBSeC to a number of existing methods and we test whether combining
methods leads to an increase in performance. The following table shows the main
results:

<table>
<thead>
<tr>
<th>Methods</th>
<th>test acc</th>
<th>test sd</th>
<th>train acc</th>
<th>train sd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benchmark</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf · idf</td>
<td>43.93</td>
<td>0.89</td>
<td>82.79</td>
<td>0.07</td>
</tr>
<tr>
<td>tf</td>
<td><strong>44.85</strong></td>
<td>0.67</td>
<td>84.84</td>
<td>0.12</td>
</tr>
<tr>
<td>LSA</td>
<td>7.89</td>
<td>0.35</td>
<td>9.16</td>
<td>0.20</td>
</tr>
<tr>
<td>LDA</td>
<td>11.78</td>
<td>0.37</td>
<td>14.15</td>
<td>0.15</td>
</tr>
<tr>
<td>TBSeC</td>
<td>36.03</td>
<td>0.60</td>
<td>46.96</td>
<td>0.26</td>
</tr>
<tr>
<td>FastText</td>
<td>32.62</td>
<td>0.79</td>
<td>41.26</td>
<td>0.24</td>
</tr>
<tr>
<td>P-mean</td>
<td>39.78</td>
<td>0.63</td>
<td>90.33</td>
<td>0.1</td>
</tr>
<tr>
<td>Feature comb.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf &amp; TBSeC</td>
<td>47.83</td>
<td>0.54</td>
<td>86.17</td>
<td>0.13</td>
</tr>
<tr>
<td>tf &amp; FastText</td>
<td>47.07</td>
<td>0.48</td>
<td>85.81</td>
<td>0.14</td>
</tr>
<tr>
<td>tf &amp; TBSeC &amp; FastText</td>
<td><strong>47.86</strong></td>
<td>0.42</td>
<td>86.38</td>
<td>0.10</td>
</tr>
</tbody>
</table>

The fact that term frequency has comparable performance to tf · idf rein-
forces the notion that all terms within an invoice description are important and
that term-specific weighting has limited value. Moreover, techniques that are
concerned with dimensionality reduction (LSA, LDA) perform worse, arguably
because they truncate a large amount of information, most of which should have
been retained. Sentence embeddings and our method TBSeC perform relatively
well, with accuracy scores nearing performance levels of tf · idf and term fre-
quency. The best performance is achieved by the combination of the new method
with tf/idf and FastText. Combinations of features were found to reach a bound
in effectiveness, highlighting that methods ultimately start impeding each other.
Businesses can use our method to derive actionable insights from online user gen-
erated content such as firm-specific tweets, online reviews and customer chats
logs. Future work could test TBSeC on larger texts, since it offers more room to
differentiate from sentence embeddings.
Abstract

Very short texts, such as tweets and invoices, present challenges in classification. Although term occurrences are strong indicators of content, in very short texts, the sparsity of these texts makes it difficult to capture important semantic relationships. A solution calls for a method that not only considers term occurrence, but also handles sparseness well. In this work, we introduce such an approach, the Term Based Semantic Clusters (TBSeC) that employs terms to create distinctive semantic concept clusters. These clusters are ranked using a semantic similarity function which in turn defines a semantic feature space that can be used for text classification. Our method is evaluated in an invoice classification task. Compared to well-known content representation methods the proposed method performs competitively.

1 Introduction

Bag-of-words approaches (Harris, 1954) to text classification rely on measures of term occurrence, or co-occurrence, such as $tf \cdot idf$ (Salton and Buckley, 1988), log-likelihood (Dunning, 1993), and mutual information (Church and Hanks, 1990). Such methods, despite their enduring popularity, are well known for their shortcomings in dealing with numerous natural language issues, such as morphological, semantic, and other types of variation and ambiguity (Augenstein et al., 2017). These issues become more critical in very short texts, such as microblogs, chat logs, reviews and invoices, because of the lack of data and context that could provide more reliable measures and a source for semantic disambiguation.

