

Classifying Spend Descriptions with off-the-shelf Learning Components

Saikat Mukherjee Dmitriy Fradkin Michael Roth
Integrated Data Systems Dept.

Siemens Corporate Research, U.S.A.

{saikat.mukherjee,dmitriy.fradkin,michael.roth.ext}@siemens.com

Abstract

Analyzing spend transactions is essential to organizations for understanding their global procurement. Central to this analysis is the automated classification of these transactions to hierarchical commodity coding systems. Spend classification is challenging due not only to the complexities of the commodity coding systems but also because of the sparseness and quality of each individual transaction text description and the volume of such transactions in an organization. In this paper, we demonstrate the application of off-the-shelf machine learning tools to address the challenges in spend classification. We have built a system using off-the-shelf SVM, Logistic Regression, and language processing toolkits and describe the effectiveness of these different learning techniques for spend classification.

1 Introduction

Organizations are engaged in various kinds of procurement activities. Such procurement, or *spend*, could be associated directly with product manufacturing or with indirect internal purchases. Direct spend is more typically centered on certain fixed commodities and possibly pre-defined suppliers. However, indirect spend is much more diffuse and cuts across the spectrum of goods and services. The purchase of office furniture, buying airline tickets and car rentals are some examples of indirect spend.

One effect of the diffuse nature of indirect spend is that often local units within an enterprise conduct their individual purchasing independently without any global coordination. This results in problems such as proliferation of suppliers for identical goods and services, inability of the enterprise to identify purchasing synergies between units, failure to identify the major classes of purchases and their amounts, and inability to strike better bargains with suppliers. Hence, it is imperative for organizations to maintain an *integrated global view* of the total spend activity.

Approaches to integrating spend involve associating

spend transactions to a family of *commodity codes* where each code represents a particular class of product or service. Typically these commodity codes are arranged hierarchically from generic to more specific classes. Examples of such commodity codes are UNSPSC¹, eCI@ss², and ESN. Figure 1 shows a fragment of the ESN commodity coding system which is widely used and has a broad coverage of products and services. The ESN scheme is a three level hierarchy where each code consists of at most three letters denoting increasing specificity of the product or service. For instance, in Figure 1, the code *M* represents “Electrical Products”, while its child code *MB* represents “Power Products”, and finally the leaf codes *MBL* and *MBM* represent “UPS Systems” and “Inverters for Power Supply” respectively. Once spend transactions have been associated to their respective commodity class, the integrated data can be analyzed by business intelligence software or different kinds of data mining tools.

Manually associating spend transactions to their corresponding classes is, however, not scalable due as much to the large number of transactions as to the diversity of classes within any single commodity coding scheme. It becomes even more difficult with increasing size and spread of organizations. Furthermore, manually associating large number of transactions is subject to human mistakes and varying skills of data experts. Consequently, there is a need to develop *automated* techniques for spend classification.

In this paper, we describe the application of off-the-shelf machine learning techniques for automated classification of spend transaction to commodity codes. In our work, we have used only the text descriptions of spend transactions to classify them to appropriate commodity classes. We have also used the ESN hierarchy as the target commodity structure. For instance, a spend on purchasing a Dell Latitude notebook could have an associated human written description “15 inch Dell Latitude” and it needs to be classified to code *NNC* of the ESN structure which represents spend on laptops.

¹<http://www.unspsc.org>

²<http://www.eclass-online.com>



Figure 1. Portion of the ESN Commodity Coding Scheme

Various aspects of classifying spend descriptions make it more challenging than typical text categorization tasks.

