**Non-functional Requirements Classification – Transfer Learning Approach**

Alexander Stoyanov

[astoya1@students.towson.edu](mailto:astoya1@students.towson.edu)

Department of Computer and Information Sciences

Towson University

**Final Report**

**May 10, 2019**

**Abstract:**

In this research we look at how the transfer learning methodology can be applied to text classification, particularly to classify non-functional requirement datasets with models previously trained on a big data corpus like hundreds of Wikipedia articles. The purpose of this research is to establish a concrete benchmark of how successful an open-source data can be applied for training deep neural network when there is a lack of desirable training data. It is really hard to find a sufficient number of non-functional requirements for training deep neural network, so we substitute those with human-written sentences from Wikipedia articles. Trained binary and multiclass models are tested on a corpus of non-functional requirements, results are recorded and analyzed to draw a conclusion. Wikipedia data corpus is found to be a good training substitute for broadly scoped class like security, but less so for narrow-scoped classes like operability and usability. This conclusion comes with a lot of caveats, as well as opportunities for future work and model improvements.

1. INTRODUCTION

**Background:**

In order to develop quality software, software engineers derive requirements for that software from various resources like charts, graphs, or written natural language. It quickly becomes time-consuming the bigger the software is because requirements need to be split into functional and non-functional, and non-functional to be split further in many subcategories. Unfortunately, requirement statements are mostly written in natural language and thus can be uncertain and ambiguous. Natural Language Processing (NLP) helps to reduce those problems and improve computer understanding of natural language written requirements [8].

Requirements are also not as abundant as data scientists would want them to be. In the supervised learning approach, Machine Learning (ML) algorithms need a lot of labeled data to train on in order to accurately classify alike data. However, in the requirements engineering field there is not enough well-labeled data. This project is to serve as a workaround of that problem by utilization of Transfer Learning (e.g. [5], [6], [7]), and Synthetic Minority Over-sampling Technique (SMOTE) (e.g. [2], [3], [4]). As in [6], Transfer Learning is done by training the ML algorithm on some data that is abundant and close (in terms of context) to requirement statements, and applying the algorithm to the existing and scarce requirement statements that are desired to be classified. On the other hand, according to [3], SMOTE is used to enlarge the training dataset by generating artificial (synthetic) statements that can be used with original statements to help ML algorithm be more precise.

**Motivation:**

The project involves the use of different techniques such as NLP, SMOTE, and Transfer Learning in order to successfully classify scarce non-functional requirement statements. The problem to be solved by this project is the successful, high-precision multi-class classification of scarce non-functional requirement statements.

Automated classification of non-functional requirements (NFRs) will help reduce the time and resources this process would need otherwise, leading to the more efficient system performance, reducing the effort of classifying NFRs manually and instead putting this energy in other areas of needs.

1. APPROACH

**Design:**

There are several concrete categories of NFRs, which made them to be the perfect fit for this project. We focus on the main five categories of NFRs: security, usability, operability, performance, and maintainability. Since the data that we use for training, validation, and testing is labeled, the model training falls under the supervised learning category, where the model is told what the desired output should be based on labels. Furthermore, as a technical stack we utilize Tensorflow machine learning library [9], with Keras high-level neural networks API [10] on top of TensorFlow. We also use Scikit-learn library [11] to help us vectorize the data, and calculate all the metrics necessary for determining model’s success.

**Data:**

1. *Training data:*

As our main source of training data, we decided to use Wikipedia. Wikipedia has several advantages which made this source to be the primary data candidate: it is written by humans, it is publicly available; it is big; it has an easy-to-work-with API, and no restrictions on using it. Wikipedia is perhaps the only resource of data with such characteristics, and even though it is very big and has an abundance of relevant data for some of the classes that we work with (e.g. security), unfortunately it does not have enough NFR-focused data for the successful classification of every of the five considered categories.

In order to label such a huge corpus of unlabeled Wikipedia data, we went with the assumption that an article belongs to a certain class if the name (a string) of this article contains the name of that class. This is a solid strategy of auto-labeling the data since there is a strong correlation between the article title and its content.

