WhatsOut

Nicolò Verardo

Università degli Studi di Milano, Milano, Italia
nicolo.verardo@studenti.unimi.it
nicoloverardo.surge.sh

Abstract. In this work we address a multilabel classification task on a dataset of activities and events. Although the dataset is made available offline, we will address the requirement to implement a solution that will work in an online learning context. We will perform text cleaning and other manipulations on the dataset. Then, we will train KNN and Random Forest as if we were in an offline context, and then SVM, Neural Network and our own implementation of the Matrix Regression algorithm. We will compare the cross-validated F1-score and we will show that the imbalancement in the data will make our models not capable of spotting the least frequent categories.

Keywords: text-classification · multilabel-classification · online-learning

1 Introduction

Classification is a problem of identifying which of the target categories a data sample belongs to. Training data consist of a set of features and an associated target class or class label. The most common type of classification is binary classification: each of the data sample belongs or not to the only one target label. If we have many labels and each data sample belongs still to only one of the labels, we are in the multiclass classification case. However, in real world applications, there may be several cases in which each data sample belongs to more than one target labels at the same time: this is the case of multilabel classification.

If we are in an online learning context, where data comes in a stream or where we might need to update our algorithm, standard machine learning algorithms may not provide a good answer to this request.

In this work we are facing a multilabel classification problem where the labels are not independent among themselves, where there is imbalancement in the data and where and our goal is to label data samples with one or more categories.

2 Research question and methodology

2.1 The dataset

Each record in the dataset represents an event, identified by its own ID. Each event has one title, description and author but it can be associated to multiple dates, places and categories. We have a total of 10.760 events belonging to one or more of the 15 categories and 12 feature columns. Data are highly unbalanced: there are categories with very few examples and others with many examples, as we can see in the picture below.

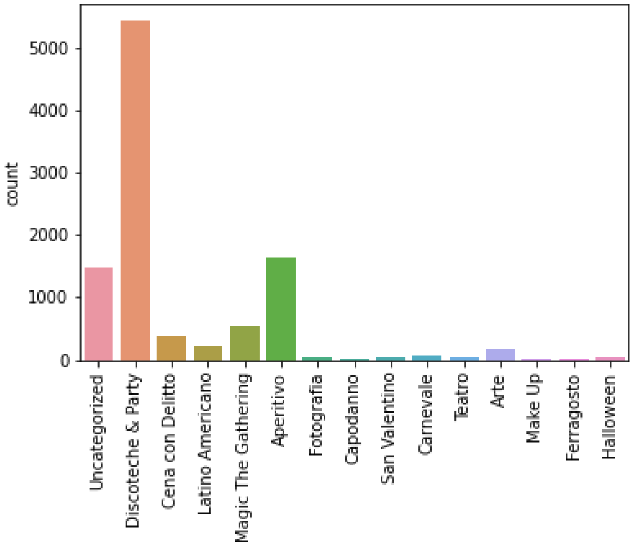

Fig. 1. Categories distribution

2.2 Data pre-processing

We proceeded to clean the corpus. We removed unuseful information like punctuation, numbers, symbols, urls, stopwords using the `nltk` package; moreover, every word was converted to lowercase. Then, we tokenized the text in unigrams and we created document-feature matrices using both Bag of Words (BoW) and Term frequency-inverse document frequency (TF-IDF) representations. In the BoW representation the importance of a term is measured simply by counting how many times the word is present in the document (we will use the term *document* or *event* interchangeably). On the contrary, with the TF-IDF, we consider a word important if it is peculiar to a specific document, but not to the rest of the corpus. We computed the relative frequency of a term as:

$$\mathrm{TF} = \frac{f_{ij}}{\sum_j f_{ij}} \tag{1}$$

where f_{ij} is the number of occurrences of term j in document i, while $\sum_j f_{ij}$ is the sum of the occurrences of the term in all documents.

Instead, the inverse document frequency was computed as:

$$\text{IDF} = \log\left(\frac{\text{n}^\circ\text{ of documents}}{\text{n}^\circ\text{ of documents containing term}}\right) + 1 \tag{2}$$

Thus, the TF-IDF is finally computed as:

$$\text{TFIDF} = \text{TF} \times \text{IDF} \tag{3}$$

We merged the title with the description of the event in order to have a single textual feature, that we then represented using BoW or TF-IDF as specified above. We filtered out events that are not written in the italian language using the `polyglot` package. Since some records are the same in every feature but differ only for the label, we merged them by ID and title in order to end up with a multilabel dataset. We created four datasets with different representations:

1. BoW on the full text
2. BoW on the full text + one hot encoding of `place` and `user`
3. TF-IDF on the full text
4. TF-IDF on the full text + one hot encoding of `place` and `user`

We end up with a dataset of 8.971 events with three features of interest, namely: the id of the author of the event, the id of the place of the event and the full text (description and title) of the event.

