Newspaper articles

Michael Hodel, Andrin Rehmann

May 2020

Contents

0 Introduction

Language is a very fundamental concept to humanity as it allowed us to create complex social constructs. As fundamental and intuitive language may be to us, it is puzzling and hardly graspable for computers. They may exceed our capabilities to juggle with numbers by a huge margin, but they struggle even with the most simple concepts in language recognition.

In this project we plan to expand our knowledge in the field of natural language processing by analysing a data set with over 200'000 newspaper articles from HuffPost published between 2012 and 2018 as well as make some qualitative statements about the data set. The main goal is to predict the categories of the articles based on their headlines and descriptions.

Chapter 1 'Exploratory Data Analysis' documents '1_eda.ipynb'.
Chapter 2 'Naive Model' documents '2_naive-model.ipynb'.
Chapter 3 'Word and Sentence Embedding' documents '3_embeddings.ipynb'.
Chapter 4 'Similarity Graphs' documents '4_similarity_graphs.ipynb'.
Chapter 5 'Predicting Categories' documents '5_predicting_categoires.ipynb'.
Chapter 6 'Web Scraping' documents '6_web_scraping.ipynb'.

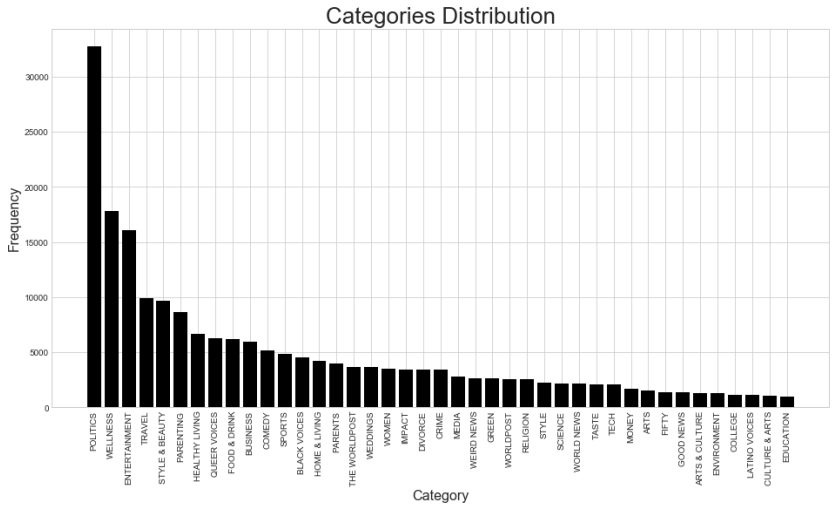
1 Exploratory Data Analysis

First we read the file as it was downloaded from `https://www.kaggle.com/rmisra/news-category-dataset` as a single string and created a list that contains all observations. Then we created a pandas dataframe containing the observations as rows and all the properties 'category', 'headline', 'authors', 'link', 'short description' and 'date' as columns. The data set contains 200853 observations and 41 categories.

1.1 Analysis of Categories

1.1.1 Static Analysis

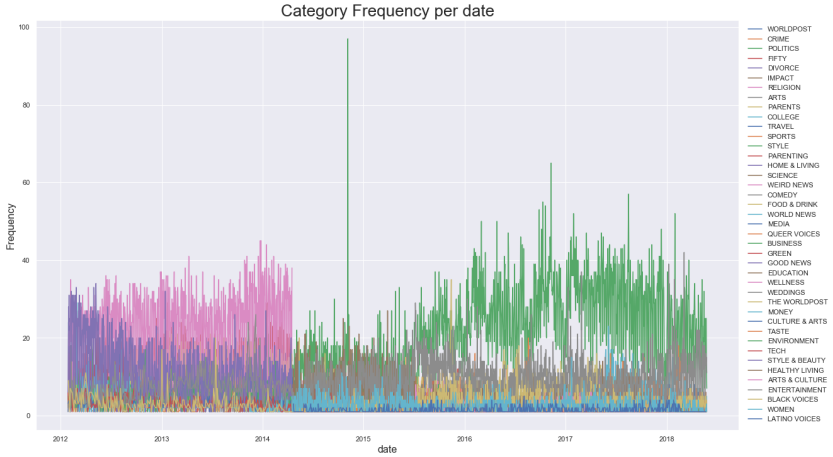
We performed an extensive analysis of the frequency distributions of the number of articles per category. Counting how many observations there are for each category and plotting the numbers in descending order yielded the following graph:



The graph shows a nice exponential decay, as one might have expected for such data, but it also indicates that we are dealing with highly imbalanced data, which might be a problem when trying to predict the categories. See chapter data set balancing.

1.1.2 Temporal Analysis

One way answered our question 'Which topics peak at a specific time (interval) and to which world events may they be related?' was by performing a temporal analysis of the categories distribution. For that we counted how many observations there are in each category for the years, the months and the days and then plotted the resulting time series in three corresponding plots. The following graph shows the number of observations per category per date.



One can see a change in the frequencies in the beginning of 2014. We inspected whether this is because HuffPost made a change to their categories by

counting how many categories were abandoned and how many new categories were introduced at that point in time. We found that in April 2014, 10 categories were abandoned and 17 new categories were introduced.

The number of 'POLITICS' articles peaks around the year change from 2016 to 2017. This is likely due to the US presidential elections that were held in November 2016 and the presidency of Donald Trump starting in January 2017.

Further, the graph shows a massive outlier in the number of 'POLITICS' articles, namely in the end of 2014. On average, ca. 22 articles in 'POLITICS' are published per day. On the date of the outlier, 2014-11-05, 97 'POLITICS' articles were published. This is most likely due to the 2014 United States elections that were held on were held on November 4, 2014.

1.2 Analysis of Words

1.2.1 Static Analysis

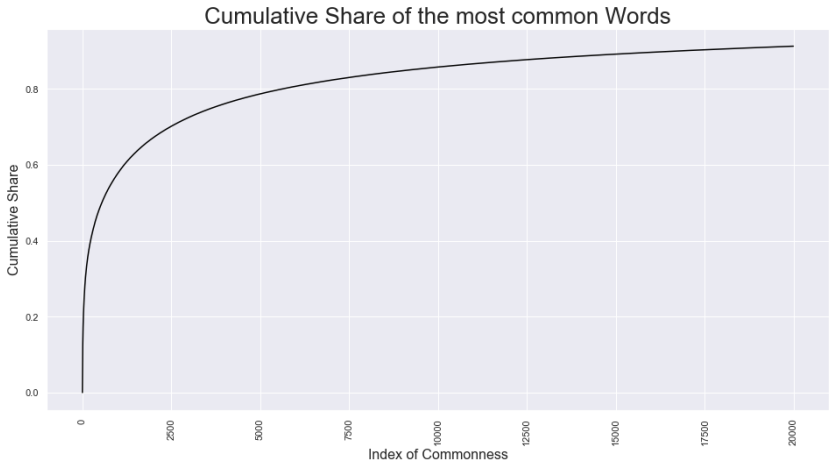
We also performed a similar analysis of the words distribution. The total number of words is 1911627, the number of unique words is 118098. First we inspected if Zipf's Law [1] holds for the word distribution of our data set. To do so, we counted the relative frequency of each unique word, ordered the tuples of words and their relative frequencies descending by relative frequencies and summed up the relative frequencies of the first n most common words, where n = 2, 7, 51, 120, 545. This got us the following results:

n	2	7	51	120	545
cumulative share	5 %	10 %	25 %	33 %	50

Analyzing the most frequent words, again sorted descending by their relative frequencies, yielded a rather nice exponential decay. Since we are also interested which frequent words are specific to our data set and not just generally common words, we plotted the same distribution without considering the 1000 generally most common English words, based on the Brown Corpus. [2]. This second plot yielded a similar distribution, although exhibiting a sharper bend and straighter tail. We also plotted the cumulative share of the most common words, which shows a graph that very nicely resembles a logarithmic function.

[1] Zipf's Law states that the frequency of a word is inversely proportional to its frequency rank.