- Classifying spend descriptions involves hierarchical text categorization due to the structure of the commodity coding schemes. Typically, these hierarchies are extensive which makes the classification task even more challenging. For instance, the ESN hierarchy has 2185 classes spread across 3 levels.
- The descriptions tend to be short pieces of text with typically less than 5 words which is unlike usual text categorization data sets.
- Due to the sparsity of words in spend descriptions, classifying them correctly to commodity classes requires that individual features within them are not erroneous. However, because these descriptions are manually written they suffer from different kinds of mistakes such as: (a) spelling errors when a word is mis-spelled (b) merge errors when multiple words are joined together, and (c) split errors when a single word is broken up into multiple words. For instance, “MicrosoftOffice” is a merge error where the two words “Microsoft” and “Office” have been joined together. Similarly, “note book” is a common error where the single word “notebook” has been split up into two words.
- Spend descriptions in global organizations could be in multiple languages. This makes it difficult to use linguistic techniques suitable for one particular language but not for others.
- The amount of spend data in large global organizations is enormous and could approach half a million new

samples monthly. This makes training classifiers for such large data sets computationally challenging.

- The classifiers need to be retrained periodically not only because of new training samples but also because commodity coding structures undergo revisions whereby new classes are added and some old ones dropped. Yet using millions of old samples, as well as the new ones, during retraining could again be computationally difficult.

In this paper, we describe ways in which we have applied machine learning techniques to address these challenges. In particular, we have focused on using and modifying off-the-shelf learning tools for *spend classification* and *noisy feature correction*. [26] and Zycus³ are recent examples of work in the research community and in commercial software for spend analytics. However, they typically do not address the challenges of hierarchical commodity classification or the presence of noisy features in data.

In spend classification, we have experimented with Support Vector Machines (SVM) as well as Bayesian Logistic Regression classifiers. We have applied these classifiers in a hierarchical setting and, for SVM, explored ways to improve performance beyond the baseline. Our results demonstrate the superiority of Logistic Regression over SVM both in terms of precision and computational efficiency.

We model the feature correction problem as an instance of *noisy channel* similar to [2, 16]. In noisy channel approaches to language correction, it is assumed that the observed word O is a corruption of an intended word C due to the presence of a “noisy channel”. We have focused on correcting only spelling mistakes and merge/split errors in features by this approach and have assumed 1 error per word for computational purposes. Our experiments demonstrate the utility of the noisy channel technique for feature correction in spend description data sets.

The primary contribution of our work lies in using off-the-shelf machine learning tools for solving a very practical problem – classifying spend descriptions to hierarchical commodity classes. We demonstrate performance and trade-offs between off-the-shelf SVM and logistic regression classifiers for this task as well as a practical application of noisy channel model for feature correction. We have brought this diverse suite of techniques within a single system to address challenges in automated spend classification.

The rest of the paper is organized as follows: Section 2 describes our experiments with SVM and logistic regression classifiers for spend categorization. In Section 3, we describe the technical details and results of correcting noisy features. Related work for classifiers and feature correction are referred to in their own sections, Section 4 briefly describes the system and Section 5 concludes the paper.

³<http://www.zycus.com>

2 Spend Classification

2.1 Data

The total size of the ESN hierarchy is 2185 nodes, not counting the root, of which most (2105) had at least 1 description. There are 17 first-level nodes, 192 second-level nodes and 1976 leaf nodes. Note that actual expense descriptions can only be assigned to the leaf nodes.

Our data consists of 128695 descriptions believed to be accurately assigned to the leaves. In order to evaluate different approaches, we randomly split this collection into 2/3 training and 1/3 test set. The size of the training set was 85953, while the test set was 42742. The training set contains cases from 418 leaf nodes, while the test set has cases from 380 leaf nodes. Additionally, the descriptions of ESN hierarchy nodes in 5 languages added 14981 examples (for 2105 nodes) to the training set. The full dictionary for the training set consisted of 69429 terms.

2.2 Feature Representation

In text classification, a document is usually represented by a sparse vector with coordinates (features) corresponding to individual words in that document. Information retrieval and text classification literature [25, 20] describe many ways of assigning weights to features based on various term statistics, such as term-frequency in a document, inverted document frequency in the collection and so on.

