Multiclass Classification: A Case Study on Videogame Reviews

Aleksander Westergaard Karlsen
Bachelor's in informatics / 2020
aleksawk@stud.ntnu.no

Abstract

This paper studies different approaches for non-binary text classification. This includes different techniques for feature extraction like Term Frequency-Inverse Document Frequency and other Bag-of-Words methods, as well as different classification algorithms. The goal of the study was to determine which text vectorization-classifier combination that could achieve the highest accuracy classifying videogame reviews. The research method applied was to try all combinations with various input sizes. It is found that Support Vector Machines and the Random Forest classifier outperform the Naïve Bayes classifier in this case, and that the classifiers perform far better given a smaller subset of the data when using feature extraction.

1 Introduction

Videogame reviews written by the users themselves often are a hot topic in various videogame communities. On the online marketplace Steam users are free to comment on the videogames they have bought. These reviews often get criticized for being off topic or sometimes just plain spam (ThangmusTheCalm, 2015). Text classification is the study of organizing text into categories (or classes). The driving motivation behind this paper was to see how effective the Steam reviews would be as data when trying to classify videogames based on text. The problem then being: given a text about an unspecified videogame, predict the videogame title. The goal was to achieve as high of an accuracy as possible for all the different combinations of text vectorization and classifiers (specified in section 2). The research methodology applied was set up an environment where one could test the different combinations with the same parameters, and enable easy parameter tweaking for each classifier separately. Other than providing the answers for the problem, the accuracy percentages, the results highlighted known strengths and weaknesses of the classifiers.
The paper starts with explaining the different text vectorization techniques and machine learning classifiers used in this paper and gives an overview of the machine learning pipeline's architecture. Later sections explore related work in the field of text classification, and the research method in detail. The paper concludes the research section with a post experiment discussion. The last section is the conclusion which presents the main contributions and discusses possible future work that can be done on this case study.

2 Background

This section introduces the text vectorization techniques and machine learning classifiers used in this study. The first subsection presents the topic of feature extraction, while section 2.2 present the different classification algorithms.

2.1 Feature Extraction

The machine learning algorithms take as input a document-term matrix consisting of numerical data, also called vectors (Bird, Klein & Loper, n.d.). Because of this, when trying to train an algorithm to recognize patterns in the data, we can't give it raw text. The task of converting text to numerical data

that machine learning algorithms can use is called feature extraction, or vectorization. The methods for feature extractions used is this study were Term Frequency and Term Frequency-Inverse Document Frequency.

2.1.1 Term Frequency

The feature extraction technique called Term Frequency, also called Token Frequency, is the task of converting documents into dictionaries or arrays where each index represents a term, and the value of said index is the frequency of that term in the document (Bengfort, Bilbro & Ojedo, 2018, p. 57-58). Take for example these two documents:

"The historical accuracy of the game is not the best. Accuracy is key."
"The game is amazing."

Converting the first one into a term-frequency matrix would result in this:

accuracy	amazing	best	game	historical	Is	key	not	of	The
2	0	1	1	1	1	1	1	1	3

Figure 1: A term-frequency matrix; each word is assigned a number equal to the occurrence of said word in the sentence

Without any pre-processing of the text, this way of extracting features provides bad performance because of the overwhelming amount of common words shared between all documents in the corpus (Bengfort, Bilbro & Ojedo, 2018, p. 59). This makes it hard for the classification algorithm to distinguish between different documents, resulting in worse accuracy. Removing the so-called stop-words from the documents would result in the term "accuracy" getting the highest frequency; teaching the classifier that the term is more significant than the others.

2.1.2 Term Frequency-Inverse Document Frequency

One of the disadvantages with Term Frequency is that it does not take the entirety of the corpus into account. In 1972 Karen Spärck Jones proposed a statistical interpretation of term specificity (Jones, 1972, p. 1). This would later be referred to as Inverse Document Frequency (Jones, 2004, p. 1). The purpose of IDF is to consider the other documents in the corpus, more specifically their terms, to find the words that provide the most information about the document.
TF-IDF consist of two parts: Term Frequency and Inverse Document Frequency. Term Frequency can be calculated as in section 2.1.1, but in this study an alternative TF called sublinear-TF was applied. Sublinear-TF prevents bias towards terms that appear often in a document (Bengfort, Bilbro & Ojedo, 2018, p. 62-63). The formula for Term Frequency is given by figure 2.

$$\mathrm{wf}_{t,d} = \begin{cases} 1 + \log \mathrm{tf}_{t,d} & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}.$$

Figure 2: Term Frequency logarithmically scaled to prevent bias towards high frequency terms.

