Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 159 (2019) 736–745

Procedia Computer Science

www.elsevier.com/locate/procedia

23rd International Conference on Knowledge-Based and Intelligent Information & Engineering Systems

Dealing with Data Imbalance in Text Classification

Cristian Padurariu[a,b], Mihaela Elena Breaban[a,b*]

[a] *Faculty of Computer Science, Alexandru Ioan Cuza University of Iasi, Romania*
[b] *SenticLab Research, Iasi, Romania*

Abstract

Many real world datasets don't offer enough training input for regular classifiers: some classes are more represented than others. Imbalanced data raises problems in Machine Learning classification and predicting an outcome becomes difficult when there is not enough data to learn from. The object of classification in our study is data coming from the field of Human Resources, consisting of short descriptions of work experiences which must be classified into several highly imbalanced classes expressing job types. We perform an extensive experimental analysis using various representations of text data, several classification algorithms and balancing schemes to derive a model that achieves highest performance with respect to metrics such as precision and recall. The contribution is twofold: a) with a comprehensive experimental design, the analysis is focused on studying the interactions between classification algorithms, text vectorization choices and the schemes to deal with data imbalance at several degrees of imbalance; b) besides state-of-the-art balancing schemes, we propose and analyze a cost sensitive approach formulated as a numerical optimization problem where the costs are derived with a Differential Evolution algorithm in two steps: in a first step costs are optimized at the class level and in a subsequent step costs are refined at the data instance level. The results indicate that the use of cost-sensitive classifiers where the cost matrices are optimized with a Differential Evolution algorithm brings important benefits on our real-world problem.

Keywords: imbalanced text classification; oversampling techniques; cost-sensitive methods; text vectorization; differential evolution

1. Introduction

Classification of imbalanced data is an important topic of research triggered by the frequent real-life situations where some classes in data are underrepresented and by the incapability of standard classifiers to properly discriminate

* Corresponding author.
E-mail address: pmihaela@info.uaic.ro

1877-0509
10.1016/j.procs.2019.09.229

the poorly represented classes. Although many solutions were proposed in literature to deal with data imbalance, there is not a unique approach among the rest that shows best results, but their performance depends on the intrinsic characteristics of the data [1].

In the Human Resources area we deal with high volumes of unstructured data coming from CVs, job postings, evaluation forms. All these documents are processed in order to extract information which is further stored as structured data. In a first step, these documents are segmented into sections that express different types of information (i.e. personal information, education, job experiences in a CV). A very important task in building automatic recruitment systems is to identify all the work experiences a person describes in his CV and to match these experiences into a predefined hierarchy of job types. The current paper illustrates the difficulties raised by imbalanced distributions in data and the solutions we used to deal with at this step, when building a classifier system for the job type a person has accomplished given a short description of the work experience extracted from the CV.

When tackling imbalanced text data classification, decisions must be made at several distinct stages: How to represent the text information? What is the classifier algorithm that would give the best results? How to deal with the imbalance in data?

Of course, the decision variables above (text representation, classification algorithm, mechanism to deal with imbalance) are not independent, situation which generates a complex optimization problem with a large selection/decision space. We tackle it by a comprehensive experimental analysis that highlights interesting properties and leads us to a solution that combines several distinct algorithms, including meta-heuristics.

The paper is structured as follows. Section 2 briefly presents the methods generally used in NLP to represent text as fix-sized numerical data, methods which are also investigated in our experimental analysis. Section 3 reviews solutions proposed in literature to deal with imbalance in data classification. Besides state-of-the art methods we employ in our study, this section also describes the cost-sensitive approach we implement, based on Differential Evolution. Section 4 motivates the choice of the classifier algorithms, presents the experimental settings and discusses the results obtained. Conclusions are drawn in Section5.

2. Numerical representations for text data

When dealing with classification of text documents, a first step is to obtain a representation of the data that can be used within the learning algorithm. For this purpose, several ways exist which transform a collection of text documents into a numerical dataset.

