Multi-classification Text mining using Žurnal24 articles

Haris Berbić

No Institute Given

1 Introduction

Text mining is automatic and semi-automatic extraction of implicit, previously unknown, and potentially useful information and patterns, from a large amount of unstructured textual data, such as natural-language texts. One of the main challenges in text mining is to classify textual data with such high dimensionality. In addition to high dimensionality, text-mining algorithms should also deal with word ambiguities such as pronouns, synonyms, noisy data, spelling mistakes, abbreviations, acronyms and improperly structured text. Text mining algorithms are two types: Supervised learning and unsupervised learning. In our research we used supervised learning with sci-kit package and supervised deep learning with tensorflow package.

2 Feature extraction

Text collections contain millions of unique terms, which make text-mining process difficult. Therefore, feature-extraction is used when applying machine- learning methods like Logistic Regression to text categorization. A feature is a combination of attributes (keywords), which captures important characteristics of the data. A feature extraction method creates a new set of features far smaller than the number of original attributes by decomposing the original data.

Unsupervised algorithms like Principal Components Analysis (PCA), Singular Value Decomposition (SVD), and Non-Negative Matrix Factorization (NMF) involve factoring the document-word matrix, based on different constraints for feature extraction. In this task TruncatedSVD is used for feature extraction. The goal of text classification is to construct a model as a function of the values of the attributes using training examples/documents, which would then predict the target values of the new examples/documents as accurately as possible.

3 Implementation

The first step in text mining is **text gathering**. The process of text gathering from Žurnal24[Ref] is done with BeautifoulSoup Python Package. The Žurnal24 dataset contains 3965 articles in 5 categories.

Text **preprocessing** , as the second step, involves extracting the actual text context, loading web data into text data, document indexing and feature extraction. In order to load the web documents into text data , the folder for each category have to be created for saving documents for each category. **Document Indexing** is normally done using following steps: tokenization, filtration, stemming, and weighting. In this work following techniques were used:

Tokenization means text is parsed, lowercased and all punctuations are removed. For tokenization Tweet tokenizer from sci-kit library is used.

Filtration refers to the process of deciding which terms should be used to represent the documents so that these can be used for describing the document's content, discriminating the document from the other documents in the collection, and frequently used terms or stop words (is, of, in ...) are removed from term streams. In this work Slovenian stop word were used for filtration

Lemmatization, reduces the inflected words properly ensuring that the root word belongs to the language. In Lemmatization root word is called Lemma. A lemma (plural lemmas or lemmata) is the canonical form, dictionary form, or citation form of a set of words. Slovenian lemmatizer was used for this task.

Weighting means terms are weighted according to a given weighting model. Count Vectorization (One hot encoding) and TF- IDF Vectorizer were used in this task.

The **class imbalance** problem typically occurs when, in a classification problem, there are many more instances of some classes than others. In such cases, standard classifiers tend to be overwhelmed by the large classes and ignore the small ones." Since we are working with imbalanced classes three different techniques were used: SMOTE (Synthetic Minority Over-Sampling Technique), RandomOverSampler for oversampling and RandomUnderSampler for downsampling

In this task TruncadedSVD algorithm was used for **dimensionality reduction**. Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables. It can be divided into feature selection and feature extraction.

The third step in text mining is **data analysis**, which was done using the logistic regression algorithm (LR) and Stochastic Gradient Descent(SGD) algorithms.

The fourth step in text mining is **visualization**. The extracted information is shown to users by making tables.

The final step is **text evaluation**, which is accomplished by applying the models on test data sets.

4 Data Optimal parameter search procedure on sci-kit models

Building classification models requires optimizing certain parameters to arrive at the best accuracy. A typical approach to exploring alternative model configurations is by using what is commonly known as a grid search. Although fairly

simple and straightforward to carry out, a grid search is quite costly because expense grows exponentially with the number of hyperparameters and the number of discrete levels of each. To tackle this problem, we build and process pipelines in the following procedure:

The **first set** of pipelines represents classification algorithms with different weighting. The best pipeline is chosen and class imbalance techniques were added resulting in **the second set** of pipelines. If better model is found, the best pipeline is forwarded to dimensionality reduction pipeline, otherwise previous best model is used and forwarded to dimensionality reduction as **the third set** of pipelines. The result of finished procedure is model (pipeline) with all parameter optimized and the best accuracy achieved.