The logarithmical scaling also follows for the calculation of IDF.

$$\mathrm{idf}_t = \log \frac{N}{\mathrm{df}_t}.$$

Figure 3: The IDF of a term is the product of N, the number of documents, divided by the number of documents in the corpus that contain said term.

Given figure 2 and 3 one can compute TF-IDF in its entirety. This formula is shown in figure 4.

$$\text{wf-idf}_{t,d} = \text{wf}_{t,d} \times \text{idf}_t.$$

Figure 4: The TF-IDF formula, constructed using figure 2 and 3.

2.2 Classification Algorithms

The problem presented in this paper is what is called a supervised machine learning problem (Bird, Klein & Loper, n.d.). This is because the data used is already labeled beforehand, and one can therefore use this label combined with various features, for example text, to train a machine learning algorithm to make predictions. The labels represent the different classes that our data consist of; hence the name text classification. In this case the labels (or classes) are the videogame titles we are trying to identify.

2.2.1 The Naïve Bayes Classifier

The Naïve Bayes classifier is based on Bayes' Theorem developed by Reverend Thomas Bayes (Scikit-learn, n.d.). It decides which label to give an input by calculating the prior probability, using the training data, and multiplying this with the features from the input. This results in the features decreasing the probability of a label being the correct one, by multiplying the chance of that feature occurring in a feature set with that label. This is illustrated in figure 5.

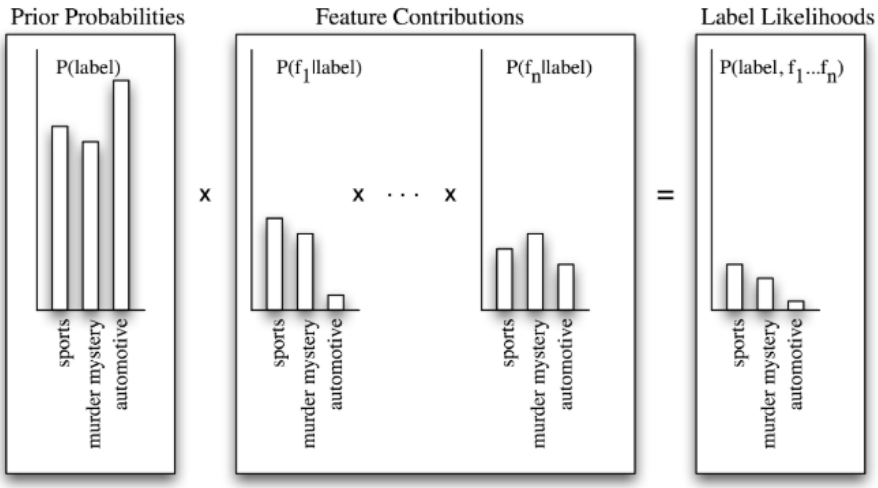

Figure 5: How the Naïve Bayes classifiers decides on a label for a given feature set (Bird, Klein & Loper, n.d.).

A weakness of the Naïve Bayes classifier is its approach to dependent features. The reason the classifier is not called "Bayes classifier" is because it assumes that all features are independent of one another. This is called the independence assumption (Bird, Klein & Loper, n.d.). Assuming that all features are independent is intuitively unreasonable in a real-world scenario, and this is something the user should be aware of when using the algorithm.

2.2.2 Support Vector Machine

The support vector machines classify data by finding a (n-1)-dimensional hyperplane which separates the datapoints in a n-dimensional feature space. A hyperplane is a subspace with n-1 fewer dimensions than the n-dimensional space it is in. It works out the maximum margin possible which separates the datapoint groups. It does this by calculating the distance from all the datapoints to the line and then chooses the smallest distance as the maximum margin. The point from which the margin is created is called a support vector, hence the name. When the margin is max, the generalization error is lower (Scikit-learn, n.d.). Figure 6 illustrates this separation.