Distributional semantic models (Baroni and Lenci, 2010) such as pre-trained word embeddings (Pennington et al., 2014; Grave et al., 2018) encode words as fixed sized real-valued vectors. Embeddings may address the issues of data sparseness and lack of extensive context, by providing a semantic representation model that is not as sensitive to literal word occurrence in the data, thus providing semantic information coverage of out-of-vocabulary words (Bojanowski et al., 2017). This is because embeddings may allow for any given word to be mapped to a real-valued vector (e.g. by using character n-grams), even if it hasn’t been observed during training. Additionally, embeddings implicitly capture semantic, syntactic and lexical properties, thereby representing implicit relationships. In this way, embeddings provide a rich representation that is otherwise difficult to attain. In a short text scenario, despite early findings that a small corpus size may not be sufficient for representing the relationships between words and documents (Song et al., 2014), availability of pre-trained embeddings makes this issue less of a concern. Despite these advantages and growing popularity, embeddings trained on general language data usually do not perform well in (i) specialised domains (Liu et al., 2018; Kameswara Sarma, 2018) and in (ii) languages with richer morphological variation than English (Zervanou et al., 2014). In this work, we attempt to address these issues while exploiting the advantages of pre-trained word embeddings in a text classification task for very short, domain specific texts in a morphologically rich language, i.e., invoices in Dutch.

Our aim is to classify short invoice descriptions, in such a way that each class reflects a different group of products or services, as illustrated in the examples in Table 1. When considering such texts, the augmented information offered by embeddings is crucial, because such texts abound in ellipsis, grammatical errors, misspellings, and
2 Related work

Text or document classification is defined as the assignment of text sections or entire documents to a predefined set of categories (Feldman and Sanger, 2007). For this purpose, algorithms process various types of text representations which are used as features for describing content. To our knowledge, invoice text classification has not been investigated previously. Work related to text classification of short texts has been applied for microblogs (Singh et al., 2016; Ren et al., 2016; Missier et al., 2016), email subject classification (Alsmadi and Alhami, 2015) and spam detection (Bahgat et al., 2016).

Initial approaches to document content representation used counts of term frequency and inverse document frequency, $tf \cdot idf$ (Salton and Buckley, 1988), whereby frequently occurring terms are assumed to represent document content and inverse document term frequency scores select the most distinctive terms for a given document within a collection. Various subsequent approaches use variants of term occurrence measures with probabilities, such as $\chi^2$-test, log likelihood (Dunning, 1993) and mutual information (Church and Hanks, 1990), or attempts to combine statistical measures with various types of linguistic and stop-word filters, so as to refine the keyword results. Considerations regarding term ambiguity and variation also led to rule-based approaches (Jacquemin, 2001) and resource-based approaches exploiting existing thesauri and lexica, such as UMLS (Hliaoutakis et al., 2009), or WordNet (Aggarwal et al., 2018). Knowledge poor statistical approaches, such as Latent Semantic Analysis (Deerwester et al., 1990) and Latent Dirichlet Allocation (Blei et al., 2003) attempt to detect document content in an unsupervised manner while reducing the dimensionality of the feature space of other bag-of-word approaches, but are also sensitive to sparse data and variation in short texts.

The advent of large semantic resources in the form of pre-trained word embeddings (Pennington et al., 2014; Grave et al., 2018) gave rise to a new line of approaches employing word embeddings for document content representation, such as Word2Vec (Mikolov et al., 2013), FastText (Bojanowski et al., 2017), GloVe (Pennington et al., 2014), and AutoML (Grave et al., 2018). These models provide a dense vector representation of words, which can be used for various downstream tasks, such as text classification, paraphrase identification, and entity recognition.

An approach reported by Bartoli et al. (2010) focuses on image features of scanned invoices rather than the invoice text.

