Summary Source Prediction

Sébastien Meyer
École polytechnique, France
`sebastien.meyer@polytechnique.edu`

This report details the different approaches and strategies that I have developed in order to get the best possible score I could on the summary source data set. We were given a dataset made up of a gathering of original documents as well as their summary, either human-written or generated by a machine. There were also additional documents. The main task was to predict whether the summary was generated by a machine or not. We had a training set and we were asked to submit a csv file containing a label associated to each summary.

It has been a really challenging project for me. In order to best describe how I tackled this problem, this report will follow my advances. Firstly, I will remind the reader with a brief description of the task and the data. Then, I will go on with my first attempts. Since my first tries were not satisfactory, I took inspiration from past projects to get new ideas. In the **Appendix**, I gathered my models with some details on the pipeline.

1 The dataset

The dataset is made up of 8000 training points and 3200 test points. It appears that the training set is not very large. Therefore, we must be prepared to face overfitting. Our task is a classification task, corresponding to the *label* variable, which is binary. The training set is perfectly balanced, as each class accounts for 4000 of the 8000 samples (see **Figure 1**).

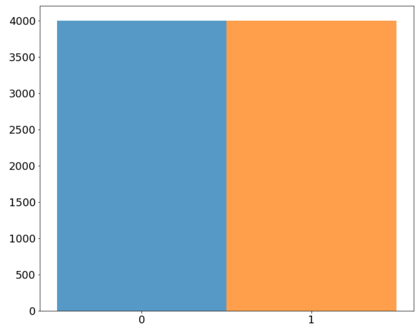

Figure 1: Repartition of the training labels.

For each sample, we recognize a document, which is the original document from which the summary is created, and the corresponding summary. Some basic information about the training documents and summaries are gathered in **Table 1**. We also show an example of summary associated with label 0 and another one associated with label 1 in **Figure 2**. We can directly observe that there are some differences between the two summaries. Indeed, the first summary lacks coherence between its two sentences. Also, there are a lot of spaces and special characters in the summary. On the contrary, the second summary is associated to label 0 and seems pretty clear and well-written. There is a clear correspondance between words, verbs and pronouns.

Feature	Mean	Std	Min.	Max.
Doc. len.	3.974	2,052	297.0	12,052
Sum. len. (0)	304.6	78.91	68.0	675.0
Sum. len. (1)	308.0	134.3	64.0	3,524

Table 1: Description of numerical variables.

The two moms from “ Big Bang Theory ” guest star this week . “ Backstrom , ” “ Blue Bloods ” are among the week ’s season finales . Comedy “ Younger ” is gaining big buzz .

Nicola Sturgeon today warned Ed Miliband that time is running out on her offer. The SNP leader said she is prepared to work with Labour to put Mr Miliband in Number 10. She said every day that passes risks fuelling the impression that Mr Miliband would rather see the Conservatives return to government than work with the SNP.

Figure 2: First summary has label 1, second has label 0.

These observations led me to create some features from the documents and summaries. Recall that our features are gathered in the *src/preprocessing/features* folder. I computed different features under the name *X_Y* where *Y* can be *count* for the number of observed instances, *ratio* for the ratio between the values of *count* for the summary and the document and *overlap*. The *overlap* is defined as follows: all items are selected and put into a set, the intersection of the set of the document and the set of the summary is computed and its length is the value associated to the feature. The first feature for this case is *char_Y* for the total length of the text. In addition, I computed similar features for words under *word_Y*, for sentences under *sent_Y* and for groups of consecutive words beginning with an upper letter under *group_Y*. On top of that, I added some minor features such as *upper_char*, *upper_word*, *numeric* and *dash* which address special characters, as well as *is_first_upper* and *is_last_ponct* to look at specificities.

2 First attempts

My first models are based on the features presented in the previous section as well as tf-idf features. The baseline logistic regression with 10,000 tf-idf features yields an accuracy of approx. 0.62 on the test set. Therefore, there is much room more improvement. Also, since I started working with binary and categorical variables, I decided to work with trees. Let X be a feature gathering binary variables and thus having values from 1 to p. When splitting at a specific node, a decision tree can make a decision based on $X > a$ or $X \leq a$. We see that such decision implies that values between 1 and a have a relationship, which is the case for a lot of features that we computed.

