Amazon Reviews: Testing and Experimenting with Title Classifications

Murilo Brugger Stockinger [1]

1. Introduction

This work's motivation is the creation of an API that access a classifier capable of delivering predictions of categories given a product title. This classifier is going to be achieved by using product reviews data-sets provided by Amazon as training, validation and testing sets.

Experiments and validations will also be shown to analyse the model's performance and efficiency when data from the same domain, but not used for training, is used to generate classifications.

2. Motivations and Technologies

The main focus of technology searching was to find a proper algorithm that was planned, from it's core, to better fit NLP classification problems. For that matter, BlazingText (Gupta & Khare, 2017) appeared to be the best available state-of-art solution that had a plug-and-play environment for testing and training.

To use the environment to it's full potential, AWS SageMaker was used as a base framework for all following experiments. By using SageMaker it was possible to access Amazon S3 services for data storage, including created models, input files and auxiliary files produced during the workflow.

While Amazon has very powerful computational-wise virtual machines, due to the usage of a free experimental account it was impossible to choose those machines. Amazon's guide to experimental machine learning workflows suggests the usage of 'ml.c4.4xlarge' instances for training/validating and 'ml.m4.xlarge' instances for the API and those were the configurations selected.

3. Classifier

BlazingText (Gupta & Khare, 2017) algorithm works in two different modes, being one for unsupervised learning and the other for supervised learning; for this study the latter was used.

To create a classifier based on supervised learning, BlazingText uses a re implementation of the fastText classifier (Joulin et al., 2017). SageMake further extends this model by leveraging GPU acceleration using CUDA kernels, along with other add-ons like Early Stopping and Model Tuning. This comes with a plenty full of tools to help with the hyperparameters.

To achieve a fast and accurate model, fastText uses a multinomial logistic regression described by:

$$-\frac{1}{N}\sum_{n=1}^{N} y_n log(f(BAx_n))$$

Firstly a matrix A is built as a look-up table over the words. The words are then averaged into a text representation, which is in turn fed to a linear classifier. The text representation is an hidden variable which can be potentially be reused. A softmax function is used to compute the probability distribution over the predefined classes. For a set of N documents, a minimization-oriented execution is done on the formulation. x_n is the normalized bag of features of the n-th document, y_n the label, A and B the weight matrices.

This model, originally, could be trained asynchronously on multiple CPUs using stochastic gradient descent and a linearly decaying learning rate.

4. Data Processing

In this experiment two data-sets were used, both from amazon containing reviews from products. Both data-sets are english-based, being one from reviews made in the United Kingdom and the other from reviews made in de United States.

Some processing was required before data-sets were ready to use. The steps can be replicated as follows: i) All commas were replaced by semicolons; ii) All reviews data-sets were converted to comma-separated-values files; iii) All columns except title and category were removed; iv) All title duplicates were removed. With this step it was possible to reduce significantly both data-sets while maintaining important data.

The next step in data processing was to prepare the titles to enter NLP formats. To achieve this, all words were tokenized, followed by a stop-word and punctuation removal and stemming process. Every NLP processing was made with *nltk* library, available on SageMaker framework.

5. Training and Validation

The processed UK reviews data-set was used to train and validate the classifier. As shown in Figure 1, the class distribution does not follow regular distribution, where the classes Music and Video contain more then half of all entries.

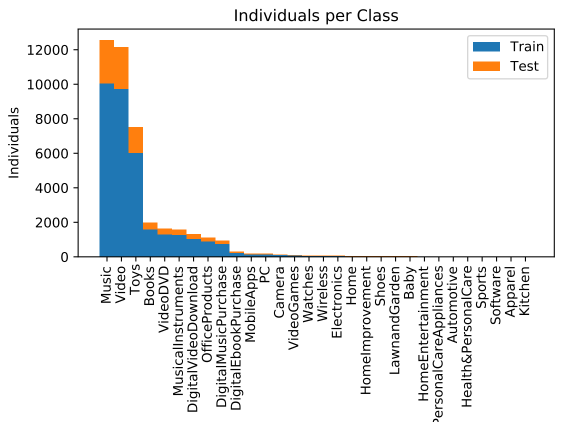

Figure 1. Classes distribution on training/validation set

The data-set distribution forced a few measures to be taken before the set was split in training and validation. Since some classes had few entries, the splitting could lead to all examples from a class being allocated to the validation set, not allowing the model to train on such classes and disrupting the accuracy metric, once all these rows would never be trained, therefore never classified correctly during the validation processing. To avoid such scenario, every class that appear less than 100 times have all it's rows inserted into the training set. For every class with more rows than the minimum threshold, 80% appends the training set and the remaining 20% appends the validation set.