---

<table>
<thead>
<tr>
<th>Text</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vervoer Almere-Lille</td>
<td>Travel expenses</td>
</tr>
<tr>
<td>Transport Almere-Lille</td>
<td></td>
</tr>
<tr>
<td>600GB SAS interne harde schijf</td>
<td>Automation hardware</td>
</tr>
</tbody>
</table>

Table 1: Example documents and classes
Within this line of approaches, methods have also been developed using word embeddings specifically for document classification, such as task-oriented word embeddings (Liu et al., 2018) and word-sense based embeddings (Jin et al., 2016). Embedding models encoding words in documents or document sections have also been developed, such as Doc2vec (Le and Mikolov, 2014), Infersent (Conneau et al., 2017), Skip-thought (Kiros et al., 2015) and FastSent (Hill et al., 2016). Such type of embeddings can be employed to calculate the semantic similarity between texts, but the risk is that intricate word-specific semantic relationships are lost. Methods originating from text similarity research using word rather than document embeddings, such as Kusner et al. (2015); Kenter and De Rijke (2015); De Boomen et al. (2016) attempt to address this issue. Embeddings have also been used for keyphrase extraction via supervised (Mahata et al., 2018) and unsupervised (Bennani-Smires et al., 2018) approaches.

Finally, the problem of variation and data sparsity with very short texts has been addressed in the past with query expansion approaches (Vechtomova, 2009). For text similarity purposes query expansion techniques have been used for document term augmentation, exploiting relevant search results and a matrix representation (Sahami and Heilman, 2006; Abhishek and Hosanagar, 2007) or a combination of search results page count difference and lexico-syntactic patterns derived from text snippets (Bollegala et al., 2007).

3 The TBSeC methodology

Our proposed method, TBSeC, consists of two stages: In the first stage we use class-specific textual information to build semantic concept clusters. Concept clusters are vector representations of strongly related terms that are distinctive for a certain class. In the second stage, we compute cluster similarity scores on generated concept clusters for a given description, thereby forming a semantic feature space. This serves as a ranking function that can be used in both unsupervised and supervised learning tasks.

3.1 Concept clustering

Concept clustering starts by extracting distinctive terms, particular to a class\(^3\). Distinctive terms are extracted based on normalized term frequency. For a given class, word embeddings belonging to found terms are used to form numerous clusters. Hence, for each class multiple clusters are created and each cluster can be seen as a group of terms that are closely related. Each cluster is transformed into a concept cluster vector by taking the respective word embeddings of terms and computing the mean over each dimension. This process is illustrated in Figure 1, with actual examples included and word embeddings indicated as vectors with dots.

Specifically, the method works as shown in Algorithm 1 for \( n \) distinct classes, \( k \) most frequent terms to be incorporated as cluster and normalized term frequency (i.e., Bag-of-words, \( B_{ij} \)) threshold \( t \). A more detailed description is given underneath the algorithm. Algorithm line numbers refer to steps as denoted in Figure 1.

Terms that provide a clear distinctive value to a class are retrieved using term frequency and L1 normalization over rows and columns. Terms not appearing in the vocabulary of the embedding model, or having a score below the normalized threshold \( t \) are filtered out. Word embeddings for these selected terms are employed to create concept clusters for each class using the DBSCAN clustering model (Ester et al., 1996). Terms can either be included in a multi-term cluster through DBSCAN clustering or can be added as a single-term cluster when occurring frequently enough based on \( k \). Ultimately, for each class \( i \), concept clusters are created as a single vector equal to the averaged embedding of included terms.

3.2 Semantic cluster similarity

We adapt the similarity measure by Kenter and De Rijke (2015), which is based on the BM25 framework (Robertson et al., 2009), to propose our semantic cluster similarity measure. We combine idf scoring with word by word cosine similarity to calculate a weighted similarity score.

The Kenter and De Rijke (2015) function for calculating semantic text similarity between two sentences \( s_t \) and \( s_s \) is as defined as follows:

\[ S_{BM25}(s_t, s_s) = \frac{\sum_{w \in s_t} \log \left( \frac{N}{n_w} \right) \cdot \cos(w, s_s)}{\sum_{w \in s_t} \log \left( \frac{N}{n_w} \right)} \]

\( \log \left( \frac{N}{n_w} \right) \) is the idf score of word \( w \) in sentence \( s_t \), and \( \cos(w, s_s) \) is the cosine similarity between word \( w \) and the entire sentence \( s_s \).