The following list is the complete set of parameters used in searching for the best model:

```
TfidfVectorizer n-gram:
max_df : 0.25, 0.5, 0.75, 1.0
min_df: 2, 3
n-gram_range: (1,2), (2,3)
smooth_idf': True, False
norm': l1, l2, None]
max_features': 1000, 5000, 10000

TfidfVectorizer word:
analyzer: word
max_df : 0.25, 0.5, 0.75, 1.0
min_df: 2, 3
n-gram_range: (1,2), (2,3)
smooth_idf': True, False
norm': l1, l2, None]
max_features': 1000, 5000, 10000

TfidfVectorizer char:
analyzer: char
max_df : 0.25, 0.5, 0.75, 1.0
min_df: 2, 3
n-gram_range: (1,2), (2,3)
smooth_idf': True, False
norm': l1, l2, None]
max_features': 1000, 5000, 10000

SMOTE imbalance:
k_neighbors: 1, 5, 10

RandomOverSampler:
Default values
```

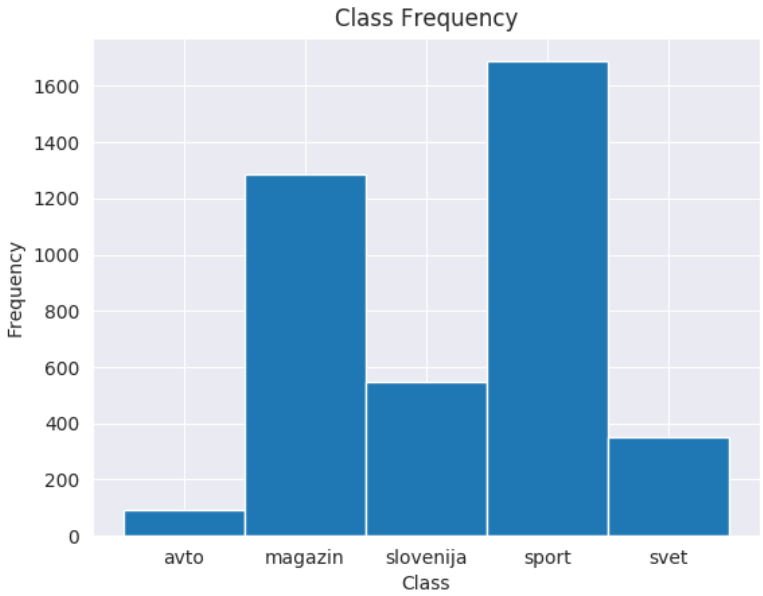
```
RandomUnderSampler:
Default values

Truncated SVD:
n_components: 10, 50, 100, 500
n_iter: 10, 20

Logistic Regression:
C: 1.0, 0.1, 0.01
Loss : log
Penalty: l2
Max_iter: 500, 100
Tol: 1e-3, 1e-4
Solver: lbfgs
Multi_class: multinomial

Stochastic gradient descent
C: 1.0, 0.1, 0.01
Loss : log
Penalty: l1, l2
Alpha: 1e-6, 1e-3
Max_iter: 500, 100
Tol:  1e-3, None
Eta0: 0.1, 0.001
```

4.1 Experiments and results

The total data set was divided into two subsets. One subset is used for training the algorithm and the other subset is used for testing the algorithm. 70% of the total data is used for training and the remaining 30% percent of the data is used for testing. Some details of the data set are given below:

Total documents: 3965

Approximate number of words in each document: 321



4.2 Results of Text mining with LR and SGD

LR results with CountVectorizer: accuracy: 90.42

```
Vectorizer settings: analyzer=word,
max_df=1.0, max_features=None, min_df=1, ngram_range=(1, 1)
Model settings: C=1.0, max_iter=100, penalty='l2', tol=0.0001
```

