Winner prediction for Dutch news headlines

Arno Deceuninck

September 7, 2022

Contents

1 Introduction

1.1 Previous research

Soubry [7] analysed a set of over 2000 potential Dutch news headlines for over 900 articles. Each of those articles has at least two candidate headlines of which only one got labeled as "winner" using AB tests. In her research, she manually analyzed different charecteristics of all those potential headlines - activity, length, questions, punctuation, duality, emotions, forward reference, signal words, article words, adjectives, proper names, directly adressing the reader, first and last names, numbers and quotations. [7]

1.2 This research

This project tries to automatically predict those winners given a set of candidate news headlines, which might be useful for recommending journalists which of their candidate headlines has the highest probability for getting clicked on. To do this, I first looked at Natural Language Processing techniques (NLP) to find the manually labeled characteristics automatically, looked at potential extra features and tried different (classification) methods to find the headline that would generate the most clicks.

1.3 Code

The code used for this is split along different Jupyter Notebooks and can be found on Github. You can often look at the extra comments in the code for more details about some parts.

2 Label classification

The first step is automatically detecting the characteristics Soubry [7] labeled. To test the different models, a train-test split was generated by first sampling one headline per article [1] and then taking 25% of this as test set and 75% as training set.

2.1 Concepts

Some NLP concepts are the same for a lot of those classification tasks, so I'll explain them first.

2.1.1 Preprocessing

The text is often preprocessed e.g. by removing special characters, removing stopwords and/or making everything lowercase.

[1] I started by generating the test set using all candidate headlines (without dropping the ones that weren't sampled), since the labels should be independent of the other headlines in the test set. However, headlines in the same test set often look very similar and thus share the same values for a lot of the features, which lead to a train and test set that weren't independent from each other (and thus also the impression of giving a better performance, even though this wasn't the case).

2.1.2 Tokenizer

Once we have our preprocessed text, we can tokenize it. A token is a basic unit that won't get decomposed any further. You can e.g. take words or characters as token or combine multiple consequent words or characters into compound tokens (n-grams). [9]

Stemming Stemming maps words containing the same stem (e.g. "unicorn" and "unicorns", "creation" and "create") onto the same token (e.g. "unicorn-", "creat-"). [5]

2.1.3 Vectorizer

The vectorizer converts the text to a meaningful numerical representation that enables the machines to understand the textual contents. [6] This numerical vector representation can then be used on e.g. already known classification vectors that take a numerical vector as input.

Bag of Words Bag of Words (also called CountVectorizer) genertes a vector containing the number of occurances in the input text of the word given position in the vector represents. [5]

TF-IDF The Term Frequency-Inverse Document Frequency (TF-IDF) is similar to Bag of Words, but also the number of occurances of given words in the entire document collection into account (so often occuring words like "the" become less important). [5]

2.1.4 Extra features

Sometimes some extra features where added to the input vector for the classifier (together with the output of the vectorizer). Often those extra features were based on NLP libraries like SpaCy and NLTK [2], which extract features of the text like part-of-speech tags, named entities or dependency parse trees. [8]

2.1.5 Classifier

The classifiers take a vector of numbers as input and outputs a class label. I mainly used a SVM classifier here, but other options might give better results[3].

SVM classifier Each of the n-dimensional input vectors can be seen as a point in an n-dimensional space. Support vector machines (SVM) try to find the best soft margin (based on validation set) which acts as a seperator line/hyperplane between the different classes. They transfer the data points to a higher dimensional space, where it might be easier to find a seperator line. The kernel function is used to transform the data points to a higher dimensional space. In my implementation, only a linear kernel is used[4]. [3]

2.1.6 Simple functions

Sometimes just using a simple functions (e.g. whether the headline contains a question mark) gave better results than a more complex method. However, we have to be carefull for overfitting with this.

2.1.7 Metrics

An evaluation metric tries to measure the quality of a classification model. Since we're working with binary classifications (positive or negative), we have four possible classification outputs. If the data is correctly classified, we have a True Positive (TP) or True Negative (TN). If the prediction is wrong, we have a False Negative (when it should be positive, but got classified as negative, FP) or False Positive

[2] Mainly on SpaCy, since the performance of NLTK on Dutch headlines was quite bad.

[3] I used the SVM classifier because it occured in one of the first text classification tutorials I found. However, you can keep endlessly searching for better classification methods and since this was only a small part of the goal for this project, I didn't test other classifiers.