The specifics of our application affect the representations that we examine. Since descriptions are short, repetition of terms is likely not a useful indicator, so term frequencies are not used. Also, feature importance will change between sibling nodes and between different levels of the hierarchy, so a single weighting scheme such as inverted document frequency is unlikely to be useful.

The two representations we examine are a simple binary weighting scheme (a term has weight 1 if it occurs in the description and has weight 0 otherwise), and a node specific feature weighting representation leveraging the hierarchical structure of the ESN. The weight of a feature f at a node n , denoted by $wt(f, n)$, is:

$$wt(f, n) = (N_f^n / N_f) \times (1 + \log(N^n / N_f^n))$$

where N_f^n is the number of leaf nodes within the subtree rooted at n which has at least one positive sample with f , N_f is the total number of leaf nodes in the entire ESN tree which has at least one positive sample with f , and N^n is the number of leaf nodes in subtrees rooted at n as well as all its sibling nodes. Intuitively, (N_f^n / N_f) represents a subtree weight indicating the relative importance of the subtree at n compared to the rest of ESN tree for f . The second part of $wt(f, n)$ indicates the proliferation of f within leaf nodes rooted at n and it penalizes features which occur in more

leaf nodes. It is a variation of the traditional *idf* feature weighting but adapted to the hierarchical structure. Thus, rare features would be weighted higher in upper levels of the ESN tree and the weights decrease down the tree. This helps in making the classifiers at the higher levels more accurate.

In discussions below, we add "W" to the name of the method if it uses node-specific feature weighting.

2.3 Classification Methods

Support Vector Machines (SVM) [28, 8] often show good results in text classification [15]. We restricted our attention to linear SVM since they are known to achieve high performance in text classification [15] while being faster and requiring less computational resources than non-linear SVM. The particular implementation we used is LIBSVM [5]. It is freely available and widely used. The multiclass approach implemented in LIBSVM is one-vs-one: a classifier is constructed for each pair of classes. During classification, each classifier votes for one of its two classes. The sample is assigned to a class with the most votes. This and alternative approaches were examined in [14]. However, due to increased memory and time required to perform train multiclass SVM models, we were not able to evaluate them in this project - LIBSVM would run out of memory on many of the multiclass subproblems or take a very long time. LIBSVM also allows classes to have different weights, specified with "-wi" option, where "i" is the class label. We experiment with balancing classes, by assigning each class a weight inversely proportional to the fraction of training samples that belong to it. Results obtained with this approach will be marked with letter "B".

We also experiment with penalized logistic regression classifiers, implemented in BBR (binary version) and BMR (multiclass version) programs [12, 21]. These methods have been previously shown to be competitive with SVM on text classification problems [12]. BBR stands for Bayesian Binary Regression, which is an alternative way of looking at penalized logistic regression. Under Gaussian prior ("-p 2") for the model parameters, BBR is equivalent to L2-penalized, or "ridge", logistic regression, while under Laplace prior ("-p 2") it is L1-penalized, or "lasso" [27], logistic regression. The latter method has sparseness-inducing properties, since weights for many features are forced to zero. Below we will refer to these two methods as BBR L1 and BBR L2. BMR implements multinomial (multiclass) logistic regression under L1 and L2 penalties - it is implicitly a one-vs-all approach, where each class is assigned a score, and the class with the highest score is picked.

Both SVM and BBR/BMR have regularization parameters that affect their performance. Ideally, these parameters should be selected by estimating performance for different values with cross-validation or on the hold-out set. How-

	BBR L2	BBR L1	SVM	BBR L2 W	SVM W	SVM WB	# positive cases
N	94.29	95.19	85.73	93.20	93.45	93.09	12897
Q	94.97	95.31	84.66	94.01	93.98	93.88	12232
M	89.48	91.41	79.39	87.16	87.14	86.67	5178
O	93.40	93.99	80.18	94.55	94.63	94.01	4893
F	92.59	94.25	86.97	89.32	88.21	86.96	4519
J	97.15	97.78	83.08	96.43	95.01	93.95	2842
A	59.21	61.84	51.32	31.58	46.05	26.32	76
P	28.26	36.96	26.09	31.11	26.67	2.22	46
I	84.09	81.82	68.18	81.40	86.05	72.09	44
K	42.86	50.00	64.29	42.86	0	0	14