2.1 Cross-validated gridsearch

I implemented an automated gridsearch pipeline using *optuna* package to select the parameters of any model I decided to use. *Optuna*[1] takes ranges of values for model parameters as entry, and performs an optimization to find the best parameters according to the cross-validation score. The cross-validation was 5 folds large, with accuracy as target. Typically, I would run a gridsearch for 15 to 50 trials, depending on the complexity of the model and the size of the dataset. This would not take much more than 30 minutes.

2.2 RandomForestClassifier, ExtraTreesClassifier and LightGBM

The first models that I used were RandomForestClassifier and ExtraTreesClassifier from *scikit-learn*[3] API. These models are widely used in the case of classifications with more than two possible labels. For instance, RandomForestClassifier relies on the averaging of several base decision trees. These decision trees are built in a top-bottom fashion. Each split builds two nodes or regions R and $\bar{R}$. Let us denote by p_k^R and $p_k^{\bar{R}}$ the proportions of samples from class k falling respectively in regions R and $\bar{R}$. Split is made by minimizing a criterion, the most used criterion being Gini index defined as:

$$C(R, \bar{R}) = \sum_k p_k^R (1 - p_k^R) + \sum_k p_k^{\bar{R}} (1 - p_k^{\bar{R}})$$

which is the sum of Gini impurities in both R and $\bar{R}$. Other criterions can be used and have been tried during gridsearch, namely entropy criterion, however Gini index yields best results in almost all runs. ExtraTreesClassifier is based on trees that differ from usual decision trees in two points:

1. Bootstrapping is disabled by default for ExtraTreesClassifier

2. Splits are chosen among randomly drawn cuts given a subset of features

In some cases, such extremely randomized decisions can reduce overfitting, but it does not appear to be the case for our data set. Later, I started using boosting models such as LightGBM[2]. Although LightGBM is based on the same principle as other boosting models, that is, updating a model by successively assigning weights to wrongly classified data points, it is designed to be much more faster on large datasets than its counterparts.

A basic gridsearch for RandomForestClassifier lead to an accuracy of 0.8206, other models having lower scores. This score was relatively good, considering that I did not start extensive feature engineering and that my models were rather simple. However, tweaking these models did not bring much improvement. I had to start engineering new features.

2.3 Stacking

Even though RandomForestClassifier appears to outperform other models, Boosting models such as LightGBM and XGBoost[4] perform relatively well on the dataset and yield similar results to RandomForestClassifier. One of the methods used to combine different models that can be good at predicting specific aspects of the data is through Stacking. In the *scikit-learn* API, several base models are fitted on the training dataset. Then, a final estimator - usually a LogisticRegression - is fitted on base estimators predictions over training data with cross-validation to avoid overfitting. A summary schema is shown **Figure 3**. In this model, I tried to combine different models such as RandomForestClassifier, XGBoost and LightGBM.

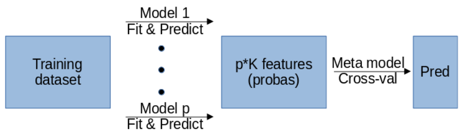

Figure 3: Stacking pipeline.

Since Stacking uses several models, its training is much more longer than for any base estimator. Therefore, running a gridsearch for the Stacking model would be computationally too expensive for my computer. I decided to fix the parameters of the base estimators using the best parameters of their own gridsearch.

3 Taking inspiration from past projects

3.1 New features

Until now, I generally separated the features either from regex expression or from tf-idf decomposition. A simple improvement to my results came from the concatenation of these features. In order not to have 10,000 tf-idf features against around 10 regex features, I performed truncated Singular Value Decomposition, which approximates a PCA for sparse matrices such that the ones we get from tf-idf decomposition using *scikit-learn* API. Moreover, I implemented a composition function to allow us to compute normalizations[5] of tf matrices. However, the different compositions did not bring much improvement in regard to the natural term-frequency matrices. In addition, I computed the cosine similarity between the summary and its original document as a new feature. Together with an ExtraTreesClassifier model, I was able to reach an accuracy of 0.84937.

