High-Precision Phrase-Based Document Classification on a Modern Scale

Ron Bekkerman
LinkedIn Corporation
rbekkerman@linkedin.com

Matan Gavish
Stanford University
gavish@stanford.edu

ABSTRACT

We present a document classification system that employs lazy learning from labeled phrases, and argue that the system can be highly effective whenever the following property holds: most of information on document labels is captured in phrases. We call this property *near sufficiency*. Our research contribution is twofold: (a) we quantify the near sufficiency property using the Information Bottleneck principle and show that it is easy to check on a given dataset; (b) we reveal that in all practical cases—from small-scale to very large-scale—manual labeling of phrases is feasible: the natural language constrains the number of common phrases composed of a vocabulary to grow *linearly* with the size of the vocabulary. Both these contributions provide firm foundation to applicability of the phrase-based classification (PBC) framework to a variety of large-scale tasks. We deployed the PBC system on the task of *job title classification*, as a part of LinkedIn's data standardization effort. The system significantly outperforms its predecessor both in terms of precision and coverage. It is currently being used in LinkedIn's ad targeting product, and more applications are being developed. We argue that PBC excels in high explainability of the classification results, as well as in low development and low maintenance costs. We benchmark PBC against existing high-precision document classification algorithms and conclude that it is most useful in *multilabel* classification.

General Terms

Text classification

Keywords

multilabel classification, high-precision classification, natural language theory

1. INTRODUCTION

Automatic classification of text documents plays a preeminent role in numerous industrial and academic data mining applications. Text classification has been explored for two decades at least [7], and machine learning techniques have proved successful for a variety of text classification scenarios [10, 14]. There are four possible text classification setups:

- ***Eager learning* from labeled documents**—the most common setup, in which a generic machine learning classifier is trained on a set of labeled documents.
- ***Lazy learning*** [1] **from labeled documents**—a case-based classifier (e.g. k-nearest neighbors) is build based on labeled documents. This setup is fairly rare in the text domain as it is highly inefficient in many practical cases.
- **Eager learning from labeled features**—usually applied in conjunction with learning from labeled documents (see, e.g. [6]).
- **Lazy learning from labeled features**—an ensemble of decision-stump-like classifiers is constructed, each of which is triggered if a document contains a certain feature, such as a word or a phrase (i.e. an $n$gram of words). This setup is in the focus of our paper—we term it *Phrase-Based Classification (PBC)*. Variations of PBC have been studied for years in Information Retrieval (see, e.g. [8]). PBC is very natural for *multilabel* text classification (when each document may belong to a number of classes), as a number of decision stumps may be triggered for a document. Another advantage of PBC is in *explainability* of classification results, which is an important feature of commercial text classification systems: the fact that a document d is categorized into a class c can be easily explained by the fact that d contains a phrase t that was manually categorized into c.

Obviously, PBC is not the ultimate solution for every text classification task. In this paper, we characterize the class of tasks in which PBC is set for success. We introduce the property of *near-sufficiency*: the information about a document's class label is mostly concentrated in a few $n$grams (usually noun phrases) appearing in the document. Intuitively, the near-sufficiency property holds if the categorized documents are short (such as news headlines, helpdesk requests, Twitter tweets, Quora questions, LinkedIn user profile summaries, etc)—just because short documents have little content beyond a few phrases. Still, longer texts can hold this property as well.[1] In Section 2 we formalize this property in terms of the Information Bottleneck principle [12].

Despite that text classification in general, and PBC in particular, have been around for a while, the field has lately

WOODSTOCK '97 El Paso, Texas USA

[1] A (rather extreme) example would be to categorize pieces of code according to the programming language this code was written in. Regardless of the code's length, just a few keywords may be enough to reveal the class label accurately.

enjoyed a fresh breeze of change, caused by two factors:

• **A quantum leap in the data scale.** The social media boom brings huge amounts of textual content on the daily basis. Blogs, reviews, news feeds, and status updates (such as tweets) are often publicly available and ready for mining. Analyzing hundreds of millions or even billions text items is the modern reality. However, their vast majority is unlabeled and thus intensive manual labeling is often necessary.

• **Availability of cheap editorial resources (Crowdsourcing).** Companies such as Amazon (mturk.com), Samasource, CrowdFlower, and others provide means for quickly labeling large-scale data. Crowdsourcing can also be used for evaluating classification results. The cost of labeling a data instance can be as low as a fraction of a cent. Apparently though, the quality of crowdsourcing results is in many cases quite low. Some workers are underqualified, some simply cannot stay in focus for long, and therefore their output is often noisy and inconsistent. This causes some frustration among researchers who apply crowdsourcing to text classification (see, e.g. [13]). Nevertheless, manual labeling of hundreds of thousands data instances (which was impractical just a few year ago) is now perfectly practical.

We introduce Phrase-Based Multilabel Classification as a process consisting of the following steps: (a) given a dataset D and a set of classes C, construct a *Controlled Vocabulary* V of words relevant for categorizing D into C; (b) from the dataset D, extract a set T of frequently used phrases that are composed out of words from V; (c) categorize phrases T into C via crowdsourcing; (d) map each document d onto a subset of phrases from T and assign d into a subset of classes based on classification of the phrases. In Section 4 we describe the phrase-based classification in greater details.

Crowdsourcing classification of phrases has two important advantages. First, humans are very good at categorizing short pieces of text. If properly trained, human annotators can perform the task accurately and fairly quickly—mostly based on their common knowledge. In contrast, to create a common knowledge base in an automatic classification system is notoriously difficult (see, e.g. [9]). Second, humans are much better than machines at multilabel classification. Questions such as *"How many classes does a data instance belong to?"* or *"Should this data instance be assigned to any class?"* are usually easy for a human annotator.

We see three potential drawbacks of crowdsourcing classification of phrases. First, crowdsourcing classification would never be consistent—we apply quite a heavy-lift consistency check on the crowdsourcing results (see Section 4). Second, human annotators cannot cope with too many classes.[2] Third, the number of phrases to be labeled might be overwhelmingly large and infeasible for human annotation, even in a crowdsourcing setup.