[4] Other kernels might give better results, but you have to see how far you want to keep searching, since the search for the optimal classifier might be almost endless.

(when it should be negative, but got classified as positive). Those four values can be found in the confusion matrix. [1]

Accuracy The accuracy is the proportion of correct classification among the total number of classified items. [1]

Data imbalance If the data is imbalanced, accuracy will give missleading results. E.g. if 99% of our labeled items is negative, we can achieve an accuracy of 99% by classifying everything as negative. The f1-score is an evalution metric that solves this data imbalance problem.

We'll mainly focus on the f1-score when evaluating the characteristics detection.

2.1.8 Data imbalance

Sometimes, the data might be imbalanced, meaning that we have a lot more samples from one of the classes than from the other. You can try to solve this by adding extra samples from the minority class (oversampling), removing samples from the majority class (undersampling) or a combination of both.

2.2 Characteristics

We'll now take a look at the different labels in the dataset from Soubry [7] and try to predict those labels automatically.

2.2.1 Activity

This label represents whether the sentence was writen in active form. If this is false, the sentence is written in passive form. You can find the results in table 1.

Custom functions I initially tried to detect passive sentences using a simple function that tries to find past particles (only checks for words starting with "ge-" (e.g. "gevochten")), the word "door" or forms of "zijn" or "worden". Here is the earlier mentioned danger of overfitting applicable. Since a lot of words starting with "ge-" aren't a past particle, I also tried only checking on the word "door" or forms of "zijn" or "worden", which worked a lot better.

For the next approach, I used SpaCy to detect whether any of the words in the headlines is a past particle. If that was the case, it got classified as a passive sentence. This also worked slightly better than my own functions.

TF-IDF words Applying TF-IDF using words as tokens gave the best results. I also did an analysis of the most important words in this model. With this, I expected a lot of past particles as words on the negative (passive) side, but that wasn't the case as much as I expected. The top 5 most important words for classifying something as passive are "opgepakt", "twee", "dj", "omdat" and "geschiedenis" accoording to this model. "Opgepakt" is the only past particle of those words.
The top 5 most important words for classying something as active are "geeft", "onze", "vlaanderen", "politie" and "auto". "Geeft" is the only one I would have expected here, since it's hard to put that word in a passive sentence.

TF-IDF chars Another approach was by using character grams of length one to four. I thougth this would generalize better to detect past particles by looking at char grams (taking white spaces into account) like " ge", "d ", " ver" or "door". This performed similar to TF-IDF for words (slightly worse). The top 5 char grams for classifying something as pasive were " ak", " era", "ng ", "erd" and " d". Some of those indeed occur often in a past particle (like "erd"), but I don't see why e.g. " ak" has the largest impact.

Method	F1-score	Accuracy
Simple function (ge-, door, zijn/worden)	58.42%	46.93% (107/228)
Simple function (door, zijn/worden)	77.14%	64.91% (148/228)
Past particle SpaCy	79.42%	68.86% (157/228)
TF-IDF (words)	91.04%	83.70% (190/227)
TF-IDF (chars (1-4))	90.29%	82.38% (187/227)
Combination (SpaCy + TF-IDF chars)	90.51%	82.82% (188/227)
Combination (SpaCy + TF-IDF words)	90.78%	83.26% (189/227)

Table 1: Detecting label "Actief"

Method	F1-score	Accuracy
Simple function (length $\geq$ 76)	98.34%	97.81% (223/228)
Logistic regression (length as feature)	97.95%	97.37% (222/228)

Table 2: Detecting label "Lang"

Combination I also tried combining the TF-IDF methods with an extra feature whether the headline contains a past particle accoording to SpaCy. I wouldn't expect this to improve for the word based TF-IDF, since this can already detect past particles using the entire words, but the character based TF-IDF couldn't do this, so I expected a better performance there. There was indeed a slight increase in performance for the character based TF-IDF (but nothing significant), but strangely also a small decrease in the word based TF-IDF (but also nothing significant).

2.2.2 Length

Longer headlines can give more information to the reader. Soubry defined as long if it contains 76 or more characters. [7] You can find the results in table 2.

Simple function The definition is clear, so I could just write a function for finding all the exact matches, of which you would expect a perfect accuracy. However, there were still a few misclassifications, which were wrongly labeled in the dataset of Soubry.