---

\(^3\)In the case of invoice classification, each class refers to a specific expense type.
Figure 1: Concept clustering process diagram. Upper panel: Preparing normalized bag of words using descriptions by class. Lower panel: Employing found values to create concept clusters for a given class

\[ f_{sc}(s, c) = \sum_{w \in s} IDF(w) \cdot \frac{\text{sem}(w, c) \cdot (k_1 + 1)}{\text{sem}(w, s) + k_1 \cdot (1 - b + b \cdot |s|/\text{avg}sl)} \]  

(1)

Parameters \( k_1 \) and \( b \) are inherited from the BM25 framework and serve as smoothing parameters. Parameter \( k_1 \) influences what semantic text similarity value is approached asymptotically, thereby limiting the influence that semantic term similarity can have. Parameter \( b \) provides a degree of importance to sentence length ratio \( |s|/\text{avg}sl \), comparing sentence length to the average sentence length \( \text{avg}sl \) of sentences being ranked. The function \( \text{sem} \) returns the semantic term similarity of term \( w \) with respect to text \( s \), as follows:

\[ \text{sem}(w, s) = \max_{w' \in s} f_{sem}(w, w') \]  

(2)

where \( f_{sem} \) is a vector similarity function that is typically computed as cosine similarity.

In TBSeC, the Kenter and De Rijke (2015) function is adapted to compare each sentence to all composed concept clusters. In our implementation, parameter \( b \) is redundant, because each concept cluster is represented as a single embedding and the measure is computed by a single cosine similarity score. Moreover, we normalize our similarity score with the number of terms appearing in the sentence to allow for use in a supervised learning task. Finally, we remove term-specific weighting, for four reasons: First, idf scoring imposes a hefty constraint on the terms that can be used because of the predefined vocabulary. Second, we argue that the limited amount of terms appearing in a description justifies the exclusion of term-specific weighting. Each term in the description holds an important piece of information and differentiating is not essential. Third, terms that don’t hold considerable semantic importance are not likely to steer the score towards an incorrect class. Each concept cluster is created to serve as a distinctive concept, thus making it unrealistic that unimportant terms will relate to it well. Fourth, terms in descriptions are subject to frequent misspellings and personal abbreviations, making idf scores inherently unreliable in this setting.

Based on the changes to Equation 1 discussed above, our function for calculating semantic cluster similarity \( f_{sc}(s, c) \) between text \( s \) and concept cluster \( c \) is defined as follows:

\[ f_{sc}(s, c) = \frac{1}{|s|} \sum_{w \in s} \frac{(\text{sem}(w, c) \cdot t(w)) \cdot (k_1 + 1)}{(\text{sem}(w, c) \cdot t(w)) + k_1} \]  

(3)

where \( t(w) \) is the term frequency of term \( w \) in the text. The score is normalized by the number of terms \( |s| \) in text \( s \). In addition to smoothing parameter \( k_1 \), two other hyper parameters \( \text{sem}th \) and \( \text{semsq} \) are added to influence scoring. These hyper parameters affect the result of \( \text{sem} \) as follows.

\[ \text{sem}(w, c) = \begin{cases} f_{sem}(w, c) & \text{if } f_{sem}(w, c) \geq \text{sem}th \\
\frac{f_{sem}(w, c)^2}{\text{sem}th} & \text{if } f_{sem}(w, c) < \text{sem}th \\
0 & \text{if } f_{sem}(w, c) < \text{sem}th \end{cases} \]  

(4)
Algorithm 1 Concept Clustering

Parameters:
\( k \) - No. of most frequent terms to be incorporated as cluster
\( t \) - Normalized bag of words threshold

Input:
\( D_i, i \in \{1, \ldots, n\} \) - Merged descriptions for each class

Output:
\( C_i, i \in \{1, \ldots, n\} \) - Concept clusters for each class

\[ B_{ij} \leftarrow \text{Calculate bag of words using } D_i, \text{ with found vocabulary set } V, \quad i \in \{1, \ldots, n\}, j \in \{1, \ldots, |V|\} \]