In Section 3, we reveal that crowdsourcing classification of phrases is actually feasible in all practical situations: the natural language imposes a (very restrictive) upper bound on the effective number of phrases which can possibly be composed out of a controlled vocabulary. Based on a language model estimated over a huge text collection, we discover that the effective number of phrases grows linearly—rather than exponentially—in the vocabulary size. To the best of our knowledge, this is an observation that has not been made before. It allows extremely high-precision classification of multimillion document collections that obey the near-sufficiency property.

We implemented phrase-based classification at LinkedIn on the task of *Job Title Classification*. The task is to categorize job titles of LinkedIn users into a couple of dozen classes according to a job function performed by those titles' holders. In Section 5 we describe the developed system—it dramatically increased precision and coverage of the previous version of job title classification. Our offline evaluation shows that we achieve about 95% classification precision. In Section 6 we compare our phrase-based classification with Support Vector Machine (SVM) classification [4], and show that our method significantly outperforms four SVM versions on the multilabel job title classification task.

Our system has been successfully deployed in LinkedIn's ad targeting product.[3] Increasing the coverage resulted in increasing the user profile inventory available for targeting. Since LinkedIn is in the pre-IPO period, we cannot comment on the direct revenue impact of our system.

Our contributions can be summarized as follows. First, we formalize the near-sufficiency property of the data that guarantees high-precision phrase-based classification (PBC). Second, we show that PBC will always be feasible, and not shy of dealing with multimillion and potentially billion data instance collections. Last, we present a working use case of phrase-based classification on LinkedIn data, and provide insights on building a successful PBC system.

2. PHRASE-BASED CLASSIFICATION

2.1 Problem Setup

The problem of multilabel text classification is defined as follows. Each document from an unlabeled corpus D is to be categorized into one or more classes from a class set C. More formally, a rule $L : D \to 2^C$ is to be learned from training data. Also available is a labeled test set $D_{test} = \{(d_i, C_i^*)\}$, where $|D_{test}| \ll |D|$ and each $C_i^* \subset C$ is a set of d_i's ground truth classes. Performance of the classification rule L is evaluated on the test set by comparing each $L(d_i)$ with C_i^*.

In classic text categorization, the rule L is learned given a training set D_{train} of documents and their labels. However, the categorization framework does not restrict the training set to contain only documents. This paper deals with *phrase-based classification (PBC)*—a text classification framework in which the training data consists of labeled *phrases*, or $n$grams, extracted from the (unlabeled) documents.[4]

2.2 The Property of Near Sufficiency

It is sometimes possible to identify a collection of words (which we term *Controlled Vocabulary*, V), such that the collection of $n$grams composed from V is a feature set that contains enough information about the document labels to allow high-precision classification using these features alone. In this section, we formalize and quantify this property.

Let T denote the set of all $n$grams ($n = 1, 2, \ldots$) from a controlled vocabulary V that are commonly used in a document collection D. In large-scale corpora, $|T| \ll |D|$ for any reasonable choice of V, that is, the size of the phrase feature set is significantly smaller than the number of documents.

[2]However, if the classification is done for a user-facing product, too many classes are not necessary because the end users will not be able to cope with them either.

[3]`http://www.linkedin.com/advertising`

[4]We will be using terms *phrase* and *ngram* interchangeably.

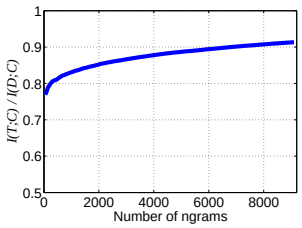

Figure 1: $I(T;C)$ as a percentage of $I(D;C)$ in the LinkedIn test data (see Section 5) plotted over the number of $n$grams in T. The $n$grams are sorted in the descending order of their frequency.

In Section 3 we show that $|T|$ is always small enough to be feasible for manual classification.

In order to measure the extend to which a given collection of phrases T is *sufficient* for classification (i.e. the extend to which we can replace any document by a set of extracted phrases without significantly increasing misclassification rate), we appeal to the Information Bottleneck principle [12]. See [11] for a recent survey of this approach. Information Bottleneck considers an input variable X and an output variable Y. (In our case, the document variable is input and the class variable is output). The value of Y is to be estimated given an observation X. In order to compress the variable X to a simpler variable, or a variable with fewer levels, a variable T is sought (in our case, the *phrase* variable) which achieves an optimal tradeoff between sufficiency for Y (attributed by a high value of the *mutual information* $I(T;Y)$) and simplicity of representation (attributed by a low value of $I(X;T)$). The Data-Processing Inequality [5] states that $I(X;Y) \geq I(T;Y)$ so that the compressed variable T cannot hold more information on Y than the original variable X. As shown in [11], whenever the $I(X;Y) = I(T;Y)$ equality holds, T is a *sufficient statistic* for Y. In other words, all the information for estimating Y is in the reduced variable T.[5] However, we do not expect the equality to hold in many practical cases.

Definition 1. *Let D with $L : D \to 2^C$ be a labeled document corpus, and V a controlled vocabulary. We say that the phrases T constructed from V are* ϵ-nearly-sufficient *for L, if $I(T;C) \geq (1-\epsilon)I(D;C)$ for some $\epsilon > 0$.*

To measure *near-sufficiency* of a candidate phrase collection T, we need to define a joint probability distribution based on the labeled corpus $D_{test} = (D, C)$ and the control vocabulary V. Put the uniform distribution on D. Let $\mathbf{DC}$ be the document-class matrix where $\mathbf{DC}_{i,j} = 1$ if document d_i belongs to class c_j and 0 otherwise. Let $\widetilde{\mathbf{DC}}$ be the row-normalized version of $\mathbf{DC}$. We think about $\widetilde{\mathbf{DC}}$ as an empirical conditional distribution of labels given document. Now let $\mathbf{DT}$ be the document-phrase matrix with $|T|$ columns and let $\widetilde{\mathbf{DT}}$ be its row-normalized version. Consider the joint distribution $p(D, C, T)$. It is easy to check that the marginal joint distribution $p(D, C)$ documents-classes is given by $\widetilde{\mathbf{DC}}/|D|$ and that the marginal joint distribution $p(T, C)$ of phrases-classes is given by $\widetilde{\mathbf{DT}}^\top \widetilde{\mathbf{DC}}/|D|$. This allows us to compute the mutual information $I(D;C)$ between the "original" documents variable and the class variable, and the mutual information $I(T;C)$ between the "compressed" variable (i.e. phrases) and the class variable. The closer $I(T;C)$ is to $I(D,C)$, the more sufficient T is for classification, prompting the use of our phrase-based classification method. Note that this criterion is easy to check in practice.