2.1 Bag of Words (BoW)

The simplest way to transform text documents into numerical data, known as Bag of Words, is to consider a number of features equal to the number of words in the corpus. Then, for each document, the value of each feature is given by the number of occurrences of that word within the current document.

2.2. TF-IDF

A common thing in text mining is to measure how important a word is to a document. One way to find this is by computing the term frequency (tf), but in many languages there are stop words that will most likely be the most frequent. In order to identify words that are specific to some documents *inverse document frequency* (idf) is introduced, which has low values for the most common words and increases the weight of rare words.

The final solution for a relevant weighting of words is using the tf.idf numerical statistic which is computed using the formula: $tf.idf_{t,d} = tf_{t,d} \times idf_{t,d} = tf_{t,d} \times \log(\frac{N}{df_t})$ where $tf_{t,d}$ is the number of occurences of term t in document d, df_t is the number of documents containing t and N the total number of documents.

2.3. GLOVE embeddings

GLOVE is a count-based model that learns their vectors by essentially doing dimensionality reduction on the co-occurrence counts matrix [2]. As a first step, a large matrix of co-occurrence information is constructed, where the rows represent words and columns represent the context found in the corpus. One value in the matrix represent the occurence of a word in a specific context. This matrix is factorized and reduced to a lower-dimensional (word x features) matrix, where each row represents a vector representation for each word.

2.4. DOC2VEC

Doc2Vec [3] is a well known document embedding technique for generating numerical representations for texts. Doc2Vec is built on word2vec [4], which is a technique for generating numerical representation vectors for words,

that is able to capture relationships between words, such as synonymy, antonymy or analogies. Word2Vec uses a neural network to derive the embedding and can be implemented in two ways:

- CBoW (Continuous bag of words): This approach creates a sliding context window around the current word, so *cbow* learns the context of a word to try and predict it.
- Skipgram: This algorithms is the opposite of cbow, because it predicts the context words surrounding a specific word.

On top of the above algorithms, doc2vec adds another layer which corresponds to paragraph representation. When training the word vectors, the document vector is trained as well, and in the end of training, it holds a numeric representation of the document.

2.5. Word2Vec with char n-grams (Fasttext)

Fasttext is a library deriving word embeddings and implementing a very efficient text classifier, developed by the Facebook Research team [5]. Fasttext generates word embeddings similar to the Word2Vec method, but unlike word2vec, it introduces char n-grams that facilitates the learning of rare words.

3. Dealing with imbalanced data in classification

When classes are imbalanced, standard classifiers are usually biased towards the majority class. In this case, a shift is necessary from the general paradigm that optimizes the overall classification accuracy to one that emphasizes the trade-off between precision and recall.

The issue of imbalanced data has been an important subject of research for a while and a number of solutions have been developed so far [6]. They can be grouped in two distinct categories: one category corresponds to methods that operate on the dataset in a preprocessing step preceding classification while a second category modifies the classification algorithm in order to put more emphasis on the minority class. The methods in the first category are known as resampling methods. Within the second category cost-sensitive learning is a paradigm that emphasizes the incorrect classification of instances from the minority class during the training process while being minimal intrusive for the classifier in the sense that the algorithm does not require important changes. In the following we briefly describe the methods in the two categories.

3.1. Resampling methods

Resampling methods aim at modifying the dataset in order to reduce the discrepancy among the sizes of the classes. In this regard, two scenarios are proposed: one that eliminates instances from the majority class - called undersampling, and one that generates instances for the minority class - called over-sampling. Random undersampling showed to give poor results in our preliminary experiments, therefore we list in the following the over-sampling methods evaluated in the experimental section.

3.1.1. Random oversampling

This is a simple approach where we take samples at random from the small class and duplicate this instances so that it reaches a size comparable with the majority class.