\( C_i \leftarrow \text{Instantiate empty concept cluster array, with } i \in \{1, \ldots, n\} \)

\[ \text{for } i \in \{1, \ldots, n\} \text{ do} \]
\[ B_{ix} \leftarrow \text{L1 normalize } B_{ix} \quad \triangleright \text{Normalize by class} \]
\[ \text{end for} \]

\[ \text{for } j \in \{1, \ldots, |V|\} \text{ do} \]
\[ B_{xj} \leftarrow \text{L1 normalize } B_{xj} \quad \triangleright \text{Normalize by term} \]
\[ \text{end for} \]

\[ \text{for } i \in \{1, \ldots, n\} \text{ do} \]
\[ \text{terms} \leftarrow \text{Array of found terms that appear in word embedding vocabulary} \]
\[ \text{end for} \]

\[ C_i \leftarrow \text{CREATE CLUSTERS} (\text{emb}, k, \text{terms}) \]
\[ \text{end for} \]

\[ \text{return } C \]

\[ \text{function CREATE CLUSTERS} (\text{emb}, k, \text{terms}) \]
\[ \text{Instantiate DBSCAN clustering model with } \text{eps, min_samples} \text{ and metric } = \text{cosine similarity} \]
\[ \text{Fit emb on DBSCAN model} \]
\[ \text{clusters} \leftarrow \text{formed clusters as collections of word embeddings} \]
\[ \text{for term in } k \text{ most frequent terms do} \]
\[ \text{if term not used in clusters then} \]
\[ e \leftarrow \text{word embedding of term} \]
\[ \text{cluster.append}(e) \]
\[ \text{end if} \]
\[ \text{end for} \]

\[ \text{concepts}_c \leftarrow \text{empty concept cluster array with } c \in \{1, \ldots, |\text{clusters}|\} \]
\[ \text{for } c \in \{1, \ldots, |\text{clusters}|\} \text{ do} \]
\[ \text{concepts}_c \leftarrow \text{coordinate mean over each dimension} \]
\[ \text{end for} \]

\[ \text{return concepts} \]
\[ \text{end function} \]

The semantic threshold \( \text{sem}_{th} \) serves as a way to add a semantic similarity bound above which it will be presumed to hold importance. Parameter \( \text{sem}_{th} \) achieves that when \( f_{\text{sem}}(w, c) \) is under the set threshold value, that \( \text{sem}(w, c) \) equals 0. Squaring \( f_{\text{sem}}(w, c) \) through \( \text{sem}_{sq} \) increases term importance of terms that are a near match and lowers importance of terms that match to a lesser extend. Squaring \( f_{\text{sem}}(w, c) \) therefore promotes the divergence of semantic similarity scores.

Semantic cluster similarity \( f_{\text{scs}} \) produces features for use in supervised learning applications, but the initial performance of TBSeC is measured without a predictive model. For this reason, a similarity score for each class is required in order to rank the classes. This is calculated as \( \text{scs} \) for each class \( i \), by extracting the maximum score over all concept clusters \( c \in C_i \) (see \( C_i \) in Algorithm 1):

\[ \text{scs}(s, C_i) = \max_{c \in C_i} f_{\text{scs}}(s, c) \quad (5) \]

4 Experimental setup

This section covers data description, data processing and the experimental setup for our method.

4.1 Data

Our invoice data originate from an auditing company. In the data, as illustrated in the examples in Table 1, each class refers to a particular type of expenses. Available data is accessed from a data directory, where each file is specific to a client. For our purposes, only the invoice description and class assignment are relevant. The volume of the entire data directory amounts to approximately 1.5 million instances. There is a total of 111 unique classes to which assignments are made. The five classes that are least represented have 24, 106, 178, 418 and 452 entries respectively. Invoice descriptions on average contain 2.80 terms, with a standard deviation of 1.55.

4.2 Word embeddings

Pre-trained Dutch FastText word embeddings\(^3\) are used for sentence embedding construction and for use in semantic similarity computations (Bojanowski et al., 2017). The FastText embedding model was trained on Dutch Wikipedia.

\(^3\)https://github.com/facebookresearch/fastText/blob/master/docs/pretrained-vectors.md
4.3 Data processing

Descriptions are processed using a procedure similar to the one used in training the FastText model. Special characters are replaced with a whitespace, stopwords in both the English and Dutch language are dropped, digits are removed and finally terms are retrieved by splitting on any sequence of unicode whitespace characters. When creating validation sets special care is taken to remove duplicates and to include data from all individual clients and all classes in a randomized manner. As a result, largely balanced validation sets are formed with data from various sources.