Length as feature Since we didn't get a perfect match with the previous method, I also tried giving the length as only feature to a logistic regression classifier (so it could find it's own optimal border length). I would've expected slightly better results with this, but it was actually slightly worse than the previous function, probably because it is overfitting slightly on the training set.

2.2.3 Questions

Question should generate more clicks, since it makes people curious about the answer. [7] You can find the results in table 3.

Simple function A questions usually comes with a question mark, so just checking for this might already give good results. This still (relatively) gave quite some false positives. When checking what the false positives in the training set were, it gave mainly retorical questions, questions in quotations or question that already contain the answer in the headline itself.

2.2.4 Punctuation

Sourby mentions three kinds of punctuation: exclamation marks (!), ellipses (...) and quotation marks. You can find the results in table 4.

Method	F1-score	Accuracy
Simple function (contains question mark)	78.95%	96.48% (219/227)

Table 3: Detecting label "Vragen"

Method	F1-score	Accuracy
Simple function (contains !, ... or ")	57.14%	92.07% (209/227)

Table 4: Detecting label "Interpunctie"

Method	F1-score	Accuracy
Simple function (contains "." or ":")	90.52%	90.31% (205/227)

Table 5: Detecting label "Tweeledigheid"

Simple function Since this contains a clear definition, it can be easily checked using a function that checks for any of the three mentioned punctuation methods. I expected a perfect result with this, but that was not the case. It was also unclear why some classifications were wrong. My guess is that this is because of misclassifications in the original dataset or that there's an extra condition to be labeled for punctuation which I don't see.

2.2.5 Duality

Some headlines consists of two parts, like a headline as first part and a quote as second part. Those two parts are often seperated by a double points or a point. [7] You can find the results in table 5.

Simple function This also contains a clear definition, so I'll just check whether the headline contains a "." or a ":".

2.2.6 Emotions

Some headlines contain emotions to trigger te reader to find out more about the entire story. [7] You can find the results in table 6.

EmoRoBERTa EmoRoBERTa is a model for emotion detection on English sentences. It doesn't just detect wheter a sentence contains an emotion, but also assigns the detected emotions with their corresponding probabilities to the sentences. However, this won't work on Dutch sentences of course. I tried to translate the sentences to English and then feed them to EmoRoBERTa. This was able to correctly classify the emotions itself, but less good at predicting our binary emotion label (since Soubry e.g. didn't see "curiosity" as an emotion, but adding those more neutral emotions also didn't fix this). Translating the sentences is also a bit cheating, so we won't use it anymore.

TF-IDF TF-IDF might also work good, since it just needs to learn the word that have an emotional connotation. However, this predicted everything as negative. Adding stemming slightly improved this, but we still have a larger problem of data imbalance.

Data imbalance In our trining dataset, only 69 of the 681 headlines contain an emotion, which makes the classes of containing or not containing an emotion highly unbalanced. I tried some different approaches like reducing the number of features, different methods for oversampling, different methods for undersampling and the combination oversampling and undersampling. However, the f-score stayed quite low.

BERTje BERTje is the Dutch version of BERT, which is a transformer based machine learning technique. [2] I tried finetuning this on our binary emotion classification task. It took quite some time to train to result in slightly worse results than just using TF-IDF and it also had a lower f-score.

2.2.7 Forward Reference

Forward references keep a part of the article unknown for the reader, until they click it open. It's often in the form of words like "this", "he"/"she", "why", "who" ... . Also not further specified headlines get seen as forward references. [7] I tried to search some papers about detecting clickbait (since this also

Method	F1-score	Accuracy
EmoRoBERTa	35.29%	75.77% (172/227)
TF-IDF	00.00%	87.28% (199/228)
TF-IDF (stem)	33.33%	89.47% (204/228)
TF-IDF (stem + low max features)	00.00%	87.28% (199/228)
TF-IDF (stem + high max features)	33.33%	89.47% (204/228)
TF-IDF (stem + oversampling (random))	35.90%	89.04% (203/228)
TF-IDF (stem + oversampling (SMOTE))	35.90%	89.04% (203/228)
TF-IDF (stem + oversampling (Borderline SMOTE))	35.90%	89.04% (203/228)
TF-IDF (stem + oversampling (ADASYN))	35.90%	89.04% (203/228)
TF-IDF (stem + undersampling (random))	41.41%	71.49% (163/228)
TF-IDF (stem + undersampling (NearMiss v2))	37.50%	64.91% (148/228)
TF-IDF (stem + undersampling (Condensed Nearest Neighbors))	40.91%	88.60% (202/228)
TF-IDF (stem + undersampling (TomekLinks))	37.84%	89.91% (205/228)
TF-IDF (stem + resampling (SMOTEENN))	22.57%	12.72% (29/228)
TF-IDF (stem + resampling (SMOTETomek))	35.90%	89.04% (203/228)
BERTje	27.03%	88.16% (201/228)