3.1.2. SMOTE

Synthetic Minority Over-sampling Technique [7] is an oversampling method based on creating synthetic instances for the minority classes. The algorithm takes each minority class sample and introduces synthetic samples along the line joining the current instance and some of its k nearest neighbours from the same class. Depending on how much oversampling is needed, the algorithm chooses randomly from the k nearest neighbours some of them and forms pairs of vectors that are used to create the synthetic samples. The new instances create larger and denser decision regions. This helps classifiers learn more from the minority classes in those decision regions, rather than from the large classes surrounding those regions.

3.1.3. SMOTE-SVM

The idea of this method is applying the SMOTE algorithm only on those instances that belong to the support vectors generated by a SVM classifier. To do this we first need to train an SVM (for this we used a linear kernel) on the data

before any oversampling. The SVM model will give us the samples belonging to the support vectors, which will then be oversampled using SMOTE. This approach doesn't equalize the number of instances, but should enforce the border between the two classes.

3.1.4. WEMOTE/CWEMOTE

Regular SMOTE needs to determine the nearest neighbours for the minority class samples. This requires a time complexity of $O(n^2)$ (n - number of training instances). WEMOTE [8] is a more efficient method of generating synthetic samples. The idea is that you pick randomly two vectors corresponding to the minority class and compute their mean vector. This new vector will count as a new sample for this class. The complexity of this method is $O(n_c)$, where n_c is the number of instances that need to be added to the minority class c.

Taking into account the in-class imbalance issues, CWEMOTE clusters the training samples from the minority class and then applies WEMOTE on the new clusters. The clustering is done using k-means. After this, the number of new instances for each cluster is decided base on the following formula: $n'_j = \frac{n_i}{\sum_{j=1}^{m_c} n_j} n_c$ where n_i is the size of cluster i, m_c the number of clusters, n_c the number of new samples to be generated.

3.2. Cost-sensitive learning

3.2.1. Thresholding

Thresholding is a post-processing operation applicable for any classifier that produces probability estimates [9]. The idea behind this method is very simple: it selects the probability that minimizes the total missclassification cost on the validation instances as the threshold for predicting testing instances. When applying classifiers to predict labels, they also return a probability associated to that label, which represents the confidence of the prediction. This confidence threshold is usually 0.5 for a binary classifier for example, but when dealing with imbalanced data this threshold is not optimal and that is why the *Thresholding* helps ajust this probability. Thresholding can thus convert any non-cost sensitive learning algorithms into cost sensitive ones.

3.2.2. Cost-based direct methods

Oversampling techniques are a good approach when it comes to imbalance data, but things might get problematic when, for example the minority class is so small that the synthetic examples generated with SMOTE are not representative of the class.

Classifiers can be made cost sensitive using a squared cost matrix where each entry M_{ij} is the cost of classifying an instance with actual class i as class j. This gives the opportunity to penalize misclassification of small classes, because classifiers usually mistakes them for majority classes, which is the issue of imbalance. One advantage of using such approach is that there is no need to remove data or add synthetic samples that might negatively affect the integrity of the data. One of the most difficult tasks of this cost matrix approach is finding an optimal matrix than can best depict the misclassification costs.

Evolutionary algorithms were proposed to search for the cost-matrix that optimizes a given objective function. In [10] the authors use Particle Swarm Optimization to optimize the cost matrix in the case of binary classification while in [11, 12, 13] the authors use a Genetic algorithm to learn the misclassification costs in multi-class classification.
Learning the misclassification costs is basically a numerical optimization problem which we choose to tackle with Differential Evolution (DE) due to the good performance this meta-heuristic showed in continuous domains. The version we implement is a classical DE/rand/1/* variant. A mutant vector y_i is constructed based on selecting three random elements from the population. We use one of these as a base vector, x_{r1}, and the other two, x_{r2} and x_{r3}, are combined and added to the base vector using a scaling factor F: $y_i = x_{r1} + F \cdot (x_{r2} - x_{r3})$. The trial element z_i is obtained using a mix of the mutant y_i with the x_i element. For this we use a binomial crossover variant which constructs the trail vector in a random way, by tacking elements from the mutant vector, but also from the current element as follows:

$$z_i^j = \begin{cases} y_i^j, & \text{if } U_j < CR \text{ or } j = k \\ x_i^j, & \text{otherwise} \end{cases}$$ where U_j is uniformly generated value for each j, over $[0, 1]$.