The test set of the LinkedIn data (see Section 5) holds the near-sufficiency property, in the sense that $\frac{I(T;C)}{I(D;C)} > 0.9$ for a labeled data set of size 19000, controlled vocabulary of size 1485, and the set of phrases of size 9111 (see Figure 1). It is important to note that in the LinkedIn dataset, using the controlled vocabulary itself (i.e. unigrams) is not enough for a successful classification. Indeed, $\frac{I(V;C)}{I(D;C)} = 0.53$, which suggests that most of the information is concentrated in phrases T rather than in single words V. This implies that we cannot avoid the tedious process of labeling phrases, where unfortunately $|T| \gg |V|$. Luckily, labeling phrases is feasible, which we prove in Section 3 below.

3. FEASIBILITY OF PHRASE LABELING

Since our classification framework is heavily dependent on the manual process, we need to make sure that the process will be manageable in a generic case: the number of $n$grams to annotate will not exceed human capabilities. We are only interested in annotating those $n$grams that are *frequent enough* in the language—otherwise we will lose generality of our framework.[6] Despite that the combinatorial number of all possible $n$grams made of words from a vocabulary V is growing exponentially with n (the length of $n$gram), it is quite obvious that the language will not allow most of them to be frequently used. Let us show that the number of *frequent enough* $n$grams is reasonably small in all practical settings. First, let us explain what we mean by "frequent enough" $n$grams.

Definition 2. *We define the* frequency of a set X *to be the frequency of its least frequent item:* $F(X) = \min_{x \in X} F(x)$.

We say that a set T consists of frequent enough $n$grams if $F(T) \geq F(V)$, i.e. the set of $n$grams is as frequent as the vocabulary out of which those $n$grams were composed. Here we solely aim at conditioning the frequency of T on the frequency of the controlled vocabulary. We might have imposed a different condition, e.g. $F(T) \geq \frac{F(V)}{\alpha}$ where $\alpha > 1$. This would lead to similar results but make our reasoning more complicated after introducing an extra parameter α.

To show that the set T of frequent enough $n$grams is feasibly small, we make use of the Web 1T dataset.[7] The Web 1T data was released by Google in 2006. It consists of frequency counts of unigrams, bigrams, trigrams, fourgrams, and fivegrams, calculated over a dataset of *one trillion* tokens. The data is large enough to be a good model of a natural lan-

[5] In fact, in the unilabel classification scenario, Shamir et al. [11] prove that the misclassification rate of the maximum conditional likelihood rule using the reduced variable T instead of the full variable D decreases *exponentially* with the number of labeled points—with $I(T;Y)$ in the exponent.

[6] Obviously, we can artificially limit the number of $n$grams to be labeled. For example, we can take a list of only ten thousand most frequent $n$grams. This will hurt the coverage though, as many documents would not be mapped onto any $n$gram from the list.

[7] `http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2006T13`

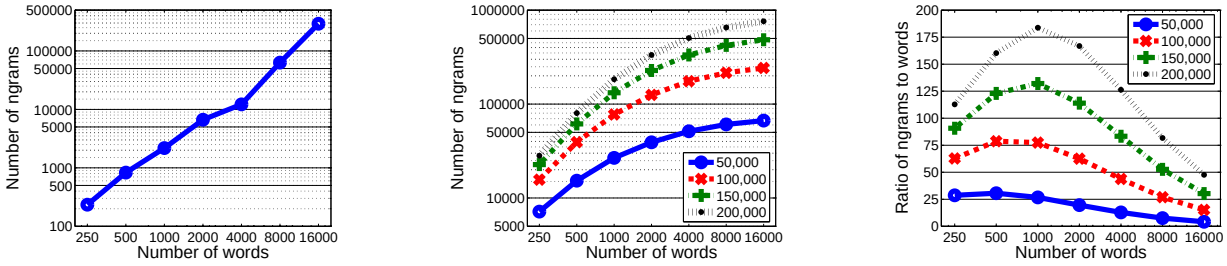

Figure 2: (left) $|T^*|$ as a function of $|V^*|$; (mid) $|T^{**}|$ as a function of $|V^{**}|$; (right) $\frac{|T^{**}|}{|V^{**}|}$ as a function of $|V^{**}|$.

guage.[8] The unigram portion of the Web 1T data consists of about 7.3M words (we lowercased letters and ignored punctuation), with their frequency counts in a range from about $2 \cdot 10^{10}$ down to $2 \cdot 10^2$. Needless to say, most of these words are not in our lexicons. In fact, only the upper 50K or so contain words that are in our everyday use.

We preprocessed the data by first omitting $n$grams that appeared less than 1000 times in the Web 1T data, removing stopwords and alphabetically sorting words in each $n$gram (which leads to ignoring the words' order). For example, given a fivegram *"of words from the dictionary"*, we removed stopwords *of*, *from*, and *the*—and then alphabetically sorted the remaining words into a bigram *"dictionary words"*. We ended up with about 2.5M (unique) unigrams, 13M bigrams, 10M trigrams, 4M fourgrams, and 1.4M fivegrams.

To establish the upper bound on the number of frequent enough $n$grams, we first ask the question what would be a vocabulary of size k out of which the *maximal* number of $n$grams can be composed. Obviously, this is an NP-hard problem, as we would need to construct $\binom{N_1}{k}$ vocabularies, where N_1 is the number of unigrams in our data. We apply a greedy approximation. Let $T = U(V)$ be the set of $n$grams composed out of a vocabulary V, where U is an $n$gram construction function, provided to us by the natural language. Let $V = U^{-1}(T)$ be the vocabulary out of which the set of $n$grams T is composed. First, we take a set T^* of all the most frequent $n$grams in the Web 1T data, such that $|V^*| = |U^{-1}(T)| = k$, i.e. the underlying vocabulary V^* is of size k. Second, we take a set $\widehat{T}^* = U(V^*)$ of all $n$grams composed of V^*, which is as frequent as V^* itself, i.e. $F(\widehat{T}^*) \geq F(V^*)$. This way, we assure that $T^* \subseteq \widehat{T}^*$. The fact that $n$gram frequencies follow a power law—and that $\widehat{T}^*$ contains *all* the most frequent $n$grams—guarantees high quality of our approximation.