4.4 Learning algorithm

During supervised learning a Support Vector Machine⁴ is used with regularization parameter $C = 0.1$ and a linear kernel. This classifier performed best when compared to other feasible classifiers (e.g. random forest), given a local working memory bounded set-up, and allows for the use of sparse matrices. Regularization parameter $C$ regulates the importance of focusing on correctly classifying training samples in favor of realizing a hyperplane with a large minimum margin. A high $C$ can lead to overfitting, a low $C$ can lead to the inability to learn meaningful decision boundaries. We set $C$ to 0.1 since it appears to offer a good balance on the basis of the main validation set in terms of limited running time and general performance.

4.5 Parameter tuning

Prior to including our framework in a supervised learning task, we optimize the parameters (see section 5.1 for results). We construct an initial set of concept clusters using preset values $k = 5$ and $t = 0.8$. Parameters are set such that the model offers a well-performing baseline with low chances of overfitting. Initial concept clusters are used to tune semantic cluster similarity parameters. This order of parameter tuning is chosen, because it is relatively straightforward to pick sensible values for $k$ and $t$, as opposed to $f_{scs}$ hyperparameters. Table 2 lists the attempted combinations of parameter settings for $f_{scs}$.

After parameter tuning for semantic cluster similarity, employed concept clusters are reconsidered. Values in the range from 5 to 50 with a step size of 5 are attempted for the $k$ most frequent terms incorporated as cluster.

Validation is performed on a dataset with approximately 5,000 entries. Performance differences for $f_{scs}$ are evaluated using three performance measures: (1) accuracy, (2) ranking loss and (3) standardized score. Accuracy is calculated by picking the class with the highest semantic similarity score as predicted class and finding the percentage of correctly classified instances. Ranking loss is calculated by obtaining the rank of the true class. The standardized score is calculated by standardizing all scores for a given instance and retrieving the score of the true class. By standardizing it can be observed how the score for the true class is positioned against all other scores. Ultimately, the objective is to maximize the accuracy and standardized score and to minimize ranking loss. Afterwards, we investigate the influence of parameter $k$ for concept cluster construction on the basis of accuracy and dimension size. Accuracy is calculated as an unsupervised score as well as a 5-fold cross validated supervised score. Both scoring methods use best values for $k_1$, $sem_{th}$ and $sem_{sq}$ which are found in the previous step. Results are compared to determine an appropriate value for parameter $k$ for use of features in a predictive model.

4.6 Invoice classification

We use the proposed semantic cluster similarity matching method, to measure performance in a classification task. We compare the performance to existing methods and we test whether combining methods leads to an increase in performance.

The parameters for generation of semantic cluster similarity scores are set in accordance with values obtained during parameter tuning.

For validation, a dataset containing approximately 20,000 entries is used. The data is used to perform 5-fold cross validation to determine overall performance. For all tests, we chose to use accuracy as performance quality measurement. Only one out of 111 classes shows significant imbalance, which for testing overall method performance is deemed negligible. Consequently, no balancing is performed and no alternative quality measure, such

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Distinct values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1$</td>
<td>[1.2, 1.6, 2.0]</td>
</tr>
<tr>
<td>$sem_{sq}$</td>
<td>[true, false]</td>
</tr>
<tr>
<td>$sem_{th}$</td>
<td>[0, 0.1, 0.2, 0.3, 0.4]</td>
</tr>
</tbody>
</table>

Table 2: $f_{scs}$ parameter settings

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameters</th>
<th>Dimension size</th>
</tr>
</thead>
<tbody>
<tr>
<td>tf · idf</td>
<td>-</td>
<td>117,766</td>
</tr>
<tr>
<td>tf</td>
<td></td>
<td>117,766</td>
</tr>
<tr>
<td>LSA</td>
<td># components: 100</td>
<td>100</td>
</tr>
<tr>
<td>LDA</td>
<td># topics: 200</td>
<td>200</td>
</tr>
<tr>
<td>FastText</td>
<td>-</td>
<td>300</td>
</tr>
<tr>
<td>P-mean</td>
<td>P-values: {−∞, 1, 2, 3, ∞}</td>
<td>1500</td>
</tr>
</tbody>
</table>