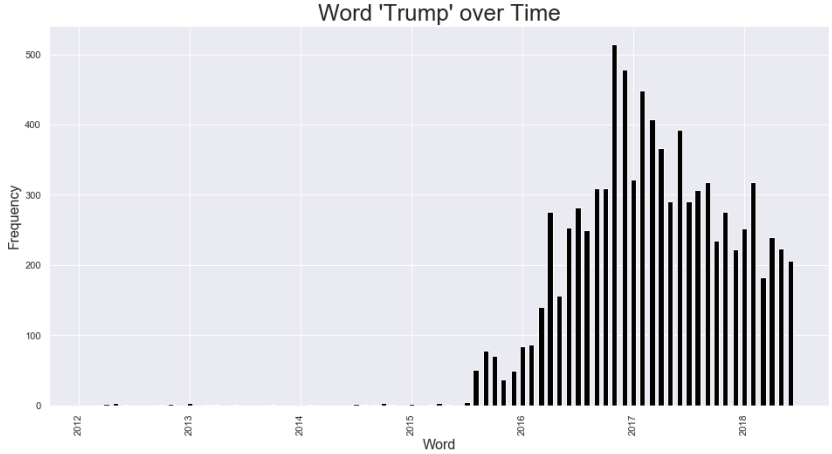
[2] The Brown Corpus is the first digital corpus of English words (containing a million words)



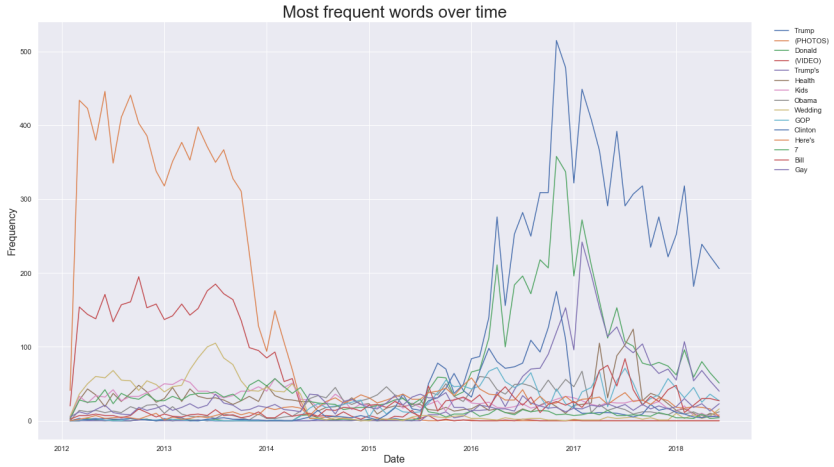
We also explored how the difference in number of observations between the biggest and smallest categories relate to the difference in the corresponding number of unique words. We found that the biggest category, 'POLITICS', has ca. 33 times more observations than the smallest category 'EDUCATION', but only about 10 times more unique words.

1.2.2 Temporal Analysis

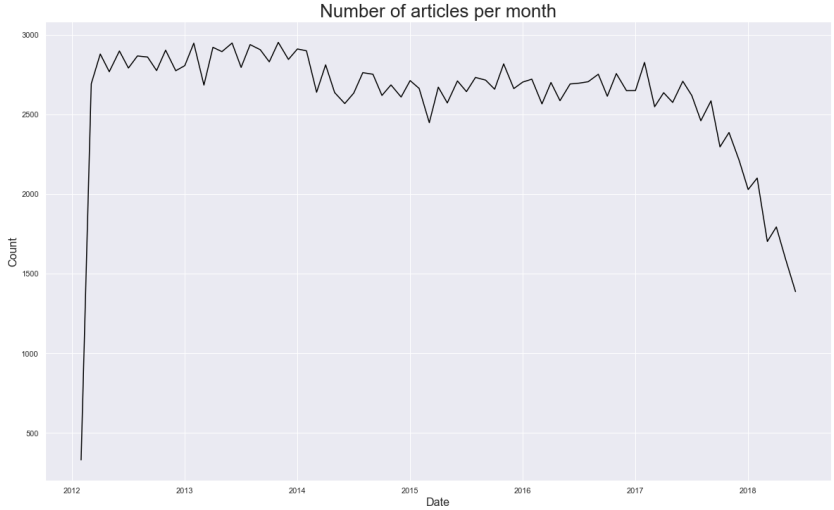
We also performed a temporal analysis of the words distribution by first plotting the most common word (that is not also generally common in the English language), namely 'Trump', over time.



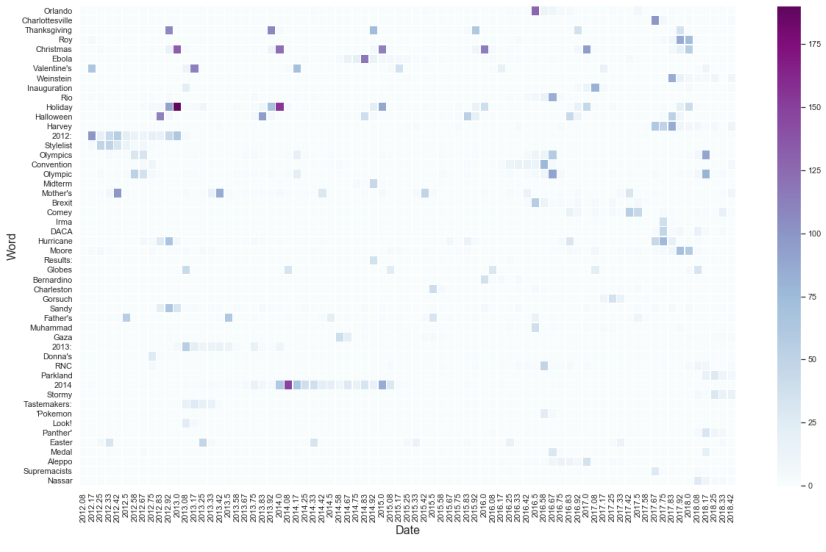
The spike in the word 'Trump' is probably best explained by the 2016 presidential campaign of Donald Trump and his presidency. We also plotted the 15 most common words (again without considering generally common words) and plotting their frequencies over time.



The above graph first gave us the impression that there is period where the most common words occur significantly less frequently, namely around 2015, indicated by the clearly visible low in the middle of the graph. But this turned out to be merely an 'illusion' and due to the fact that the frequency of the tag-words '(PHOTOS)' and '(VIDEOS)' diminish and the Trump-words, due to the presidential elections, only rise after the mentioned period. Nevertheless, we also plotted the number of articles per month to be sure that there is no drop in the number of articles in this time period, which would have been the obvious explanation for the low.



It does not look like there is a decrease in the number of articles in this time period. However, this statement only concerns the articles in the data set. We don't know whether the author included all articles that were published between 2012 and 2018 in the data set. If that is actually the case, then there is a massive decline in the monthly number of articles starting from around 2018.



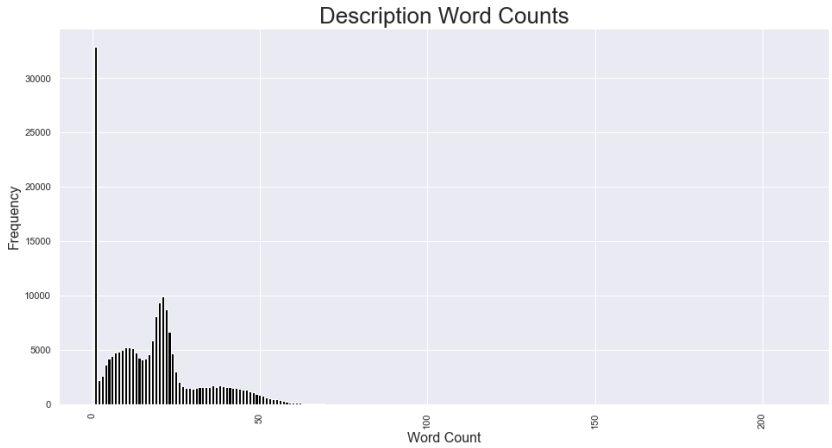
Since simply plotting the most frequent words is very beneficial for headlines that dominate the news for longer periods of time, we needed another strategy to extract events from the data set. One possible way to do so is to look at the words with the highest variance. The problem with this approach is that also fill words like 'the' 'to' seem to peak. To counteract this, we came up with following formula:

$$score_{word} = std(freq_{word}) - 0.03 \times sum(freq_{word}) \tag{1}$$

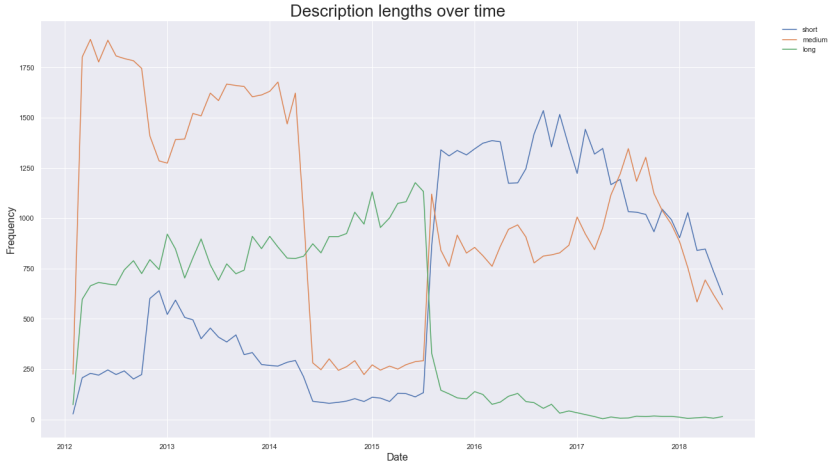
which simply rewards high standard deviation and punishes high frequency of the word. The factor of 0.03 was selected by trial and error. In the above heatmap, we simply plotted the 50 words with the highest scores and their frequencies per month. Interestingly we can now observe things such as: Rio Olympics, Charlottesville right wing protests, Ebola outbreak, Brexit, DACA protests, Weinstein scandal, destruction of Aleppo and also some reoccurring events such as the general olympics, Christmas, Valentine's Day, holidays, etc.

1.3 Headline and Description Lengths

We also analyzed the distributions of the number of words both for the headlines and the short descriptions. For that we plotted the distribution of the word counts. The histogram of the headline word counts hints at a rather nice normal distribution. The histogram of the description word counts, however, has a really unique distribution with three peaks, as can be seen below.



While we were at first puzzled about this strange distribution, after some thinking we came up with the hypothesis that this could be due to a change in the guidelines for journalists or whoever else wrote the short descriptions. We inspected this accordingly by determining the two local minima present in the description word counts plot. The minima are 14 and 29 and, judging visually from the area of the three areas formed by splitting the whole area at the two minima, separate the number of articles for each area into three roughly equally-sized categories. We then plotted a line for each of the three description word count ranges depicting the corresponding number of categories over time.



It looks like indeed, a lot, if not all, of the characteristic distribution in the description lengths can be explained by changes over time: It looks like the description lengths dropped mid 2016, which can be seen in the decrease of articles in the third spike and increase of articles in the first spike. Also interestingly is the interim drop in the number of articles of medium length.

The very high first bar of the description word counts plot turned out to be due to missing descriptions, meaning descriptions only containing single char-

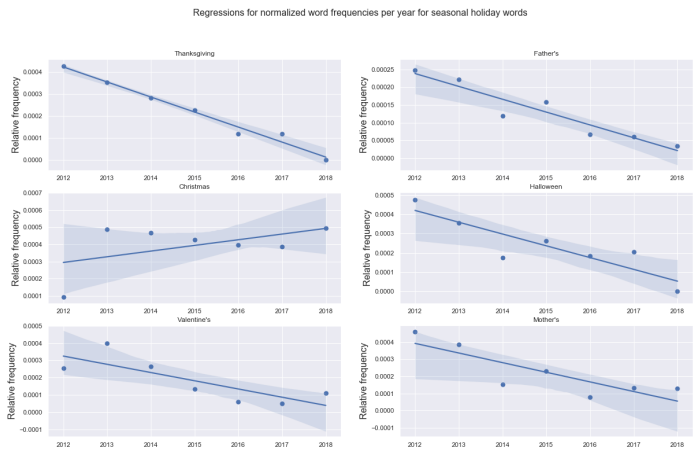
acters. As assumed, the vast majority of the short descriptions with only 0 or 1 words in length are missing descriptions, more precisely, empty strings and double backslashes. Further, there are some interjections and ASCII characters among them. The share of bad descriptions, meaning, descriptions with only 0 or 1 words, is about 16.39 %. The average headline has 9.52 words. The average description, not considering the bad descriptions, has 21.19 words. Thus, average description is about 2.23 times as long as the average headline in terms of number of words. The headline lengths have a variance of 9.65. The description lengths have a variance of 173.88. Thus, the variance of the description length is about 18.01 times greater than the variance of the headline length.

We further noticed that sometimes, there are all-caps words (tags) in brackets at the end of the headlines, probably indicating what kind of content the articles contain. Most prevailing are the tags 'PHOTOS', 'VIDEO' AND 'PHOTO'.

1.4 Trending Terms

While we were examining the data, we noticed some interesting behaviours considering the word frequencies over time. The following word frequencies per year are all normalized, to counteract underrepresenting the first and last year in the data set. At this point we have to mention that we can in no way make conclusions about cultural changes, since it could be that the observations only apply to articles of Huffpost and cannot be extended to other newspapers or a shift in political and individual views. For example, Huffpost could have decided at one point to focus more on specific news articles for whatever reason, which would drastically have shifted the word frequencies.

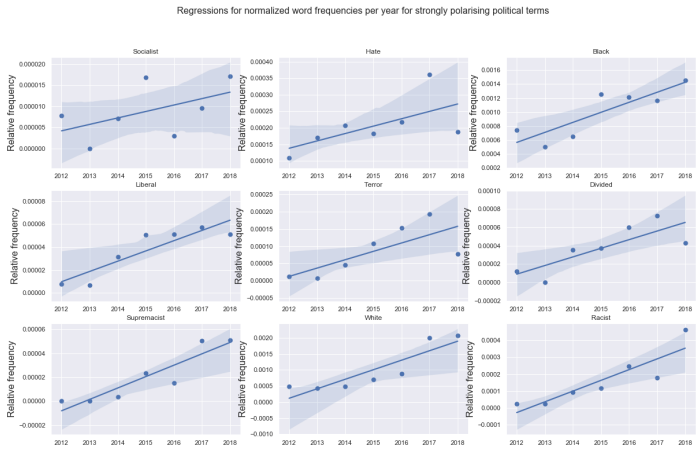
1.4.1 Traditional Holidays on the Decline



The plots indicate that especially Thanksgiving, Fathers and Mothers day as well as Halloween appear less often over time. It is free to speculation what

the reasons for these declines are. On the other hand, due to the values for 2012 and 2018, Christmas does not have such a clear trend.

1.4.2 Political Polarisation on the Rise



Since the recent polarisation of the US' political landscape is a widely debated subject we wanted to examine this further. We have chosen nine words which we associate with polarisation and plotted their frequencies. Interestingly, we found an increase for all the words.

2 Naive Model

As described in our project proposal, the simple approach we planned for prediction was to categorize the headlines based on some sort of weighted majority vote of the categories in which the individual words of the headline occur most frequently. We called this approach 'naive model', since it does not include any machine learning algorithm but rather a more brute-force method. What we did here is counting the most frequent words for each category and then considering the most frequent nouns, again, without the nouns that are generally common in the English language. We decided to limit ourselves to the nouns since we thought they probably capture the meaning of the headlines best.

We wrote a function called naive_model() that takes the hyperparameters sample_size (number of observations that are predicted), nr_nouns (number of most popular nouns to consider) and reward_strength. Given a word sequence, the reward score is computed for each category, as the sum of the scaled ranks. The rank is computed as the difference of the number of nouns considered and the index of the noun in the list of most common nouns for the respective category. The scale is simply a power. The reason for scaling the ranks is that we wanted to check if we could improve the performance of this model by penalizing

higher ranks either sub- or sup-linearly. The model then simply predicts the category which yielded the highest reward score. We also wrote a function called evaluate_model() that takes a sample size, a list nrs_nouns of parameters for nr_nouns and a list reward_strengths of parameters for reward_strength. The function then performs a grid search by iterating over each combination of those parameters and returns tuples containing the hyperparameters as the first value and the accuracy as a percentage as the second value, sorted descending by accuracy.

The reasons for only using small sample sizes and not using the whole data set are the long runtimes of the models and the fact that the naive model was not our main focus. The naive model managed to predict up to a third of the categories correctly. The reward strength did not seem to have any significant influence on the performance. The number of nouns considered too did not have any significant influence, even though models using higher number of nouns tended to perform slightly better. This might also just very likely be due to the small sample sizes. Also, we did not perform any splitting of training and testing sets here since the model is hard-coded and does not do any training.