In Figure 2 (left), we plot $|\widehat{T}^*|$ as a function of $|V^*|$. As we can see on the plot, $|\widehat{T}^*| = |V^*|^\beta$, where the exponent β is very small: $1 \leq \beta < 1.3$. This leads us to the conclusion that the language will only allow a reasonably small number of frequent enough $n$grams, to be composed out of vocabularies of reasonably small sizes.

The procedure for creating a vocabulary V^* described above is artificial, and quite unrealistic in practice. While this does not matter for establishing the upper bound, we can tighten it if we apply a more natural scenario. We start with an observation that controlled vocabularies are usually built of words that come from the same topic (or a few topics). We simulate the creation of a topic-oriented controlled vocabulary V^{**} in the following process: we first sample a word w_1 from a multinomial distribution defined over the set V_m of m most frequent words in the Web 1T data. The word w_1 plays the role of a seed for the topic of V^{**}. Then we add a second word $w_2 \in V_m$ that is most co-occurring with w_1, and we add a third word $w_3 \in V_m$ that is most co-occurring with $\{w_1, w_2\}$, and so on until we populate our vocabulary V^{**} with k most co-occurring words.[9] We then build the $n$grams $T^{**} = U(V^{**})$, such that $F(T^{**}) \geq F(V_m)$—the $n$grams are as frequent as the space of words from which V^{**} was built (note that $F(V_m) \leq F(V^{**})$). To overcome the non-determinism of seeding the topic, we repeat the process 10 times and report on the mean and standard error.[10]

We probe various sizes k of the controlled vocabulary, from 250 to 16,000 words. This spans over arguably the entire spectrum of possibilities, as a set of 250 words appears to be too small for a high-quality controlled vocabulary, while a set of 16,000 is larger than the entire lexicon of an average language speaker. We also probe different values of m—the number of most frequent words from which V^{**} is built. We tried four values of m: 50K, 100K, 150K, and 200K. We argue that the utility of using $m > 200$K is very low, as words in the tail of that list are completely obscure, and therefore useless from the generalization point of view.

Our results are shown in Figure 2 (middle): we plot $|T^{**}|$ over $|V^{**}|$ and see that it looks curvy on the log-log scale, which may imply that the number of $n$grams is no longer super-linear in the number of underlying words. To verify this claim, we expose an additional view of the same data: we plot $\gamma = \frac{|T^{**}|}{|V^{**}|}$ over $k = |V^{**}|$, and show that $\gamma < 200$ for all realistic values of k and m, which leads to the following formal result:

COROLLARY 3. *The natural language imposes a linear dependency between the number of words in a controlled vocabulary V and the number of frequent enough $n$grams composed out of words from V.*

We observe an interesting saturation effect on the right plot: for large vocabularies ($k > 1000$), the γ ratio decreases. We can explain this as follows: high-frequency words are well represented in vocabularies of any size. Low-frequency words, however, are mostly represented in large vocabularies. Ngrams that contain lower-frequency words are even

[8] In our future work, we plan to utilize the newly released Books $n$gram dataset `http://ngrams.googlelabs.com`.

[9] The fact that the words in V^{**} highly co-occur with each other assures that V^{**} is built around a particular topic—which is more realistic than in our previous setup. The fact that the words co-occur *as much as possible* allows us to establish the *upper bound* on the number of $n$grams composed out of V^{**}.

[10] The standard error is too small to be visible on our plots.

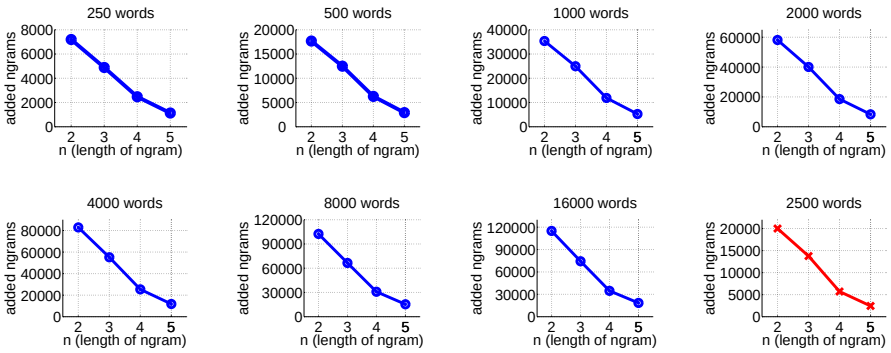

Figure 3: Additions to the set of $n$grams T, as obtained from portions of the Web 1T data, sorted from bigrams to fivegrams. The last plot is for the real-world controlled vocabulary.

less frequent—they fall below our threshold, which leads to fewer $n$grams in T^{**} per word in V^{**}.

One could argue that since the Web 1T data does not contain frequencies of high-order $n$grams (above fivegrams), then the data is not complete and no conclusion can be made on the overall number of frequent enough $n$grams composed out of words in V. While the data is not complete indeed, we are confident that it contains enough information to let us reason about the total size of T. Figure 3 justifies our claim. The figure shows the number of $n$grams added after every processing step: first, after scanning Web 1T bigrams, then after scanning trigrams, then fourgrams and fivegrams. As we see from the plots, the number of $n$grams to be added goes down dramatically as the $n$grams' order goes up. It can be easily explained by the fact that the probability of an $n$gram to be frequent enough decreases exponentially with the $n$gram's length. As the number of added $n$grams is already small at the fivegram landmark, we do not anticipate a significant addition to the total size of T that would come from higher-order $n$grams.