Table 3: Benchmark methods

<table>
<thead>
<tr>
<th>$k_1$</th>
<th>$sem_{sq}$</th>
<th>$sem_{th}$</th>
<th>acc</th>
<th>loss</th>
<th>std score</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0</td>
<td>true</td>
<td>0.0</td>
<td>21.46</td>
<td>28.08</td>
<td>1.5773</td>
</tr>
<tr>
<td>2.0</td>
<td>true</td>
<td>0.1</td>
<td>21.46</td>
<td>28.09</td>
<td>1.5759</td>
</tr>
<tr>
<td>1.6</td>
<td>true</td>
<td>0.0</td>
<td>21.26</td>
<td>28.12</td>
<td>1.5412</td>
</tr>
<tr>
<td>1.6</td>
<td>true</td>
<td>0.1</td>
<td>21.24</td>
<td>28.13</td>
<td>1.5397</td>
</tr>
<tr>
<td>2.0</td>
<td>true</td>
<td>0.2</td>
<td>21.24</td>
<td>28.35</td>
<td>1.5127</td>
</tr>
<tr>
<td>1.6</td>
<td>true</td>
<td>0.2</td>
<td>21.26</td>
<td>28.40</td>
<td>1.4739</td>
</tr>
</tbody>
</table>

Table 4: Best $f_{scs}$ configurations

Figure 2: Concept clustering parameter tuning

as macro-averaged recall or F1-score is used.

Our benchmarks consist of other state-of-the-art content feature generation methods: term frequency ($tf$), $tf \cdot idf$, Latent Semantic Analysis (LSA, Deerwester et al. (1990)), Latent Dirichlet Allocation (LDA, Blei et al. (2003)), concatenated power mean word embeddings (P-mean, Rücklé et al. (2018)), and FastText sentence embeddings (Bojanowski et al., 2017). All benchmark methods, parameter settings and dimension sizes are shown in Table 3.

Feature transformation models ($tf$, $tf \cdot idf$, LSA, LDA) are trained on the entire training directory. Additionally, when using such transformation models, words are stemmed to reduce inflectional word forms to their word stem.

5 Results

This section discusses our results from parameter tuning and the subsequent supervised classification task.

5.1 Parameter tuning

5.1.1 Semantic cluster similarity

Parameter combinations have been attempted and results have been retrieved for the used dataset. For each quality measure, the 3 best scores have been retrieved, returning all instances that conform to that score. The results are shown in Table 4.

Table 4 tells us that squaring the semantic similarity score works well. High accuracy scores are retrieved, accounting for the fact that evaluations are made based on maximum similarity scoring instead of a predictive model. As we are able to retrieve six configurations, we note that quality measures tend to score parameter settings similarly. The configuration with $k_1 = 2.0$, $sem_{sq} = true$ and $sem_{th} = 0.0$ across quality measures is the best performing.

5.1.2 Concept clustering

Next, we study the influence of parameter $k$ on concept cluster construction. For each setting, we present the accuracy scores and dimension sizes in Figure 2.

With increasing $k$, more concept clusters of unused single terms are added. As a result, the dimension size steadily increases. We can see that accuracy also shows an increasing trend with the value of $k$. When more unused single terms are added, we run the risk of overfitting. The ratio of single terms to broader concept clusters increases with $k$, thereby relatively shifting the focus from broader concepts to frequently occurring distinctive individual terms. This behaviour is also reflected in the graph, in the relative performance difference between unsupervised and supervised scoring. Although supervised accuracy is undoubtedly higher with lower number of concept clusters, both accuracy scores converge with increasing $k$. By adding unused single terms, the unsupervised ranking method is able to capture an increasing number of edge cases, leading to a convergence in performance. This behaviour is an indication of overfitting.