Nevertheless, a major breakthrough originated from a very simple feature. Indeed, I computed a feature which counts the number of spaces before a ponctuation using a regex expression. As shown **Figure 4**, almost none of the generated summaries present spaces before ponctuation. Therefore, this feature helped us reach a new highscore. In a Stacking model including RandomForestClassifier, LightGBM, XGBoost and ExtraTreesClassifier, I reached an accuracy of 0.89937.

Other improvements came from minor features. For instance, recall that sentences can be transformed into Part-of-Speech tagging. In this case, each token is associated with a tag which must describe its role in the text, for example noun or adjective. I used all the available tags in *nltk* package and computed lots of features, again with *count*, *ratio* and/or *overlap* versions. These new features with a Stacking model allowed me to reach an accuracy of 0.90625.

Another type of features which I looked into is latent semantic analysis. The *scikit-learn* package offers different ways of performing latent semantic analysis. In one case, Latent Dirichlet Allocation or LDA allows to find important topics in a corpus, thanks to tf-idf decomposition. I manually computed the 3 most important topics and used some or all of them as features. This allowed me to reach an accuracy of 0.91312, using LightGBM. In fact, LightGBM started to show better results than Stacking when sufficiently extensive grid search was performed.

Finally, I found GLTR[6]. This model is based on existing models, for instance BERT and GPT-2, and uses them to predict whether if words in a sentence were likely to be generated by this type of models. I used the estimated probabilities for all the words in a summary to compute bins of fractions of probabilities and of topk values. With LightGBM model, I reached an accuracy of 0.91875.

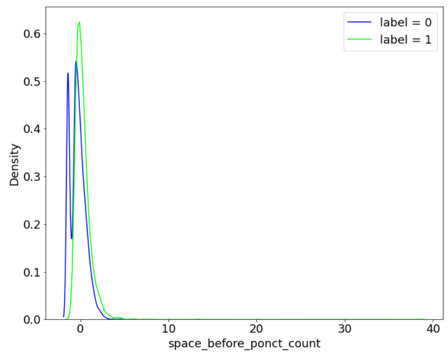

Figure 4: Repartition of training labels wrt *space_before_ponct_count*.

4 Last improvements

4.1 Failed attempts

There are two major paths that did not lead me to good results. On the one hand, I tried to use embeddings. I performed the usual tokenization and padding of the summaries, and I used either the GloVe[7], GoogleNews[8] available embeddings, or I trained my own Word2Vec vocabulary and embeddings using *gensim*. Then, I used different neural networks, from feed-forward to bidirectional LSTMs. However, I was not able to go over an accuracy of 0.68 in a train-test-split setting.

On the other hand, recall that very powerful models for text generation already exist. Also, my recent findings on GLTR made me think that I could use existing models for detecting generated summaries. To that extent, I used the huggingface[1] library in order to find models shared by the community. Here, I faced two major issues. Firstly, most of the models are not fitting on my GPU, which is capped at 6GB. Secondly, there were not a lot of available models that were built for our classification task. I tried with some models such as DistilBERT and DistilroBERTa. After some epochs, it is possible to achieve competitive results, however, I did not continue with these models.

4.2 Feature selection and Catboost

My last improvements come from the *mlxtend*[9] package, which offers sequential feature selector. I also tried other well-known models, namely CatBoost[10]. CatBoost is a boosting model particularly adapted to categorical features. My final accuracy is 0.92812. Similar feature selection with a Stacking model gathering RandomForestClassifier, XGBoost, LightGBM and CatBoost brings me to an accuracy of 0.92187, which is not more than the one achieved with CatBoost alone.

[1]https://huggingface.co/

Conclusion

To conclude, I have learnt some practical skills from this project. Most of the time, it is difficult to get a precise idea of a model or even a pipeline to address a dataset. One has to try and search different combinations. All in all, we can keep in mind that once a baseline is found by using an usual model, other less "conventional" models can be tested out. In terms of the competition itself, one of the most important aspects was to design pertinent features. This challenge allowed me to learn more about data analysis ideas and to be more careful about the models we use on a regular basis. Despite my satisfactory results, I think that things can still be done to improve my models. If I had more time, I would have tried to build more features, especially from the semantic analysis which remained mysterious to me. In addition, more computational power would allow for more aggressive gridsearch and the use of other huggingface models!

