CIKM AnalytiCup 2017 – Lazada Product Title Quality Challenge: Constructing Features for a Diversified Ensemble of Classifiers

Massimo Nicosia[◇] and Alessandro Moschitti
[◇]DISI, University of Trento, 38123 Povo (TN), Italy
Qatar Computing Research Institute, HBKU, 5825, Doha, Qatar
{mnicosia,amoschitti}@gmail.com

ABSTRACT

Feature engineering and stacked generalization are at the core of our model. Several classifiers are trained to produce first level predictions, which are fed with the original features to selected classifiers. Second level predictions are simply averaged to obtain the final class probabilities. Our submission ranked #9, despite the simplicity of the model.

1 INTRODUCTION

E-commerce web sites contain product pages which are directly created by product sellers. Sometimes the latter may write titles and descriptions which are full of keywords and thus result unnatural, in an attempt to game the e-commerce search engine. The Lazada Product Title Quality Challenge consists in helping Lazada to detect rogue listings by building a classifier that scores product titles and pages according to two criteria: clarity and conciseness.

A product page is clear if, in a short time, the user can understand the title and the product, together with its key attributes. The page is concise if it is short, and contains all the necessary information. The task of the competition participants is to build a model capable of producing a probability value associated to the clarity and conciseness labels. Submissions are evaluated using the Root-Mean-Square Error (RMSE), defined as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Submissions are then ranked by averaging the RMSE computed on the two labels:

$$RMSE_{overall} = \frac{RMSE_{conciseness} + RMSE_{clarity}}{2}$$

2 RELATED WORK

The automatic classification of the quality of textual content has received an increased amount of attention, especially after the explosion of user-generated content.

Spam detection is probably the most popular and ancient task [2, 4]. It is, as all the task described in this section, an instance of Text Classification. In particular, the goal is to detect email or web pages that contain unsolicited commercial content, or may be deceptive for the reader. The majority of spam detection (and text classification) approaches transform the sequence of words in a document into features, such as word counts, that can be used in a classifier.

Bad content can also comprehend bad product reviews [5, 6] and bad forum posts [8]. In addition to words, the cited pieces of work try to model an higher level notion of readability. This goal is achieved by designing specific metrics that should capture the level of formality or difficulty of textual content.

A more recent problem is clickbaiting: messages that circulate in social media, and are forged to spark curiosity and to elicit a click from the user. Automatic attempts to clickbait classification try to model the syntactic and semantic features of the content [1].

All the previous tasks may benefit from signals not directly coming from the content to classify. For example, in spam detection, the IP address or the email of the sender are very discriminative features. Reviews or forum posts are often associated to specific authors, and profiling the authors usually improves the classification accuracy. Clickbait titles are often linked to web pages containing the complete article, which can also be analyzed and used as a source of features. In the Lazada Product Title Quality Challenge, similar data is not included and cannot be easily modelled.

3 METHODOLOGY

The development of our model can be divided in two key parts: feature engineering and the construction of the ensemble classifier.

3.1 Feature engineering

Product titles are preprocessed to obtain a cleaned version: we remove a given set of punctuation characters, and we apply a very light product name normalization. The original titles are also processed with the spaCy [1] tokenizer and tagger, to obtain tokens and part-of-speech tags. The majority of features is extracted from product titles.

Engineered features. We briefly describe the features used in our model, indicating with **(r)** when we also compute the feature ratio, e.g., how many times a specific class of characters or words appear in the title, with respect to the total number of characters or words.

From the original unprocessed titles we extract:

- the number of characters;
- the number of words;
- the number of uppercase, lowercase, non alphanumeric, numeric characters **(r)**;
- a boolean feature indicating the presence of number, parentheses, plus or minus symbols;
- a boolean feature indicating if the title is exactly contained in the description;
- the spam score computed with the antispam[2] package.

From the cleaned title we extract:

- the number of occurrences of the most repeated word;

[1] http://spacy.io
[2] https://pypi.python.org/pypi/antispam/0.1

- the number of words that are repeated **(r)**;
- the number of words matching a color, a buzzword, or a word in an English dictionary[3] **(r)**;
- the sum of the occurrences of repeated words **(r)**;
- the average number of word characters.

In addition, we extract from the title a set of readability features [6] through the textstatistics[4] package:

- the number of syllables;
- the average number of syllables per word;
- the count of polysyllable words;
- the Flesh Reading Ease score;
- the Flesh Kinkaid Grade level;
- the Smog index;
- the automated readability index;
- the Linsear Write Formula score;
- the number of difficult words (words which are not contained in a list of easy words);
- the proportion of long words.

From the tokenized titles, we extract the ratio of words which appear only once in the training set. Then, for a subset of count features, we compute their counts divided by the maximum count value observed in the training set, the deviation from the mean, the deviation from the median, the difference with the maximum.