Table 6: Detecting label "Emotie"

Method	F1-score	Accuracy
TF-IDF (SMOTETomek + stem)	47.73%	79.47% (181/227)

Table 7: Detecting label "Voorwaartse Verwijzing"

often contains forward references), but without much succes. You can find the result of the TF-IDF approach in tble 7

TF-IDF TF-IDF (with SMOTTomek as resampling method and stemming) still gave quite a bad f-score. I would've expected better performance with this, since there are often the same forward referencing words. The most important stems where indeed "dez-", "dit", "zo", "waarom" ... which is what you would expect in forward references.

2.2.8 Signal words

Some headlines contain signal words like "LIVE" or "SPORT". [7] You can find the results in table 8.
'

Simple function Those signal words are often full caps, so I started by classifying all headlines containing a full caps word as positive.

TF-IDF Since the signal words are often the same occuring words, I would expect good performance here. The accuracy is indeed higher than the previous method, but the f-score is lower.

Extra features Along TF-IDF, I also added three extra features: whether there is a dot after the first word, whether the first word is full caps and whether it contains any full caps word. This resulted in a better accuracy and f-score.

Method	F1-score	Accuracy
Simple Function (any full caps)	53.85%	94.74% (216/228)
TF-IDF	40.00%	97.37% (222/228)
Extra Features (TF-IDF, dot, first full caps, any full caps)	85.71%	99.12% (226/228)

Table 8: Detecting label "Signaalwoorden"

Method	F1-score	Accuracy
Simple Function (articles)	58.99%	75.00% (171/228)
Simple Function (articles, no quotes)	66.13%	81.58% (186/228)
TF-IDF	45.07%	82.89% (189/228)
TF-IDF (1-2 words)	49.38%	82.02% (187/228)
TF-IDF (1-3 words)	45.78%	80.26% (183/228)
TF-IDF (no quotes, 1-2 words)	62.34%	87.28% (199/228)

Table 9: Detecting label "Lidwoorden"

Method	F1-score	Accuracy
Bag of Words (SMOTETomek)	37.70%	66.67% (152/228)
TF-IDF (SMOTETomek)	17.65%	75.44% (172/228)
TF-IDF (SMOTETomek + stem)	18.92%	73.68% (168/228)
SpaCy (contains adjective)	44.54%	44.30% (101/228)

Table 10: Detecting label "Adjectieven"

2.2.9 Article words

Articles are often omitted from news headlines. In this analyses, they only looked at articles that can be omitted and not articls in fixed expressions (e.g. "uit de biecht klappen", "Bart De Pauw"). [7] Just detecting articles is easy, the hard part is detecting whether an article could be omitted. You can find the results in table 9.

Simple function Just detecting whether there is some article ("de", "het", "een") in the headline might already give quite some good results, though they were still lower than I initially would've expected. I also tried to do the same, but after removing text between quotation marks (since that counts as fixed expressions). That gave as expected an increase in accuracy and f-score.

TF-IDF Since it just needs to detect articles, I would expect TF-IDF to give a similar performance as my first simple function. The accuracy was slightly better but the f-score was slightly worse. The articles still got seen as the most important words in this model. I also tried TF-IDF on multiple words, since this might be better to detect fixed expression. However, there weren't any expressions in the most important word pairs. I also tried here first removing the quotations from the headline and then applying TF-IDF.

2.2.10 Adjectives

Adjectives give extra information and are often used to add extra emotions. The hard part here is that not all adjectives count, only important ones that have a significant meaning and difference from the other candidate headlines. [7] You can find the results in table 10.

TF-IDF We just need to know what adjectives are, so using Bag of Words or TF-IDF might already give quite good results. The results where lower than expected, certainly the f-score.

Simple function SpaCy can also recognize wheter a given word in a sentence is an adjective, so we can just check this for any word.

2.2.11 Proper names

Proper names might give the reader a better connection to the article. [7] You can find the results in 11.