Furthermore, the articles are picked randomly from Wikipedia, which guarantees different result scores with each run, but with the small enough variance for results to be consistent.

1. *Validation data:*

Validation dataset is a small ratio (usually 5-10%) of randomly selected instances from the training set. Although the best practice is to use somewhere around 20% of the training data for validation purposes, we wanted to maximize the training data for classes that struggled to get enough training data from the Wikipedia in order to achieve higher scores.

1. *Testing data:*

Testing dataset is the RE challenge dataset combined with the PROMISE dataset. The dataset consists of several functional and non-functional requirement groups, from which we only considered five, as described in the design section. The dataset consists of requirement sentences, where each sentence is followed by the appropriate label of the requirement class to which this sentence belongs to. This dataset is never used for anything other than testing already trained model in the scope of this project.

**Data Preprocessing:**

In order to achieve high model accuracy, it is necessary to preprocess the data. All sentences that are used for either training the model or testing it are augmented in four different ways.

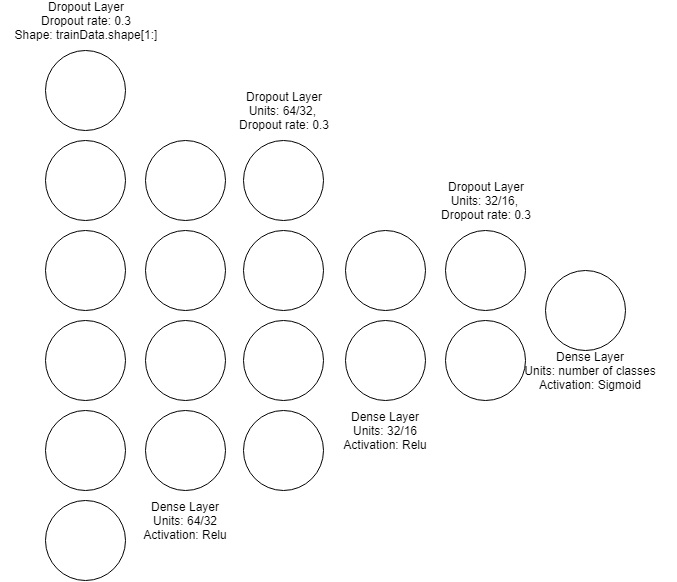
All the words that are longer than 18 characters are ignored since they could be gibberish artifacts from Wikipedia articles. Unnecessary characters, including any punctuation symbols and numbers, are removed from the sentences because they do not carry any value for our model.

Now the NLP tool comes into play which provides a list of common stop-words that are filtered from our data corpus. Stop-words are words like ‘a’, ‘me’, ‘have’, etc., that are general to any data corpus and hence not only they do not carry any value for the model, but also can be a catalyst to potential decreased model performance.

Lastly, NLP also provides a stemmer, which assists with word augmentation by merging two words that are very likely to have the same meaning into one-word instance. For example, words ‘plays’, ‘played’, ‘playing’ will be reduced to ‘play’ throughout the corpus.

**Machine Learning Models:**

**Figure: Visualization of Deep Neural Net**

****

Binary classification model – deep neural network represented by sequential model implemented with Keras API. It consists of the dropout layer as its input layer, enabling some of the features to be ignored with each new training epoch. The output layer is the dense layer with sigmoid activation function, making the range of the output be [0, 1]. Also, there are four hidden, alternating layers of types dropout and dense.

First two hidden layers are 64-dimensional, while last two are 32-dimensional. The dropout rate of the dropout layers is 0.3, meaning that 30% of the neurons are ignored with each training epoch to prevent the model from overfitting. Lastly, the classifier’s loss function is binary-crossentropy, with adaptive moment estimator (Adam) optimizer as a method of stochastic optimization, and a default learning rate of 0.001.