The first seven of the eight plots in Figure 3 show the $n$gram addition data over the same values of k as in Figure 2. The last, eighth plot shows an example of a real-world V of size 2500—the one described in Section 5—with the frequency constraint $F(U(V)) \geq F(V)$. While the curve behaves very similarly to the previous seven curves, the overall number of $n$grams is about 3–4 times lower, as compared to the artificial setups with $k = 2000$ and $k = 4000$. This implies that our upper bound is actually not very tight.

4. ARCHITECTURE OF PHRASE-BASED CLASSIFICATION SYSTEM

In this section we propose a system for phrase-based classification, which achieves very high precision (see Section 5) and outperforms popular classification methods (Section 6). The system is composed of two modules: offline and online. The offline module aims at constructing and labeling phrases which are then used in the online module for categorizing documents. Clearly, the online module is optimized for throughput such that most of the heavy data processing is done in the offline module.

4.1 Offline Process of Phrase Classification

The schematic view of the phrase classification module is shown in Figure 4. Let us walk over the module's components one-by-one.

Agree upon the Taxonomy of Classes C. Each classification system that is built in response to a particular business need in a commercial organization starts with defining the taxonomy of classes. Any text corpus can be categorized into a variety of class taxonomies, however not all of them fit well the data at hand. If our data consists of, say, people's job titles, but our classes are models of Japanese cars, then we will fail to meet the system's coverage requirement. A good taxonomy is constructed in a negotiation between business and technology groups within the organization. Note that the success of this step as well as most of the next steps could not be achieved without the involvement of a *Content Manager*—a domain expert who is in charge of keeping the data organized and presented in the best possible way.

The next step in the process is to **Create a Controlled Vocabulary** V of words that are supposed to characterize classes in the taxonomy. Controlled vocabularies are routinely used in computational linguistics to investigate particular language patterns, as well as in library science to organize and index knowledge. In our case, the creation of a controlled vocabulary is a semi-manual feature selection process where, at first, words in the dataset are sorted in the descending order of their frequency, and then the content manager scans through the sorted list and uses her domain expertise to decide whether a word can be included or not. Tools can be built to improve productivity of this work, given a particular task at hand.

Once we agreed on the content of the controlled vocabulary, the next step is to **Build Phrases** T. We are only interested in $n$grams of words from V that frequently reappear in the data (rare phrases are useless from the generalization point of view). We ignore the order of words in phrases, so that, for example, *text classification* and *classification (of) text* will be considered the same phrase.[11] In Section 3 we proved that the language will not let the set T be too large for manual labeling. The process of building the set T can be fully automatic: (a) extract all phrase candidates from the data D; (b) sort them by frequency; (c) cut off the least frequent ones. We have to take care of the following issues:

• **Filter out compound phrases.** We can use NLP techniques (such as *shallow parsing*), to identify phrases that are frequent enough, but are clearly composed of two (or

[11]One may argue that in some cases the order of words is actually crucial, e.g. *Paris Hilton* and *Hilton, Paris* are not the same. However, those cases are rare from the statistical point of view, and therefore for all practical purposes words' ordering can be ignored in text classification (see, e.g. [2]).

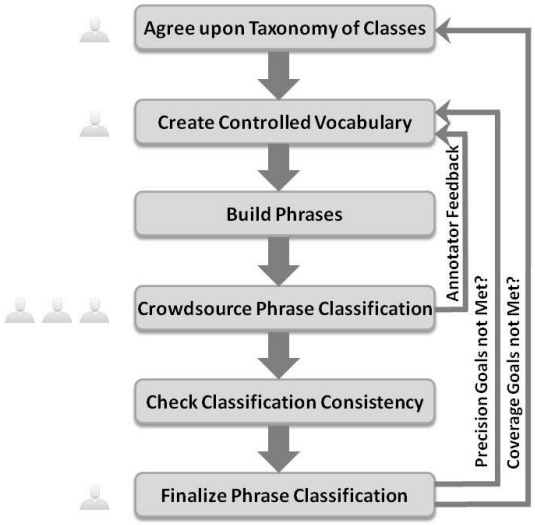

Figure 4: Phrase Classification. A single human icon means involvement of the content manager; multiple human icons mean crowdsourcing.

more) stand-alone phrases. For example, a phrase *"Machine Learning or Information Retrieval"* can be frequent enough, but keeping *machine learning* and *information retrieval* in one phrase might distract our annotators, who may make an arbitrary decision whether the phrase belong to one class, two classes, or none.

• **Filter out too specific phrases.** Some phrases, such as, e.g. *"Content-Based Collaborative Filtering"*, can be frequent enough in the data but too specific for an annotator to understand their meaning. It makes more sense to remove such phrases given that a more general phrase (i.e. *collaborative filtering*, in this example) is in T.

• **Filter out near duplicate phrases.** Some phrases, such as *"Recommendation System"* and *"Recommender System"*, are practically identical—not only in their form but also in their meaning. We may use *Levenshtein distance* to identify those pairs and then reason whether one of them can be mapped on the other. Some help from the content manager may be needed with this task.

Now, when we have a feasible number of phrases, we can **Crowdsource Phrase Classification**. Crowdsourcing sounds simple but it is actually not. A straightforward approach would be to submit the task to a crowdsourcing tool, asking the workers to assign each phrase[12] into zero, one, or more classes from the predefined set C. If priced properly, the task will be done quickly but chances are strong that the results will be unsatisfactory (in some cases, practically random). The voting mechanism (i.e. let a number of workers annotate each phrase and then take the majority vote) does not seem to help much as the mode sometimes emerges from a seemingly uniform distribution. The mechanism of *qualification tests* (which a worker needs to pass in order to take the task) is helpful though. The best results are achieved when (a) trust is established between the requester and every worker; and (b) every worker is well educated about the task to be performed.

[12]Despite that we do not preserve words' ordering in the phrases, we need to present them to annotators in the form that is most commonly used in our data.

As a byproduct of the classification process, workers can detect nonsensical phrases, as well as undesired words in the controlled vocabulary. Their feedback can be plugged back into the system and necessary changes can be made.