In order to evaluate the results, we will use mainly the F1-score since our data is very unbalanced and therefore we cannot rely on accuracy. The F1-score is the harmonic mean of precision and recall:

$$\text{F1} = 2 \cdot \frac{(\text{precision} \cdot \text{recall})}{(\text{precision} + \text{recall})} \tag{4}$$

where *precision* can be seen as “the ability of the classifier not to label as positive a sample that is negative”:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{5}$$

and *recall* can be seen as “the ability of the classifier to find all the positive samples”:

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{6}$$

2.3 K-Nearest Neighbor

Given a training set S, the k-nearest neighbor (knn) algorithm generates a classifier that predicts every point in the training set with its own label and predicts any other point with the label of the point in the training set which is closest to it.

If there is more than one point in S with smallest distance to x, then the algorithm predicts with the majority of the labels of these closest points. If there is an equal number of closest points with same label, then the algorithm predicts a default value among the labels.

For a multiclass classification problem, the prediction is – just like the previous case – the label corresponding to the majority of the k closest data points. For a regression problem, instead, the prediction can be defined as the average target of the k nearest neighbors.

This classification is done through some notion of distance given by a metric function. It can be, for example, the Euclidean distance:

$$||\boldsymbol{x} - \boldsymbol{x}_t|| = \sqrt{\sum_{i=1}^{d}(x_i - x_{i,t})^2} \tag{7}$$

The NN algorithm has some problems:

- It is not very *scalable*: since it stores the entire training set, if the latter is huge, then a huge amount of memory is required.
- It is *computationally inefficient*: computing distances for each point in a huge training set may require some time.

2.4 Support Vector Machines

SVM is a learning algorithm for linear classifiers which, once fixed a linearly separable training set $(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_m, y_m) \in \mathbb{R}^d \times \{-1, +1\}$, generates the linear classifier corresponding with the single solution $\boldsymbol{w}^* \in \mathbb{R}^d$ of the following optimization convex problem with linear constraints:

$$\min_{w \in \mathbb{R}^d} \frac{1}{2}\|\boldsymbol{w}\|^2 \quad \text{s.t. } y_t \boldsymbol{w}^T \boldsymbol{x}_t \geq 1, t = 1, ..., m \tag{8}$$

where $\boldsymbol{w}^*$ geometrically represents the separating hyperplane with maximum margin. The *support vectors* are those $\boldsymbol{x}_t$ on which $\boldsymbol{w}^*$ has margin equal to 1, that is $y_t(\boldsymbol{w}^*)^T \boldsymbol{x}_t = 1$.

In the non linearly separable case, the SVM problem becomes:

$$\min_{w \in \mathbb{R}^d} \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{m}\sum_{t=1}^{m} h_t(\boldsymbol{w}) \tag{9}$$

where $h_t(\boldsymbol{w}) = \left[1 - y_t \boldsymbol{w}^T \boldsymbol{x}_t\right]_+$ corresponds to the hinge loss. A regularization parameter λ is introduced in order to balance the two components. Also in this case, as for the linearly separable case, the solution $\boldsymbol{w}^*$ belongs to the subspace of the linear combinations of example of training set multiplied by their labels.

In order to solve the above minimization problem, we can apply OGD, that is the online version of the Stochastic Gradient Descent.

It is worth mentioning that SVM does not natively support multilabel classification. Moreover, reducing the problem to a one-vs-all classification would have implied treating each category as independent from the others, and this is not true in reality. Thus, we exploited the Classifier Chain problem transformation of the `scikit-multilearn` package. This method constructs a bayesian conditioned chain of per label classifiers; for L labels it trains L classifiers ordered in a chain according to the Bayesian chain rule. The first classifier is trained just on the input space, and then each next classifier is trained on the input space and all previous classifiers in the chain in order to retain the dependency among labels.

2.5 Random Forest

Random forest is an ensemble method. Ensemble methods are meta-algorithms that generate multiple weak predictors on the same training data. These predictors are then combined into a single, more powerful, predictor. Adding randomization and then combining predictors has an averaging effect on the performance of the learning algorithm, which corresponds to reducing variance and increasing bias. For this reason, ensemble methods are often used with non-parametric algorithms with little bias, such as tree predictors.
Random forest uses only decision trees as base algorithms. The split of a node is the best split over a random subset of the features and no longer the best split among all features. Thus, the bias increases, but the decrease in variance usually compensates for it, hence yielding an overall better model. The reason lies in the fact that while some trees may be wrong, many other trees will be right; therefore, considering them as a group, helps us to move towards the correct prediction.

