Amazon Reviews - classification of a multi-class dataset

Artur Juraszek

Project goal

The dataset being the center of this project is a small-ish (about 1.5 GB) subset of reviews produced by Amazon users throughout the last two decades. It, among other things which this time are of no interest to us, contains two types of data - an overall rating, in form of 1 to 5 stars, which, as one could guess, represent the given user's attitude into a particular product, and, more importantly, a (usually) short corresponding text, consisting of the user's review written in English. The main question we would like to ask is: **Knowing the review's text content, can we predict which 1 to 5 stars rating it gives?**

Overview of the dataset

The review database, provided through courtesy of Julian McAuley from UCSD stores exactly 1,697,533 records taken from *Movies and TV* Amazon's store category. Unfortunately, it's extremely unevenly distributed - 5 stars ratings account for the vast majority of it, and 2 'lowest' classes are visibly underrepresented.

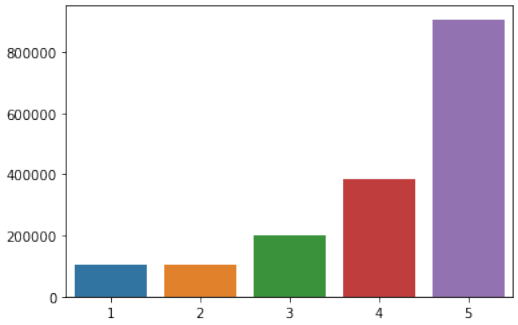

Figure 1: Class distribution, 1 to 5 stars

An arbitrarily split has been made, to 75% of data being a training sample and another 25% - a testing set, keeping their uneven distribution the same as the full dataset.

Three approaches has been leveraged to tackle this balance problem:

- Keeping the class proportions unchanged
- Discarding some arbitrary data from overrepresented classes, i.e. randomized undersampling, presented below

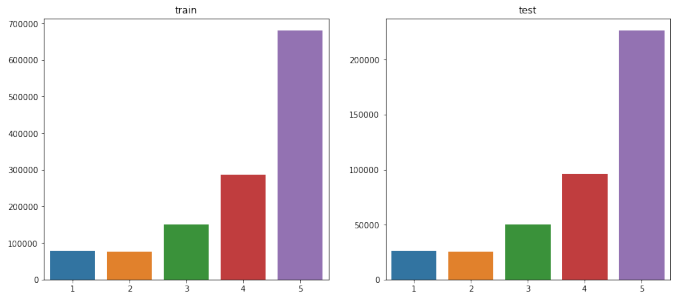

Figure 2: Shard's distribution has been preserved

- Augmenting the training dataset in form of copying the minor-iest classes a few times

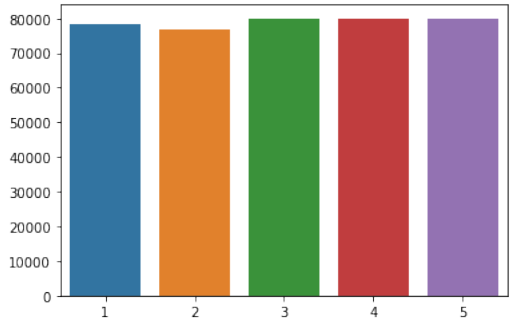

Figure 3: Class distributions after randomized downsampling

Classification accuracy measure

Due to dataset's imbalance, the tempting thought to use the simple percentage-based accuracy score as a measure to compare classifiers on might in reality be not the best one - a smart, but not so much correct hypothetical model could strongly favor the 5-star class and score higher than more 'smooth' ones, in extreme case, even by always predicting '5'.

To mitigate this risk, a so called **balanced accuracy score** has been used, defined as an average of single-class accuracies. A custom metric measuring seriousness of the errors was also calculated - due to classes having a trivial order defined on them ($1 > 2$, $5 > 3$ and so on) one could reason about the errors being made in terms of averaged distance of the error, namely:

$$orderedErr(class) = \frac{\Sigma \mid pred_i - true_i \mid}{\#classifiedIncorrectly}$$

Where *i* are samples which belong to class *class*.