From the description, we extract a set of boolean features capturing the seller effort in formatting the text:

- the presence of html lists (presence of <ul> and <li> tags);
- the presence of a style tag;
- the presence of the ampersand HTML entity.

Additional features include the logarithm of the price, a binary feature indicating expensive items, and the proportion of each part-of-speech tag which appears in the title.

Categorical features. Each listing contains a subset of fields that we treat as categorical features, namely:

- the country where the product is marketed;
- three hierarchical product categories;
- the product type.

Word features. We extract word counts from the lowercased titles. More specifically, we compute the TFIDF of unigrams considering: words, lemmas and stems obtained with (i) the Porter Stemmer, (ii) the Snowball stemmer. Our intention is to learn base classifiers on various word forms, in order to capture different views of the text, and enhance diversity in the final ensemble.

3.2 Ensemble learning

In this section, we describe the base classifiers that we trained on our features, we explain how we tuned the model hyperparameters, and how we built the final ensemble.

Base classifiers. We select a set of base classifiers by measuring their performance in a 5-fold cross-validation setting, and by considering their diversity. We cross-validate hyperparameters and perform feature selection (we add a feature to the final set of features, if it improves the cross-validation performance) for each target label, to obtain:

- a Logistic Regression classifier with the regularization term C set to 1.0;
- an ElasticNet model, clamping predictions in the [0,1] range, and with the term C set to 1.0 and alpha set to 0.0001;
- XGBoost [3], a Gradient Boosting Classifier, using 50 trees, a learning rate of 0.1, and a maximum tree depth of 10;
- LightGBM[5], another Gradient Boosting Classifier package.

The Logistic Regression and ElasticNet implementations are from the scikit-learn package [7].

Mixing diverse models such as linear and decision tree classifiers is common practice when constructing ensembles, and it is at the core of stacked generalization [9]. Diverse model may produce different predictions, and make orthogonal errors. In this case, an ensemble model may correct such errors, if it is able to learn when one of the model fails, and the other succeed.

First level predictions. We train each classifier on each target label, and we use the entire training set to obtain predictions on the test data. To obtain correlated predictions on the training set, we compute the out-of-fold output of each classifier. This means that we split the training set into 50 folds, we train each model on 49 folds, and we classify the examples contained in the remaining fold to obtain the associated predictions.

Second level predictions. We perform stacked generalization by augmenting the original feature set for the training and test data with the out-of-fold and test predictions from the base classifiers. Intuitively, this process should also pick up some correlation between conciseness and clarity. These concepts share similar traits, so it may be useful for a model trained on conciseness targets, to know what another model thinks about the clarity of the same examples (and viceversa).

We retrain the same classifiers used to produce first level predictions on the agumented feature set, and we apply them on the test set. The final set of predictions is produced by simply averaging the probabilities of each target label.

Computational resources. All the experiments are carried out on a laptop with a 2.50GHz Quad-Core CPU, 16 GB of RAM, and an Nvidia GTX860M GPU with 2 GB of memory. The classifiers with the highest training time are XGBoost and the neural networks, especially for the conciseness label. The training time is dominated, as expected, by producing the out-of-fold predictions: 40 minutes for XGboost and the convolutional network, and 4 hours for the recurrent network. The feature computation time was instead dominated by producing the TFIDF counts. For this reason, it was vital to implement a caching mechanism to store the features for each partition extracted from the datasets. We are referring to the entire training and test data, plus each training partition from the cross-validation process. Caching count (and other) features, saved us a considerable amount of computation time across iterations of the models, enabling us to experiment much faster.

[3] https://github.com/dwyl/english-words
[4] https://pypi.python.org/pypi/textstat
[5] https://github.com/Microsoft/LightGBM

4 LESSONS LEARNED

In this section, we itemize some general insights and thoughts on the models, and the competition.

Classifiers not included in the ensemble. We tried several classifiers in the hope of bringing diverse predictions and thus diverse errors into the ensemble. We evaluated Randomized Decision Trees, a Lasso model fit with Least Angle Regression, Random Forests, a Linear SVM calibrated with the Platt method to obtain probabilities, and a K-Nearest Neighbours classifier. Interestingly, we also trained a convolutional neural network on the clarity labels, and a recurrent neural network on the conciseness label. Both networks used as input only GloVe word embeddings. The predictions of such models did not help in the final ensemble.

A third-level classifier. Averaging the predictions for the second-level classifiers may be sub-optimal. Ideally, an additional simple model could learn how to combine those predictions and give better results. We tried training a meta-classifier on the second level predictions, applying again the out-of-fold method. This approach did not give the desired outcome, and performed similarly to averaging.

Multiple word surface forms. Although we computed base classifier predictions for a diverse set of TFIDF counts (on words, lemmas, stems), we ended up using only Porter stems. This particular choice gave us the better cross-validation performance. Adding signals from the counts on other surface forms (adding out-of-fold predictions in the second level classifier) did not improve the final outcome. This was surprising, since this technique helps most of the time.