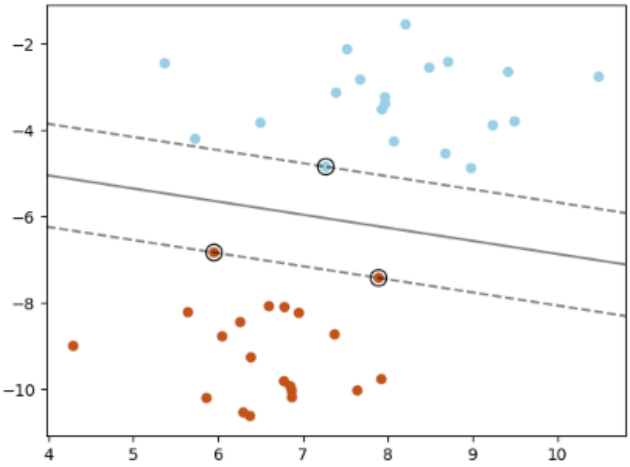

Figure 6: A 2-dimensional feature space with a 1-dimesnional hyperplane separating the classes. The circled dots are the support vectors. (Scikit-learn, n.d.)

In practice, most datasets are not linearly separable in the feature space. SVMs handles this by still calculating the maximum margin but allowing some points on the wrong side. This is called a soft margin (Chen, 2019).

2.2.3 The Random Forest Classifier

A random forest classifier is a collection of many decision tree classifiers (Koehrsen, 2018). A decision tree classifies feature sets by asking a series of true-false questions. These questions are formed during training, with the goal of identifying differences in features, that then divide the data into classes. This series of questions forms a tree structure where the leaf nodes are the classes, and the input will end up in one of these nodes when passed to the root.
The random forest classifier consists of two concepts: a forest and randomization. The forest is a collection of many decision trees that together classify data by making the average prediction, amongst all trees, the winner. The concept of randomization is incorporated because each tree gets a random subset of the training data, and each node a random subset of features to make a question from.

3 Related Work

An important factor for classification performance in this paper is the way feature extraction was used to convert text into vectors. There are other methods for working with text in a machine learning context and therefore a lot of related studies using these methods. One that has caught special interest is a study from 2015 by Masoumeh Zareapoor and Seeja K. R, comparing feature extraction to feature selection for a model to classify phishing emails. Compared to this paper, their study constructs a pipeline for testing both feature extraction and feature selection performance on a classifier, instead of different vectorization-classifier combinations. The main difference between feature extraction and -selection is that in feature selection a subset of the original feature space used as the feature, instead of converting the existing feature space into a new one (e.g. vectorization). This is illustrated in figure 7:

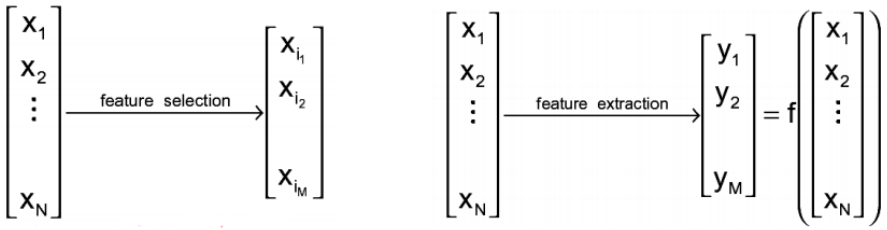

Figure 7: Feature selection compared to feature extraction (Schrater, 2009).

A disadvantage with feature selection is that it can be computationally expensive to find the optimal subset of features to use (Kenji, Rendell, 1992, p. 130). However, when using feature selection, you maintain data interpretability, compared to feature extraction (Hira, Zena & Gillies, 2015, p. 10). Regardless of the disadvantages, this approach is discussed more in section 6 as part of future work.

4 Architecture

Figure 8 is an illustration of the pipeline that the data from the reviews-dataset travels through. First the documents get extracted from the corpus. Then the documents are processed before getting fed to a feature extractor. The output from the feature extractor gets combined with the corresponding label and fed to the different classifiers.