The fitness of each individual in the population is computed by training the classifier using the costs it encodes and evaluating the performance of the model on the validation set.

In a first stage we use DE to learn the missclassification cost for each class. In a second stage, we try to improve the classifier by learning misclasification costs at instance level. Because in this second stage the search space is

huge, the initialization makes use of the matrix learned in the first stage: for each instance in the training dataset its misclassification cost is set to be the one of the class it belongs to (as optimized in the first stage) altered by adding or subtracting a random value in the range (0, (max-min)/10), where *min* and *max* are the smallest and the largest values in the cost matrix extracted in the first stage.

4. Experimental analysis

In order to study the interaction between the degree of class imbalance, the type of text representation, the resampling scheme and the classifiers we evaluate the methods both in the context of binary classification and multi-class classification.

We use 5-fold cross validation with the folds created using stratified sampling: we train on 4 of the 5 folds and test on the 5^{th}, and we do this for each combination of 4 folds, so at the end we report the results for all the instances in our corpus.

For the binary classification tasks we report the F-measure computed for the minority class as the harmonic mean between precision and recall:

$$F - measure = \frac{2 * precision * recall}{precision + recall}$$

where $precision = \frac{TP}{TP+FP}$, $recall = \frac{TP}{TP+FN}$, TP represents the number of items that are correctly classified in the minority class, FP the number of items that are incorrectly classified as belonging to the minority class, FN the number of items that belong to the minority class but are incorrectly classified in the majority class.

For the multi-class case we report the macro-average F-measure.

4.1. Classification algorithms used

As classification algorithms we chose both linear and non-linear classifiers:

- Logistic Regression;
- SVM with linear and radial kernel; the results obtained with radial kernel are not reported in the experimental section because of the inferior performance;
- Decision Tree Classifier.

In the multi-class classification setting we used for the first two classifiers the *one versus one* approach which shows superior results especially in the case of imbalanced data because it does not increase the imbalance.

The Decision Tree Classifier uses the *Gini impurity index*. No pruning was used because it is detrimental when the degree of imbalance is high.

4.2. Data

We used a corpus of 4235 work experiences related to the financial and accounting sectors. The entries are short texts (74 words on average not counting the stop-words) in Italian. The data is split into 17 classes representing subcategories of the larger sector. The distribution of the data within the classes is illustrated in Table 1.

4.3. Text pre-processing

Before beginning the experiments, some data pre-processing is required:

Class	Label	#Instances
Asset Manager	A	34
Credit Manager	B	163
Director, Finance Control	C	197
Financial director	D	213
Administrative Manager	E	751
Head of General Accounting	F	304
Head of Management Control	G	239
Financial manager	H	193
Fiscal Responsible	I	48
Head of Treasury	J	71
Administrative Specialist	K	114
Industrial Accounting Specialist	L	69
Management Control Specialist	M	1319
Financial Specialist	N	288
Internal Audit Specialist	O	22
Credit Recovery Specialist	P	147
Temporary Manager	Q	63

Table 1. Data distribution

- We iterated through all the work experiences and removed all punctuation and numbers by replacing all of them with an empty string.
- Next, the stop words were removed from the experiences.
- Then the experiences were stemmed, for a more uniform corpus. For this we used from *nltk* the SnowballStemmer[14].

4.4. Parameter settings

Text representation

For BOW and tf-idf we used a vector size of approximately 12000 (which represents the number of distinct words found in the training data) after eliminating the stopwords. For the Fasttext embedding the vector dimension was set to 50 and length of char *n-grams* were set to 2.