Measuring area under the **Receiver Operating Curve** looked promising too, but due to apparent lack of consensus on which flavor of it (1 vs 1 or 1 vs rest) should be used in a multi-class scenario, it has been left alone.

Text preprocessing

Three methods has been tried:

- Keeping the data dirty
- So-called "stop words" removal
- Stemming, in simple words removing the word suffixes introduced by English language's inflection, so, for example 'fishing' and 'fisher' both end up in the same bucket, namely 'fish'

Another popular technique, foreign accents removal was immediately discarded, as the data had already been represented as 7-bit ASCII, so no room for improvement (or the opposite, possibly) here. Also, after initial experimentation (with Naive Bayes approach, in belief that it will be representative enough to deduct anything from these manual experiments) it turned out that stop-words removal based on popular 'stop-word lists' showed no improvement, and in most (all) cases produced worse results than the dirty data alone. This can be explained by appearance of words such as "however" in this lists, which in our case would probably suggest that the review author has some "BUT" - thus indicating this review is not a 5-star one. To avoid combinatorial explosion of methods being used on not-so-fast hardware, it has thus been discarded too.

Word representation

Again, three vectorization approaches has been tried:

- Treating text chunk as a position independent vector of word occurences in it - reviews span rows, words used across the whole training dataset make up columns
- A small twist to the method above - ignoring word repetitions, i.e. storing either 0 or 1 instead of a proper occurence count
- Popular TF-IDF, more or less a transformation in which we replace word/term occurence counts with a product of **t**erm **f**requency and **i**nverse **d**ocument **f**requency - i.e. information how often the word/term occurs in the document (review) is "divided" by information on how often it is across the whole dataset

Note that words not necessarily mean words here, they could as well be N-grams and in fact are, in experiments conducted on best-promising estimators.

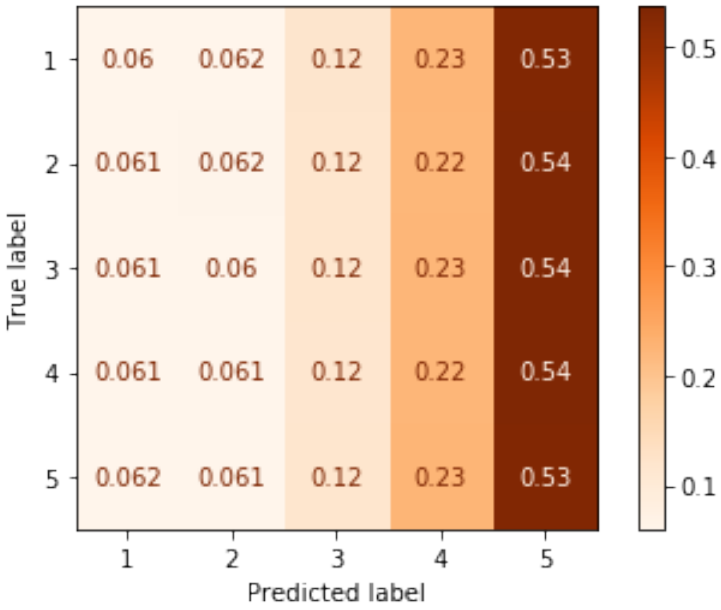

Figure 4: Confusion matrix of the baseline estimator

Baseline estimator

A really dumb, randomly guessing classifier has been prepared as a baseline for other estimators. Not surprisingly, it can be summed up with:

- Balanced accuracy of **about 20%**
- "Real-world" accuracy, not taking class distribution into account (from now one we will refer to it via similar expressions) of **35.645%**
- Very heavy bias towards classes 5 or 4 - the overrepresented ones. Easily noticable on the confusion matrix provided below.

Naive Bayes

First classification technique tried was Naive Bayes - a few nested loops were ran, resulting in the cartesian product of two sampling methods (randomized downsampling and untouched dataset), two word representations, either stemming usage or not preprocessing the data at all, and last, but not least - two hyperparameters to NB, specifically **alpha** and a boolean of whether prior probabilities assigned to classes should be uniform, or deducted from the training dataset.