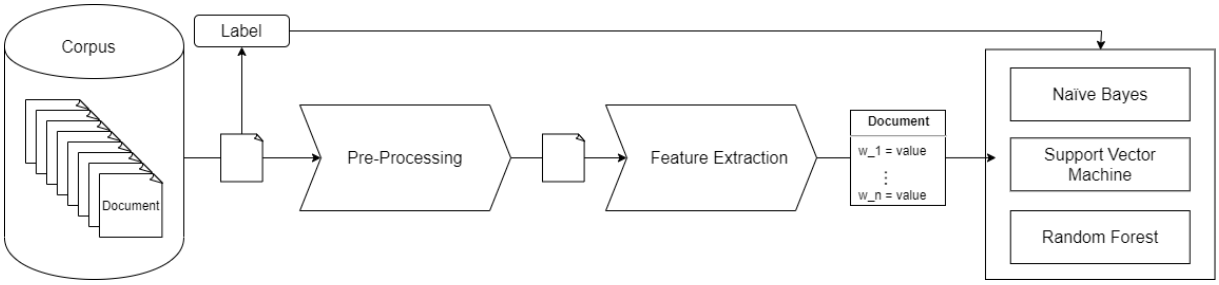

Figure 8: The complete pipeline used for training classifiers in this study.

5 Experiments and Results

Since the goal of the study was to achieve as high of an accuracy as possible for a model tasked with classifying game reviews, one had to try many possible approaches. TF-IDF and TF were chosen as feature extraction methods, and Naïve Bayes, SVM and RF as classifiers. For SVM and RF, Scikit-learn's implementation was used, and for Naïve Bayes, Nltk's implementation. The plan was to test all possible combinations of feature extraction and classifiers to find the one that performed the best, and to find which training set sizes that were optimal.

5.1 Experimental Setup

This section explains the steps taken to conduct the experiments. This includes data management, text pre-processing, feature extraction- and classifier parameters and usage.

5.1.1 Data Management

The dataset was downloaded from Kaggle. It contains about 450 thousand reviews about games on Steams "top selling games" list. Because of technical limitations it was decided to reduce the number of topics to 4. This was done using Excel spreadsheets, since the file is on the csv format. Excel was used to randomize the entries and sort out the reviews that did not belong to one of the four topics:

- PLAYERUNKNOWN'S BATTLEGROUNDS
- Rust
- Grand Theft Auto V

- Dead by Daylight

In Python, the data was partitioned into a 90/10 spilt. Four sets of two array pairs where on is the reviews and the other the corresponding labels.

5.1.2 Pre-processing

The pre-processing consisted of looking at each review and removing noisy elements from it. This included: stop words, symbols, numbers and tokens with a length equal to 1. Furthermore, the text was converted to lower case and stemmed. Stemming is the process of reducing a word to its root. This can be done using nltk's Snowball Stemmer. This stage of the process was also partially inspired by a paper by Guyon and Elisseeff from 2003 that produced a list of suggestions for doing variable and feature selection. In this list they suggest that if you have domain knowledge you should use this to make a set of ad hoc features. In this paper, the authors domain knowledge led to the construction of a short custom stop words list. This list included the words "game" and "games" since they would not be of any help in differentiating

5.1.3 Feature extraction- and classifier parameter/usage

When it comes to feature extraction using TF-IDF, two parameters were set: sublinear-TF and max-df. Given that user created reviews were the data, the chance of someone spamming the same word in a review was high; hence sublinear-TF was applied to combat this. Additionally, the max-df parameter was set to 0.5 to remove words that have a frequency higher than 50%.
Both the Support vector machine and the random forest classifier got some of their parameters set:

- For the random forest classifier, the default dept of 100 was set, and the square root function was used to make feature subsets. Bootstrap was set to "True" to give each tree a random sample of the training data.
- The SVM used the linear kernel, and the C-value was set to 1. The C value is the parameter that controls the margin and therefor the amount of misclassification allowed for the training data.

The Naïve Bayes classifier and the TF vectorizer did not get any parameters set.