3 Word and Sentence Embedding

In order to effectively use traditional classifiers on words and sentences, they have to be converted into numerical representations. We could in theory map each word to a unique number, but then the question of order arises. Lets say we would use an alphabetical or any other arbitrary mapping between words and integers, even if the numerical representation of some word abbreviates by a value 5, which is a rather small number considering a standard vocabulary of 20'000, the error in meaning is huge. The same principle is valid for one-hot word embeddings.

We solved this problem by mapping the words to a multidimensional vector space, where similar vectors ideally also represent words with similar meanings. Interestingly, this embedding space can have a relatively low number of dimensions compared to the absolute number of unique words in the text.[3]

3.1 Manual Word Embedding

The process of creating embeddings is an interesting concept and it generally seems to take hold in the machine learning world. Therefore we decided to give it a go and create our own word embeddings using the vocabulary of our data set. In order to achieve this, we built a neural network with one hidden layer, which tries to predict the context for each word in the text. There are two commonly used methods to interpret the concept of predicting context, Skip Gram and CBOW.

[3]Word Embeddings: https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa

3.1.1 CBOW vs. Skip Gram

CBOW (continuous bag of words) takes a number of surrounding words as input in order to predict the word in the center. The number of surrounding words is generally referred to as window size. We will used a window size of 2 for the following example. Given the sentence: *Data science at UZH rocks.* we would generate following relations:

input	**output**
science, rocks	data
data, at, UZH	science
data, science, UZH, rocks	at
science, at, rocks	UZH
at, UZH	rocks

Skip gram on the other hand tries to predict the surrounding words given the centre word. For the example sentence we would get:

input	**output**
data	science, at
science	data, at, UZH
at	data, science, UZH, rocks
UZH	science, at, rocks
rocks	at, UZH

Since skip gram is known to work better for infrequent words and small data sets, we decided to use skip gram for our embeddings.[4]

3.1.2 Subsetting and Preparing the Data Sets

As one may already have suspected from the previous example, it does not really make sense to associate a word with a very common word such as "at". Since "at" can appear pretty much everywhere and is generally one of the most frequent words in the English language, it would have associations with almost every other word. A commonly used method to counteract this is to delete a word with a probability relative to its appearance frequency. The word "at" has a very high frequency and would be very prone to be removed, meanwhile "UZH" probably only appears a few times in a data set, and thus has a very low probability of being removed. [5] Besides subsetting the data set, we removed special characters and made all words lowercase. Then we prepared the data set using the skip gram method. In order to gain an intuitive understanding how such a data set will look like before training the neural network, we printed a small sample of 6 words.

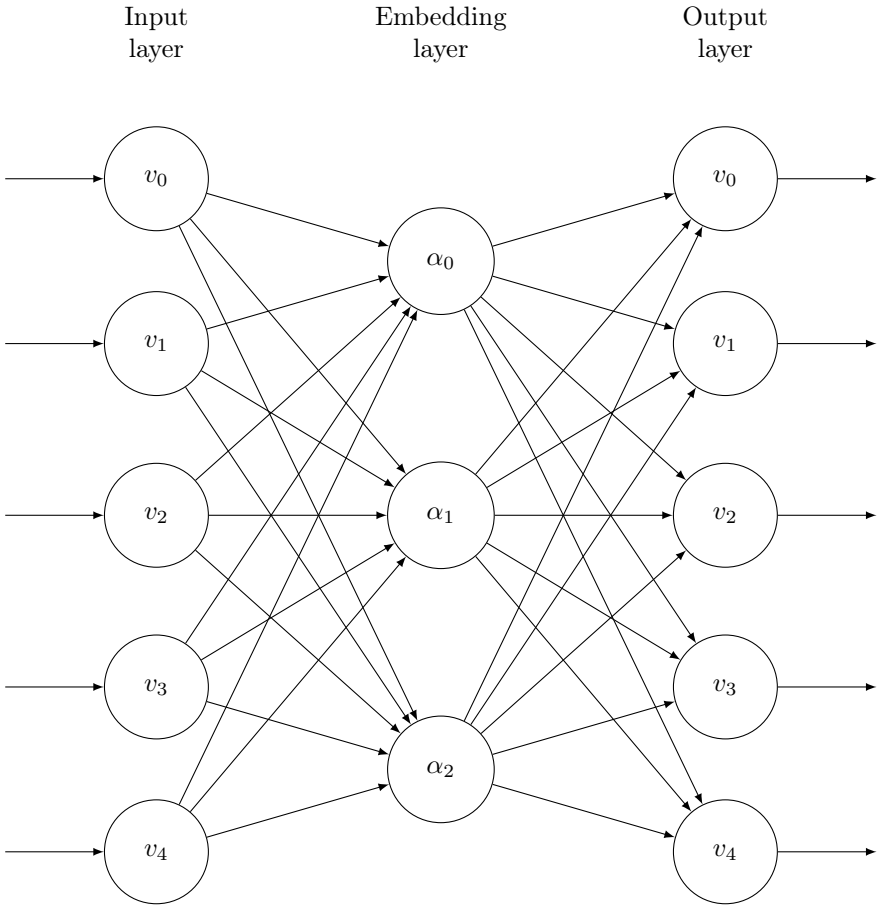
[4]Skip Gram, CBOW: `https://towardsdatascience.com/nlp-101-word2vec-skip-gram-and-cbow-93512ee24314`

[5]Subsetting: `https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/`

input X	wedding	appears	marriage	going	based	fact
output y	night	favor	favor	sweep	popular	point

As the reader may notice, not all the relations make sense, but since the neural network will be searching for a global optimum and therefore to small errors in the training data, the network will still perform well.

3.1.3 Constructing the Neural Network



The model is a straight forward shallow neural network where the dimensions of the in- and output layers are both equal to the number of unique words in the document. The size of the embedding layer is usually fixed such that it is small enough for the emebdding vectors not to be prone to effects commonly known as

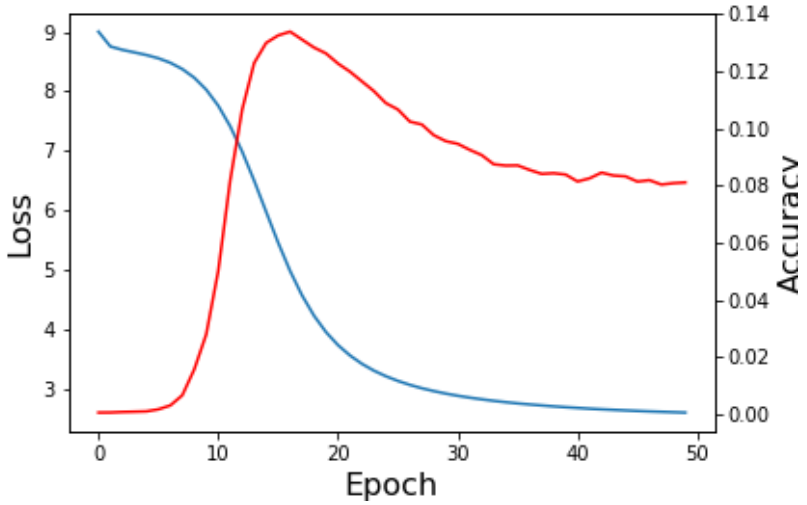
curse of dimensionality [6] but big enough to have sufficient information encoded. After the model was fully trained, we used the weights from the embedding layer as our embedding matrix. We used an embedding size of 300, as this yields the best improvement of accuracy compared to speed loss. [7] We applied a relu activation functions to all hidden layer as well as a softmax function to the output layer of the neural network. Finally, in order to calculate loss, we used a categorical cross-entropy function which seems to be the best choice for multi-categorical classifications. To implement the neural network, we used the tensorflow and keras libraries as they are fairly easy to use for beginners.

3.1.4 Training the Neural Network

Considering our vocabulary consisting of the five words from our example [at, data, rocks, science, uzh] and the one hot encoding vector being constructed by looking up the alphabetical ordering of the vocabulary, we would get the following vector pairs for the word: data.

$$\left(\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \right), \left(\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \right) \tag{2}$$

3.1.5 Overfitting of Word Embeddings



[6] Curse of dimensionality: https://towardsdatascience.com/the-curse-of-dimensionality-50dc6e49aa1e

[7] Global Vectors for Word Representation https://nlp.stanford.edu/pubs/glove.pdf/

Because we want to fit our vocabulary as good as possible, and we do not really care about words which are not part of the vocabulary, we did not worry about overfitting in this model. That is also the reason we did not introduce a train test split in this step.