Results were sorted on two independent conditions - a) balanced accuracy, b) real-world accuracy
A clear supremacy of **TF-IDF** representation could be observed when evaluated by balanced acc, among top 100 estimators, 84 were using this representation. Surprisingly enough, it turned out to be the opposite after sorting by real-world acc - all 100 of top 100 estimators were based on simple count-based vectorization.

Similar correlation was observed in the matter of training dataset - 100 out of top 100 balance-acc-sorted estimators were trained on either the randomized, forcefuly balanced downsampled dataset or the oversampled one, while 100 out of 100 real-world-acc sorted ones were trained on the full, untouched dataset. This **can be easily explained - exploiting the class imbalances, which are better reflected in the "raw" training set will lead to higher real-world accuracies** (and probably the opposite for balanced-acc). No significant value in having the input text stemmed was observed, thought no negative effect, neither - stemmed and non-stemmed classifiers were interleaved among the top ones.
Best of the achieved results was:

- For the balanced-accuracy-proritizing estimator, **47.3654%** balanced accuracy with **48.8517%** of real-world, biased accuracy.

  Averaged order-aware errors: **#1: 1.619118, #2: 1.246848, #3: 1.309533, #4: 1.316007, #5: 1.673930, AVG: 1.433087**

  With **alpha = 0.5**, randomly downsampled training set, no preprocessing and **TF-IDF**

- For the real-world-prioritizing estimator, **40.7792%** balanced, **60.3153%**

  Their confusion matrices are presented below.

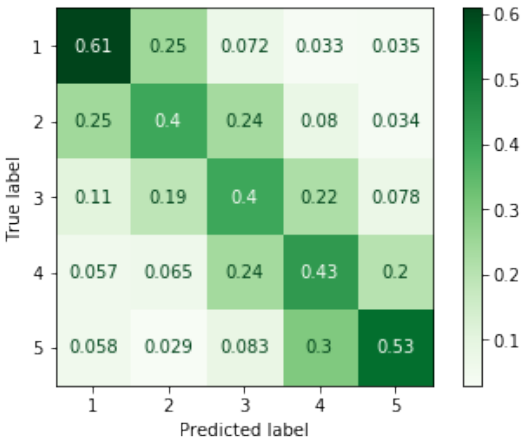

Figure 5: Naive Bayes winner when estimators are rated via their balanced accuracy, note the more even accuracies

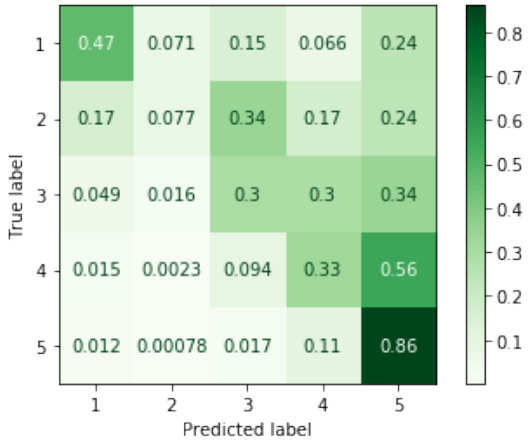

Figure 6: Strongly biased, bias-blind accuracy winner

In the perfect world, it would be easy to find a good compromise between these two accuracies, however, as presented below, the dependencies are not that straightforward.