Multiclass classification model – just like the binary classification model, with the exception of the loss function, which cannot be binary, since it is not a two-way classification. Instead, the loss function is sparse-categorical-crossentropy. Also, since the multiclass model is working with smaller dataset due to data limitations of several NFR classes, it has reduced dimensionality of the hidden layers (32 and 16 dimensions for first two and second two hidden layers respectively).

In order to understand the underlying difference between different categories of requirements, we use Term Frequency – Inverse Document Frequency (tf-idf) vectorizer, which extracts features from the data corpus such that the least frequent words get the weight representing the highest value of these words within the corpus, while most frequent words get the weight representing the lowest value. In this project we use a maximum of 20,000 features in order to balance between choosing too many features, which would increase model training time, and too little features, which would decrease model’s accuracy, and overall performance.

**Training and Testing:**

Tuples created by tf-idf vectorizer are used as a primary source of training and testing data for the model. After the training has been done, the model is tested against the testing data which is described earlier in the paper. The testing scores such as accuracy, precision, recall, and f-score (a.k.a. f1-score) are calculated and recorded for a future reference and analysis.

1. RESULTS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Binary Model | Accuracy | Precision | Recall | F-score |
| Security | 0.7825 | 0.7907 | 0.7684 | 0.7794 |
| Usability | 0.6497 | 0.6442 | 0.6688 | 0.6562 |
| Operability | 0.6176 | 0.6154 | 0.6275 | 0.6214 |
| Performance | 0.5 | *Irrelevant* | *Irrelevant* | *Irrelevant* |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Multiclass Model  (Security, Usability, Operability) | Accuracy | Precision | Recall | F-score |
| Round 1 | 0.5723 | 0.5941 | 0.5723 | 0.5804 |
| Round 2 | 0.5904 | 0.5873 | 0.5904 | 0.5885 |
| Round 3 | 0.5948 | 0.5990 | 0.5948 | 0.5949 |

The models, both binary and multiclass, best classified requirement sentences that belonged to ‘security’ class. Those kinds of test requirements achieved the highest scores amongst all 4 different scoring formulas, maxing out at 80% of accuracy, precision, recall, and f-score. Wikipedia is abundant of security articles (approximately 8,000 unique articles), which is great for the model because it will never run out of data to train on. However, this is not the case for other four classes of NFRs, because none of them have more than a hundred articles to train on, which makes training dataset for the multiclass model greatly unbalanced. Oversampling with SMOTE does not solve this problem as one would expect because oversampled data is too close by its context to the original data, making the model train on the same data without learning any new context about the specific NFR class. Under-sampling the data for multiclass model for security, usability, and operability NFRs yields to 60% amongst all four metric scores.

The other two classes, maintainability and performance, are not considered for multiclass model training because they are greatly decrease the results by not having nearly enough relevant data to train on.

Data relevance also plays a major role in this project. Security models perform so well because security data extracted from Wikipedia represents a general security topic. Security is more or less the same everywhere, so there is no issue with learning about security from Wikipedia and then applying this knowledge to narrow-scoped NFR testing data. Performance from the Wikipedia, on the other hand, by meaning and context is not the same performance that one would expect to find in the narrow-scoped NFRs. The fact that the performance can’t be as generalized as security makes it much harder to apply transfer learning methodology to existing NFR classification problem. These are the reasons why performance classification comes down to a random guess.

Maintainability class simply does not have enough data to be used for model training. There are only 3 Wikipedia articles that are labeled as maintainability, which is not sufficient by any means for training a deep neural network.

Overall, the models’ performance meets the expectations. Multiclass model’s results are well within the margin of error, and with a small enough variance to safely conclude that transfer learning is a consistent approach for text classification. The best results belong to the security class, because security is very broad topic, and rarely security-related NFR sentences can be worded in a different way from general security sentences. This is why the models’ performance is so great with security class, but not so great with classes like usability, and operability. Usability and operability NFRs can vary a lot depending on the software that those NFRs belong to. For example, usability NFRs for medical software can be much different from usability NFRs for satellite software. We call these classes narrow-scoped, since they have much less of a general pattern, and more inclined towards the software that they represent.