Every classifier contained in SageMaker's framework has a built-in fit method that allows hyperparameters setting using a Bayesian search, where the problem of finding the best parameters is transformed into a regression problem. The hyperparameters tested and their windows are shown in Table 1 and the descriptions in Table 2. All the values are suggested by Amazon, and the number of combinations of hyperparameters tried was limited to the free account's limitation, being possible to use 500 tries to find the best parameterization.

Table 1. Parameters and values used in tuning process.

HYPERPARAMETER	WINDOW	TYPE
BUCKETS	[1000000-10000000]	INTEGER
EPOCHS	[5-15]	INTEGER
LEARNING_RATE	[0.005-0.01]	CONTINUOUS
MIN_COUNT	[0-100]	INTEGER
VECTOR_DIM	[32-300]	INTEGER
WORD_NGRAMS	[1-3]	INTEGER

Table 2. Parameters Description

PARAMETER	DESCRIPTION
BUCKETS	THE NUMBER OF HASH BUCKETS TO USE FOR WORD N-GRAMS.
EPOCHS	THE MAXIMUM NUMBER OF COMPLETE PASSES THROUGH THE TRAINING DATA.
LEARNING_RATE	THE STEP SIZE USED FOR PARAMETER UPDATES.
MIN_COUNT	WORDS THAT APPEAR LESS THAN MIN_COUNT TIMES ARE DISCARDED.
VECTOR_DIM	THE DIMENSION OF THE EMBEDDING LAYER.
WORD_NGRAMS	THE NUMBER OF WORD N-GRAM FEATURES TO USE.

The training process of BlazingText model just output's the accuracy of every training with given hyperparameters for evaluation purposes. The best hyperparameterization reached an accuracy of 77% using the values 7684899, 15, 0.009, 3, 91 and 1 for the parameters buckets, epochs, learning_rate, min_count, vector_dim and word_ngrams, respectively. The hyperparameter's fitting process took 10 hours to finish.

6. Computational Experiments

To test the efficiency and performance of the trained classifier, the processed amazon reviews' data-set of users from United States was used. The distribution present in testing data-set, shown in Figure 2, shows a discrepancy in classes inputs, similar to the one presented in Figure 1.

When comparing the number of rows, even though the US data-set, in it's raw form, had close to 3GB of data, after the removal of all title duplicates, the number of entries gets close to the UK data-set, with close to 800MB. This shows that the number of uniquely reviewed products is similar in both sets.

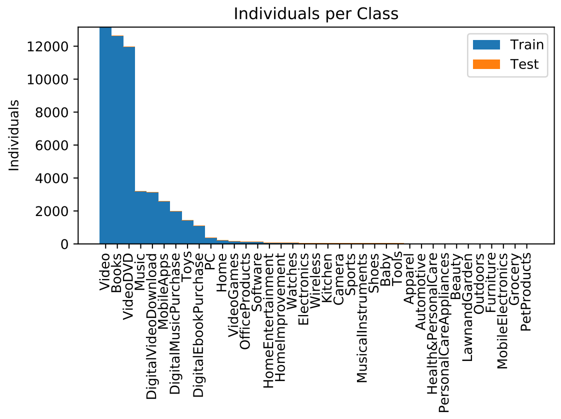

Figure 2. Classes distribution on testing set

Table 3 presents possible values for every metric used to evaluate the classifier performance when tested against the US data-set. *Accuracy* is the ratio of correct predictions to the total number of input samples. The higher the *Accuracy*, the higher the number of correct classifications, but biased if classes distributions are not equal. *Logarithmic Loss* penalizes the model for false classifications, better suited metric for multi-class classifications, the lower the value the better. *Precision* is the ratio of correct positive results by the number of positive results predicted by the classifier. *Precision* can extract the capability of not false-classifying, the higher the value the better. *Recall* is the ratio of correct positive results by the number of relevant samples. Relevant samples are the entries that should belong to the class. *Recall*, then, evaluates the model's capability of correctly classify instances from the same class, the higher the value, the better. *F1-Score* is the Harmonic Mean between *Precision* and *Recall*.

Table 3. Metrics Values Window.

Metric	Window
Accuracy	[0,1]
Logarithmic Loss	$[0,\infty)$
Precision	[0,1]
Recall	[0,1]
F1-Score	[0,1]

Table 4. Classes metrics evaluation for testing without filtering classes not trained.