Classifiers

For SVM we report the results obtained with the linear kernel, with the regularization constant (C) set at 1 and tolerance set at 0.001. We use the OVO (one vs one) method when we are doing both binary and multi-class classification. Logistic Regression is also used with $C = 1$ and the classification method is OVO.

The Decision Tree Classifier uses the *Gini impurity index*, with an *unpruned* structure.

Tackling data imbalance schemes

For random oversampling we created instances in the minority class until the size of the majority class was reached. For SMOTE we used 5 nearest neighbours to construct the synthetic samples and for SMOTE-SVM we call the SVM classifier that is used to find the support vectors with identical parameters as the SVM classifier used for final classification. WEMOTE has no specific parameters while in the CWEMOTE we choose the number of clusters dynamically by using Silhouette Width (SW): kMeans is run severeal times on the training folds varying the number of clusters and the partition with the highest SW score is used.

Cost-sensitive learning

The parameters of the DE algorithm used for deriving the costs of misclasification are: F= 0.4717 and CR = 0.8803. In the first stage, when the misclassification cost at the class level is optimized with the DE algorithm, the population size was set to 20 and the number iterations to 30. In the second stage, when the costs are refined at data instance level, a larger population is required and was set to 50, for a number of 30 iterations. Macro-average F-measure is

used as fitness function.
Each experiment is repeated for a number of 6 times, which, in conjunction with 5-fold cross validation, generate 30 values for each dataset.

4.5. Results for binary classification using resampling

In order to study the influence of the degree of imbalance on the performance of the evaluated methods, we chose the largest class (Management Control Specialist) in conjunction with the smallest one and other two of intermediate sizes to form 3 classification problems that correspond to higher and lower imbalance:

- *Management Control Specialist* vs. *Internal Audit Specialist* (M vs. O) corresponds to the highest imbalance with a ratio IR=1319/22=59.95;
- *Management Control Specialist* vs. *Financial Specialist* (M vs. N) corresponds to an imbalance ratio IR=1319/288=4.57;
- *Management Control Specialist* vs. *Administrative Manager* (M vs. E) corresponds to an imbalance ratio IR=1319/751=1.75.

Figures 1, 2 and 3 present the F-measure recorded for the minority class. We dropped off the results obtained with Glove and doc2vec representations which were significant worst compared to the others.

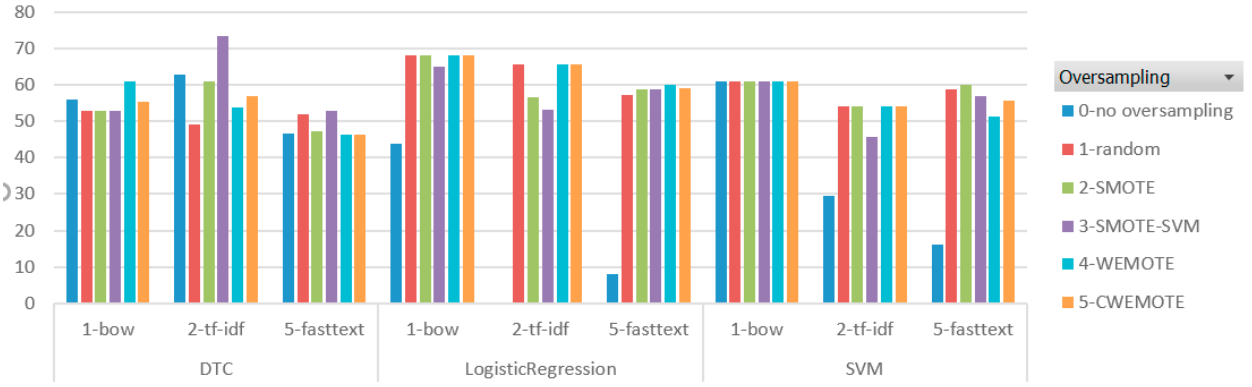

Fig. 1. Classification results for the M (1319 instances) vs. O(22 instances) pair of classes: F-measure obtained for the minority class