Neural networks. The inclusion of neural networks into the final ensemble did not have the desired effect. Probably, more time should have been allocated into tuning the network architectures, and into incorporating the hand-engineered features in addition to word embeddings.

Model diversity. The most noticeable improvements (other than the feature engineering effort) were obtained by combining the predictions of different types of classifiers.

Domain. We modelled few aspects of the domain language of product descriptions (product features, names, brands, etc.), and most of the time with indirect features. Probably, more effort should have been spent into extracting a semantic view of the product title, i.e., associating a semantic interpretation to each token. This would have reduced the vocabulary sparsity of the product domain. We believe that incorporating a notion of product/brand popularity into the model would have also been helpful.

The language of titles, and inter-annotator agreement. Most of the times, titles consisted in a set of keywords, so it was problematic to apply natural language processing techniques which rely on syntax. In addition to that, annotations appeared to be quite subjective in many cases. It would be interesting to know the inter-annotator agreement on the competition dataset.

Model	Overall RMSE
a	0.2859
b	0.2733
c	0.2712
d	0.2669

Table 1: Models and overall RMSEs on the development set.

5 ANALYSIS

In this section, we describe how the scores evolved over time. Table 1 shows the models, identified by a letter, and the corresponding overall RMSE, obtained by averaging the RMSE on the clarity and the conciseness target labels.

Model (a). Our first entry, a Logistic Regression model with few features: TFIDF unigram scores and simple character/word features.

Model (b). The first ensemble model, with XGBoost and Logistic Regression as base learners. Regarding second level predictions, a Logistic Regression model was used for obtaining clarity labels, and an XGBoost model for obtaining conciseness labels.

Model (c). Ensemble model, with tuned first level XGBoost classifiers, and tuned features.

Model (d). Ensemble model, with some more feature engineering and selection. In addition to that, this time we added the ElasticNet model, and we averaged the second level predictions.

The main reasons for the increase in performance were, apart from the constant feature engineering and selection, the introduction of diverse classifiers in the ensemble, model tuning, and the averaging of second level predictions.

In the final ensemble on the test dataset, we included an additional gradient boosting model, LightGBM. Then, instead of just averaging the predictions, we weighted the 4 models differently: the two gradient boosting models had a weight of 0.3, while the two linear models had a weight of 0.2.

Our final ranking on the testing leaderboard was #9, with an overall RMSE of 0.2908.

6 CONCLUSION

In this report, we described our entry in the CIKM AnalytiCup 2017 Lazada Product Title Quality Challenge. Our approach consisted in designing general text classification features, and features specific to the problem; then, we trained a diverse set of base classifiers that we used in a stacked generalization fashion. Second level predictions from the same classifiers were averaged to obtain the final class probablities.

Possibile future directions for improving the model could include (i) the use of external data, from Lazada, or from web sites such as Amazon; (ii) modelling the neighborhood of a title, using distance measures from similar instances or ranking methods; (iii) addressing the sparsity of the vocabulary obtained from the listings. More specifically, a way to map each word into a category would be beneficial for a better understanding of the features of a title. For example, some time could be spent to manually label tokens into categories such as product features, product serial numbers, brand names, product measures/sizes, and so on. This way, a tagger could be trained on a subset of the data, and then used to tag the titles. The predicted categories would then be used as a source of features.

REFERENCES

[1] Prakhar Biyani, Kostas Tsioutsiouliklis, and John Blackmer. 2016. "8 Amazing Secrets for Getting More Clicks": Detecting Clickbaits in News Streams Using Article Informality. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI'16)*. AAAI Press, 94–100. http://dl.acm.org/citation.cfm?id=3015812.3015827

[2] Xavier Carreras, LluÃŋs Marquez, and Jordi Girona Salgado. 2001. Boosting Trees for Anti-Spam Email Filtering. (2001).

[3] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[4] William W Cohen et al. [n. d.]. Learning rules that classify e-mail.

[5] Nitin Jindal and Bing Liu. 2007. Review Spam Detection. In *Proceedings of the 16th International Conference on World Wide Web (WWW '07)*. ACM, New York, NY, USA, 1189–1190. https://doi.org/10.1145/1242572.1242759

[6] Michael P. O'Mahony and Barry Smyth. 2010. Using Readability Tests to Predict Helpful Product Reviews. In *Adaptivity, Personalization and Fusion of Heterogeneous Information (RIAO '10)*. LE CENTRE DE HAUTES ETUDES INTERNATIONALES D'INFORMATIQUE DOCUMENTAIRE, Paris, France, France, 164–167. http://dl.acm.org/citation.cfm?id=1937055.1937097

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[8] Luca Ponzanelli, Andrea Mocci, Alberto Bacchelli, Michele Lanza, and David Fullerton. 2014. Improving low quality stack overflow post detection. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. IEEE, 541–544.

[9] David H. Wolpert. 1992. Stacked Generalization. *Neural Networks* 5 (1992), 241–259.