Method	F1-score	Accuracy
TF-IDF (SMOTETomek)	29.41%	89.47% (204/228)
SpaCy (contains proper noun)	25.43%	43.42% (99/228)

Table 11: Detecting label "Eigennamen"

Method	F1-score	Accuracy
TF-IDF (SMOTETomek)	29.41%	89.47% (204/228)
SpaCy (contains proper noun)	25.43%	43.42% (99/228)

Table 12: Detecting label "Betrekking"

TF-IDF Since the same proper names occur often in the news, TF-IDF might do quite a good job, but also here is the f-score lower than expected. Here is also the danger of overfitting on names that were relevant at the time the data got collected and new names won't be detected. The most important fatures are "Kevin", "Temptation" (Island) and "Blind" (Getrouwd), but "zijn" and "tegenslag" make less sense.

Simple function Spacy can appearantly also detect proper nouns. However, this results in a very low accuracy. Some of them might be labeld wrongly in the dataset (e.g. "Gert" or "Verhofstadt"). SpaCy also marks words like "Manneke Pis" as proper nouns, but they aren't in the dataset.

2.2.12 Adressing the reader

Some headlines directly adress the reader, like "How can I sleep in this warm weather?" or "How to get rid of pimples?". [7]. You can find the results in table 12.

TF-IDF TF-IDF might recognize some words that get often used in those sentences. The most important positive words are as expected "je", "waarom" and "bent". Adding stemming didn't make a difference here.

2.2.13 First and Last name

This label checks whether both the first and last name is used instead of only one of the two. [7] You can find the results in table 13.

TF-IDF This learns the names of the people that were in the news when the data was collected. When using only one word tokens, most names have a positive impact, except for Trump, what makes sense, since there is often news about him, so most people already know him, so they almost never say Donald Trumps. Only allowing bi- or trigrams makes more sense to detect the entire name, however this strangely leads to a worse performance.

Simple Function SpaCy can detect proper nouns. To find a first and last name combination, we can simply check for two consecutive proper nouns. This resulted in a better f-score, but a lower accuracy. Things like "Mnneke Pis", "Club Brugge" and "Stille Ocean" remain false positive, but they might be easily filtered out by still adding TF-IDF as extra feature. Adding this, makes it indeed better, but strangely it is still better with unigrams than with bigrams.

Method	F1-score	Accuracy
TF-IDF	19.05%	92.54% (211/228)
TF-IDF (2-3 words)	9.52%	91.67% (209/228)
SpaCy (consecutive proper nouns)	52.46%	87.28% (199/228)
Combination (TF-IDF (1 word) + SpaCy (consecutive proper nouns))	66.67%	93.86% (214/228)
Combination (TF-IDF (2 words) + SpaCy (consecutive proper nouns))	57.14%	90.79% (207/228)

Table 13: Detecting label "Voor+Achternaam"

Method	F1-score	Accuracy
TF-IDF	48.57%	84.21% (192/228)
TF-IDF (1-6 chars)	52.17%	85.53% (195/228)
Simple Function (digit or number text in headline)	80.00%	90.35% (206/228)
SpaCy (contains number)	77.78%	89.47% (204/228)

Table 14: Detecting label "Cijfers"

Method	F1-score	Accuracy
Contains quotation	96.45%	97.81% (223/228)
TF-IDF + Contains quotation	96.45%	97.81% (223/228)
TF-IDF + Contains quotation + #quotation words	96.45%	97.81% (223/228)
TF-IDF + Contains quotation + #quotation words + Double point	97.14%	98.25% (224/228)

Table 15: Detecting label "Quotes"

2.2.14 Numbers

Numbers make the content of the article predictable. It can make the headlines more informative or add more emotion. [7] You can find the results in table 14.

TF-IDF If there are enough numbers in the training set, TF-IDF might already give quite good results with words. I would've expected different numbers as most important words, but that's not really the case. The top 5 is "jaar", "000", "euro", "graden" and "24". Things like "jaar", "euro" and "graden" all make sense and is a more out of the box approach. Since most numbers consist of the same characters (0-9) or contain the same textual parts (e.g. "zes", "-tig" in Dutch), working with char-grams might also give quite good results. The most important char grams were as expected mainly digits, but also brackets (since e.g. the age of people is often put inside brackets).

Simple function I also tried an own function that checks whether the headline contains a digit or contains a word with one of a small list of numbers as part of it (e.g. "drieëndertig" contains both "drie" and "dertig"). Here we have to be carefull for overfitting.