Table 1. Performance (Precision-Recall Break-Even Point) of Binary Classifiers on all the First Level Nodes (except for G which has only 1 positive example). The best results at each node are in bold. The size of the test set is 42742.

ever, in our application, it would be impractical to do this for every classifier since the training would then take too much time. Therefore, we rely on default settings. For SVM, the default value of C is 1. For BBR/BMR the prior variance parameter is set to be the ratio of the number of distinct features to the mean 2-norm of the training examples. We also use default settings for other parameters, unless stated otherwise.

2.4 Hierarchical Classification

Hierarchical classification problems occur frequently in text analysis, however, unlike "flat" classification, to the best of our knowledge there aren't any off-the-shelf implementations of hierarchical classification methods. Therefore, one is forced to make use of the flat classification tools to solve hierarchical problems. There are two approaches that one can take:

- build a single "flat" classifier that would map an instance directly to a leaf; or
- build a hierarchical classifier by constructing a model at every node.

The first approach essentially ignores the hierarchical structure of the data which usually leads to worse results [10, 17]. In [10] the flat approach performed worse than a hierarchical one when classifying web documents into a two-level category hierarchy, using SVM to build individual models. In [17], Bayesian methods and separate feature selection for individual classifiers led to better results than flat approaches or hierarchical approach without individual feature selection. Thus, building a multiclass classifier at each internal node is a preferred approach.

Many multiclass classification approaches consist of combining binary classifiers to select a single class. Such

approaches include error-correcting codes [9], one-vs-each and one-vs-all [11, 29, 7]. Many binary classification methods, including SVM and logistic regression, can be reformulated for multiclass problems using these ideas. These reformulations, while elegant, can lead to increased memory and computational requirements, compared to explicit construction of a multiclass classifier from binary classifiers, as some of our experiences with this project show.

Some recent work focused on developing methods specifically for hierarchical text classification [3, 24]. Both of these approaches involve new formulations of SVM that take into account hierarchical structure of the classes. While the results are encouraging, implementations of these methods are currently not easily available.

2.5 Experiments

We initially consider using binary classifiers at each node (i.e. each classifier makes a prediction whether a particular description belongs to the node or not). The node with the highest score is the one to which a particular description is assigned. This is a one-vs-all multiclass classification scheme. The specific methods we consider are: BBR L2, BBR L1 and SVM on binary features; and BBR L2 W, SVM W and SVM WB - i.e. using node-specific weighting, and, in the last case, weight balancing for classes.

Table 2.2 shows performance (as Precision-Recall Break-Even Point) of these methods on the first level nodes of the ESN hierarchy. These results show that BBR L1 gives the best results. Applying BBR L2 to weighted features gives worse results than applying either kind of BBR to binary features. SVM methods appear comparable when applied to weighted features - SVM on binary features performs worse than all the other methods. Results on second and third level nodes (a sample of second level results in shown in Table 2.5) suggest that BBR L1 is at least as good

	BBR L2	BBR L1	SVM	BBR L2 W	SVM W	SVM WB	# positive cases	# cases
FF	75.47	75.47	64.15	80.39	78.43	74.51	53	4519
FE	85.11	80.85	68.08	80.44	78.26	78.26	47	4519
FO	95.49	96.20	92.07	94.01	93.90	93.90	2655	4519
FP	93.35	93.51	87.63	92.87	92.69	92.21	1711	4519
NO	96.04	96.13	88.02	94.48	94.38	94.50	5685	12897
NN	94.65	95.16	84.08	91.33	92.43	92.08	2915	12897
NJ	90.96	91.87	75.30	88.22	86.10	88.22	332	12897
NR	82.47	79.38	43.30	80.85	76.60	76.60	97	12897
NK	69.23	61.54	41.03	41.03	43.59	43.59	39	12897
NP	48.15	66.67	51.85	61.54	53.85	38.46	27	12897
OC	96.56	95.95	87.55	96.41	96.63	96.46	1801	4893
OM	97.68	97.26	87.10	98.65	98.65	98.40	1206	4893
OE	92.11	90.51	76.55	94.40	93.56	93.92	875	4893
ON	93.46	93.22	74.06	96.39	95.91	96.15	428	4893