1. RELATED WORK

Many of the relevant experiments have already been done in the area of application of Transfer Learning methodology to NLP and text classification. Some works that could be useful in future research, and that we recommend to be acquainted with are [5], [6], [7].

1. FUTURE WORK

Some suggestions that could be done to improve the existing NFR classification models:

* Combining existing binary and/or multiclass models with third-party pretrained models like BERT [12].
* Modifying existing models from convolutional neural network (CNN) to recurrent neural network (RNN) [13], so that models will consider the context of words in training sentences.
* Finding additional accessible sources of relevant NFR data that can be used to balance the existing Wikipedia data corpus.
* Converting training data corpus with doc2vec technique might improve the results of neural networks since doc2vec could pay more attention to the context within data corpus. This also means that currently implemented tf-idf vectorizer would not be used, so training data would require more sanitation in order to remain on par with previously used vectorized data.
* Not trusting the labels of labeled data sets. It has been observed that NFRs from security and privacy categories share a lot in common, and it is extremely easy to confuse one for the other.
* Applying any sort of technique that considers the context of overall data corpus might be extremely helpful, because context in non-functional requirements carries a lot of weight when deciding to what class of NFRs a certain requirement belongs.

1. REFERENCES
2. Google Inc. (2018). Text Classification Guide. Retrieved from https://developers.google.com/machine-learning/guides/text-classification/
3. Chawla N.V., Bowyer K.W., Hall L.O., & Kegelmeyer P.W. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Retrieved from https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume16/chawla02a-html/chawla2002.html
4. Aridas C., Oliveira D., Nogueira F, Lemaitre G. (2016). imblearn.over\_sampling.SMOTE. Retrieved from https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over\_sampling.SMOTE.html
5. Amunategui. (n.d.) SMOTE – Supersampling Rare Events in R. Retrieved from http://amunategui.github.io/smote/
6. Sarkan D. (2018). Deep Transfer Learning for Natural Language Processing —  Text Classification with Universal Embeddings**.** Retrieved from https://towardsdatascience.com/deep-transfer-learning-for-natural-language-processing-text-classification-with-universal-1a2c69e5baa9
7. Martigny P. (2018). Transfer Learning in NLP. Retrieved from https://blog.feedly.com/transfer-learning-in-nlp/
8. Hulstaert L. (2018). Applying transfer learning in NLP and CV. Retrieved from https://towardsdatascience.com/applying-transfer-learning-in-nlp-and-cv-d4aaddd7ca90
9. Collobert R., Weston J., Bottou L., Karlen M., Kavukcuoglu K., Kuksa P. (2011) Natural Language Processing (Almost) from Scratch. Retrieved from http://www.jmlr.org/papers/v12/collobert11a.html
10. Abadi M., Agarwal A., Barham P., Brevdo E., Chen Z., Citro C., Corrado G.S., Davis A, Dean J., Devin M., Ghemawat S., Goodfellow I., Harp A., Irving G., Isard M., Jozefowicz R., Jia Y., Kaiser L., Kudlur M., Levenberg J., Mané D., Schuster M., Monga R., Moore S., Murray D., Olah C., Shlens J., Steiner B., Sutskever I., Talwar K., Tucker P., Vanhoucke V., Vasudevan V., Viégas F., Vinyals O., Warden P., Wattenberg M., Wicke M, Yu Y., and Zheng X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Retrieved from <https://www.tensorflow.org>
11. Fran, C., and others. (2015). Keras. Retrieved from https://keras.io
12. Pedregosa F., Varoquaux G., Gramfort A., Michel, V., Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., Vanderplas J., Passos A., Cournapeau D., Brucher M., Perrot M., and Duchesnay E. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. vol. 12, pp. 2825-2830.
13. Devlin J. Chang M-W., Lee K., and Toutanova K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Retrieved from https://arxiv.org/abs/1810.04805
14. Mikolov T., Karafiat M., Burget L., Cernocky J., and Khudanpur S. (2010). Recurrent Neural Network Based Language Model. Retrieved from https://www.iscaspeech.org/archive/interspeech\_2010/i10\_1045.html