SpaCy SpaCy can appearantly also recognize functions, which is slightly worse than the previous aproach, but probably less vulnerable to overfitting.

2.2.15 Quotations

Some news headline contain a quote from someone. You can find the results in table 15.

Simple function Quotes are almost always between quotation marks, so we can use a functionto that checks whether there is a quote present in the headline. This gave as expected quite a good performance. Some of the misclassifications were just a single word between quotation marks.

TF-IDF Quotes are often combined with e.g. forms of "to speak", so I added the TF-IDF vector as extra feature along with our previous function, but that didn't make any difference.

Number of words Since most misclassifications (in our training set) were single words between quotes, so I also added the number of words between quotes as an extra feature, but that also didn't make any difference. Adding whether there is a double point in the headline as extra feature make a slight difference (but only one extra correct classification).

3 Extra features

All previously mentioned features won't be enough to classify everything correctly. Take for example test case 16. This contains 8 different candidate headlines, of which 6 (including the winner headline)

contain exactly the same feature vector. So if we only need to choose a winner based on those feature vectors, we have to randomly sample one of those, since they would look exactly the same to our model. Let's add extra features to solve this.

3.1 Length

Up to now, the feature length was binary and was true (1) once the headline contained more characters than a certain treshold value. Adding the number of characters itself as an extra features already makes each feature vector more unique.

3.2 Sentence embedding

Using BERTje, we could generate a vector representing the meaning of the different headlines. This vector could also be used as extra feature.

3.3 Word differences

Up to now, I've focused on each of the headlines independently. However, in order to become the winner, you must be the best headline in the test set, so adding extra features based on how the given headlines is regarding the other headlines in the same test set will probably give better results. For this I looked at the words for each headline that weren't part of any of the other headlines in the same candidate headlines set. Based on those words, I added some extra features.

Number of words The number of those words might give an idea how different the sentence is from the other sentences in the test set.

Average word length The average length of those words might give an idea of how important those words are (since a lot of short words often don't have much added meaning).

Max word length The length of the longest of those words, since this might give an indication of how important the words are.

Difference embedding A sentence embedding of the words that were different, which might give some extra meaning of them to the classification model. Instead of seeing those words as one embedding, I also tried to create an embedding of each of the words and pooled those different vectors (e.g. using average pooling), but this didn't make a difference.

3.4 Results

I tried those different parameters on both an Naïve Bayes or XGBoost winner predictor. Adding length didn't make much difference, which was strange since the binary length label seemed one of the more important labels (5th place) with a correlation test. Sentence embeddings made the results worse for both of our models[5]. The different word differences techniques also didn't seem to make a lot of difference. Again here the embedding made the models worse. A combination of all those extra parameters (except for the embeddings) seemed to make the models slightly better, but this wasn't the case anymore when I tested it for multiple runs[6].

4 Winner prediction

In the previous section, we mainly focussed on trying to automatically predict the different features. Now we want to predict the winner based on those features. As train-test split, the headlines where seperated such that all headlines of a specific test were either in the train set or all in the test set. You can find a summary of all models in table 16.

[5] Even though XGBoost started having embeddings starting from the 6th place of most important features

[6] Probably because the model for those runs at the end were finetuned using only the labels that were mentioned by Soubry

4.1 Goal

In our use case, we want to predict for a given news headline whether it's the winner or not, so instead of actually predicting a class, we'll predict the probability the class winner would get assigned and give the candidate headline with the highest such probability the winner label for that test set.

The methods I tried didn't take the other headlines in the same test set into account, but tried to independently[7] predict for each of the headlines whether it's a winner or not (and take the one with the highest probability as actual winner). [8]

The results of a baseline random classifier varied a lot, with a mean of 41% (lowest value in 100 runs was 28%, highest value was 60%).

4.1.1 Accuracies

The accuracies are based on whether the correct candidate headline got selected[9]. There was a lot of variance in the accuracies per model and I didn't always take the average of multiple runs at the start, so not all results of something getting better where correct. In the overview at the end, I took the best tuned models and tested them multiple times to get a better idea of the actual accuracy and standard variation.

4.2 Hyperparameter tuning

I've used some different approaches for finding the best parameters for the model. A grid search tests all possible combinations of the parameters I suggested. Random search randomly selects some, but doesn't necessarily check all possible combinations. Besides the random search of sklearn, I've also used the hyperopt library, which does distributed asynchronous hyperprameter optimization and uses some extra features to select the parameters based on the previous results that gave good scores in the objective function, so not completely random.