Table 2. Performance (Precision-Recall Break-Even Point) of Binary Classifiers on some of the Second Level Nodes. The best results at each node are in bold. The size of the test set varies for different nodes.

as any other method. Note that, not surprisingly, results tend to be worse for smaller classes than for larger ones.

BBR models were much faster to train, taking at most several minutes as opposed to up to half an hour for training SVM. Furthermore, BBR L1 models are much smaller than BBR L2 or SVM models due to storing only a vector of weights for relatively few features, due to the sparseness inducing prior. The total number of weights in the 2185 models built with BBR L2 was 4184500 (or 1915.1 per model), while with BBR L1 it was 159656 (or 73.1 per model). In comparison, the average number of support vectors per model using SVM W approach was 339.649 per model, with the average of 3.43 features per support vector. It is clear from these numbers that SVM models are at comparable to BBR L2 models in size, and are much larger than BBR L1 models.

However, when we evaluate the full hierarchical classifier, we find that performance of BBR deteriorates rapidly as we move from first to the leaf levels, while SVM performance decreases much slower. (These results are in Table 2.5). The explanation for this discrepancy is the following. We use "-b" option in LIBSVM, which leads to SVM being trained to compute probability estimates. Therefore, the scores produced by SVM classifiers at different sibling nodes are all calibrated probabilities and are comparable across different classifiers. The BBR scores, while also technically probabilities of class membership, are not calibrated and therefore are not directly comparable when produced by different classifiers. Thus, while individual BBR classifiers order the test examples better than binary SVM classifiers, their predictions do not compare well across sibling classifiers - they do not produce a reliable multiclass

classifier. The problem becomes more noticeable at lower levels due to occurrence of rare classes. In these circumstances, a text will have a higher score for a more frequent class model with which it has no terms in common than for a rare class with whose model it has some common terms. This problem could be resolved by adding calibration post-processing to BBR models or possibly just adjusting the intercept in some way. However, this would have entailed additional implementations and evaluation of different calibration techniques [31, 30].

At this point we decided to switch to a multiclass version of BBR: BMR. This software builds weight vectors for all classes simultaneously, requiring more memory. In our experiments it still however ended up being much faster than training multiple binary SVM classifiers. The number of weights per class in BMR models tends to be larger than in BBR models: 4095763 (or 4707.77 per class) and 89110 (or 102.425 per class) based on 870 classes which were included in the models. (Some classes were omitted since there was no test data for them). However, BMR L1 models is still much smaller than SVM models, where most of the training examples would end up as support vectors. It turns out (see Table 2.5 that using BMR L1 leads to the best performance in terms of the leaf level accuracy, and accuracy at each level. BMR L2 was the second best method.

2.6 Summary

Our comparison of the off-the-shelf SVM and BMR classifiers showed that for the SPEND application BMR leads to much faster performance and smaller models than SVM while giving an improved performing. While performances

Methods:	SVM W	SVM WB	BBR L2	BBR L1	BMR L2	BMR L1
Top Level Accuracy	91.64%	89.90%	92.98%	93.35%	93.80%	95.00%
Second Level Accuracy	87.58%	85.84%	23.83%	24.00%	91.14%	92.10%
Leaf Accuracy	81.80%	80.36%	9.43%	9.50%	84.26%	85.66%

Table 3. Accuracy of Different Methods

of these methods can likely be improved by careful parameter selection, the practical constraints of the problem leave little room for that.