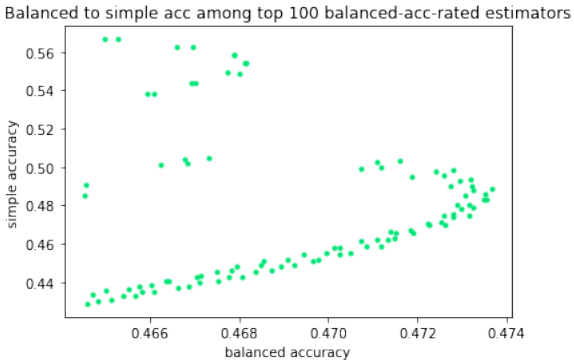

Figure 7: Balanced to simple accuracy

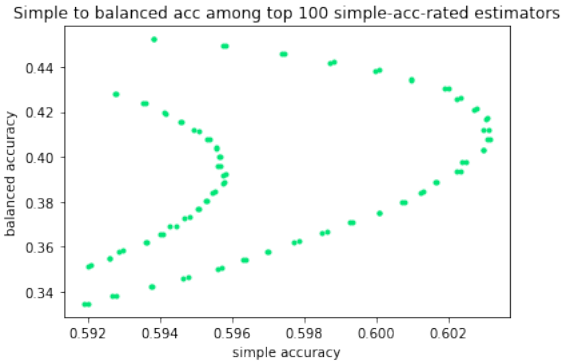

Figure 8: Simple to balanced accuracy

Logistic Regression

Next, a linear approach in the form of Logistic regression was tried. Same algorithm-independent parameter variations

as in Naive Bayes trial has been set, in addition two them, two log-regression specific hyperparameters were also included in the grid search - **C**, a regularization parameter, in theory preventing the model from overfitting and **mc**, the approach to generalizing log-regression to multiclass scenario, taking one of two values - we either train one binary classifier per every class, i.e. leverage a One-vs-Rest method or consider the model being multionmial and train over all classes at once.

Due to hardware constraints, optimization process lying underneath the algorithm was set to stop after 100 iterations, regardless of how much has it really converged - however, even with that constraint the log-regression approach proved better than Naive Bayes. Also, the 10 highest scoring hyperparameters combinations were re-trained with a much higher limit of 1,000 iterations.

Again, TF-IDF representation proved to behave better than simple counting - when the result list has been sorted by estimators performance, tf-idf based ones accounted for nearly half of the top results.

Also, One-vs-All approach had better results on average than Multinomial.

Two best classifiers on 100 iterations:

- Balanced accuracy of **53.7067%**, standard accuracy of **59.4022%** (**C = 0.05, stemming, One-vs-All, TF-IDF, oversampled training set**)

- Balanced accuracy of **51.9371%**, standard accuracy of **60.3992%**

And the best on 1000 loops iteration limit, surprisingly, having all of the hyperparameters the same as the 100-iterations winner with the exception of doing Multinomial instead of OvR multiclassing, which actually stopped after 545 loops:

- Balanced accuracy of **53.8411%**, standard accuracy of **58.8100%**

  Averaged order-aware errors: **#1: 1.663587 #2: 1.231656 #3: 1.275698 #4: 1.214740 #5: 1.614087 AVG: 1.399954**

So the variation between these two metrics in best-predicting estimators is much, much less than before, in Naive Bayes. Thus, choosing the top 1 instance would not result in such a big tradeoff. Picture below illustrates the regularization parameter's behavior on estimators class chosen as best on average.