During the hyperparameter tuning, I initially used the accuracy on the winner column (with true/false as labels assigned by the classifier (but where the number of true values for a specific test wasn't always exactly one), but changed it later to the accuracy of the headline id's of the winning headlines (by taking the one with the highest probability for winner to be true as actual winner).

4.3 Correlation

In order to find a model that predicts the winner, there must be a correltion between the feature you get as input and the headline that got selected as winner. Using a pearson test on the input features, I noticed that there was a clear correlation between quotes and duality, which makes sense since headlines with quotes often contain the actual quote as one part of the headline and some other text as another part, what makes duality true. Using a chi squared test, I could find the most important labels for predicting the winner, which were forward reference, activity, adjectives and length.

4.4 Naïve Bayes

Naïve bayes searches the class that maximizes the probability of finding the class given the input features, while making the naïve assumption that the input features are independent of each other. It looks for each of the features at the probability of that feature event occuring, the probability that features occurs in a winner headline and the probability that feature occurs in a non-winner headline. If those probabilities where similar, the feature doesn't have much effect on determing whether the headline is the winner or not accoording to this model. Some important features where activity, emotion and adjectives.

[7] Except for some features

[8] An idea for future research might be to add the features of another candidate headline to the feature vector, so a classification algorithm like a random forest can compare the headlines more.

[9] This means the accuracies are counted as one per correctly predicted test and not one per correctly predicted winner label (which is either true or false and would result in a higher accuracy)

4.5 Random Forest Classifier

A decision tree predicts a class by looking at conditions for features in the input vector (e.g. length $\geq$ 42) in multiple splits. This is however highly sensitive to the training dataa and might fail to generalize. A random forest solves this by letting multiple decisions tree vote for the class. This classifier seemed to give similar accuracies as the Naïve Bayes classifier. Hyperparameter tuning didn't seem to make a lot of difference (only a slight increase (+1%)). After trying some grid search on multiple parameters, I tried to tune the parameters independent of each other (e.g. by finding the best number of estimators with all other parameters fixed). This again seemed to give slightly better results, but that might also be because of the high variance in the accuracies instead of actually better parameters.

4.6 Multi Layer Perceptron Classifier

Intially, the MLP classifier seemed to give again slightly better results than the naïve byes and a random forest, however, this wasn't the case anymore when I averaged the accuracies over multiple runs. Sklearn contained besides an MLP classifier also an MLP regressor, which I also tried since we where rather predicting a score to sort the headlines on to find the winning headline instead of actually classifying each headline independently as winner or not a winner, but this didn't gave any better results.

4.7 Gradient Boosting Classifier

XGBoost is a classifier that has won a lot of Kaggle competitions, so could also give good results for us, but that wasn't the case (it gave worse results than the previous methods). However, during hyperparameter optimization the results on the validation set where a lot better, but that wasn't the case for the actual test set (which means it was overfitting while training the hyperparameters). I tried solving this by calculating the validation score based on multiple folds instead of only one, which made the results of the validation set lower (as expected), but didn't give a better test score. The most important features accoording this model where forward reference, activity, emotion, proper nouns, adressing the reader and quotes. Note that this order is slightly different from the order during the correlation test.

4.8 Proximity Trees

The other models up to now looked at each candidate headline independently, without actually comparing the winner and loosing headlines of the same candidate sets with each other. Proximity Trees try to solve this problem.

4.8.1 Main idea

The idea is based on an earlier paper and is similar to a random forest classifier, but with another kind of decision trees. (section 3 of [4]) Each of the trees is a binary decision tree. For each split, you take a random test set and take the winner headline as one branch and randomly sample a non-winner as the other branch. When choosing a branch, you take the one that is most similar to the input vector. By comparing each headline with a winner and loser from the same candidate set, is the result based on the difference between those two headlines. Just like in a random forest classifier, you take multiple such trees, where each tree is trained on another subset of the complete training set (bootstrapping) and using only a randomly chosen subset of the features (random subspace sampling).

4.8.2 Implementation

I've written my own implementation on this in Python, which takes a lot more time to make prediction than e.g. the sklearn random forest. The model itself is a ProximityForestClassifier, which consists of multiple ProximityTreeClassifiers. Here is an overview of the different hyperparameters.

n_trees The number of trees the forest uses to make a prediction.