2.6 Neural Networks

An artificial neural network is a model of computation inspired by the structure of neural networks of the brain. It receives an input, changes its internal state (activation) according to that input, and produces an output depending on the input and activation. A neural network can be described as a directed graph whose nodes correspond to neurons and edges correspond to links between them. Each neuron receives as input a weighted sum of the outputs of the neurons connected to its incoming edges.
Neural networks are structured in layers, which are sets of nodes. The first layer is called *input layer*, those in the middle of the network are called *hidden layers* and their nodes *hidden nodes* (or *hidden units*), since we do not directly observe them during training; the final one is called *output layer*. A neural network that has more than one hidden layer is called *deep neural network*.
There are several activation functions. Among the most used ones we can find

the Sigmoid, the Softmax and the ReLU.

There exists two types of neural networks: feedforward and feedback neural networks. Feedforward neural network is a non recursive neural network. Neurons in this layer are only connected to neurons in the next layer, and they do not form a cycle. In feedforward signals only travel in one direction, i.e.: towards the output layer.
Feedback neural networks contain cycles. Signals travel in both directions by introducing loops in the network. They implement a two-pass process called *back-propagation*: first, current weights are fixed and the predicted values are computed (forward pass); secondly, errors are computed and then back-propagated via back-propagation equations (backward pass).
Feedback cycles can cause the network's behavior to change over time based on its input.

Although neural networks have proven to be very powerful in many scenarios, they are still subject to some problems.
Bad starting values for the weights, for instance, may lead to bad performance of the network. If we set starting values that are exactly zero, then the algorithm never moves; on the contrary, using too large initial values for the weights may lead to poor solutions.
Neural networks may also suffer from overfitting: using too many weights for our model is one of the causes.

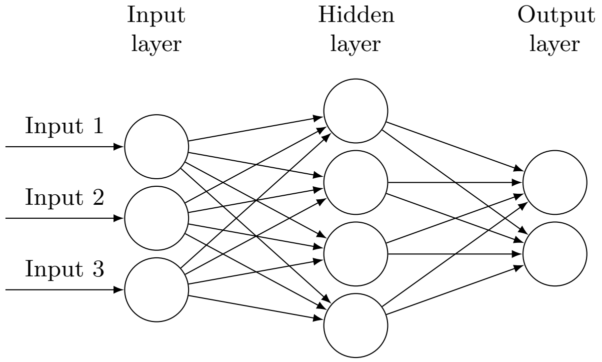

Fig. 2. A feed-forward neural network with a single hidden layer

2.7 Matrix Regression

Presented in [1], it is an attempt to cover a multilabel classification problem where the documents belong to several categories and the number of possible categories is very high. Moreover, this algorithm is incrementally updatable, so it can be used in an online learning context.

It consists of a matrix W where each row is a unique word of the corpus of the events, and each column is a category:

$$W = \begin{bmatrix} w_{11} & w_{12} & ... & w_{1C} \\ w_{21} & w_{22} & ... & w_{2C} \\ ... & ... & ... & ... \\ w_{T1} & w_{T2} & ... & w_{TC} \end{bmatrix} \tag{10}$$

where T is the set of unique words, C is the set of categories and w_{ij} is a sum of the tf-idf weight of term i and it is initially set to zero. For each event in the training collection, we obtain the list of categories with which the event is labeled. For each term of the current event vector and for each category we identify the associated counter w_{ij} in the matrix. We increment the counter with the tf-idf weight of the term.

In the prediction phase, the output of the algorithm is a vector of length C where the elements correspond to weights associated to each possible category. The choice of the categories is then made through a threshold value that is used to filter them. Since the authors do not specify a fixed threshold or a fixed solution to compute it, we decided to implement our own simple strategy. First, we scale the values of the output vector to be in the range $[0, 1]$. Then, we have two cases: in the first case, we let the user choose the threshold manually; otherwise, the threshold is automatically chosen as the median of the vector, since it represents the center of the mass distribution. Thus, the predicted categories are those that have the associated weight greater than the threshold value.

We implemented it from scratch in python and the code has been made publicly available on GitHub [4].

3 Experimental results

The experiments were run on a free Google Colab instance running Ubuntu that provided a quad-core Intel Xeon E5-2650 v3 that run at 2,30 Mhz and 25GB of RAM. Moreover, it provided also a NVIDIA Tesla K80 as GPU that we used to train the Neural Networks using `keras` with `Tensorflow` as backend.
We set a random seed to ensure the reproducibility of the results.

We did a stratified split of the four datasets mentioned in the previous section, reserving 30% of the records as test set. We ran a 5-fold cross-validation for each algorithm on each of the training sets (except for Matrix Regression, that could only use the TF-IDF representation) and we computed the F1-score in order to evaluate the final results. SVM, KNN, and Random Forest were run without altering the default parameters' values in their `scikit-learn` implementation. The Neural Network architecture consisted only of dense layers: starting with an input of 1.024 neurons, then we gradually reduced the number of neurons by powers of two in the four hidden layers with `ReLU` activation, using a dropout of 0.2 between each hidden layer. The output layer has the number of neurons equal to the number of categories. We used `adam` as optimizer, `categorical_crossentropy` as loss, 20 epochs and a batch size of 64. We also implemented early stopping in order to reduce overfitting.