Even when the workers are well trained for the task, the results are likely to have two issues: *systematic errors* and *consistency errors*. A systematic error occurs when the majority of workers make the same incorrect decision. Systematic errors are hard to discover and fix. One way to reduce the systematic error rate is to choose workers of different backgrounds and geographic locations, and use the voting mechanism. Fortunately, systematic errors are not very common.

Consistency errors are more common, but also easier to tackle. A consistency error occurs when one choice is made for an item, and a different choice for a similar item. Consistency errors occur even when the voting mechanism is used. When the consensus of voters is weak, it is often enough for a single worker to make an inconsistent choice, in order to introduce a consistency error. To dramatically reduce the consistency error rate, it is desirable for each worker to perform the *entire* task. This increases the systematic error rate, but—again—those are quite rare. Even when each worker performs the entire task, consistency errors are common, primarily because people do not remember all the choices they have made. Another approach to reduce the consistency error rate is to present the workers with sets of similar items. Even then, the results are rather inconsistent.

This observation leads us to the next step in the pipeline: **Check Classification Consistency**. We represent each phrase t as a feature vector over the aggregated Bag-Of-Words of all documents mapped onto t. This representation is supposed to be dense enough for machine learning to be applied. If it is still too sparse, we can make use of additional data, such as, e.g., documents hyperlinked from our dataset D (e.g. shared links in tweets, etc).

Once represented as a feature vector, each phrase passes a consistency check, which utilizes a simple k-nearest neighbors (kNN) model: each phrase t_i is assigned to a class c_i^{knn} based on the class majority vote of its neighbors in the vector space. Whenever $c_i^{knn} \in C_i$, where C_i is a set of classes obtained for t_i via the manual classification, we consider the classification to be consistent. A more interesting case is when $c_i^{knn} \notin C_i$. Crowdsourcing the task of deciding between c_i^{knn} and C_i is unlikely to produce good results, as people are bad at choosing between two legitimate options.[13]

We use an SVM model as the judge. We train it on the consistently categorized phrases and apply it to the unresolved cases. Since SVM model is fundamentally different from the kNN model (one is eager learning, another is lazy learning), we believe that the two models are unlikely to produce a systematic error. If the SVM judgement c_i^{svm} falls within $\{c_i^{knn}\} \cup C_i$, we go with the SVM result. If not, we declare the case too difficult and leave t_i uncategorized.

By this, we approach the last step of the pipeline: **Finalize Phrase Classification**. Even after such an elaborated error detection process, some problems may remain in the phrase classification results. It turns out however that spot-checking the existing classification results is a much easier task for a human than coming up with the original classification. The content manager can do a good job of fixing

[13]We created a test of 20 such cases and asked over 200 workers to take it—none of them passed a 75% accuracy barrier.

last issues by simply scanning through the created phrase classification. Of course, not all the issues can be detected this way. However, this process can go on long after the classification system is deployed. Given that the phrase classification task is by no means large-scale, it has the potential of eventually converging to the 100% precision.

4.2 Online Process of Document Classification

The online part of the PBC system is to map each document to a subset of phrases from T, and then the document classes are obtained by a simple look-up of classes of the corresponding phrases. Since the set of phrases T was created out of most common phrases in the data D, many documents from D will be mapped onto at least one phrase. The mapping process is fully automatic, and fairly similar to the process of creating T. For each document d_i, we (a) extract controlled vocabulary words $V_i = V \cap d_i$; (b) given V_i, construct all possible phrases $T_i \subset T$; and (c) filter some irrelevant phrases if any. The latter step would also incorporate some shallow NLP, for example, if the document is *"Text Analytics and Image Classification"*, then we must not map it onto the phrase *text classification*. Note that before we start the mapping process, we need to perform some preprocessing, such as taking care of word misspellings and abbreviations—to be mapped onto their full, correct form.

Another important aspect of the document-to-phrases mapping is that we would prefer mapping documents onto the most specific phrases available (filtering in the offline process prevents us from getting too specific). For example, if a document is mapped onto three phrases: *text*, *classification*, and *text classification*, then we would choose the more specific phrase (*text classification*) and filter out the others. This is done because the more specific the phrase is, the more precise decision the annotator can make.

5. JOB TITLE CLASSIFICATION SYSTEM

Our use case for testing the proposed PBC methodology is classification of LinkedIn users' job titles. On their professional profiles, LinkedIn users are free to specify their job titles in any form they wish. For example, we found almost 40,000 different ways in which users specified a title *"Software Engineer"*.[14] Table 1 shows a few very common cases.

A showcase task we describe here is to categorize the job titles into a few dozens of classes according to job functions. Examples of the classes can be *Engineering, Marketing, Sales, Support, Healthcare, Legal, Education*, etc. For the offline module (phrase classification), we used a dataset of about 100M job titles. Obviously, many of the titles repeat in the data, especially those generic titles such as *"Manager"*, *"Director"* etc. After we lowercased all titles and identified duplicates, we created a dataset of unique titles which turned out to be about 5 times smaller than our original dataset. Let us first discuss the offline part of our PBC implementation:

• **Agree upon Taxonomy of Classes.** The set of classes was provided by the business owner which we accepted.

• **Create Controlled Vocabulary.** There are three most important types of words in job titles: (a) job names, such as *engineer*, *president*, *comedian* etc; (b) seniority words, e.g. *senior*, *junior*, *principal*; and (c) function words, such as *sales*, *research*, *financial* etc. Seniority words are the smallest bucket; function words are the largest bucket. Altogether, our controlled vocabulary consists of about 2500 words. Also, we created a translation look-up table, where abbreviations, common misspellings, and foreign language words were translated into the controlled vocabulary words.

Note that classification of foreign titles is quite straightforward because our domain is very narrow: simple word-by-word translation usually creates a meaningful English phrase that annotators can successfully categorize. Two corner cases that we needed to take care of are translation of one word into multiple words (typical in German), and translation of multiple words into one word (typical in French).

