NLP Report

Maja Świerk and Jakub Konieczny

Abstract

The aim of this project was to analyze a collection of comments concerning drinkable and edible products and then use ML algorithms for comments classification.
We began with preprocessing, which included among others removing useless characters, and capital letters and lemmatizing all terms. After that, we vectorized each opinion with different configurations and selected useful features.
In the end, we run three classification algorithms: SVM, Random Forest and XGBoost. We achieved the best results using TFIDF + N-grams + POS tagging vectorization with the XGBoost algorithm with accuracy at a level of 0.75.

Keywords: NLP, TFIDF, Classification, Random Forrest, SVM, XGBoost

1 Problem description

The purpose of this project is to predict ratings for the products, based on the text of the review and the short conclusion.

The dataset contains 10 different features:

- Id
- ProductId
- UserID
- ProfileName
- HelpfulnessNumerator
- HelpfulnessDenominator
- Score
- Time
- Summary
- Text

and over 50 000 samples.

Throughout the project, we have focused on 'Summary' and 'Text'. On them, we have done prepossessing and four different types of vectorization.

2 Methods and materials

2.1 Prepossessing

In prepossessing step we applied following procedures:

- Removing capital letters
- Removing useless characters
- Removing duplicated words (followed by each other)
- Removing duplicated characters (followed by each other
- Removing contractions
- Lemmatizing words

In the TFIDF we added the stop_word parameter to filter English stopwords. Additionally, we tried correcting spelling errors, but this process was taking too long and we abandoned it.

2.2 Vectorization

For vectorization we used the TfidfVectorizer method from the sklearn library. For each of the configurations, we used different vocabularies for the 'Text' column and the 'Summary' columns. Furthermore, for both of them, we added parameter min_df, that makes the algorithm take into account only the words that appeared more times than specified value. For 'Summary' it was set at 2 and for 'Text' at 5.

2.2.1 TFIDF

```
ct = make_column_transformer(
    (TfidfVectorizer(min_df=2, stop_words='english'), 'Summary'),
    (TfidfVectorizer(min_df=10, stop_words='english'), 'Text'))
```

2.2.2 TFIDF + N-grams

In this configuration we added n-grams with n between 1 and 3.

```
ct_ngram = make_column_transformer(
    (TfidfVectorizer(min_df=2, ngram_range=(1,3), stop_words='english'), 'S
    (TfidfVectorizer(min_df=10, ngram_range=(1,3), stop_words='english'),
```

2.2.3 TFIDF + N-grams + POS tagging

In this configuration we included also POS tagging, by adding part of speech tags at the end of the words (in format word_tag).

```
ct_ngram = make_column_transformer(
    (TfidfVectorizer(min_df=2, ngram_range=(1,3), stop_words='english'), 'S
    (TfidfVectorizer(min_df=10, ngram_range=(1,3), stop_words='english'),
```

2.2.4 TFIDF + N-grams + POS tagging + other features

In this configuration we provided 2 new features:

- Number of words in the record
- Number of sentences

```
def words_number(text):
  words_in_text = text.split(" ")
  return len(words_in_text)

summary_words = []
for row in df['Summary']:
  summary_words.append(words_number(row))

text_words = []
for row in df['Text']:
  text_words.append(words_number(row))


other['s_words_count'] = summary_words
other['t_words_count'] = text_words
```

2.3 Chosen algorithms

For the classification, we decided to use the SVM and Random Forrest classifiers. On both of them we carried out parametrization using RandomSearchCV with 10 iterations and cross-validation with k = 3 (in the case of SVM k = 2 in order to save time). Additionally, we applied XGBoost classifier without any changes to hyperparameters.

2.4 Trainign and testing datasets

For learning and testing, we used provided dataset, however, we used only part of it because we did not manage to correctly extract all the rows. We also tried making the learning dataset more balanced by over-sampling using SMOTE function, but it was not improving the results.

2.5 Parametrization

The SVM was parametrized using following search with parameters:

```
parameters = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf', 'poly', 'sigmoid'],
              'random_state': [0]}

gcf = RandomizedSearchCV(svm, parameters, n_iter=10, verbose=3, cv=2)
```

And the Random Forrest with following:

```
parameters = {'n_estimators': [5, 10, 50, 100, 200, 300, 500, 1000],
              'max_features': [None],
              'max_depth': [1,2, 3, 4, 5],
              'min_samples_split': [2, 5, 10, 20],
              'criterion': ['gini', 'entropy'],
              'bootstrap': [True],
              'n_jobs': [-1],
              'random_state': [0]}

gcf = RandomizedSearchCV(rf, parameters, n_iter=10, verbose=3, cv=3)
gcf.fit(X_train, y_train)
```

