Weka report

Sisällysluettelo

INTRODUCTION **3**

DATA PREPARATION AND PREPROCESSING **3**

Unnecessary variables 3
Label class 4
Text mining 4
Class imbalance 5
Missing values 6

MODEL SELECTION AND EVALUATION **6**

Selection of classifier 6
Selection of testing set 7
Classifier assessment 7
Fine-tuning the model 8
Evaluation of the final model 9

Introduction

In this report I address the second task of business scenario 1.

In the business scenario 1, the company had actually purchased a large amount of similar reviews from Facebook on the company, which were pure text without giving any numeric ratings on the service product. They wanted me to develop machine learning models and use the models derived from TripAdvisor review data to understand Facebook reviews.

The task 2 asked to use the review title and text to predict how influential a review would be.

The influence of a review is measured with the number of helpfulness votes that a review has received. The bigger the number, the more influence a review has.

In the dataset the variable *num_helpful_vote* got integer values between 0 and 30. Based on this the problem would be a classification problem. However, in theory the variable can have also decimal number values. Also, number of votes is not restricted to certain range but can be any number larger than or 0. These properties make it possible to treat these values as continuous values. When the label class gets continuous values (or values that can be treated as such) the problem turns into **regression problem**.

As our dependent variables are strings the task also included **text mining part**, which enabled us to study the effect of individual words.

Data preparation and preprocessing

Unnecessary variables

The first step was to remove any unnecessary variables from the dataset. Only variables needed were *title*, *text*, and *num_helpful_vote*. All other variables were removed.

Label class

The *num_helpful_vote* was selected as class right after the first step and was re-selected every time it moved out of its place.

Text mining

The next step was to extract the words from the title and the text. The title and the text were handled as one, meaning only one extraction was made to produce one joint word vector. This was done by using filter *StringToWordVector.* See settings in image 1.

A large number was picked for *wordsToKeep* since goal was to extract every single one of the words. All words were changed into lowercase, and only *stopwordHandler eas used*. Stemmer was chosen to leave out as it seemed to produce too messy results.