Model	manual	extra	embed	all
Random	45.8%	44.6%	44.5%	40.6%
Naïve Bayes	62.5%	61.4%	46.4%	52.3%
Random Forest	62.1%	61.5%	47.8%	59.4%
Multi Layer Perceptron	54.6%	56.2%	48.9%	54.4%
Gradient Boosting	57.5%	58.1%	43.6%	58.6%
Proximity Forest	57.2%	54.5%	42.3%	43.9%

Table 16: Winner prediction accuracy overview

sample_multiple_splits For each set, you must randomly select a candidate set (and pick a winner and loser from this set to generate a candidate pair). Instead of doing this only once for each split, you can also make such choices multiple times and select the one with the best GINI split. This parameter defines the number of random candidate pairs you generate.

max_depth The maximum number of splits that occur on the path from root to leaf in a decission tree.

4.9 Overview

In table 16 you can find an overview of the results of each of the models. This score is generated by taking the average accuracy of multiple folds[10]. The column 'manual' contains the accuracy using the labels Soubry defined as features. The column 'extra' adds extra labels from the 'Word differences' section (Number of words, average word length and max word length of the words not occuring in other headlines from the same candidate set). The column 'embed' contains the winner prediction based on only the embedding of the headline. The column 'all' contains all features mentioned for the other columns.

5 Conclusions

There are a lot of NLP techniques that work better than expected on Dutch sentences. BERTje works great for understanding the meaning of sentences, SpaCy works great for parsing and recognizing the different parts in sentences and for a lot of my scenarios, a simple TF-IDF (with or without stemming, resampling or character tokens) performed better than expected and often provides better understandibility of the most important words for classifying a sentence. There are however still some labels that are a lot harder to classify like emotion, forward reference and adressing the reader.

Some of the simpler models (like naïve bayes) might perform better than expected, so it's often worth trying them, but there performance might drop if you supply too many unimportant features (e.g. when I also added the sentence embedding to the features). Also remember that when optimizing models that have a high variation on the accuracy, you should take the average accuracy over multiple runs and be sure your train and test set are completely independent, to prevent wrongly thinking something is better.

With the current performance, none of the tested models is good enough to actually recommend to journalist which of their candidate headlines would generate the most clicks, which means the features we're currently using aren't enough for this task. We should find features that focus more on the goal, which is finding the best headline along the different headlines, so we must provide more features that show the relation of given headline to other headlines in the same dataset.

[10] 20 for each ofthe models except for the ProximityForestClassifiers, which was tested on 5 folds because it took a way longer time to run

References

[1] Rahul Agarwal. The 5 classification evaluation metrics every data scientist must know, Sep 2019.

[2] Wietse de Vries, Andreas van Cranenburgh, Arianna Bisazza, Tommaso Caselli, Gertjan van Noord, and Malvina Nissim. Bertje: A dutch bert model. *arXiv preprint arXiv:1912.09582*, 2019.

[3] Raoof Gholami and Nikoo Fakhari. Support vector machine: principles, parameters, and applications. In *Handbook of neural computation*, pages 515–535. Elsevier, 2017.

[4] Benjamin Lucas, Ahmed Shifaz, Charlotte Pelletier, Lachlan O'Neill, Nayyar Zaidi, Bart Goethals, François Petitjean, and Geoffrey I Webb. Proximity forest: an effective and scalable distance-based classifier for time series. *Data Mining and Knowledge Discovery*, 33(3):607–635, 2019.

[5] Ankitkumar Patel and Kevin Meehan. Fake news detection on reddit utilising countvectorizer and term frequency-inverse document frequency with logistic regression, multinominalnb and support vector machine. In *2021 32nd Irish Signals and Systems Conference (ISSC)*, pages 1–6. IEEE, 2021.

[6] Anita Kumari Singh and Mogalla Shashi. Vectorization of text documents for identifying unifiable news articles. *International Journal of Advanced Computer Science and Applications*, 10(7), 2019.

[7] Margaux Soubry. Deze artikels klikten we massaal aan (en dit is waarom). Master's thesis, Vrije Universiteit Brussel, Pleinlaan 2, 1050 Elsene, 2020.

[8] Yuli Vasiliev. *Natural Language Processing with Python and SpaCy: A Practical Introduction*. No Starch Press, 2020.

[9] Jonathan J Webster and Chunyu Kit. Tokenization as the initial phase in nlp. In *COLING 1992 volume 4: The 14th international conference on computational linguistics*, 1992.