• **Build Phrases.** Titles got cleaned by filtering out words that did not belong to our controlled vocabulary, and applying translations. Following our recipe in Section 4, we split compound titles (such as *"Owner/Operator"*), got rid of too specific titles, and deduplicated the resulting list. We then selected a few dozens of thousands most common cleaned titles to be our set of phrases (we called them *standardized titles*). Examples of standardized titles are *software engineer*, *senior software engineer*, *software developer*, *java programmer* etc. We verified that the list of standardized titles was comprehensive enough to cover the spectrum from very common to relatively rare titles. Practically every non-compound title that consisted of the controlled vocabulary words and appeared in our dataset more than a couple hundred times was added to the standardized title list.

• **Crowdsource Phrase Classification.** Classification of the standardized titles was the pivotal step in the project. It was critical for us to perform the job with the highest possible precision, so we decided not to outsource it. Instead, we chose the annotators to be LinkedIn employees. To reduce a possible systematic error, we needed to hire annotators of diversified backgrounds. Together with that, we had to minimize the effort of educating and training our annotators, so we organized them in groups. We created two groups of annotators in two separate geographic regions, and asked each annotator to label the entire set of standardized titles. After the task was completed, we applied the voting mechanism and found that about 15% of the standardized titles could not gain the majority vote (i.e. no class was chosen for those titles by the majority of annotators). We then performed two rounds of disambiguation and finally came to a consensus classification of every standardized title in the list.

• **Check Classification Consistency.** As described in Section 4, we applied the kNN model and found out that about 25% of the standardized title classifications were not consistent with the kNN model results. We then applied the SVM model as a judge and resolved most of the inconsistency cases. We left uncategorized the cases when SVM produced a result that differed from both the kNN result and the original manual classification result.

• **Finalize Phrase Classification.** It turned out that the process of finalizing the standardized title classification was very important for the success of the project. Spot-checking revealed a number of systematic errors and consistency errors. After the classification got finalized, the new challenge was to evaluate the standardized title classification results. We could not use our annotators for evaluating their own work. Therefore, we hired a different group of annotators, who evaluated 25% of standardized title classifications and estimated the classification precision at 98%.

[14]We did not include various subgroups of software engineers, nor software developers / programmers, in this count.

software engineer	trainee software engineer	software enginer	software engineer / consultant
software engg	software engineer trainee	software engineer (contractor)	software engineer co-op
sw engineer	sw eng	software engineer (intern)	software enggineer
software eng	software engeneer	software engineer	ingénieur logiciel
s/w engineer	consultant software engineer	software engineer/consultant	s/w engr
software engineer intern	software engr	sw engg	soft engineer
software engineer ii	software engineer i	intern software engineer	soft eng
software enginner	ingeniero de software	software engineeer	software engineer/architect
s/w engg	software engineer iv	soft engg	software engineer/project manager
software engineer iii	software engineer consultant	software engineer (consultant)	sw engr
software engineering	software engineer (contract)	software engineer in test	software engineer/developer
sofware engineer	s/w eng	software engineer 2	engenheiro de software

Table 1: A small sample of common ways to specify a job title "Software Engineer" on LinkedIn user profiles.

The online process of mapping original titles onto the standardized titles is fairly simple. For each original title, we first applied word translations and filtered out words that did not belong to our controlled vocabulary. Out of the remaining words, we composed all possible standardized titles from our list. We then applied three NLP heuristics to get rid of problematic mappings. For example, we avoided mapping a compound title *"Senior Engineer / Project Manager"* onto a standardized title *senior project manager*. We also removed generic mappings if more specific ones were available. For example, if a title was mapped onto both *senior software engineer* and *software engineer*, we removed the latter. The resulting mapping achieved more than 95% coverage on our dataset of about 100M titles.

A challenging task was to evaluate the precision of the mapping. We used one of the commercial crowdsourcing providers for this task. The provider evaluated a large set of mappings, where the evaluation set of original titles was sampled from the original title distribution (such that more common titles had a higher probability to get selected). Given the high level of duplications in our data, the evaluation set covered about 40% of all titles in our dataset. The evaluation estimated the mapping precision at 97%.

Given the precision of standardized title classification (98%), the lower bound on precision of the overall (original job title) classification process is $97\%*98\% = 95\%$. It is a lower bound because an original title can be incorrectly mapped onto a standardized title that happens to belong to the same class as the original title. We succeeded in achieving our precision goals. However, we could not meet our coverage goals which stood at 90%, because for many standardized titles (usually generic ones, such as *"Manager"*) there was no appropriate class in C. Obviously, coverage does not only depend on the system but also on the data and the taxonomy of classes, which makes the coverage goals rather vulnerable. Our system's coverage currently stands at 80%. Higher coverage can be achieved either by introducing new data classes, or by mapping the input data to more specific phrases—those that *can* be categorized.

6. COMPARISON WITH SVM

A legitimate question is how well would the traditional text classification tools cope with the problem of job title classification. As we now have an order of a hundred million job titles categorized with 95% precision, we can use any portion of this data to train a classifier and compare its results with those we obtained with PBC.

We use the SVM classifier for this comparison. Note that SVM is not a strawman—it is one of the best classification models available, particularly well suited for text (see, e.g. [14]). We followed [3] to set up a multilabel SVM framework. We used SVMlight[15] with parameters $c = 0.1$ and $j = 2$ which were chosen based on our prior knowledge.

We train an SVM on a portion of the job title data, then test it on the *entire* data and compare the classes assigned by the SVM with the classes assigned by PBC. If the SVM is within the 90 percentile from PBC, we can say that the document-based classification results are comparable with the phrase-based classification results. However, if the SVM results are farther from those of PBC, we can claim that the SVM does a poor job (given that PBC is 95% accurate).

We propose two quality measures of the classification results: a tolerant and an aggressive:

• **Partial Match.** For each data instance d_i, we choose a *single* class c_i^{svm} that has the maximal confidence score among those obtained by one-against-all binary classifiers. Note that the maximal score can actually be negative. Then we consider d_i to be successfully categorized if $c_i^{svm} \in C_i$ (i.e. if c_i^{svm} is among classes assigned by PBC). In this setup, we apply $k + 1$ binary classifiers, where the extra classifier is for "No Class"—we compose an artificial class of instances that PBC did not assign to any class.

• **Full Match.** For each data instance d_i, we build a set C_i^{svm} of classes that have positive confidence scores. We then consider d_i to be successfully categorized if $C_i^{svm} = C_i$ (i.e. SVM classes and PBC classes are identical). In this setup, we do not need an extra class for "No Class".