Fig. 2. Classification results for the M (1319 instances) vs. N(288 instances) pair of classes: F-measure obtained for the minority class

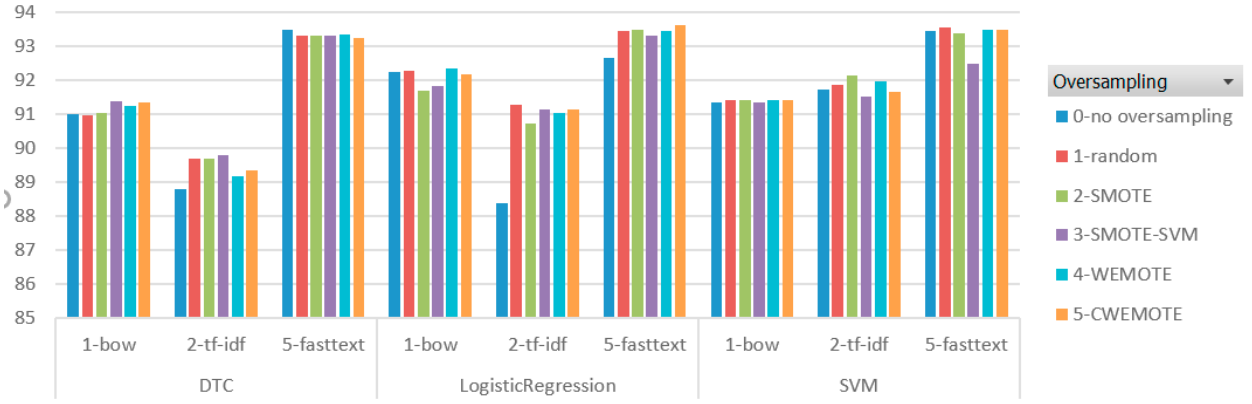

Fig. 3. Classification results for the M (1319 instances) vs E(751 instances) pair of classes: F-measure obtained for the minority class

4.5.1. The effect of data imbalance on the classifiers

It is obvious that, generally, the highest the degree of imbalance is, the highest the error is.
Regarding the performance of the classifiers on the original unmodified (not resampled) data (blue bars within the figures, labeled as "no oversampling"), the decision tree gives the best results on the first two datasets, which indicates that it is the least biased from our classifiers in the case of highly imbalanced data. Comparing the linear classifiers, Logistic regression is the most affected by imbalance, while SVM achieves better results which can be explained by the principle of structural risk minimization behind it. It is surprising that, although Logistic regression behaves the worst from all classifiers at high imbalance for every type of text representation, the effect is extreme when it is used in conjunction with tf-idf: for the first dataset all the instances are classified in the majority class.

4.5.2. Degree of imbalance over text representation performance

Regarding the impact of the text representation method, Fasttext (and implicitly word2vec) shows the highest dependence on the degree of imbalance: while for the first case corresponding to the highest imbalance, Fasttext word embeddings record the worst results, it is obvious how the performance increases with the decrease of imbalance, obtaining clearly the best results for the third data set corresponding to low imbalance, for all classifiers. Comparing the performance of the two text representations which are mainly based on word frequencies, it's surprising that for the linear classifiers (Logistic regression and SVM) BoW surpasses the more complex tf-idf method; this is also observed in the case of the decision tree for the second and third dataset.

4.5.3. The efficiency of oversampling schemes

Over-sampling schemes bring a clear advantage in case of the algorithms that suffer most from the imbalance - Logistic regression the most followed by SVM. Although Logistic regression gives the worst results when applied on the original highly imbalanced data, when oversampling schemes are used it shows to outperform all the other methods for every type of embedding. SVM with tf-idf and fasttext embeddings also benefit from oversampling while for the BoW embedding oversampling does not make a significant difference. In case of Decision trees, which show the least bias towards the majority class, the oversampling schemes do not guarantee increase in performance for the highly imbalanced datasets (first two cases), on contrary, most of the times recording worse results.
Regarding the comparative performance of the oversampling schemes, there is not a clear winner. Analyzing their performance on the linear classifiers where they are most efficient, simple random oversampling seem to work as well as other more complex schemes.