LR results with TfidfVectorizers (word): accuracy: 88.82

```
Vectorizer settings: analyzer='word',
max_df=1.0, max_features=None, min_df=1, ngram_range=(1, 1),
norm='l2', smooth_idf=True, token_pattern='\\w{1,}')
Model settings: C=1.0, max_iter=100, penalty='l2', tol=0.0001
```

LR results with TfidfVectorizers (word): accuracy: 77.47

```
Vectorizer settings: analyzer='word',
max_df=1.0, max_features=None, min_df=1, ngram_range=(1, 1),
norm='l2', smooth_idf=True, token_pattern='(?u)\\b\\w\\w+\\b')
Model settings: C=1.0, max_iter=100, penalty='l2', tol=0.0001
Accuracy: 90.33%
```

SG results with CountVectorizer: accuracy: 91.00

```
Vectorizer settings: analyzer='word', max_df=1.0,
max_features=None, min_df=1, ngram_range=(1, 1),
token_pattern='\\w{1,}'
Model settings: alpha=0.0001, eta0=0.0,  max_iter=1000, penalty='l2', tol=0.001,
```

SG results with TfidfVectorizers (word): accuracy: 86.72

```
Vectorizer settings: analyzer='word', max_df=1.0,
max_features=None, min_df=1, ngram_range=(1, 1), norm='l2',
smooth_idf=True, token_pattern='\\w{1,}')
Model settings: alpha=0.0001, eta0=0.0,  max_iter=1000, penalty='l2', tol=0.001,
```

SG results with TfidfVectorizers (word): accuracy: 90.50

```
Vectorizer settings: analyzer='word', max_df=1.0,
max_features=None, min_df=1, ngram_range=(1, 1), norm='l2',
smooth_idf=True, token_pattern='(?u)\\b\\w\\w+\\b')
Model settings: alpha=0.0001, eta0=0.0,  max_iter=1000, penalty='l2', tol=0.001,
```

The best chosen model is SGD with CountVectorizer (91.4%). In next step imbalance learning is performed.

SGD results with CountVectorizer and SMOTE: accuracy: 90.00

```
SMOTE settings: k_neighbors: ?
```

LR results with CountVectorizer and RandomOverSampler: accuracy: 92.18

```
RandomOverSampler settings: defualt
```

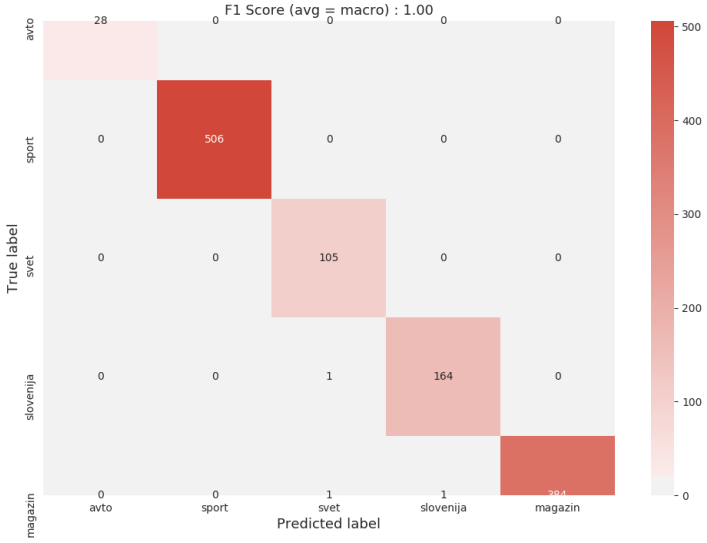
LR results with CountVectorizer and RandomUnderSampler: accuracy: 87.56