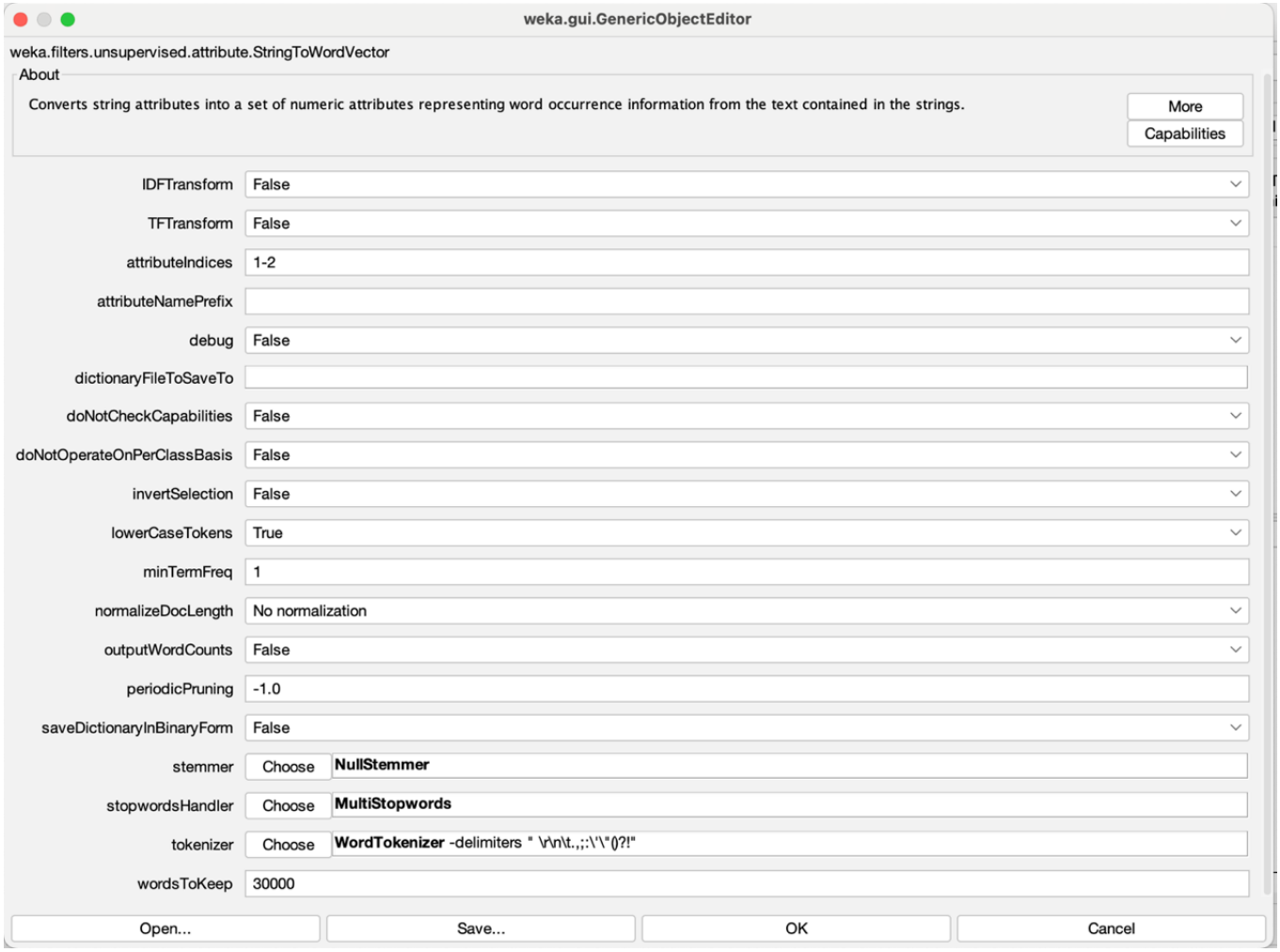

Image 1. The settings used for word extraction.

After extraction *num_helpful_vote* was re-selected as class. At this point the dataset had over 26000 attributes, so filter *attributeSelection* was used to select the best 100 attributes based on correlation and ranking. See settings in image 2.

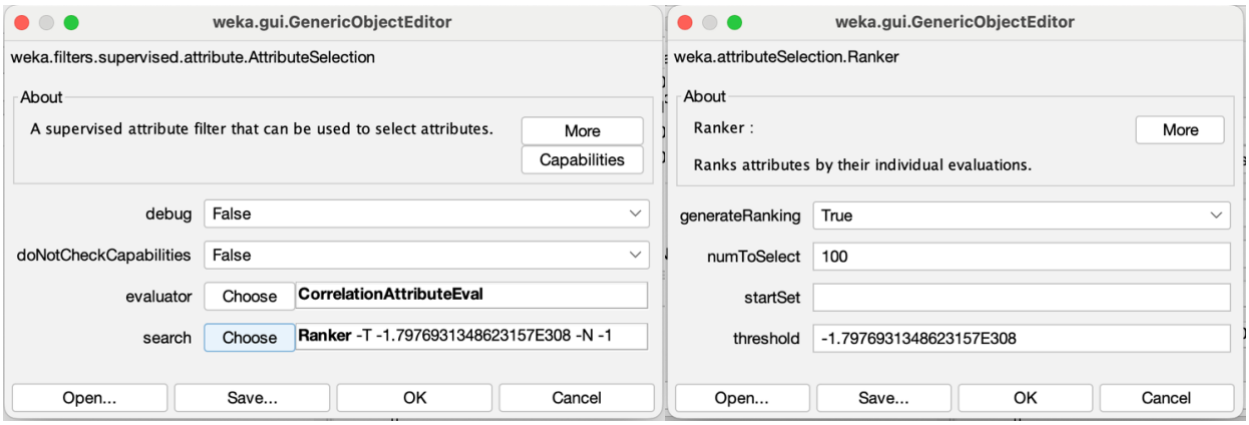

Image 2. The settings used for attribute selection.

Class imbalance

Before trying out any algorithms, the severe class imbalance in class still needed to be addressed. (See image 3.) Filter *ClassBalancer* was used to balance the data. For starters, default settings with 10 discretization intervals were used.