In the above graphic we have plotted the training loss (in blue) vs. the training accuracy (in red). The decrease in accuracy is not an issue, as long as the loss function decreases as well. It simply means that the model is becoming more "confident" in its predictions. Also, considering that this set is being trained on almost 10'000 unique words, which is equal to 10'000 categories, an accuracy of 10% is fairly good. But even after some research we did not completely understand the exact reason why loss and accuracy can decrease at the same time.

3.1.6 Extracting the Embedding Matrix

Once the model finished training, we extracted the embedding matrix and calculated the individual embedding vectors. Each node of the embedding layer is connected to all the nodes of the input layer. Thus we can extract a matrix of shape $VocabularySize \times EmbeddingLayerSize$ which can then be multiplied with all the one-hot encodings of words in the vocabulary, which results in the embedding layers.

3.1.7 Performance Issues

Since we did not have enough time to optimize some functions, especially for creating one hot encodings on the data set, we were only able to train the embedding models on very small sub-samples of the data set, which usually only contained around 3000 articles of a total of over 200'000. It is to note that 3000 newspaper articles still generate a training data set of around 100'000 in and output words. Besides preprocessing, actually training the model also took a long time, since it benefited from running for around 50 epochs, and the in- and output sizes are usually around 40'000, which results in a lot of weights.

3.1.8 Conclusion and Accuracy

As it is difficult to test the word embeddings numerically, we will simply look at 5 most similar word vectors of some sample words. Similar words are defined as words with a small euclidean distance between the target vector representations of the given words.

input	**5 most similar words**
dog	mitzvah, solita, humanly, uncofmy, corn
cat	paints, whiskers, tattooed, overdosing, frame
house	victim, catpure, ube, comity, representatives, tackling
door	victim, exit, irritating, locks, monitor
trump	which, handles, couadfounder, to, nose
pen	random, johanna, diplomacy, solve, beck

As the reader may verify, there are some connections which make sense, such as (door, locks), (door, exit), (cat, whiskers) but others which are rather nonsense, such as (dog, corn), (trump, nose). The results of our own word embeddings are acceptable, but not good enough for the further process of this project. Surely they could be improved by tweaking the training data set and using the entire data set instead of such a tiny subset, but because of time reasons, we had to move on and thus used a library for a more accurate word embedding.

3.2 Embedding with Gensim

3.2.1 Word Embedding

To embed the words into a vector space, we first tokenized all the headlines and descriptions, yielding a list containing a list of words for each observation. The gensim[8] word2vec algorithm very conveniently accepts such a list as input, applies skip-gram and CBOW models, and returns an embedding model. As parameters we used a window size of 4 and embeddings size of 300.[9] We then applied this algorithm on both the tokenized headlines as well as the tokenized descriptions. Since the descriptions on average are composed of significantly more words than the headlines, we had the hypothesis that the model trained on the description will enable a better performing classifier. Since it is unfortunately hard to make a qualitative statement about the model precision, as it is not clear how to measure the precision of the generated vectors, we ran some experiments on the two models using the provided methods by gensim to see which model better represents our own intuitive understanding of the words. Both models performed better than our manual embedding model. The vectors correspond to a large percentage with our own intuitive understanding of the words. Based on this experimental assessment of the models, the model based on the short descriptions seems to embed the words better than the one based on the headlines. The following are the 10 most similar words to 'trump' with their similarities, ordered descending by similarity, according to the gensim models:

Headline embedding :

1. 'trumps: 0.8251',
2. 'trumpus: 0.8028',
3. 'obama: 0.7783',
4. 'glover: 0.7765',
5. 'ted: 0.6814',
6. 'rubio: 0.6782',
7. 'sterling: 0.6756',
8. 'marco: 0.6585',
9. 'nra: 0.649',
10. 'cruz: 0.6487'

[8]Python library for NLP: `https://github.com/RaRe-Technologies/gensim`
[9]Global Vectors for Word Representation `https://nlp.stanford.edu/pubs/glove.pdf/`

Description embedding :

1. 'trumpus: 0.7797',
2. 'trumps: 0.7635',
3. 'obama: 0.7481',
4. 'sanders: 0.6947',
5. 'clinton: 0.6543',
6. 'glover: 0.6499',
7. 'gop: 0.6469',
8. 'rubio: 0.6312',
9. 'presidentelect: 0.6249',
10. 'sterling: 0.6205'

As one can see, the headline embedding yields words with greater similarities than the description embedding. This might be due to the headlines making up a smaller corpus than the descriptions. The description model manages have trump very similar to other presidents and presidential candidates such as Sanders and Clinton, which the headline model fails to do. Since this is obviously only an example, one can not make any conclusions on how well the descriptions embedding performs as a predictor for the categories compared to the headlines embedding.

3.2.2 Weighing Words using TF-IDF

Since the vast majority of words present in our data are generally common words and thus not specific to the headline or description they appear in, the word lists representing the observations can be considered very noisy. We reduced this noise by giving more weight to words that represent their respective categories well and vice versa. This was achieved by weighing the words with their TF-IDF (term frequency - inverse term frequency) statistic. TF-IDF, the product of TF and IDF, indicates the importance of a word for the document (in our case headlines or descriptions of a specific category) in a corpus (in our case list of headlines or descriptions). While there are different variations of TF and IDF, TF generally speaking simply tries to measure the word frequency in a document and IDF tries to measure the amount of information of the word. 'Weighing' the word vectors with their TF-IDF statistic simply means scaling them with a scalar. We implemented this 'manually', meaning without using a package. Unfortunately, the tf-idf calculation and especially the weighing of the words with their tf-idf scores had really long runtimes and, since the gensim package was not installed on the server and we could thus not utilize it in this step, we did not pursue the combined approach (headlines and descriptions) any further.

3.2.3 Sentence Embedding

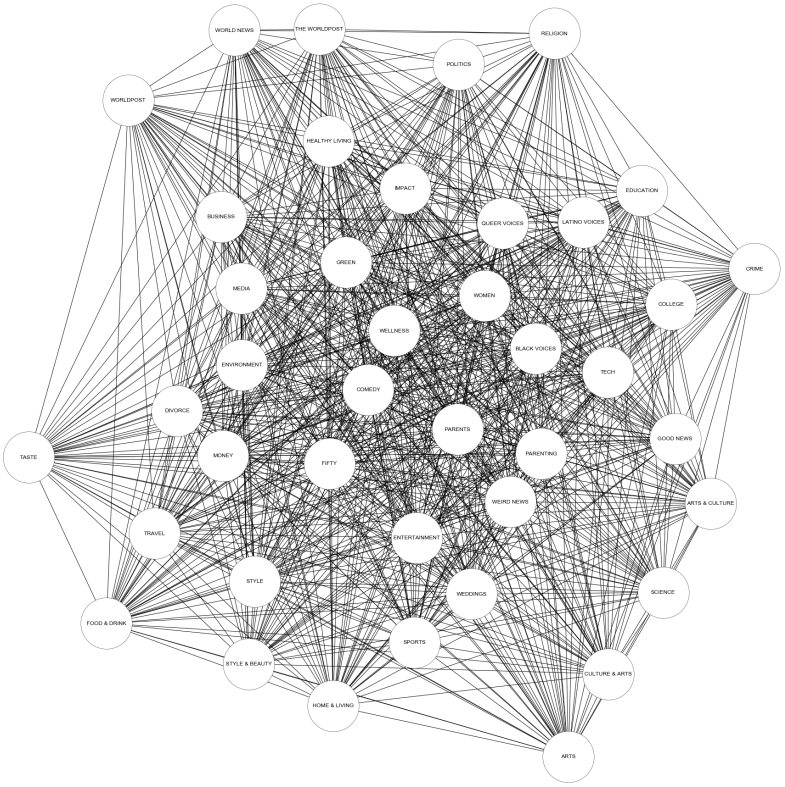
Every embedded word is represented by a vector. Since we are interested in the headlines and short descriptions, which are lists of words, we had to find a way to represent them based on their corresponding word vectors. Naively, one could represent a list of words as a list of word vectors. But since most

common machine learning algorithms take a vector of numbers as input and not a list of vectors, and flattening the list of lists to a one-dimensional list would completely distort the embedding, mainly due to varying numbers of words in the headlines and descriptions, we had to do a more sophisticated approach. One very common such approach is letting the mean of the word vectors of a text represent the text. This is what we did.