Using the two feature extractors, three pairs of document-term matrices and labels were used as input for the classifiers. The first one being documents vectorized to TF-IDF values, the second one into TF, and the third one a combination of TF and TF-IDF. The reasoning for the third one was to see if focusing on both most frequent and most informative (or least frequent) terms would lead to any better results. This resulted in a 3-dimensional array which the classifiers did not accept. Therefore, dimensionality reduction using NumPy was applied to the third document-term matrix.

5.2 Experimental Results and Discussion

From figure 9, it is observed that SVM and RF achieve the highest accuracy of 87,5% with 100 training samples, while NB has significantly worse performance overall. NB's highest accuracy is achieved with TF with 450 training samples.

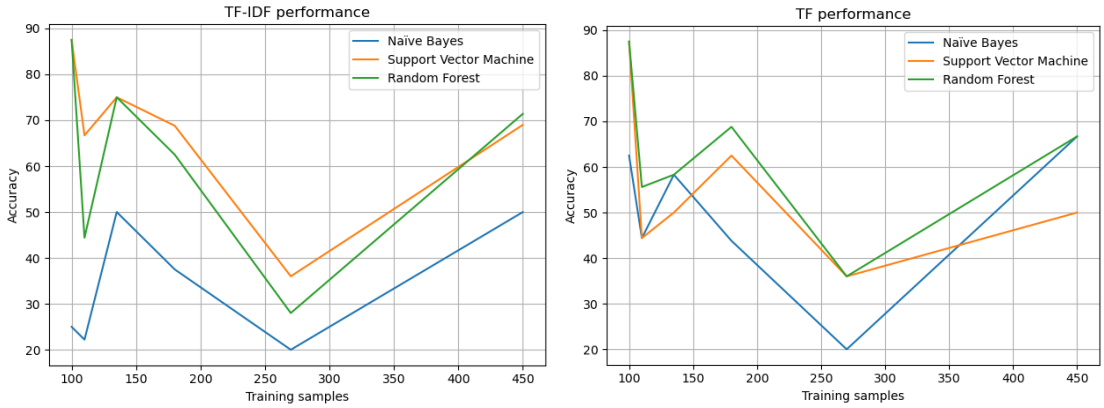

Figure 9: Feature extraction using TF-IDF, compared to using TF.

The most interesting aspects of these results is the fluctuations in accuracy when increasing the training sizes. This was not something that was expected when the experiments first started, but as more results came in it became clear that this was a trend across all the graphs. On the other hand, this trend is also found in the results given in the article by Masoumeh Zareapoor and Seeja K. R presented in section 3. They show that feature extraction gives good performance straight away with relatively few training samples, and then the accuracy decreases as the number of training samples increases. This might explain the fairly good results with few training samples. To get a more detailed picture of what is causing the accuracy, a classification report was generated.

	Precision	***Recall***	***F1***	***Support***
PUBG	**0,75**	1,00	0,86	3
Rust	**0,00**	0,00	0,00	1
GTA V	**1,00**	1,00	1,00	4
Dead by Daylight	**0,00**	0,00	0,00	0

Figure 10: The classification report for SVM at 100 training samples as seen in figure 9.

The classification report strengthens the accuracy metric because when support is 0, the precision and recall is also 0. When support is higher, the precision and recall is high.

Another interesting observation is that the graphs decrease and increase at the same time. To pinpoint the exact reason for this is difficult, but one might think that it could have something to do with the data fed at each point which “confuses” each model equally.

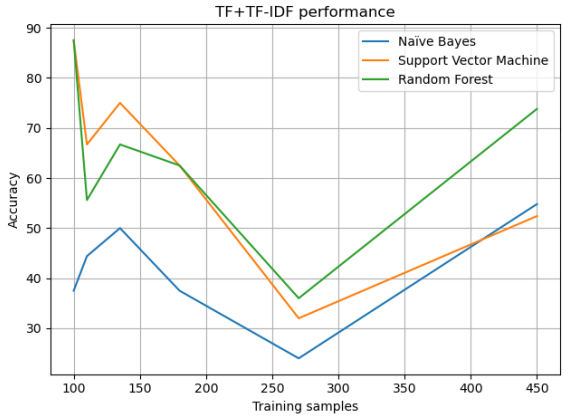

Figure 11: Performance on with both TF and TF-IDF as features.