4.6. Results on Multi-class classification using resampling

Figure 4 illustrates the performance of the evaluated methods in case of multi-class classification, where all 17 classes are used in the analysis. Regarding the performance on the original imbalanced data, generally the same

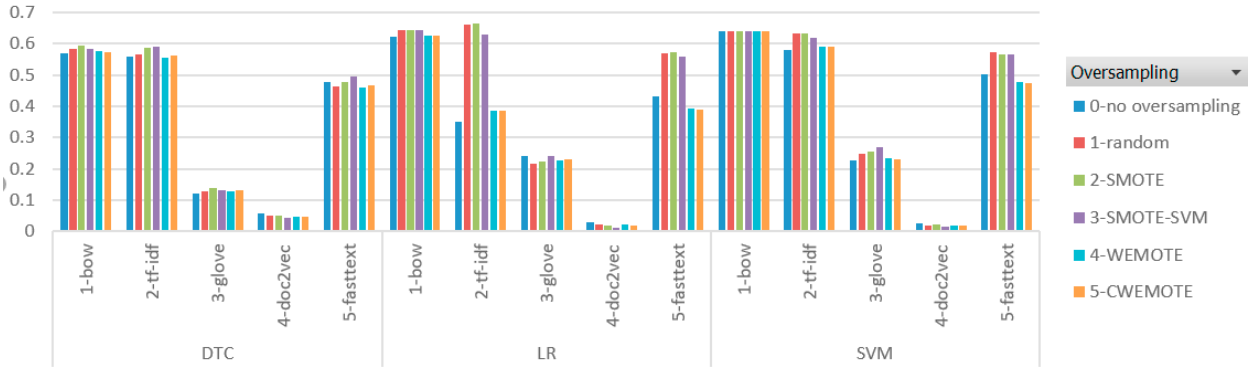

Fig. 4. Macro-average F-measure for the classification of the work experiences dataset consisting of 17 classes

observations as in the case of binary classification hold: Logistic regression is the most affected classifier from the imbalance in data when used with tf-idf and fasttext vectorizations. When using BoW, both linear classifiers surpass the decision tree, with SVM as winner. BoW is the winning vectorization type in case of all classifiers. This contardicts the last experiment in the binary classification section, where fasttext achieved the best results, but it is explained by the fact that the M-E pair of classes had lower imbalance, while the binary classifiers behind OVO classifiers (Logistic regression and SVM) still have to deal with many imbalanced datasets.
All classifiers seem to benefit to a given extent by oversampling, with highest increase in performance for the linear classifiers using tf-idf and fasttext vectorizations. Simple oversampling, SMOTE and SMOTE-SVM show the best, comparable performance.
The poor results obtained by glove and doc2vec embedding may be explained by the relatively reduced size of our datasets but also by the length of the texts in our corpus which consists of very short documents. Embeddings obtained with neural networks are very powerful tools in text classification, but they need a larger amount of data to learn context or word n-grams from. That's why they work better with more balanced and larger classes.

4.7. Results for cost-sensitive classification with DE

The goal of employing a cost-sensitive approach is to outperform the results we obtained with resampling methods for the case of multi-class classification, the problem we have to solve in practice. With this aim, analyzing the results obtained in previous experiments, we decided to use further the TF-IDF representation, which gave the best results so far.
Figure 4.7 illustrates comparatively the best results obtained using resampling versus the results obtained using cost-sensitive classification based on DE for the multi-class classification case, in both steps: when miss-classification costs are inferred at the class level (denoted as *cost-sensitive 1* in the figure) and when these are further optimized at the data instance level (*cost-sensitive 2*). There is an evident increase in performance when employing the cost-sensitive approach over the best resampling schemes, in the case of all the classification algorithms.The use of costs at the level of data instances brings small improvements over the use of costs at class level and only for logistic regression and SVM. The experiments show that SVM, a classifier aiming at minimizing the structural risk, benefits the most from deriving weights at data instance level.