4 Similarity Graphs

4.1 Category Similarity

We thought that it would be interesting visualize the similarities of the individual categories in a graph. To do so we plotted the categories as nodes and the similarities between them as weighted edges. We again based the similarity measure on the most popular nouns in the categories. For each pair of categories, we simply defined the similarity as the cardinality of the set difference of the n most popular nouns, where n is the number of nouns in the category with the least nouns, since we did not want to introduce bias from having differently sized popular nouns lists. This yielded the following graph:



Despite the rather naive approach, the graph yielded a nice result, grouping categories that we would naturally consider similar close together and ones that seem unrelated further apart. One might have been able to improve the similarity measures by considering not only nouns but also other word types such as adjectives or verbs, tinkering with the number of nouns to consider for the set differences or scaling the similarities with a non-linear function to over-proportionally penalize greater dissimilarities. Surprisingly, the two most similar categories, 'FOOD DRINK' and 'TASTE', yielded a similarity score of 1. Further, out of the 820 category pairs, there were 10 with a similarity score of 0.

The 10 most similar categories and their similarities:

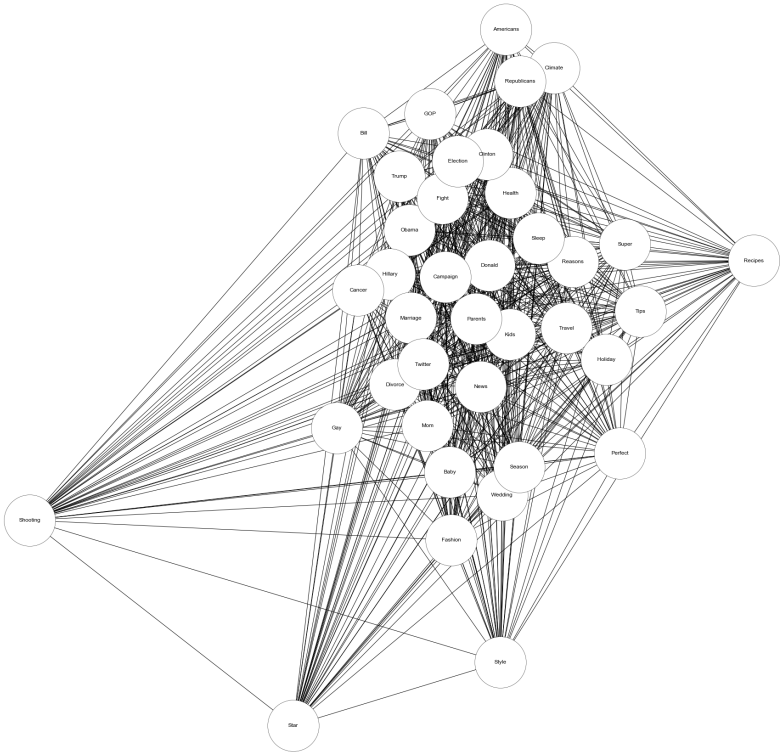
1. ('FOOD & DRINK', 'TASTE', 1.0)
2. ('WELLNESS', 'HEALTHY LIVING', 0.897)
3. ('THE WORLDPOST', 'WORLDPOST', 0.810)
4. ('GREEN', 'ENVIRONMENT', 0.776)
5. ('STYLE', 'STYLE & BEAUTY', 0.759)
6. ('PARENTS', 'PARENTING', 0.759)
7. ('THE WORLDPOST', 'WORLD NEWS', 0.741)
8. ('CULTURE & ARTS', 'ARTS', 0.707)
9. ('WORLD NEWS', 'WORLDPOST', 0.552)
10. ('GOOD NEWS', 'WEIRD NEWS', 0.483)

The 10 most dissimilar categories (similarity of 0):

1. ('FOOD DRINK', 'WORLD NEWS')
2. ('ARTS', 'TASTE')
3. ('CULTURE & ARTS', 'HEALTHY LIVING')
4. ('MONEY', 'ARTS & CULTURE')
5. ('CRIME', 'TASTE')
6. ('ARTS', 'HEALTHY LIVING')
7. ('TRAVEL', 'POLITICS')
8. ('THE WORLDPOST', 'DIVORCE')
9. ('RELIGION', 'ARTS')
10. ('MONEY', 'WORLD NEWS')

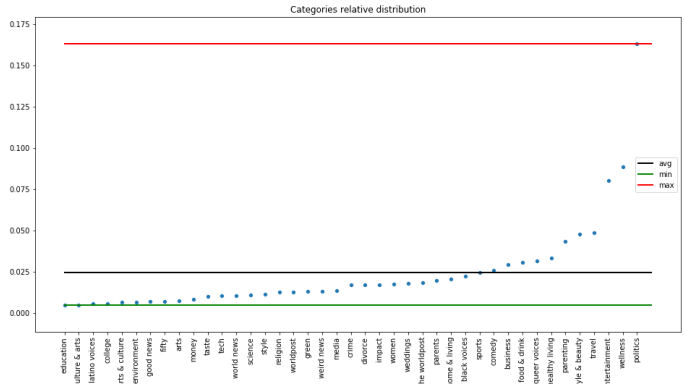
4.2 Word Similarity

Since we already had the category similarity graph and also a very nice notion of word similarity provided by our gensim embedding, we thought it would also be interesting to do a similar graph for the most popular nouns. We used the 60 most popular nouns, again without considering generally common nouns, also without considering other uninteresting words such as tags or single characters. This yielded the following graph:



5 Predicting Categories

5.1 Data set balancing



The above plot displays the relative frequencies of all categories int the data set. As already mentioned the data set is not balanced. This means that if we would implement a model which will naively predict "politics" no mater what the input is, its accuracy would be around 15% which is clearly better than if we had a balanced data set, where it would yield an accuracy of around 2.5%. Another issue with an imbalanced data sets is that the prediction accuracies for specific article categories may differ greatly among each other. We think this dynamic is negligible. Nevertheless we explored a few options to deal with this issue:

- remove categories which abbreviate largely from the mean
- balance the data set either by generating or removing data

We would have a data loss of more than 75% if we would use undersampling to remove data and make all categories as big as the smallest category. On the other hand, if we would use oversampling, we would generate an additional 570% of data, which is also problematic, as we are already struggling with the size of the data sets.

5.2 Machine Learning Algorithms

We created training and testing set for each the headlines, the descriptions, the weighted headlines and the weighted descriptions, using a 80-20 train-test split. In order to not have to run models that don't perform well anyway on the full training sets, we decided on first running models on only a handful of small categories. This way, we can get a first impression on how well different classifier

algorithms and predictors perform and then base our decision on what to pursue further based on the obtained results. The rather crucial assumption we made here is that the order of performances over the different predictors and classifiers would remain rather stable when compared to training the models on the full training sets. This assumption is obviously not entirely sound. The rows in the table below denote the classifiers used (SVC = Support Vector Classifier, RF = Random Forest Classifier, KNN = K-nearest neighbors and the columns denote the predictors used, where headlines are abbreviated as h. and descriptions as d.:

	headlines	**descriptions**	**weigh. h.**	**weigh. d.**	**mean**
SVC	0.632883	0.667368	0.666414	0.634858	65.04 %
RFC	0.586682	0.614808	0.701933	0.602689	62.65 %
KNN	0.528564	0.503112	0.632882	0.535272	55.0 %
mean	58.27 %	59.51 %	66.71 %	59.09 %	60.90 %

As expected, using the descriptions on average led to a better performance than using the headlines, however, only very slightly (< 2 %). Also, weighing the headlines led to a significant increase in the accuracy (> 8 %). Interestingly, however, weighing the descriptions actually on average led to a slight decrease in performance (< 0.5 %), leaving the weighted headlines as the best predictor that we want to pursue further. Naturally, when looking at the performance increase that weighing the headlines yielded, one might expect a similar performance gain when weighing the descriptions, which was not the case. The reason for this could be that weighing the description words with their tf-idf had a flaw somewhere. Also, on average, SVC performed best with an accuracy of over 65 % and RF performed slightly (< 3 %) worse. This might change once we optimize the hyperparameters of the models. Not least due to its comparably poor performance, we decided to not pursue KNN any further.