3 Spend Feature Correction

As mentioned in Section 1, one of the challenges in classifying spend is the presence of mistakes in words in the text descriptions. Since spend descriptions are typically short strings, these mistakes have a bearing on the classification accuracy. In this section, we describe the application of *noisy channel* techniques to correcting typos, merge, and split errors.

In the noisy channel framework, the joint probability of observing an erroneous sequence of characters O corresponding to a correct sequence of characters is

$$P(O, C) = P(O|C) \times P(C)$$

where the intuition is that first the intended sequence of characters, C , is generated with $P(C)$ which is then corrupted due to a noisy channel into sequence of characters O with probability $P(O|C)$.

Our work is based on noisy-channel based OCR post-processing [2, 16]. Most of these techniques depend on a language specific lexicon for correction (see [18] for a survey). However, spend descriptions could be in multiple languages and many of them have limited support of thesauri and lexicons such as Wordnet [23]. Hence, we have currently adopted a linguistic free approach.

3.1 Experiments

Source Model: We estimate the source language model, $P(C)$, by smoothed frequency counting of 5-grams character sequences over samples in the data set. We experimentally determined 5 to be the size of the n-grams considering computational costs and capturing local context. In order to keep the source model clean, we considered only those words which were at least 2 characters long and had only alphabetic characters. We used the CMU-Cambridge Language Modeling Toolkit [6] to create the 5-grams models with a vocabulary of alphabets and space and using Witten Bell smoothing.

Channel Model: The channel model is estimated from training data each of which is a pair of erroneous word and corresponding correct word. Rather than manually creat-

ing the training data by going over all samples, we created a set of possible training instances by first collecting all words and bigrams (2 consecutive words separated by space) such that each word contains only alphabets and has at least 2 characters. We created a spelling mistake training pair by pairing a correct word w_i with an incorrect word w_j if w_i, w_j start with the same character, has a *normalized edit distance* [22] of at least 0.8, and w_i occurs 10 times more than w_j . The assumptions were that spelling mistakes seldom happen in the first character, the correct word occurs many more times than the incorrect word, and despite the mistake the pair of words still have high character similarity. Similarly, a potential split error training pair is a bigram w_i, w_j and a word w_k such that w_k is the concatenation of the characters of w_i and w_j and occurs 10 times more than the bigram. Merge training pairs were computed by splitting a word at all its character positions and checking whether the resulting bigram occurs in the data and with relatively more frequency.

These potential training instances were manually cleaned to produce the final training data. We treat each instance as a pair of a sequence of characters from the set of alphabets, space, and ϵ which is used to denote the absence of a character. The sequence pairs in each training instance are aligned to each other as per their minimum edit distance. and $P(c_i \rightarrow c_j)$, the probability of converting character c_i into character c_j , is estimated using maximum likelihood frequency counting.

Testing: We limit the number of mistakes in any test sample to typically 1 for computational reasons. Given a test sample sequence of characters O , we identify all possible variations with 1 character corrections by substituting or deleting or inserting characters. For each variation C , we compute $P(C)$ by breaking it into independent 5-grams and using the source model. $P(O, C)$ is computed by combining $P(C)$ with the probability of the correction from the channel model. The best possible C is returned as the corrected character sequence.

Results: Table 4 shows the results of our evaluation. Automatically generating training instances for the channel model resulted in 2469 spelling mistake pairs, 84 merge pairs, and 91 split pairs. This was manually cleaned to a total number of 717 unique training pairs, present in 3004 samples, and 159 unique test cases present in 280 test samples. In Table 4, *IC-FT* denotes evaluation for 1 character correction considering each single word and bigram inde-