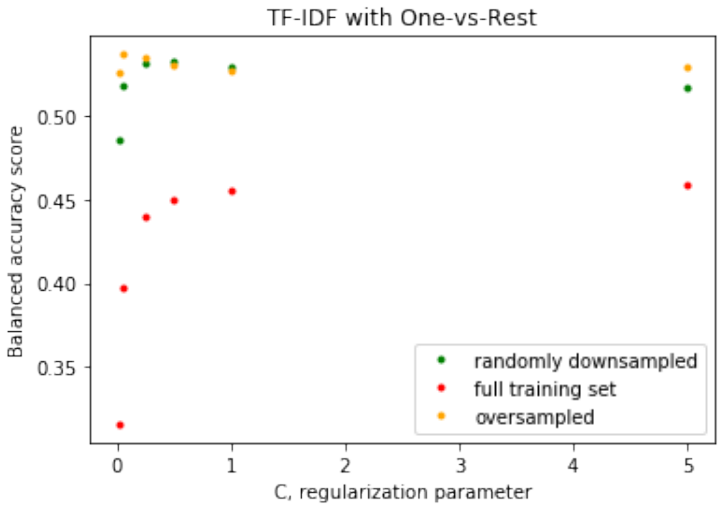

Figure 9: Regularization parameter behavior

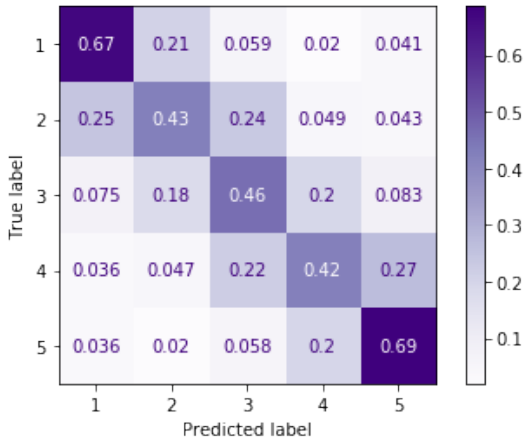

Figure 10: Best Logistic Regression estimator (when rated with balanced accuracy)

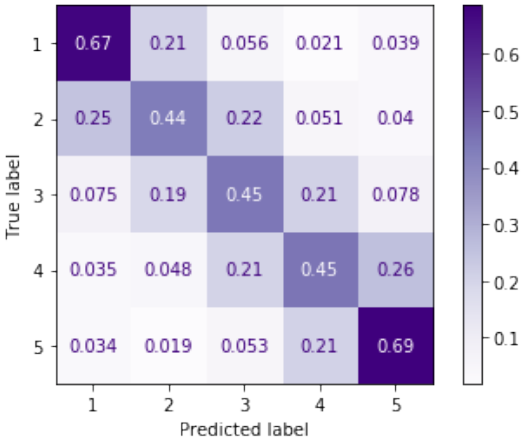

Figure 11: Best Logistic Regression estimator (when rated with balanced accuracy, retrained with 545 iterations)

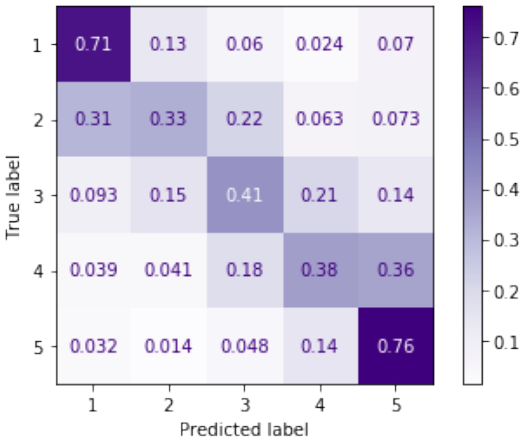

Figure 12: Best estimator when rated with simple accuracy. Note the higher bias towards extreme classses

AdaBoost

Last of the three methods applied was AdaBoosting, an ensemble technique built on one-level decision trees (aka *stumps*). An arbitrary number of estimation levels was assumed to be 10, and three different learning rates were tried. Then, the best behaving hyperparameters combination was retrained with 1,000 levels of boosting.

- Best: balanced accuracy = **33.9933%**, simple accuracy = **42.9667%**, trained on the oversampled training set
- Most promising combination after retraining = **51.0199%**, simple accuracy = **57.0877%**,

  Averaged order-aware errors: **#1: 1.830312, #2: 1.277904,#3: 1.316928, #4: 1.227841, #5: 1.716943, AVG: 1.473986**
- Best on simple accuracy: balanced = **27.2546%**, simple = **54.5826%**, trained on the full training set

Even though the simple accuracy metric is not *that* bad, it's far worse than the previous two - the heavy bias towards class 5 almost mimics the dataset distribution, so the actual results are barely better than a completely random estimator.

Conclusions

As expected, a strong bias towards classes 1 and 5 has been observed - it is probably easiest to detect extremities, as some specific words tend to appear in them - e.g. "splendid" will probably not occur in 1-star review full of curses, and the opposite would hold true for "crap". That, however, might suggests good results in a simpler, 2-class (either 1 or 5) classification task - even with the absolute imbalance between samples count the algorithms managed to do a proper separation.

