DTSE CZ - practical task - Steam Game Reviews

Stanislav Zámečník

16.1.2022

Task: *In this scenario imagine you work as a Data Scientist for Steam gaming platform. Your task is to download historical game review data from* `https://www.kaggle.com/whoiskk/steam-game-reviews` *and prepare a ML model which will be used to predict "user suggestion" category for future game reviews in production. Please use Python. There are 4 datasets in total:*

- game_overview.csv
- sample_submission.csv
- test.csv
- train.csv.

Solution: can be found here `https://github.com/stazam/DTSE-project`

Description of GitHub repository: repository is structured into 4 folders:

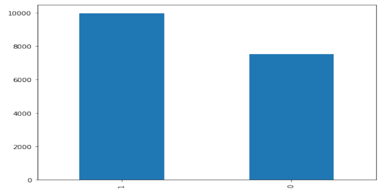
1. **Data** folder: contains
   - original data from Kaggle competition (`https://www.kaggle.com/whoiskk/steam-game-reviews`) - game_reviews.csv, train.csv, test.csv, sample_submission.csv.
   - two .pkl files - df_test_merged.pkl, df_train_merged.pkl, which were created in exploratory analysis notebook for purpose of text processing.
2. **Documentation** folder: contains Documentation.pdf file with a detailed description of exploratory data analysis and model building.
3. **Functions** folder: contains help_functions.py file with functions necessary for creating diagnostic graphs and stacking method.
4. **Model-building** folder: contains two .ipnyb notebook files
   - DTSE_steam_data_reviews_exploratory_analysis.ipynb - exploratory data analysis of original data files.
   - DTSE_steam_data_reviews_model_building.ipynb - description of the whole model building process, from text preprocessing to final user suggestion prediction.
5. **Results** - contains results.pkl file with results of modelling. It is a data frame with two columns: *review_id* and predicted *user_suggestion*.

1 Exploratory data analysis

The detailed shape, missing values, types of variables, number of categories, and much more about *Train.csv, Test.csv, game_review.csv* can be found in *DTSE_steam_data_reviews_exploratory_analysis.ipynb* notebook. Here I would summarize the main findings from the analysis:

- data files *train.csv, test.csv* contains some **missing observations** - variable *year*, which needed to be input for the purpose of some machine learning models, which are not able to deal with it. We used *kNN imputer* technique, which is based on k- nearest neighbours.

- data sets are **balanced**. The target variable *user_suggestion*, which needs to be predicted is nicely spread between both zero and one category, which we can see in the picture below. So we did not have to give special at-

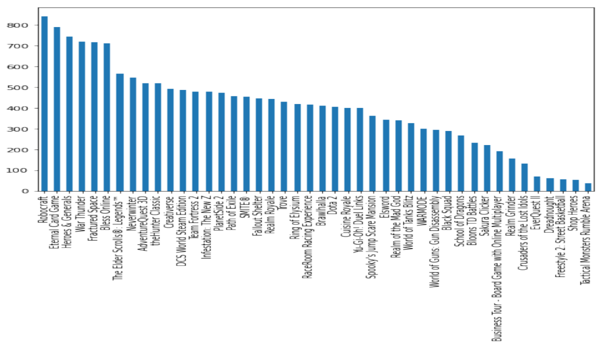


  tention to take care of unbalances. In that case, we could use different technique: undersampling, oversampling, or using some generative models.

- file *game_overview.csv* contains information - variables *developer, publisher, overview, tags*, which is not included in train, test files, and thus we did merge these files.

- there are different games (variable *title*) in train and test set. So we had to modify models to this.

During analysis, we could have also seen, that categorical variables have many different categories (we can see in figure below). In the case of neural networks it is not a big problem, but for models with much fewer parameters what is typically done, is to use "other" category, where we stored categories with few observations.



2 Model building

Model building was following these steps:

1. text preprocessing
2. training model
3. diagnostic of prediction power - using graphs and different metrics.
4. choice of the best model, based on previous diagnostic

Text preprocessing

We started by text preprocessing of merged files, which we created during exploratory analysis. For text preprocessing, we created function *preprocess_data*, which functionality was checked on one example. Function will:

- check for language - since the dominant language in *user_review* was English, I have decided to remove reviews in different languages.
- handle the camel case problem. It means, that if there are two words that are falsely connected, function will split them up, i.e. ThisIs → This Is
- handle the words with capital letters.
- remove all "vaste" characters
- remove "vaste" blank spaces.

After we did text preprocessing on the *user_review* column, we also inspected other "string" variables *title, overview, publisher, developer* but found out that all of them are in English and did not contain "vaste" characters. We also preprocessed *tags* and removed some characters.

As a next step, we bind all mentioned variables from text processing together and create one variable *text*. This was also due to the fact, that train and test files contain different games, so models like decision tree would not work if they would see only train data, because it has never seen values from test data. I also consider using two separate models - with separate parameters, one trained only on *user_review* variable and the second one on *publisher, title, developer*,... But this would not work as well and the second model would probably learn nothing since the mentioned variables are repeated via the training set multiple times (they are merged from *game_overview* file to bring some other information) once with the value of the target variable 1 and other time with 0.