	#Test Samples	All 5-grams		5-grams with Freq > 2		5-grams with Freq > 10	
		Precision (%)	Recall (%)	Precision (%)	Recall (%)	Precision (%)	Recall (%)
1C-T	280	88.20	60.38	83.78	55.36	84.74	57.50
1C-S	280	81.22	57.14	74.16	55.36	66.82	52.50
1C-DS	3284	73.52	57.10	70.85	58.62	66.38	57.17
2C-T	280	92.35	60.36	88.42	60.00	90.53	61.43

Table 4. Evaluation Results of Feature Correction

pendently in test samples instead of the whole string together. *1C-S* denotes results for 1 character correction on entire test sample strings while *1C-DS* denotes results on entire strings of both training as well as test samples. Finally, *2C-T* denotes 2 character corrections considering single words and bigrams independently in test samples. *All 5-grams* represents using all 5-grams in building the source model while *5-grams with Freq > 2* and *5-grams with Freq > 10* used only those 5-grams which appear at least twice and ten times respectively in the entire data set.

Our results indicate that using the frequency to limit words in the source model degraded performance. This could be because of the extreme sparseness of data where many good words do not appear often. As expected, it is easier to correct words or bigrams taken independently than the whole sample because of the presence of additional context. This negatively affects precision when rare words have subparts which are also present in more frequent words. For instance, “rent for company apartment” is incorrectly changed to “rent for companty apartment” because the model is biased towards the 5-gram “ty app” due to many samples of the form “quality appraisal”. However, sometimes the context helps in recall such as “fujits hornet” gets corrected to “fujitsu hornet” but not the sole word “fujits”. Thus, the recall does not drop as much as the precision between *1C-T* and *1C-S*. Finally, the results indicate that having 2 character corrections does improve the precision (*2C-T*) which is why it is important to scale to multi-character mistakes. This is also supported by Figure 2 which shows the amount of CPU time, in a machine with 1 GB RAM and 2 GHz processor, to perform 1 character feature correction increasing linearly with number of words in samples. One way to improve this is by using advanced data structures to represent the feature space [13].

4 System

Based on the above off-the-shelf learning tools and modifications to them, we have developed a system for spend classification in Java. The system lets users browse training data and ESN descriptions which could be in 5 different languages. The system is coupled to a backend Derby database which stores the training data and ESN descriptions. Users can train hierarchical SVM and BMR classifiers through the

system and specify parameters for feature weighting and selecting different training data sets. Users can test data in batch mode by directly connecting to databases. The results are displayed in the system and users can correct them and feed them back to training data. Results are also saved in a database for further examination. The system has been successfully deployed and received positive feedback from multiple users.

5 Discussions

Automated classification of spend transactions to commodity codes is important to organizations for analyzing their overall procurement in a scalable way. In this paper, we have described the application of off-the-shelf learning tools spend classification. We presented experimental results which demonstrate the performance of SVM and Logistic Regression classifiers as well as a noisy channel framework for feature correction.

We have not directly addressed here the need for frequent retraining of the classifiers due to new examples, or handling increasingly large datasets. The only way to reliably handle such situations is with incremental algorithms. Multiple incremental approaches have been developed for SVM (ex. [4, 19]). Some work in this direction has also been done for logistic regression [1]. However, the accuracy of these approaches tends to be lower than for batch methods, and off-the-shelf implementations are not readily available.

Our current classification is based solely on the spend text description. However, if related information such as *supplier* names and purchase volumes are available for spend transactions then they could also be used for better classification. Finally, our feature correction techniques are currently language agnostic and could be improved if transactions can be geographically localized and *linguistic* cues, wherever available, of corresponding languages incorporated.

References

- [1] S. Balakrishnan and D. Madigan. Algorithms for sparse linear classifiers in the massive data setting. *Journal of Machine Learning Research*, 9:313–337, 2008.