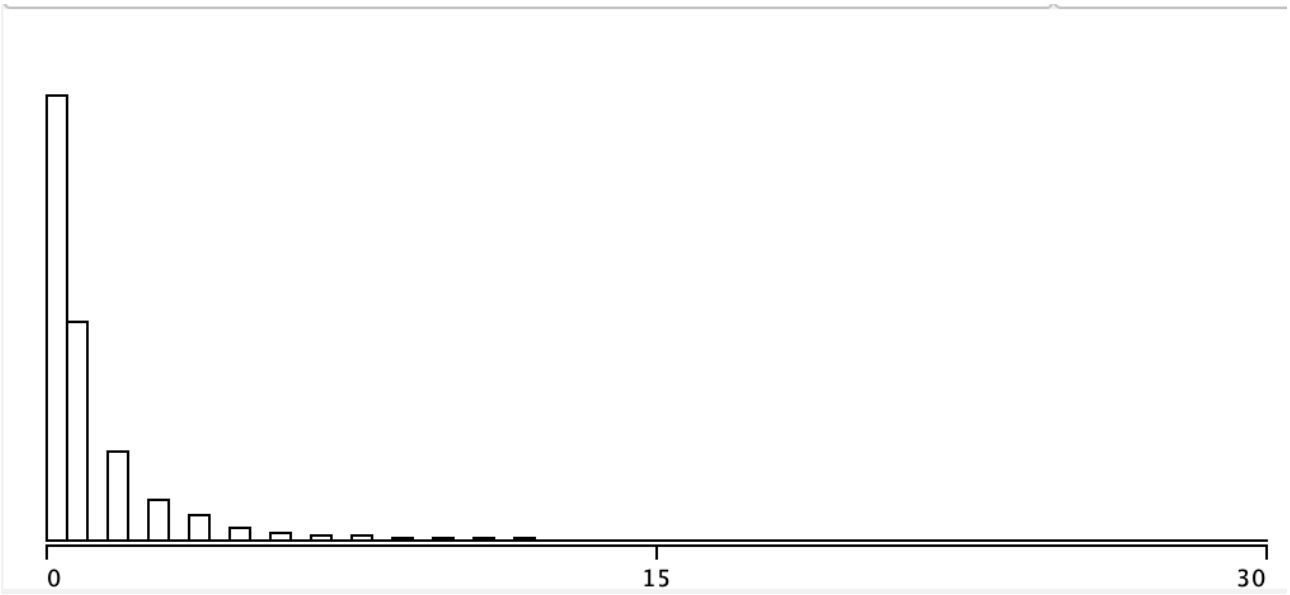

Image 3. The class *num_helpful_votes* before balancing.

Missing values

The original dataset had a lot of missing values, but these got removed during the preparation. This meant that there were no missing values to handle.

Model selection and evaluation

Selection of classifier

Six regression classifiers (*ZeroR* (for benchmarking), *LinearRegression*, *SMOReg*, *IBk*, and *RandomForest)* were tested out. They were used inside *WeightedInstancesHandlerWrapper* (setting at default besides classifier) since the wrapper was said to support weighted instances that we had after balancing the data. The default settings were used for all but *LinearRegression* and *IBk* classifiers. For *LinearRegression*, both 5M method and Greedy method were tested as attribute selection methods. For *IBk*, feature of the classifier was taken advantage of to automatically find best value for k. See used settings in image 4.

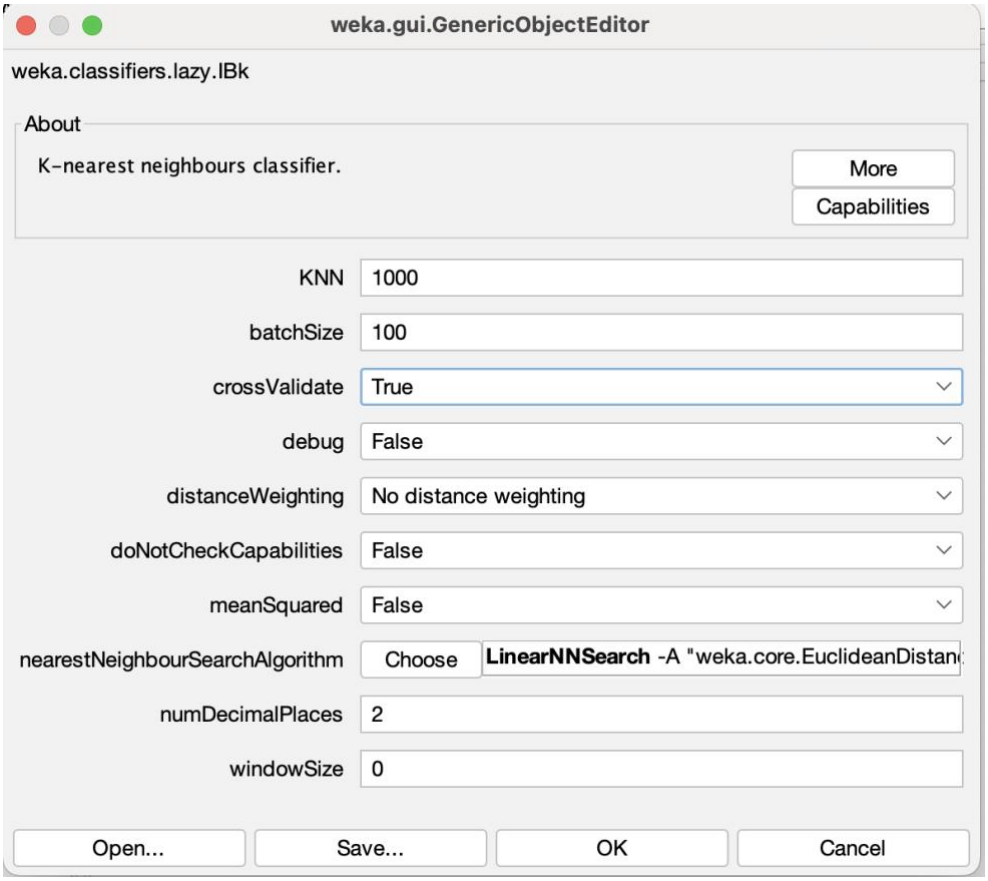

Image 4. Used settings of IBk classifier.