The hypothesis that a combination of TF and TF-IDF would yield better accuracy seems to not be correct in this case. The results evident that applying both vectorization methods at the same time only led to

slightly worse performance overall, and in the case of the SVM, resulting in a much worse accuracy when the amount of training samples were high. This can be spotted when comparing the left graph in figure 9 to figure 11. One might think this is because there is simply to many features for the classifier to make a good generalization from.

6 Conclusion and Future Work

The goal of this study was to achieve as high of an accuracy as possible for all the different combinations of text vectorization and classifiers. Trough experimentation it was revealed that SVM and RF far out-performed NB in almost all cases. Other than achieving reasonably good performance for SVM and RF, the results strengthens other results about feature extraction done in studies like the one by Masoumeh Zareapoor and Seeja K. R in 2015. In hindsight it is clear that feature selection should have been explored as an opportunity as well. This would possibly yield better results based on the amount of training data available.

References

Bengfort, Benjamin. Bilbro, Rebecca. & Ojedo, Tony. (2018). *Applied Text Analysis with Python*. O'Reilly Media, Inc.

Bird, Steven. Klein, Ewan. & Loper, Edward. (n.d.). *Natural Language Processing with Python*. Retrieved from: http://www.nltk.org/book/

Chen, Lujing. (2019). *Support Vector Machine — Simply Explained*. Towardsdatascience.com. Retrieved 26.05.20 from: https://towardsdatascience.com/support-vector-machine-simply-explained-fee28eba5496

Guyon, Isabelle. Elisseeff, André. (2003). *An Introduction to Variable and Feature Selection*. Journal of Machine Learning Research 3. Retrieved 27.05.20 from: http://www.jmlr.org/papers/volume3/guyon03a/guyon03a.pdf

Kira, Kenji. A. Rendell, Larry. (1992). *The Feature Selection Problem: Traditional Methods and a New Algorithm*. University of Illinois. Retrieved 27.05.20 from: https://www.aaai.org/Papers/AAAI/1992/AAAI92-020.pdf

Koehrsen, Will. (2018). *An Implementation and Explanation of the Random Forest in Python*. Towardsdatascience.com. Retrieved 26.05.20 from: https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76

M. Hira, Zena. F. Gillies, Duncan. (2015). *A Review of Feature Selection and Feature Extraction Methods Applied on Microarray Data*. Department of Computing, Imperial College London. Retrieved 27.05.20 from: https://www.hindawi.com/journals/abi/2015/198363/

Scikit-learn. (n.d.). *1.4. Support Vector Machines*. Scikit-learn.org. Retrieved 26.05.20 from: https://scikit-learn.org/stable/modules/svm.html

Scikit-learn. (n.d.). *1.9. Naïve Bayes*. Scikit-learn.org. Retrieved 29.05.20 from: https://scikit-learn.org/stable/modules/naive_bayes.html

Scikit-learn. (n.d.). *sklearn.metrics.classification_report*. Scikit-learn.org. Retrieved 31.05.20 from: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

Spärck Jones, Karen. (1972). *A statistical interpretation of term specificity and its application in retrieval*. University of Cambridge. Obtained 25.05.20 from: http://danigayo.info/teaching/SIW/PDF/sparckjones1972-bis.pdf

Spärck Jones, Karen. (2004). *IDF term weighting and IR research lessons*. University of Cambridge. Obtained 25.05.20 from: https://www.cl.cam.ac.uk/archive/ksj21/ksjdigipapers/jdoc04.pdf

Schrater, Paul. (2009). *Feature Selection/Extraction.* University of Minnesota. Retrieved 27.05.20 from: http://vision.psych.umn.edu/users/schrater/schrater_lab/courses/PattRecog09/Lec17PattRec09.pdf

ThangmusTheCalm. (2015). *Is Steam's User Review system reliable?* [Online Forum]. Retrieved from: https://www.reddit.com/r/truegaming/comments/3de0o3/is_steams_user_review_system_reliable/