We used the standard Precision, Recall, and F1 measures in both setups, as obtained on the entire data D, besides instances that were not covered by PBC. Note that this way the SVM is evaluated on both the training and testing data simultaneously—and it is a rare case when this is *not* a problem: since we start from scratch (no labeled data is initially provided), it does not matter which instances are categorized manually and which automatically, as soon as the results can be fairly compared. And we achieve fair comparison by training the SVM on the *same number* of instances as were manually labeled in the PBC process.

We tested four options for applying SVMs: (a) train on the set of standardized titles T, and test on D while taking into account only features from T, that is, only our controlled vocabulary V; (b) the same, but apply the word translation lookup table, for mapping some extra features of data D onto V; (c) train on the most common titles in D; (d) train on titles sampled from the title distribution over D. The reason for choosing those options was that, in traditional text classification, option (d) would be the default

[15] `http://svmlight.joachims.org/`

Setup	Multiclass SVM trained on:	Partial Match			Full Match		
		Precision	Recall	F-measure	Precision	Recall	F-measure
(a)	*standardized titles*	91.2%	83.6%	87.2%	80.9%	65.5%	72.4%
(b)	*standardized titles with word translations*	94.0%	92.1%	93.0%	82.4%	73.6%	77.8%
(c)	*most frequent titles*	95.4%	88.2%	91.7%	83.0%	73.4%	77.9%
(d)	*randomly chosen titles*	95.1%	88.0%	91.4%	80.7%	73.7%	77.1%

Table 2: SVM results on the job title classification task. SVM results are good on single-class classification (partial match columns), but are poor on multilabel classification (full match columns).

one, option (c) could be plausible, but options (a) and (b) would not be available. Our goal is to see whether the domain knowledge provided in V and the translation table can improve the SVM classification results.

Our findings are summarized in Table 2. The *Partial Match* results are very good, which proves again that the SVM classifier is strong. The *Full Match* results are much poorer. They suggest that SVMs are not able to identify all the classes an instance can belong to, while phrase-based classification copes with this problem very gracefully. Among the four training options, option (a) turned to be the worst one, which can be easily explained by the fact that T (without word translations) had the minimal number of features among the four setups. However, whenever word translations got added (option (b)), this setup became the leader, which suggests that the domain knowledge is helpful for SVM classification. Options (c) and (d) performed remarkably similarly, which was rather predictable, given that sampling from the title distribution (option (d)) would heavily prefer most common titles (option (c)).

7. DISCUSSION

In this paper, we show that phrase-based classification (PBC) can achieve extremely high precision (95%) with reasonable coverage (80%, improvable) on a large-scale text collection of over 100M instances. We characterize the class of data on which PBC can be successful (the data that satisfies the near-sufficiency property), and prove that PBC will be feasible on data of virtually any size, once crowdsourcing is used—the natural language prevents the annotation task from being overwhelmingly large. The development cost of our deployed PBC system is low (2 person years plus annotation expenses), and so is the maintenance cost (which boils down to periodically updating the controlled dictionary as well as the pool of categorized phrases).[16] Overall, we proposed a classification framework that can be directly applied to other tasks, with success guaranteed.

We believe that the main advantage of the deployed system is in the right use of human labor: domain expertise of the content manager combined with common knowledge of the crowd is leveraged in intuitive, single-action tasks such as "choose a word", "categorize a phrase", and "check a phrase/category pair".

We note that crowdsourcing is not a panacea, and is actually a much more complex task than it would have expected. Without a consistency check, the crowdsourcing classification results would be rather poor. Consider our job title classification use case, for which we hired two groups of annotators. Since we now have all final results, we can see how well each group did as compared to the final results. It turns out that one group achieved 73.2% F-measure in the *Full Match* setup (see Section 6), while the other group achieved 76.7% F-measure. Together they achieved 79.3% F-measure. This implies that the two-step consistency check contributed over 20% to the F-measure, bringing success to the entire framework.

[16]Within half a year of no maintenance, the coverage of our job title classification system decreased only by 0.3%, which assures its sustainability.

8. REFERENCES

[1] D. Aha, editor. *Lazy learning*. Kluwer Academic Publishers, 1997.

[2] R. Bekkerman and J. Allan. Using bigrams in text categorization. Technical Report IR-408, CIIR, UMass Amherst, 2004.

[3] R. Bekkerman, R. El-Yaniv, N. Tishby, and Y. Winter. Distributional word clusters vs. words for text categorization. *JMLR*, 3:1183–1208, 2003.

[4] B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of Workshop on Computational Learing Theory*, pages 144–152, 1992.

[5] T. M. Cover and J. Thomas. *Elements of Information Theory*. Wiley, 1991.

[6] G. Druck, G. Mann, and A. McCallum. Learning from labeled features using generalized expectation criteria. In *Proceedings of SIGIR*, 2008.

[7] D. D. Lewis. An evaluation of phrasal and clustered representations on a text categorization task. In *Proceedings of SIGIR*, pages 37–50, 1992.

[8] E. Riloff and W. Lehnert. Information extraction as a basis for high-precision text classification. *ACM Transactions on Information Systems*, 12, 1994.

[9] M. Sahami and T. Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *Proceedings of WWW*, 2006.

[10] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34, 2002.

[11] O. Shamir, S. Sabato, and N. Tishby. Learning and generalization with the information bottleneck. *Theoretical Computer Science*, 411:2696–2711, 2010.

[12] N. Tishby, F. Pereira, and W. Bialek. The information bottleneck method. In *Proceedings of the 37-th Annual Allerton Conference on Communication, Control and Computing*, pages 368–377, 1999.

[13] P. Wais, S. Lingamneni, D. Cook, J. Fennell, B. Goldenberg, D. Lubarov, and D. Martin. Towards building a high-quality workforce with mechanical turk. In *Proceedings of Workshop on Computational Social Science and the Wisdom of Crowds*, 2010.

[14] T. Zhang and F. Oles. Text categorization based on regularized linear classification methods. *Information Retrieval*, 4(1):5–31, 2001.