Machine Learning: Marvel vs DC

Lecturers:
Prof. Walter Daelemans
Nicolae Banari

Student:
Jeroen Van Sweeveldt
s0134162

1 Introduction

Text classification provides us a myriad of possibilities both useful and fun in terms of research, and one of these includes building a binary classifier to detect the difference between a superhero's creator, based on their profile text.

The data set for this task was retrieved on Kaggle[1]. It features 1450 entries of superheroes belonging to different creators, ranging from toy manufacturers as Lego, authors such as J.K. Rowling and George R.R. Martin, to television series (Heroes), and comics. Marvel and DC are the largest supplier of superheroes, with their shared entries totaling 1059 data points.

The feature that was used to predict the hero's creator was the "history_text" feature, a column containing their profile texts. A similar task has already been performed on this data set[2], using classical machine learning models exclusively. My task will differ in the sense that it will run only three classical algorithms, i.e. KNN, Support Vector Machines, and Naive Bayes, as well as an LSTM neural classifier (which unfortunately failed to run due to an unresolved error).

2 Workflow

Work on this data set started with getting a general overview: how many features did it contain? How many entries? Since only the text feature ("history_text") and the label ("creator") were relevant, a subset was made of these two columns. The second step was isolating the DC and Marvel creators from the others, bringing the total of 1450 entries down to 1059. I inspected

[1] https://www.kaggle.com/datasets/jonathanbesomi/superheroes-nlp-dataset/code
[2] https://www.kaggle.com/code/raislervoigt/marvel-or-dc-creators

whether either of these columns contained empty values, since these would prove to be disruptive to the experiments' results. While the label column did not have any empty values, a portion of the history_text feature did. A way to deal with this is to run a KNN Imputer to calculate a mean based for the empty cell on its neighboring cells. However, while this is useful with numeric data, we are here dealing with string data. Even though this data would eventually be vectorized, padding these cells with the average values of their neighbors would, I reason, result in the algorithms picking up gibberish that might end up disrupting the results.

For this reason, I calculated how many data points would be lost if these empty cells were to be removed: this would result in a 6.03% data loss, bringing the amount of data points down from 1059 to 997. The empty cells were removed.

Total data points	**Relevant data points (Marvel & DC)**	**Empty data points**	**Net data points**
1450	1059	62	997

I then ran a plot to check the balance of the data set. The data set was fairly balanced, with a difference of less than 20% in the number of labels.

Marvel labels	**DC labels**
583 (58.48%)	414 (41.52%)

I did not run any preprocessing step that would truncate (or pad) either of the label values. To account for the imbalance, I used the stratify argument during the train/test-split.

A word count was done to decide whether on the Vectorizer that would be used on the data. The top 30 of words in the history_text's vocabulary were stop words. Although the sklearn's vectorizers provide a stop words argument, there are preferred ways over this to perform such actions in the realm of NLP (e.g. through Spacy). I used the tf-idf vectorizer to weigh down the importance of these recurring words instead. For the neural classifier, I ended up using the count vectorizer, as tf-idf makes us lose information about word order, which is important to RNN networks. Finally, the labels were encoded uses sci-kit learn's label encoder.

The data was then split in three parts to accommodate the neural classifier, which uses a development or validation set as an alternative to cross-validation. I did not risk doing two splits, one with a train-test split and another with a train-dev-test split, as I felt it would result in unreliable comparisons between the neural and classical algorithms.

Originally I opted for a 60/20/20 split, however, this resulted in different shapes for the dev set and test sets (199 and 200), which proved adversarial for the neural classifier. Rather, the initial split of the train and test sets was 0.33%, with a further equal split into the dev and test sets, resulting in equal shapes of 667 (train), 165(dev), 165(test).