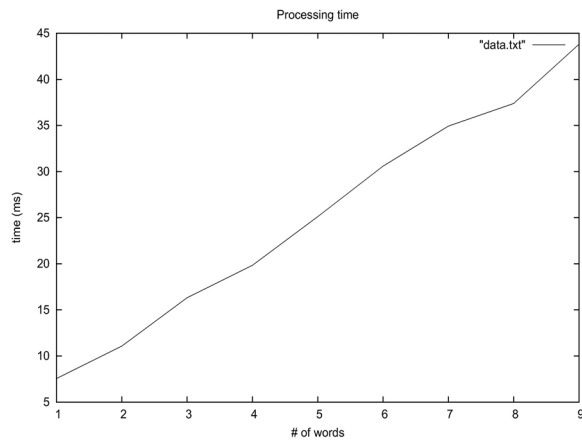


Figure 2. Time taken for feature correction with number of words in samples

- [2] E. Brill and R. Moore. An improved error model for noisy channel spelling correction. In *Proceedings of Annual Meeting of the Association for Computational Linguistics*, 2000.
- [3] L. Cai and T. Hofmann. Hierarchical document categorization with support vector machines. In *ACM Conference on Information and Knowledge Management*, 2004.
- [4] G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2000.
- [5] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [6] P. Clarkson and R. Rosenfeld. Statistical language modeling using the cmu-cambridge toolkit. In *Proceedings of ESCA Eurospeech*, 1997.
- [7] K. Crammer and Y. Singer. On the learnability and design of output codes for multiclass problems. In *Computational Learning Theory*, pages 35–46, 2000.
- [8] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 2000.
- [9] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.
- [10] S. Dumais and H. Chen. Hierarchical classification of web content. In *ACM Conf. on Information Retrieval (SIGIR)*, 2000.
- [11] J. H. Friedman. Another approach to polychotomous classification. Technical report, Stanford University, 1996.
- [12] A. Genkin, D. D. Lewis, and D. Madigan. Large-scale bayesian logistic regression for text categorization. *Technometrics*, 49(3):291–304, 2007. Software: www.bayesianregression.org/.
- [13] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.
- [14] C. Hsu and C. Lin. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 13(2), March 2002.
- [15] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European Conf. on Machine Learning*, 1998.
- [16] O. Kolak and P. Resnik. Ocr post-processing for low density languages. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, 2005.
- [17] D. Koller and M. Sahami. Hierarchically classifying documents using very few words. In *Proceedings of International Conference on Machine Learning*, pages 170–178, 1997.
- [18] K. Kukich. Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24(4), 1992.
- [19] P. Laskov, C. Gehl, S. Krueger, and K.-R. Mueller. Incremental support vector learning: Analysis, implementation and applications. *Journal of Machine Learning Research*, 7:1909–1936, 2006.
- [20] D. D. Lewis. Text representation for intelligent text retrieval: a classification-oriented view. pages 179–197, 1992.
- [21] D. Madigan, A. Genkin, D. D. Lewis, S. Argamon, D. Fradkin, and L. Ye. Author identification on the large scale. In *Proceedings of Joint Annual Meeting of the Classification Society of North America*, 2005.
- [22] A. Marzal and E. Vidal. Computation of normalized edit distance and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9), 1993.
- [23] G. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller. WordNet: an on-line lexical database. *International Journal of Lexicography*, 3(4), 1990.
- [24] J. Rousu, C. Saunders, S. Szedmak, and J. Shawe-Taylor. Learning hierarchical multi-category text classification models. In *Proceedings of International Conference on Machine Learning*, 2005.
- [25] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.
- [26] M. Singh, J. Kalagnanam, S. Verma, A. Shah, and S. Chalasani. Automated cleansing for spend analytics. In *Intl. Conf. on Information and Knowledge Management (CIKM)*, 2005.
- [27] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.
- [28] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 2nd edition, 1998.
- [29] J. Weston and C. Watkins. Multi-class support vector machines. Technical Report CSD-TR-98-04, Department of Computer Science, Royal Holloway, University of London, 1998.
- [30] T.-F. Wu, C.-J. Lin, and R. C. Weng. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5:975–1005, 2004.
- [31] B. Zadrozny and C. Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of ACM SIGKDD'02*, 2002.