Class	Precision	Recall	F1
Video	0.97	0.93	0.95
Books	0.63	0.89	0.74
VideoDVD	0.93	0.57	0.71
Music	0.56	0.90	0.69
Toys	0.62	0.44	0.51
DigitalMusicPurchase	0.62	0.31	0.41
MobileApps	0.23	0.42	0.30
DigitalVideoDownload	0.44	0.06	0.10

Of all 37 classes present in the US data-set, only 30 were present in the UK data-set. For this reason, some classes were never introduced in the training process of the classifier. Table 4 shows the classes which the classifier was able to provide truth-full outcome.

The classifier performed differently for every class, with Video, Books, VideoDVD and Music being, respectively, having the best of the set. The better performance of some labels can be explained by the UK distribution of each class in the training process. The classes with higher individuals are among the ones better classified, with a single exception of the Toys class.

Table 5 presents the values for *Accuracy* and *Logarithmic Loss* for the testing experiments without missing-classes filtering.

Table 5. All-Class metrics evaluation for testing without filtering classes not trained.

Metric	Value
Accuracy	0.64
Logarithmic Loss	0.24

A second experiment filtered all classes but the ones with more than 1000 individuals used in the training process. The reasoning behind this experiment is to check the model's performance when tested on the classes that had more individuals than the average used in training. This limits the labels to Music, Video, Toys, Books, VideoDVD, MusicalInstruments and DigitalVideoDownload. The results for the experiment are shown in Table 6 and Table 7.

Table 6. Classes metrics evaluation for testing after filtering classes not trained and classes with less training data.

Class	Precision	Recall	F1
Video	0.97	0.93	0.95
Books	0.78	0.89	0.83
Music	0.65	0.90	0.76
VideoDVD	0.94	0.57	0.71
Toys	0.91	0.44	0.59
DigitalVideoDownload	0.44	0.06	0.10
MusicalInstruments	0.00	0.00	0.00

The increase in the *Precision* metric for Books, Music, VideoDVD and Toys shown in Table 6 is due to the lesser amount of False Positives, when this classes are allocated to an individual because of the lack of the correct class in the training model. Even with a more controlled environment for testing, the model was not able to correctly classify any individual to the MusicalInstrument class.

The improvement in global *Accuracy* shown in Table 7 is due to the fact that the set of tested individuals is smaller and better mapped by the model. The change in the number of entries for testing can also justify the increase in *Logarithmic Loss* metric, since the failure in correctly classify an instance has a more severe impact than before.

Table 7. All-Class metrics evaluation for testing after filtering classes not trained and classes with less training data.

METRIC	VALUE
ACCURACY	0.73
LOGARITHMIC LOSS	0.27

7. Conclusions and Future Works

In this work Amazon's Sagemaker Framework was used to instantiate, train and deploy an API with a ready-to-use classifier. To train this classifier a data-set of product titles based on UK products reviews was processed, tokenized and used. The individuals-per-class ratio of this data-set was not optimal, as images showed, with classes being extremely unbalanced.

To test the efficiency of the classifier, a similar data-set, from US products reviews, was processed, tokenized and used through the API Gateway provided. The test data-set contained a few classes that were not available in the training set and, for that reason, two different evaluations were made. The first one included all data and classes into the metric and the second one filtered untrained classes and classes with less than 1000 individuals trained. As was to be expected, the classifier performance increased when working in a more intensively trained domain.

Empirical analysis made on the characteristics of product titles showed that, to each attributed class, titles had few to none symbols that could lead to a strong characterization of such class. To make the training process even less accurate, some set of tokens could easily be found in products of different classes (e.g. books that were adapted to movies and, later, adapted to music tracks). For the mentioned reasons, classifying products by title proved to be hard and not very accurate.

To give sequence to this work, a few more algorithms could be investigated, since all experiments were made using BlazingText's (Gupta & Khare, 2017) implementation of FastTex (Joulin et al., 2017). The usage of a bigger data-set for training with a more balanced class distribution is also a valid attempt that should increase the overall accuracy of all classes.

References

Gupta, S. and Khare, V. Blazingtext: Scaling and accelerating word2vec using multiple gpus. In *Proceedings of the Machine Learning on HPC Environments*, MLHPC'17, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450351379. doi: 10.1145/3146347.3146354. URL `https://doi.org/10.1145/3146347.3146354`.

Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pp. 427–431, Valencia, Spain, April 2017. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/E17-2068`.