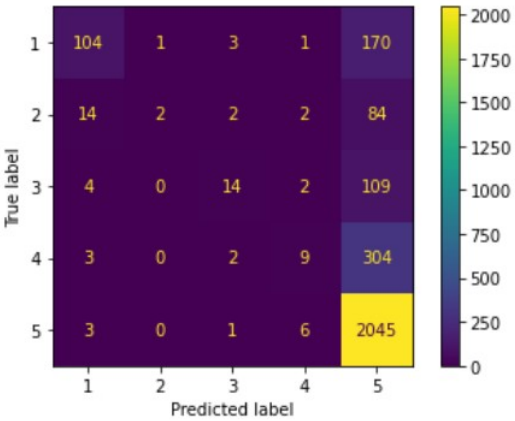
3 Experiments and results

As evaluation metrics, we used a confusion matrix, F1 score and accuracy. We had to perform clustering for three algorithms for each configuration. Results of accuracy and F1 Score are presented in below table:

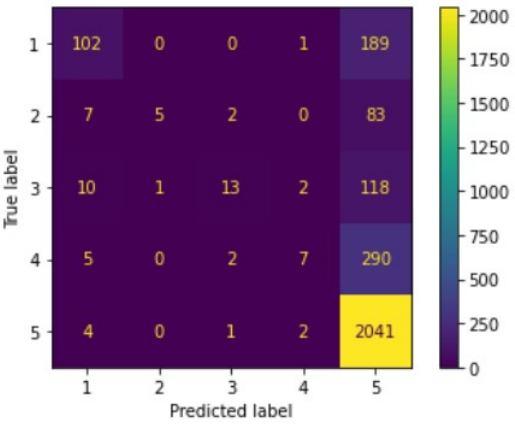
	ML Algorithm	Accuracy	F1 Score
TFIDF	SVM	0.705	0.584
	Random Forest	0.723	0.628
	XGBoost	0.754	0.676
TFIDF + N-grams	SVM	0.708	0.588
	Random Forest	0.723	0.622
	XGBoost	0.751	0.673
TFIDF + N-grams + POS tagging	SVM	0.733	0.639
	Random Forest	0.74	0.652
	XGBoost	0.754	0.681
TFIDF + N-grams + POS tagging + other features	SVM	0.74	0.652
	Random Forest	0.735	0.642
	XGBoost	0.756	0.679

To avoid too many pictures, we will include the best results of confusion matrices for each configuration:

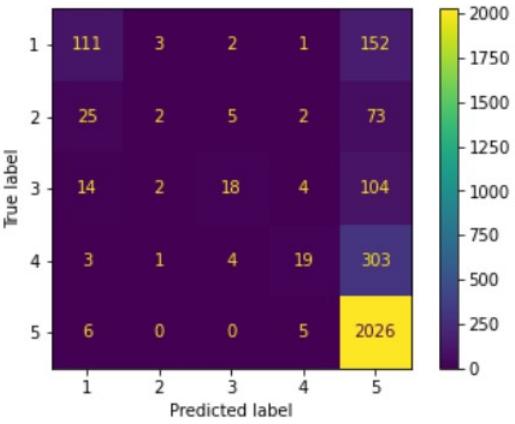
- TFIDF



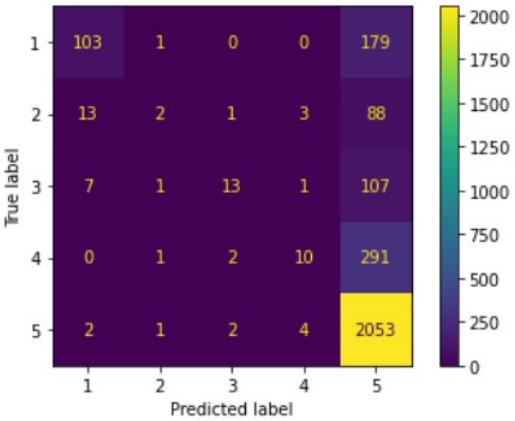
- TFIDF + N-grams



- TFIDF + N-grams + POS tagging



- TFIDF + N-grams + POS tagging + other features



4 Conclusions

We achieved the best results using TFIDF + N-grams + POS tagging vectorization. Although we made parametrization for both SVM and Random Forest, XGBoost turned out to be the best algorithm. The F1 Score was 0.68 and the accuracy was 0.75.

Those results are not great, especially taking into account that most of the data is rated at 5 (around 2/3 of the dataset), and our algorithm is highly biased for giving a rating of 5. It can also detect some 1-star ratings, but it is not able to correctly predict any of 2,3, and 4 star ratings.

For future improvements, we could find a way of extracting all the data from the csv file and in that way obtain a more diverse dataset. Moreover, we could

apply parametrization on the XGBoost algorithm and make improvements in vectorization and feature selection processes.