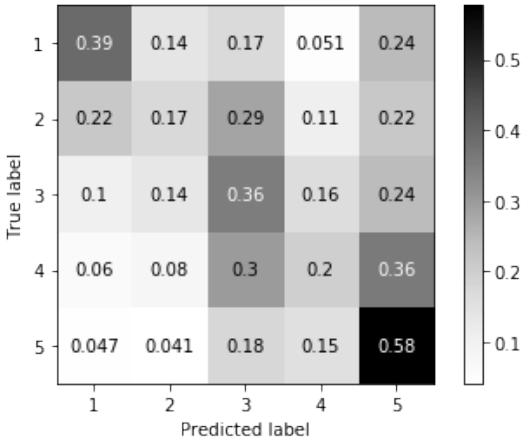

Figure 13: Best AdaBoost-based estimator

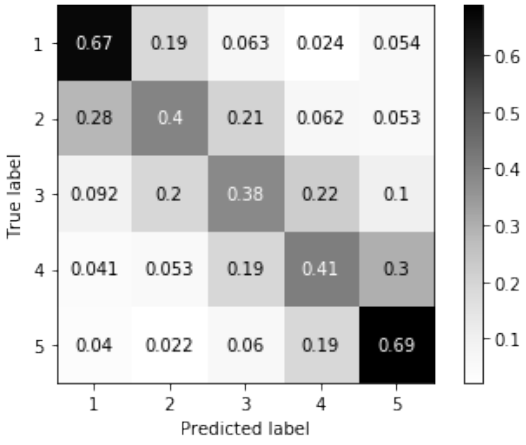

Figure 14: Best AdaBoost-based estimator

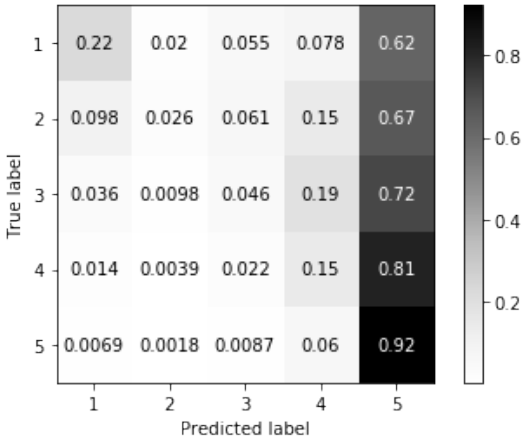

Figure 15: ”Best” AdaBoost-based estimator

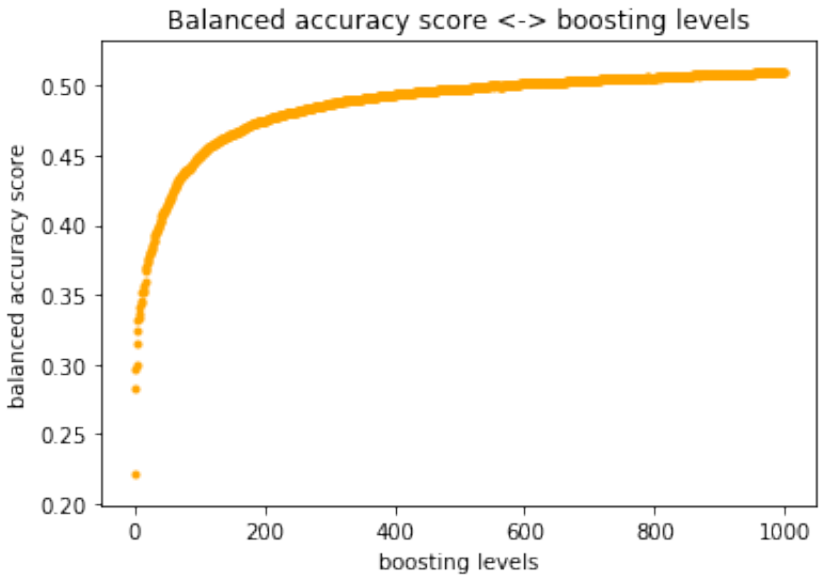

Figure 16: AdaBoost accuracy increase