Selection of testing set

For testing, cross-validation with 10 folds was used, as this helps to prevent overfitting of the model. Overfitting was very likely because the imbalanced dataset.

Classifier assessment

Regression models are usually assessed based on correlation coefficient, mean absolute error (MAE), or rooted mean squared error (RMSE). The correlation coefficient tells how well the chosen attributes predict the predictable variable, while MAE and RMSE tell how big the average prediction error is. RMSE is used when one wants to give much larger penalty to bigger errors than smaller errors. When penalties in case of MAE grow linearly, penalties in case of RMSE are squared, i.e. grow exponentially. I would recommend using MAE in our case since predicting helpful votes correctly is not so big deal compared to many other scenarios.

Based on all evaluators looks like the winner in this round is Linear regression classifier with greedy attribute selection method wrapped in *WeightedInstancesHandlerWrapper* filter.

In next phase, the idea is to fine-tune the found model to get even better model.

Table 1. Summary of evaluation metrics of tested models

Classifier	**Correlation coefficient**	**MAE**	**RMSE**
ZeroR	-0.3569	7.0308	8.7684
LinearRegression			
- 5W	0.3007	7.441	9.8627
- Greedy	0.6658	5.8134	7.1136
MultilayerPerceptron	0.6644	6.8231	9.2998
SMOReg	-0,0606	8.2142	11.2324
IBk	0.5864	7.6537	9.5541
RandomForest	0.1842	7.4681	9.7707

Fine-tuning the model

To fine-tune the previously developed model the first thing tried was to change around the number of discretization intervals in *ClassBalancer* filter. Table 2 presents a summary of these results.

Table 2. Summary of relevant evaluation metrics of tested variations

Number of intervals	Correlation coefficient	MAE	MRSE
1	0.3655	1.1407	1.9563
2	0.7186	6.1468	8.504
3	0.8668	5.4209	6.3447
4	0.7944	6.056	6.9288
5	0.7383	6.2991	7.6473
6	0.8137	5.0624	6.0569
7	0.68	6.1202	7.4636
8	0.6676	6.2911	7.5302
9	0.6883	5.6627	7.0124
10 (Initial model)	0.6658	5.8134	7.1136

One can see that the discretization interval of 3 has already made our initial model a lot better. The next thing tried was to change the setting of *forceResampleWithWeights* in *WeightedInstancesHandlerWrapper* filter from False to True. Meaning that we make filter to force resampling.

The result was even better model with 0.8857 correlation coefficient and mean absolute error of 4.3303. (See image 5.) As there are not many changes to make and the model is really good, we take it as our final model.

```
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

WeightedInstancesHandlerWrapper
================================

Force resample with weights: true
Base classifier:
- command-line: weka.classifiers.functions.LinearRegression -S 2 -R 1.0E-8 -num-decimal-places 4
- handles instance weights: true


Linear Regression Model

num_helpful_votes =

     14.2411 * truck +
      2.2125 * scottish +
      6.0517 * 24-hour +
      6.6314 * crap +
     -1.8463 * nintendo +
     10.4198 * ruin +
     15.0955 * greyline +
      6.9796 * luck +
      9.1652 * southgate +
      7.6419 * review +
      4.9045

Time taken to build model: 43.04 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.8857
Mean absolute error                      4.3303
Root mean squared error                  5.706
Relative absolute error                 39.4001 %
Root relative squared error             45.6007 %
Total Number of Instances             3118
```

Image 5. The output of the final model.

Evaluation of the final model

Correlation coefficient of 0.8-1.00 is deemed as very strong correlation. This means that our final model is a very strong fit to predict how influential a review will be.

The model reveals that words “*truck”, “scottish”, “24-hour”, “crap”, “ruin”, “greyline”, “luck”, “southgate”, and* “*review”* positively affect the level of influence the review has. On the other hand, word “*nintendo*” decreases the number of helpfulness votes. Since we used wrapper

and forced it to resample, these are the best attributes to use to predict the level of review's influence.

The average absolute error is about 4 votes. This is not bad when we consider that most reviews in our TripAdvisor data have 0 votes. Four helpfulness votes mean very little, even if the actual number of votes were 0.