In the table below we report the main results. These are F1-scores computed on the test set for each best classifier resulted from the cross-validation:

Table 1. F1-score results on the test set

Dataset	KNN	Random Forest	Neural Network	SVM	MR
BoW	78,82%	82,34%	**84,86%**	81,94%	-
BoW + place + user	79,56%	82,51%	84,00%	81,95%	-
TF-IDF	**81,71%**	**83,19%**	82,71%	**86,51%**	**59,20%**
TF-IDF + place + user	70,49%	75,80%	73,45%	61,50%	-

We can see that we achieved the highest score almost every time using the TF-IDF representation. This can be due to the nature of the TF-IDF itself: the IDF works as a weight for the words. In this way, words that are very specific to a document that belongs to a category, but not for the rest of the documents that belong to other categories, are more important for the choice of predicted category/ies.

However, we should not conclude that our prediction capabilities have such a strong impact, because for some of the categories both precision and recall are zero, as we can see in the table below:

Table 2. Classification report for Random Forest

label	precision	recall	f1-score
0	0.86	0.74	0.79
1	1.00	0.51	0.67
2	0.00	0.00	0.00
3	0.00	0.00	0.00
4	0.98	0.90	0.94
5	0.98	0.95	0.96
6	0.00	0.00	0.00
7	1.00	0.43	0.60
8	0.00	0.00	0.00
9	0.92	0.66	0.77
10	0.96	0.98	0.97
11	0.00	0.00	0.00
12	0.00	0.00	0.00
13	1.00	0.20	0.33
14	0.84	0.91	0.87

This is caused by the imbalancement in the data. Those labels that have zero precision and/or recall are the categories that have very few examples. This means that the Random Forest algorithm is very powerful on some labels but very weak on others, and this results in a biased prediction since we will be able to spot only those examples that belong to the majority category.
This behaviour is displayed by all the algorithms we tested, except for Matrix Regression.

By construction, the Matrix Regression algorithm is able to predict one or more categories at the same time for a document. This algorithm showed a high recall (99%) but a very low precision (around 15%). Having a high recall means that we are able to find all the categories to which a document belongs, while having an high precision means being able to predict a reasonable number of categories. Thus, there is a trade-off between precision and recall, that we managed to control by choosing the most suitable value of the threshold. In fact, changing the threshold value increased the precision but lowered the recall. This behaviour was expected and also reported by the original authors.

It is worth mentioning that we also tried to run these algorithms on: a TF-IDF representation that included bigrams; the randomly oversampled dataset (with SVM only); the lemmatized text; the stemmed text. Since the least frequent category had less then five documents, oversampling was not efficient at all and didn't provide any improvement. In fact, the results of all the above cases are not reported here since they did not provide any significant change to the initial results.

4 Concluding remarks

Multilabel classification where there is dependency among the categories is not an easy problem. Moreover, applying it to an online learning context, makes it even harder. In fact, this kind of problem is still being actively researched by the major tech companies. Results were not satisfactory in terms of F1-score but we expected that because of the imbalancement of the data, that made this an even more difficult problem to solve.
The first attempt that can be made in a future work in order to improve the results is to collect more events of the least frequent categories. If data are available, this should be the simplest and best way to achieve a satisfactory high accuracy in prediction. Other attempts may include trying a different cleaning of the text corpus or different architecture of the neural network (even though it is not the best algorithm in this situation). Moreover, an exhaustive parameter GridSearch could improve the results.
Regarding Matrix Regression, at the time of writing, the implementation made for this work still needs improvement, especially on the matter of concerning computational speed (i.e.: parallelization) or, more importantly, in the strategy of the treshold value' choice.

There is no best or correct solution in this case: KNN and Random Forest do not support partial fit, so they cannot be used in an online learning context; expanding the set of categories for the neural network and SVM is tricky. That being said, we propose Matrix Regression as the most suitable algorithm for this dataset since it has all the needed requirements.

References

1. Popa, I. & Zeitouni, Karine & Gardarin, Georges & Nakache, Didier & Métais, Elisabeth. (2007). Text Categorization for Multi-label Documents and Many Categories. 421 - 426. 10.1109/CBMS.2007.108.
2. Trevor Hastie, Robert Tibshirani, Jerome Friedman. The Elements of Statistical Learning. Springer Series in Statistics, 2017.
3. Shalev-Shwartz, Ben-David - Understanding Machine Learning. Cambridge University Press, 2014.
4. Matrix Regression, http://github.com/nicoloverardo/matrix-regression