5. Conclusion

Based on the experiments we have done we can say that simple text representations such as *BoW* or *TF-IDF* work better for small sets with large imbalance, rather than a more complex embedding such as Glove, doc2vec or even

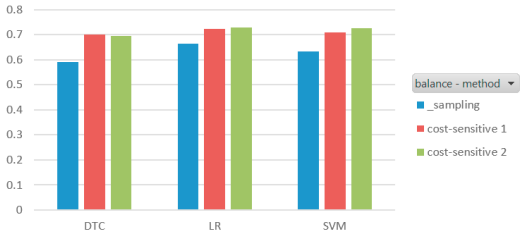

Fig. 5. Best F-measure scores using sampling methods compared to the scores obtained using the cost sensitive approach based on Differential Evolution: *cost-sensitive 1* corresponds to the first level of optimization when costs are derived at the class level and *cost-sensitive 2* corresponds to the second step when costs are derived at the data instance level

FastText. Regarding the classifiers we used, both linear ones (Logistic regression and SVM with linear kernel) are more biased in the context of high imbalance compared to decision trees and this phenomenon is more pronounced when using TF-IDF representation and less when using BoW. Oversampling schemes are necessary in the context of the linear classifiers when working with TF-IDF and Fasttext vectorizations. SVM with the BoW vectorization does not seem to be affected by the imbalance in data.

When comparing the resampling schemes with the cost sensitive approach where the misclassification costs are learned by a Differential Evolution algorithm, the cost-sensitive approach clearly give the best results in terms of F-measure.

References

[1] V. López, A. Fernández, J. G. Moreno-Torres, and F. Herrera, "Analysis of preprocessing vs. cost-sensitive learning for imbalanced classification. open problems on intrinsic data characteristics," *Expert Systems with Applications*, vol. 39, no. 7, pp. 6585–6608, 2012.

[2] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[3] J. Han Lau and T. Baldwin, "An empirical evaluation of doc2vec with practical insights into document embedding generation," 07 2016.

[4] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[5] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, 2017.

[6] V. López, A. Fernández, S. García, V. Palade, and F. Herrera, "An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics," *Information Sciences*, vol. 250, pp. 113–141, 2013.

[7] N. Chawla, K. Bowyer, L. O. Hall, and W. Philip Kegelmeyer, "Smote: Synthetic minority over-sampling technique," vol. 16, 01 2002.

[8] B. L. Q. L. J. X. Tao Chen, Ruifeng Xu, "Wemote - word embedding based minority oversampling technique for imbalanced emotion and sentiment classification," 2013.

[9] V. S. Sheng and C. X. Ling, "Thresholding for making classifiers cost-sensitive," in *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, ser. AAAI'06. AAAI Press, 2006, pp. 476–481. [Online]. Available: http://dl.acm.org/citation.cfm?id=1597538.1597615

[10] P. Cao, D. Zhao, and O. Zaiane, "An optimized cost-sensitive svm for imbalanced data learning," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2013, pp. 280–292.

[11] Y. Sun, M. S. Kamel, and Y. Wang, "Boosting for learning multiple classes with imbalanced class distribution," in *null*. IEEE, 2006, pp. 592–602.

[12] T. Perry, M. Bader-El-Den, and S. Cooper, "Imbalanced classification using genetically optimized cost sensitive classifiers," in *Evolutionary Computation (CEC), 2015 IEEE Congress on*. IEEE, 2015, pp. 680–687.

[13] C. Lemnaru and R. Potolea, "Evolutionary cost-sensitive balancing: A generic method for imbalanced classification problems," in *EVOLVE-A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation VI*. Springer, 2018, pp. 194–209.

[14] E. L. Bird, Steven and E. Klein, *Natural Language Processing with Python. O'Reilly Media Inc.*, 01 2009.