After carefully considering the points above, we proceed with value $k = 20$. This choice appears to offer a good compromise between generalizing ability and direct performance gains. The primary
<table>
<thead>
<tr>
<th>Methods</th>
<th>train acc</th>
<th>train sd</th>
<th>test acc</th>
<th>test sd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benchmark</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf · idf</td>
<td>43.93</td>
<td>0.89</td>
<td>82.79</td>
<td>0.07</td>
</tr>
<tr>
<td>tf</td>
<td><strong>44.85</strong></td>
<td>0.67</td>
<td>84.84</td>
<td>0.12</td>
</tr>
<tr>
<td>LSA</td>
<td>7.89</td>
<td>0.35</td>
<td>9.16</td>
<td>0.20</td>
</tr>
<tr>
<td>LDA</td>
<td>11.78</td>
<td>0.37</td>
<td>14.15</td>
<td>0.15</td>
</tr>
<tr>
<td>TBSeC</td>
<td>36.03</td>
<td>0.60</td>
<td>46.96</td>
<td>0.26</td>
</tr>
<tr>
<td>FastText</td>
<td>32.62</td>
<td>0.79</td>
<td>41.26</td>
<td>0.24</td>
</tr>
<tr>
<td>P-mean</td>
<td>39.78</td>
<td>0.63</td>
<td>90.33</td>
<td>0.10</td>
</tr>
<tr>
<td>Feature combin-</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf &amp; TBSeC</td>
<td>47.83</td>
<td>0.54</td>
<td>86.17</td>
<td>0.13</td>
</tr>
<tr>
<td>tf &amp; FastText</td>
<td>47.07</td>
<td>0.48</td>
<td>85.81</td>
<td>0.14</td>
</tr>
<tr>
<td>tf &amp; TBSeC</td>
<td><strong>47.86</strong></td>
<td>0.42</td>
<td>86.38</td>
<td>0.10</td>
</tr>
</tbody>
</table>
& FastText

Table 5: Invoice classification results

The goal of TBSeC is relating input to broader concepts, which is why a relatively moderate value for $k$ is preferred.

### 5.2 Invoice classification

In this section, we discuss the results of our method against state-of-the-art benchmark methods in a supervised invoice classification task. The results, consisting of cross validated accuracy scores with standard deviation (sd), are shown in Table 5 under header ‘Benchmark’. The fact that term frequency has comparable performance to tf · idf reinforces the notion that all terms within an invoice description are important to take into account and that term-specific weighting has limited value. Moreover, techniques that are concerned with dimensionality reduction (LSA, LDA) perform worse, arguably because they truncate a large amount of information, most of which should have been retained. Sentence embeddings and our method TBSeC perform relatively well, with accuracy scores nearing performance levels of tf · idf and term frequency. Furthermore, a large feature space (1500D) that is achieved with P-mean embeddings appears to have a positive influence on the amount of information that is contained. It is also found that methods tf · idf, term frequency and P-mean have a tendency to overfit on the data, having cross-validation training accuracy scores of over 80%. In comparison, TBSeC and FastText have training accuracy scores closer to test accuracy scores.

Combinations of techniques are attempted next to improve performance. The feature combinations are formed by concatenating the feature spaces of each method. The most successful combinations are highlighted in Table 5 under header ‘Feature combinations’. Combining feature generation techniques leads to surprising results. P-mean performs well as sole feature, but doesn’t yield better results when combined with other techniques. In contrast, FastText does pair well with other techniques and improves performance levels. Moreover, when TBSeC is used performance is even better. Term frequency in combination with both TBSeC and FastText does also improve performance, although slightly. TBSeC and FastText are more likely to be complementary as soon as TBSeC encompasses lower number of concept clusters, but this doesn’t guarantee better overall performance. In the current configuration it manages to perform better, but the difference is not major. The tendency to overfit for tf · idf, term frequency and P-mean is a likely cause for some of the other unsuccessful feature combinations.

### 6 Conclusion

A new feature generation framework TBSeC was presented that is suited to the prediction of well defined classes on the basis of very short texts (2.8 words on average). Generated features were proven to be able to function well independently and jointly with traditional feature generation techniques. Performance and reliability was improved by pairing multiple disjoint feature generation techniques, including TBSeC. A combination of highly specific features with more flexible ones was found to lead to the best results. Combinations of features were found to reach a bound in effectiveness, highlighting that methods ultimately start impeding each other. Businesses can use our method to derive actionable insights from online user generated content such as firm-specific tweets, online reviews and customer chats logs. Future work could test TBSeC on larger texts, since it offers more room to differentiate from sentence embeddings.

References


Izzat Alsmadi and Ikdam Alhami. 2015. Clustering and classification of email contents. Journal of King
Saud University-Computer and Information Sciences