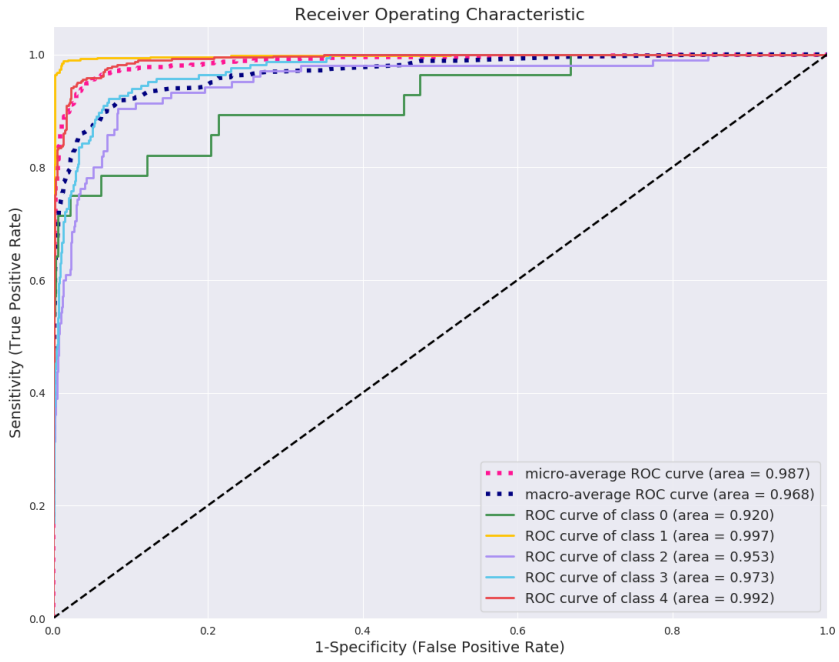
```
RandomUnderSampler settings: default
```

The best chosen model is SGD with TfidfVectorizers and RandomOverSampler (92.18%) , In next step dimensionality reduction is performed.

LR results with CountVectorizer and SMOTE and TruncatedSVD: accuracy: 99.50%

```
SVD settings: n_components:50, n_iter:10
```





From the above experiments with LR and SGD, it was found that the SGD performs better and faster than LR for text mining with Žurnal dataset. Further,

when comparing CountVectorizer and TfidfVectorizer, latter gives slightly better results. Using imbalanced learning and dimensionality reduction the performance of learning algorithm is improved.

5 Optimal parameter search procedure for tensorflow models

In this section, we briefly introduce the deep learning methods for multi-classification learning that are used in this study. In this study, we use three type of networks: DNN with embedding layer, Convolutional Neural Network(CNN) and Long Short Term Network(LSTM).

Networks were built with Python library Tensorflow. Tensorflow 2.0 was released about month ago, which means that it is still not in mature state. When searching for best optimal parameters we also tried to perform grid search, but it was impossible, since library returns error when using grid search. More about the issue one can found on https://github.com/tensorflow/tensorflow/issues/33204.

To tackle this problem, we build and process models in following procedure: We are search for the best parameters iteratively. The first set of parameters is max_len for padding sequences on same size. The second set of parameters is batch_size, defined as the number of samples that will be propagated through the network and the last set of parameter is epochs, defined as the number of passing the dataset through the network.

For learning we used Adam with default values that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data. On all networks we use relu activation fuction before output layer and softmax activation fuction in output layer.

The architecure of networks are following:

```
DNN with embedding layer:
Embeding layer, Dense layer, Output layer
```

CNN:

```
Embeding layer, Conv1D, GlobalMaxPooling1D, Dense layer, Output layer
```

LSTM:

```
Embeding layer, lstm, Dense layer, Output layer
```

The following list is the complete set of parameters used in searching for the best model:

DNN with embedding layer:

```
Outputsize (of embedding layer): 128, 248
Hidden layer size: 32, 64
```

CNN:

```
Outputsize (of embedding layer): 128, 248
Convolution size: 128, 256, 512
Hidden layer size: 10, 100, 250
```

LSTM:

```
Outputsize (of embedding layer): 128, 248
LSTM layer size: 64, 128, 256
Hidden layer size: 100, 500, 1000
```

5.1 Results of Text mining with DNNs

DNN with embedding layer: accuracy: ?

```
Outputsize (of embedding layer): ?
Hidden layer size: ?
```

CNN: accuracy: ?

```
Outputsize (of embedding layer): ?
Convolution size: ?
Hidden layer size: ?
```

LSTM: accuracy: ?

```
Outputsize (of embedding layer): ?
LSTM layer size: ?
Hidden layer size: ?
```

Above experiment is unfortunately unfinished, due to lack of time.