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta and Masanori Koyama. *Optuna: A Next-generation Hyperparameter Optimization Framework.* 2019, in KDD.

[2] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye and Tie-Yan Liu. *LightGBM: A Highly Efficient Gradient Boosting Decision Tree.* Advances in Neural Information Processing Systems 30 (NIPS 2017), pp. 3149-3157.

[3] 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. *Scikit-learn: Machine Learning in Python.* 2011. Journal of Machine Learning Research, vol.12, pp. 2825-2830.

[4] Tianqi Chen and Carlos Guestrin. *XGBoost: A Scalable Tree Boosting System.* San Francisco, California, USA. 2016. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785-794.

[5] François Rousseau and Michalis Vazirgiannis. *Composition of TF Normalizations: New Insights on Scoring Functions for Ad Hoc IR.* ACM, SIGIR 2013.

[6] Sebastian Gehrmann, Hendrik Strobelt and Alexander M. Rush. *GLTR: Statistical Detection and Visualization of Generated Text.* June 2019. (Available at: https://arxiv.org/pdf/1906.04043.pdf)

[7] Jeffrey Pennington, Richard Socher and Christopher D. Manning. *GloVe: Global Vectors for Word Representation.* 2014.

[8] Tomas Mikolov, Kai Chen, Greg Corrado and Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space.* Septembre 2013. (Available at: https://arxiv.org/pdf/1301.3781.pdf)

[9] Sebastian Raschka. *MLxtend: Providing machine learning and data science utilities and extensions to Python's scientific computing stack.* 2018. J Open Source Softw 3 (24).

[10] Anna Veronika Dorogush, Vasily Ershov and Andrey Gullin. *CatBoost: gradient boosting with categorical features support.* Workshop on ML Systems at NIPS 2017.

Appendix

Models and results

The following tables sum up the different submissions we made as well as corresponding features and tuned parameters. All runs and gridsearches were made with at least one of the following seeds: 42, 1342 and 8005.

Model	Preprocessing	Features	Tuned Parameters	Test accuracy
RandomForestClassifier	Eliminate correlated var.: No PCA: No Scaling: Standard	*char_X* + *word_X* + *sent_X* + *upper_char_X* + *upper_word_X* + *numeric_X* + *dash_X* + *group_X* + *is_first_upper* + *is_last_ponct* **Total: 26**	*n_estimators* *criterion* *max_depth* *min_samples_split* *min_samples_leaf* *max_features* *ccp_alpha*	0.82062
ExtraTreesClassifier	Eliminate correlated var.: No PCA: No Scaling: Standard	previous + *n_idf_cos* + *n_idf_pca_1* **Total: 28**	*n_estimators* *criterion* *min_samples_split* *min_samples_leaf* *max_features* *min_impurity_decrease* *ccp_alpha*	0.84937
Stacking: - RandomForestClassifier - ExtraTreesClassifier - XGBoost - LightGBM	Eliminate correlated var.: No PCA: No Scaling: Standard	previous + *space_before_ponct_count* **Total: 29**	(taken from base estimators)	0.89937
Stacking: - RandomForestClassifier - ExtraTreesClassifier - XGBoost - LightGBM	Eliminate correlated var.: No PCA: No Scaling: Standard	previous + *DT_X* + *MD_X* + *NN_X* + *RB_X* + *$_X* **Total: 44**	(taken from base estimators)	0.90625
LightGBM	Eliminate correlated var.: No PCA: No Scaling: Standard	previous + *n_lda_{1,2,3}* **Total: 47**	*n_estimators* *num_leaves* *min_split_gain* *min_child_weight* *min_child_samples* *subsmaple* *subsample_freq* *reg_alpha* *reg_lambda*	0.91312
LightGBM	Eliminate correlated var.: No PCA: No Scaling: Standard	previous + *frac_{1,..,10}* + *count_{1,..,4}* **Total: 61**	*n_estimators* *num_leaves* *min_split_gain* *min_child_weight* *min_child_samples* *subsmaple* *subsample_freq* *reg_alpha* *reg_lambda*	0.91875
CatBoost	Eliminate correlated var.: 0.95 PCA: No Scaling: Standard	feature selection	*n_estimators* *learning_rate* *grow_policy* etc.	0.92812