I ran experiments with KNN, Support Vector Machines and Naive Bayes for the classical models. I first defined two baseline models to compare their accuracy against. The results are as follow:

	Dummy (most frequent)	**Dummy (stratified)**	**KNN**	**SVM**	**Naive Bayes**
f1 (macro avg)	0.37	0.52	0.83	0.83	0.87
Precision	0.29	0.52	0.84	0.83	0.87
Recall	0.35	0.52	0.82	0.84	0.87

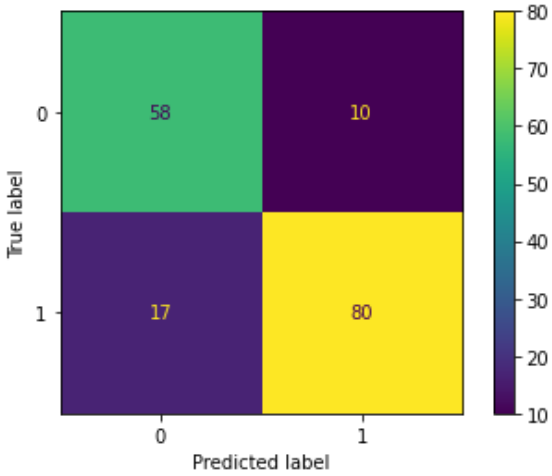
I unfortunately did not succeed in successfully building an LSTM network, producing a *RuntimeError: Length of all samples has to be greater than 0, but found an element in 'lengths' that is <= 0* traceback that I did not manage to resolve. I ran a simple, flat model to test whether the preprocessing might have had something to do with it. The model ran, but it did not produce any satisfactory results as it overfitted.

3 Error analysis

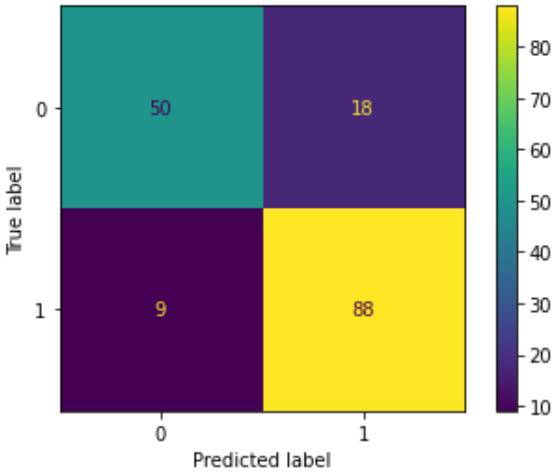
In the error analysis, I built two separate data frames to compare their respective labels and see which texts had been misclassified. I calculated the average of the number of tokens in each cell, and this revealed that erroneously classified data points had on average a lower amount of tokens.

	KNN	**SVM**	**Naive Bayes**
Avg. length correct	650.42	662.86	662.86
Avg. length error	329.52	265.96	165.35

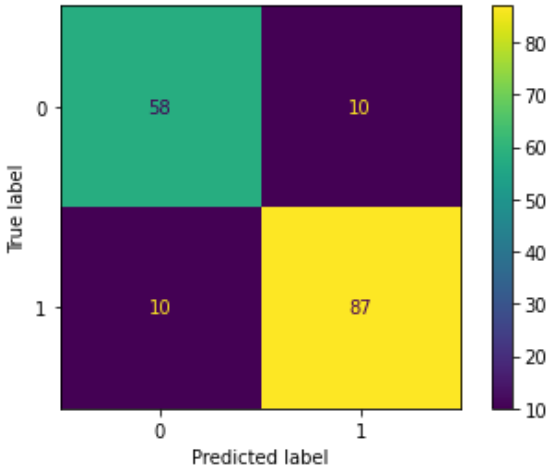
Below are the classifier's confusion matrices, where 0 stands for DC and 1 stands for Marvel Comics.



Support Vector Machines confusion Matrix



KNN confusion matrix



Naive Bayes confusion Matrix

More robust preprocessing, such as removing stop words, could help improve future experiments, I believe, as stop words might disrupt accurate analysis and prediction of the shorter texts. Finally, while all of the models performed similarly, Naive Bayes stood out with both higher precision and recall. Error analysis showed that this classifier seems more capable of dealing with shorter texts than KNN and Support Vector Machines.