5.3 Random Forests

5.3.1 Default Model Parameters

As mentioned in our project proposal, we planned to use tree based methods, random forests in particular, since they are known to perform very well. We used the five-fold cross-validation accuracies as a measure to compare the different predictors. This approach, using the full training sets, yielded the following results:

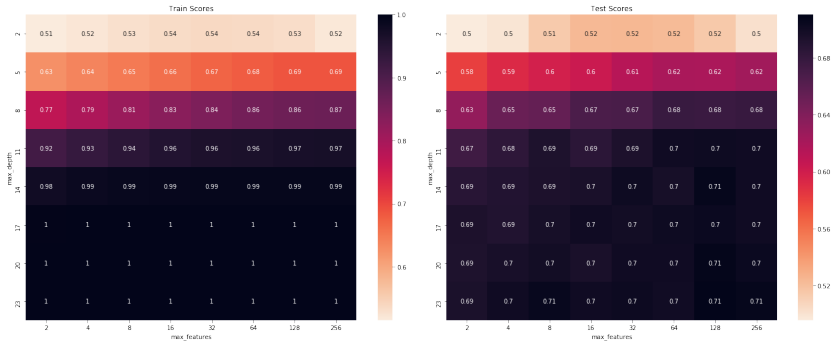
	split 1	**split 2**	**split 3**	**split 4**	**split 5**	**mean**
headlines	0.3864	0.3841	0.3851	0.3863	0.3894	38.62 %
weighted h.	0.5156	0.5158	0.5170	0.5196	0.5181	51.72 %
descriptions	0.3114	0.3123	0.3135	0.3142	0.3148	31.32 %
weighted d.	0.3137	0.3150	0.3211	0.3165	0.3208	31.74 %

Fortunately, the variance of the cross validation test accuracies for a given set of model parameters is minimal, as the accuracies always vary within an

interval of less than one percent, which indicates a stable algorithm in terms of changing training data. Since the weighted headlines as predictors clearly outperform the other three, namely by over 10 %, we only considered them for the random forest parameter optimization.

5.3.2 Hyperparameter Tuning

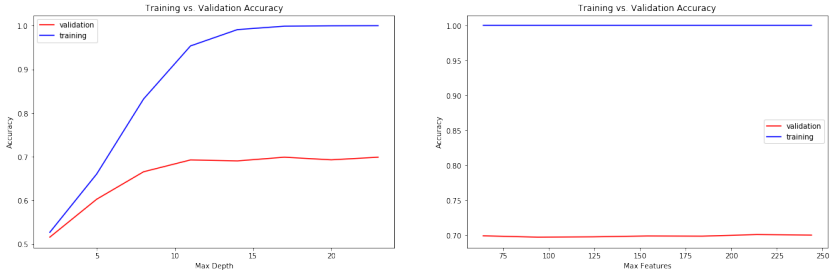
In order to tune the random forests, we again considered only a small subset of all the categories for runtime reasons, making the same assumption about the stability of the order of model performances as made earlier. We used the grid search over a parameter grid (list of values for each model parameter), which is an exhaustive technique that essentially fits models for all combinations of the provided parameters using cross validation. Since we assume all dimensions of our word-vector-space to be approximately equally important, we decided to also inspect large numbers of features (dimensions) when looking for the best split, namely adding large values for the max_features parameter. Also, we decided to examine the tree depths as a second parameter for the grid search. We used random forests with 100 trees each (n_estimators), since this is the default value for the current sklearn version. Here, the provided server came in really handy with its 16 concurrent workers, saving us a lot of waiting time. The following graphs show the results of the grid search as heatmaps for the mean train (left) and test (right) cross-validation scores, with the max_features values on the horizontal and the max_depth values on the vertical axis, darker colors indicating a better accuracy.



The grid search yielded 71.04 % as the best mean cross validation test score, corresponding to the best parameters 'max_features': 128, 'max_depth: 23. Compared to the default parameters, which led to a mean cross validation test accuracy of 65.97 %, the parameter optimization thus managed to increase the accuracy by almost 5 %. It needs to be stated, however, that this accuracy increase can mainly be attributed to the higher number of trees [10]. Considering 100 estimators as the benchmark, which yielded an accuracy of already 69.92 %, the accuracy increase is barely one percent.

[10] The version of the sklearn package installed on the provided on the server was not quite up to date and still used 10 as a default value for $n_estimators instead of 100$

Since the best max_depth parameter lies on the upper boundary of the used range, it can be assumed that even higher values for max_depth lead to an even better performance. The max_features parameter peaks at 128, which might be due to chance, but also very well due to an accuracy maximum for values of max_features between 64 and 256, the next smaller and larger values provided in the range. Correspondingly, we did a more fine-grained search over different values for max_features in the mentioned range. The following graphs show the train (blue) and test (red) accuracies for both the further explored values of max_depth and max_features.



It looks like the accuracy stagnates in the tree depth, however, the best accuracy was reached for a max_depth value of 17, which is likely due to chance. The accuracy in max_features in the range [64, 256] is almost constant, but peaks at a value for max_features of 214, which, again, is most likely due to chance.

Unfortunately, but importantly, it needs to be stated that the very good and sometimes even perfect training accuracies of the random forests indicate severe overfitting. Thus, there may be a random forest with a smaller training but higher testing accuracy. However, since the random forest is probably not the best machine learning technique for our problem (since all predictors are of about equal importance) and we did not manage to find out why the random forest overfits, we did not inspect the random forest any further.

5.4 Gradient Boosting

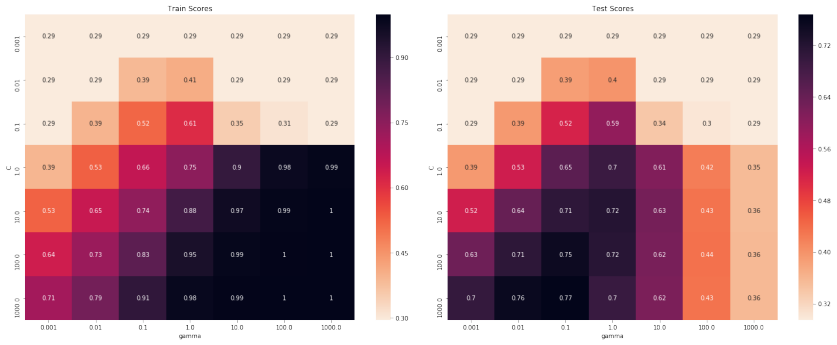
Another technique that is known to perform well and is typically tree-based is gradient boosting. As opposed to the random forest, which trains the classifiers independently (in parallel), uses fully grown trees and tries to minimize the error mainly by reducing variance, gradient boosting is an iterative (sequential) approach that uses shallow trees and tries to minimize the error by mainly reducing the bias.

Surprisingly, the gradient boosting technique with the default parameters managed to yield a mean CV test accuracy of 71.11 % - which is better than our random forest after hyperparameter tuning. Since tree based methods were not our main focus however, we did not perform any hypeparameter tuning for the gradient boosting.

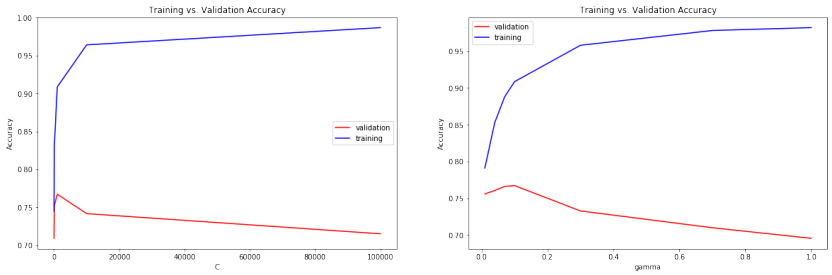
5.5 Support Vector Machines

Support Vector Machines with a linear kernel are known to be one of the best text classification techniques. One usually scales the data before applying support vector machines, since the kernel method uses a distance measure. However, we have an 'artificial' vector space that assumably does not have large differences in the values for the different dimensions. In the case of more natural features, for example a predictor price that ranges between say values of 10'000 and 100'000 and a predictor age that ranges between values of 0 and 20, scaling would obviously be crucial since the distance measures would be pretty much meaningless. Scaling each dimension would probably distort the relations between the sentence vectors. Therefore, we did not perform any scaling.