Another thing which we considered was removing stop words and to use lemmatization to improve performance. Because when we printed out the 10 most frequent words in our corpus we got words like *the, for, and, to...*, but this was expected (we can see words below). After the process, the most typical word was *game* which was again expected, since we are dealing with game reviews data (we can see words below). But loosing stop words could lead into two scenarios:

- we will improve prediction power because we lose noise from sentences,
- we will sort of loose some sense from sentences thus performance. decrease.

```
[('game', 67825),
 ('your', 68858),
 ('in', 76900),
 ('you', 82914),
 ('of', 109945),
 ('a', 110048),
 ('to', 151913),
 ('and', 192385),
 ('the', 208385)]
```

```
[('new', 19513),
 ('battle', 24014),
 ('multiplayer', 24938),
 ('player', 25798),
 ('free', 32048),
 ('world', 32864),
 ('play', 44438),
 ('game', 73434)]
```

Model training and diagnostic

After we prepared our data we started with the modelling part. We considered two models with and without removing stop words. So all in all, we considered 4 models:

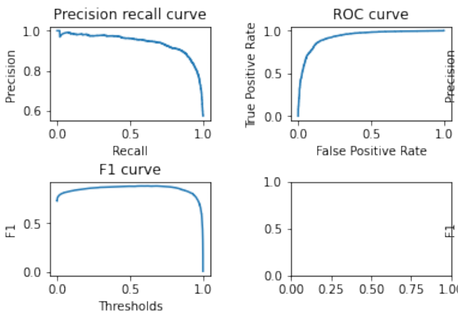
1. Bidirectional LSTM layer (try to add CNN layer) without removing stop words and lemmatization
2. Bidirectional LSTM layer (try to add CNN layer) with using stop words and lemmatization
3. BERT - transformer for text classification without removing stop words and lemmatization.
4. BERT - transformer for text classification with removing stop words and lemmatization.

The choice of BERT was pretty clear, since it is a *state-of-art model* right now, which is pre-trained and should perform well. The second choice of Bi-LSTM layers was *state-of-art* till the 2016 year (after transformers came) and I have also the most experience with using this type of model for language processing. We used four different metrics - *accuracy, F1 score, precision,*, and *recall* to compare our models and some diagnostic graphs which are all shown below.

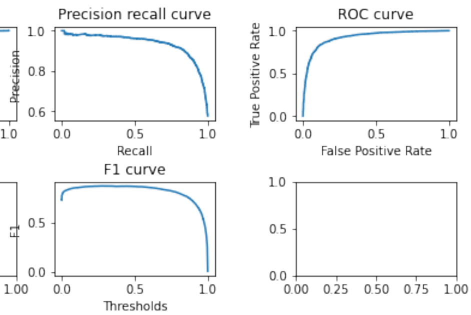
model	accuracy	F1	precision	recall
Bi-LSTM with stop words	85.7	88.1	84.0	92.7
Bi-LSTM without stop words	85.6	87.1	89.3	85.1
BERT with stop words	63.6	75.2	61.6	96.7
BERT without stop words	66.3	68.5	73.7	64.1
Bi-LSTM + xgb with stop words	87	88	87	90

We also used the stacking method to improve prediction power (the diagram of the model can be found below). We considered different models *xgb, decision tree, logistic regression function*. The results can be found up in the table as fifth model

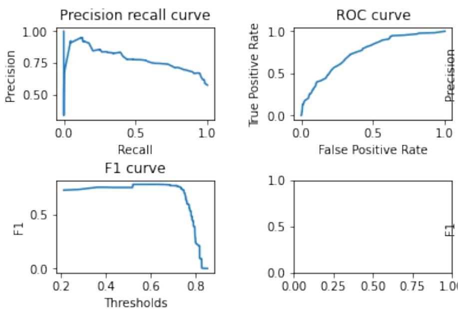
Some comments on results: we can clearly see that removing stop words and lemmatization does not improve nor drastically decrease the performance of our model. Also, pretty interesting is to find out, that BERT models did not work in this case as well as common Bi-LSTM neural networks. The stacking method also does not make big improvements.



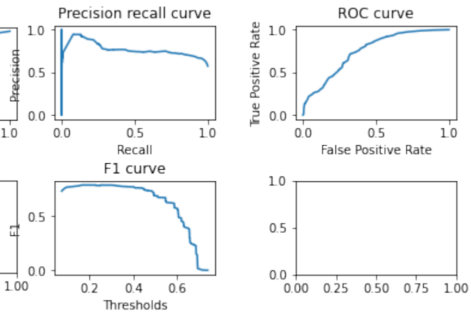
Bi-LSTM with stop words



Bi-LSTM without stop words



BERT with stop words



BERT without stop words

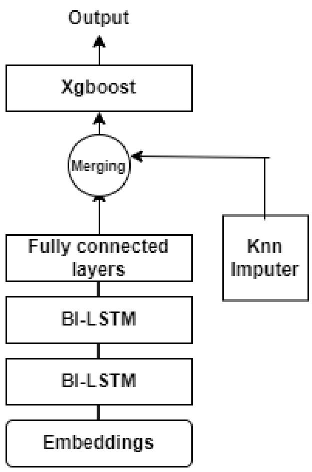


Diagram of a model stacking

Choice of the best model

From the previous table and also graphs, we can see, that it is moreless tie between Bi - LSTM model with and without stop words and lemmatization. Based on the F1 score, which was slightly in favour of model with stop words, we chose to use this model to make final predictions on *test.csv* file. The file *results.pkl* with these predictions can be found in folder results in GitHub repository.

Finally, possible improvements for all models, which we could consider to the future:

- look at badly classified examples and try to find some pattern in them and remove it or maybe hardcode some rules.
- Try to improve hyperparameters of the model.
- Use different types of regularization techniques.
- Improve text preprocessing.