Here we focused only on the weighted headlines as predictors, since they have shown to perform best in the previous approaches, and only on a few small categories, since we wanted to keep runtimes manageable for parameter tuning. The support vector classifier with the default parameters has an accuracy of 46.27 %. We again used grid search for hyperparameter tuning, here with the parameters gamma (inverse influence of a single training example) and C (importance measure for avoiding misclassifications), each with values as powers of ten, with powers ranging from -3 to 3. The best mean test cv score amounted to 76.73 %, with the corresponding parameters 'C': 1000, 'gamma': 0.1. Thus, tuning the parameters of the support vector classifier lead to an absolute accuracy increase of about 30.45 %, which is very remarkable. The following graphs show the results of the grid search as heatmaps in the same fashion as in the section about the random forests, with the C values on the horizontal and the gamma values on the vertical axis.



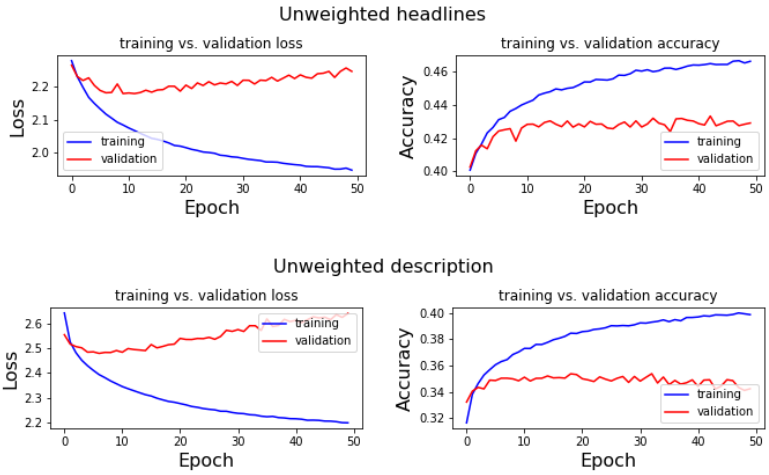
One can assume that the ideal gamma parameter lies somewhere in the interval [0.01, 1], the next smaller and larger values from 0.1 in the provided range. Since the best value for C was the largest in the provided range, larger values for C might lead to even better accuracies. Accordingly, we explored larger C values for gamma = 0.1 as well as values for gamma in the stated interval for C = 1000. The following graphs show the train (blue) and test (red) accuracies for both the further explored values of C and gamma.

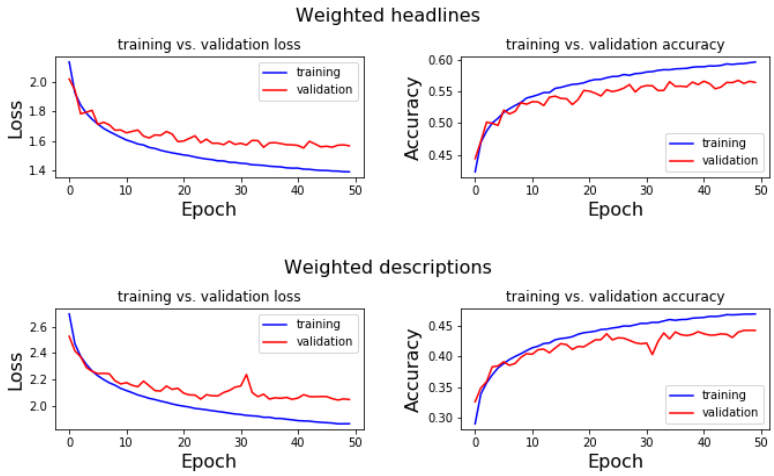


Regarding larger C values for gamma = 0.1, the accuracy peaks again at a C value of around 1000. The accuracy regarding gamma values in the range [0.01, 1] for C = 1000 also again peaks at a gamma value of about 0.1. What needs to be stated is that due to the possible interactions between C and gamma, which influences the accuracy and especially the differences in training data size, it is by all means possible that these ideal values for gamma and C are shifted when it comes to the full training set. As with the random forest, we had rather large differences in the test and validation accuracy, the train accuracy reaching around 90 % for the best test accuracy. Correspondingly, there might be an even better set of model parameters for the support vector classifier.

Given our assumption about the stability of the accuracy order over the different model parameters, we trained the the support vector classifier on for runtime reasons only a trimmed training set, namely, without considering categories for which the number of observations deviates more than half the standard deviation from the mean number of observations per category. This yielded a mean validation accuracy of 57.48 % for 5-fold cross validation.

5.6 Neural Networks





We are using a deep neural network with an input layer of the same size as our embeddings, a fully connected layer of size 200, another fully connected layer of size 150 and finally our fully connected output layer with a softmax activation function. The loss function is the categorical cross-entropy function and the optimizer is the tensorflow built in 'adam' optimizer. We tested this network on the four different data sets. The weighted headlines data set clearly works the best. Interestingly we can observe that the weighted data sets are much less prone to over-fitting.

The model which was trained on the weighted headlines training set, reached an accuracy of 56 % on the test set.

6 Web Scraping

Since it would obviously be very interesting to see how much better our models perform given the full article texts as inputs, we initially planned, as mentioned in our project proposal, to scrape the full articles from the web using the URLs provided in the data set. We managed to fetch the articles by using the Python package urllibs and extract the article text from the scraped file, which contained a lot of HTML and JavaScript code, using the package bs4. We estimated the space that would be needed and the time it would take to fetch all the 200'000+ articles. Despite the fact that according to our estimate, all article texts would probably take up less than 1 GB of space, our estimate for the required time totaled a whopping 10 days.

However, since we had to use mockup-headers for the request to avoid the error 'HTTP Error 403: Forbidden', we got concerned about the legal aspects of this approach. Correspondingly, we informed ourselves with the help of the 'Department of Data Protection at the University of Zurich' and realized, that what we planned to do, was probably not a very good idea. The main reason

for this was that in the user agreements of the HuffPost it says under 2. e. Use of services: "[...] You must not misuse or interfere with the Services or try to access them using a method other than the interface and the instructions that we provide. [...]". One may easily interpret the automatized requesting of news articles with the help of a Python script as a "method other than the interface we provide". Further, there are concerns regarding copyright, privacy and data protection laws - areas, where we simply lack the knowledge. Therefore, to not violate their terms of service of HuffPost and be on the save side, we decided against fetching the articles.

7 Afterword

7.1 Challenges

Of course we were faced with many challenges, so we believe it is important to reflect on those, such that we can improve ourselves in future projects.

7.1.1 Amount of Data

Since most language processing libraries were not available on the remote server, and the server still uses outdated package versions such as tensorflow v1 and sklearn v0.20.1, we were forced to do most of the work on our local machines. This was very time consuming, especially for the process of preparing our data set for the category classifications. Also, it hindered us to fully explore all possibilities such as training the manual word embeddings on the full data set, etc.

7.1.2 Jupyter Notebooks

We think jupyter notebooks is a great way to do simple and small data science projects, but this project consists of many thousand lines of code and processing intensive work, so the disadvantages of jupyter notebooks really began to show up. One of the biggest challanges for working in a team with jupyter notebooks is that they do not math very well with github. It is basically impossible for two persons to work on the same file an then merge the two files. Since jupyter notebook stores the variables locally, one has to be very careful not to use deprecated variables, or apply the operations twice on one variable. Also, jupyter notebooks do not allow us to define functions which can be used in multiple different files, which was a huge drawback for this project, since many operations could have been abstracted and the code left easier to read, optimize and debug.

7.1.3 Project Scope

The scope of the project was definitely on the bigger end of the spectrum. It was hard to keep the focus on the research questions and not to get lost in the

intriguing depths of natural language processing, visualisations or classifiers. Unfortunately, some questions could have been explored for much longer, as we haven not nearly exhausted its information content.

7.2 Learnings

During this project we learned a lot about data science, its processes, challenges and potential. It was really interesting to get familiar with the concept of word embeddings and other concepts in natural language processing, as they are all relatively new concepts and there is still a lot of unused potential. Overall, we greatly enjoyed the format of the course and applying the different techniques that were covered.