university of groningen / faculty of mathematics and natural sciences / artificial intelligence

Master's Thesis

Predicting online article popularity based on text-generated features

Panagiotis Giagkoulas
S3423883

Department of Artificial Intelligence
University of Groningen, The Netherlands

Internal Supervisor: Dr. M.A. Wiering (Artificial Intelligence, University of Groningen)
External Supervisor: Wouter Storteboom (theFactor.e, Friesestraatweg 215A, Groningen)

January 22, 2020

Abstract

Predicting the popularity of online content can have great value for all parties involved, from content creators and editors to technical and marketing personnel. Many methods have been developed and they either rely on the first available indicators of popularity right after publication of the content or rely solely on the content itself in order to make a prediction well before publication. This latter approach is called cold-start prediction and it will be the focus of this thesis.

Content-wise we will focus on online news articles. We will employ embedding techniques, that have not yet been investigated thoroughly in cold-start popularity prediction, to encode the main text of our articles. We will use these encodings to train simple predictive models for both binary classification and regression tasks. Our main aim is to develop a proof of concept for the suitability of our embedding methods of choice in this task.

To develop a reliable and informative proof of concept we experiment with three encoding methods to represent the texts of the articles, namely tf-idf indexes, Word2Vector averaged word embeddings and Document2Vector document embeddings. For predictive models, we experiment with Logistic Regression, Support Vector Machines for classification (SVCs) and regression (SVRs) and Linear Regression.

Our system shows promise in the binary classification task while it performs poorly on the regression task. The best performing models are Document2Vector for encoding in both tasks while SVCs are the best performing classifiers and SVRs are the regressors that showed the most potential. The quality of the embeddings can be improved to lead to better results in both tasks and more complex models can be used to utilize fully the information encapsulated in the document embeddings. In total our proof of concept shows that our methods of choice for encoding and prediction can be successfully applied on cold-start popularity prediction, although improvements are necessary for real-world usage.

Acknowledgements

I would like to express my gratitude to all the people who supported me in this project. First, Dr. M.A. Wiering, for accepting to be my supervisor and for his advice and guidance throughout the project. Second, I would like to thank my external supervisor, Wouter Storteboom, and the whole team of theFactor.e, that welcomed me, provided me with a friendly environment and all the resources and assistance I needed to conduct my research. I would also like to thank my friends and family for their continuous support in all my endeavors that helped keep moving forward and achieve my goals.

Finally, I would like to pay some tribute to the 286 cups of tea that were consumed throughout this project. You definitely added flavour to my everyday routine.

Panagiotis Giagkoulas
January 22, 2020

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We are in an era in which internet users have access to vast amounts of media content from a wide variety of sources and, more often than not, need to filter out low quality or unreliable content. Finding trustworthy sources of information and high-quality content is a strenuous process for the users but with the numerous available alternatives, it is easy for them to follow a trial and error approach and disregard unwanted sources entirely. If the personal preferences and characteristics of the users are also taken into account, then it becomes apparent that attracting and maintaining a loyal online audience is no simple task. Therefore content creators need to provide high-quality, reliable and personalized content to attract audiences. In order to achieve that, a wide variety of tools is employed with the most recent being machine learning which has already been successfully applied in recommender systems [39], offering users personalized content projection.

The Factor.e is a full-service digital agency that offers personalized web and digital services to clients. Many of their clients are content creators or have considerable online content in their websites; more specifically online articles. The goal is to develop a system that will predict the popularity of an online article prior to its publication. News articles from the Emerce website[1] will be used and their popularity will be measured by using the number of views they receive. Being able to assess in advance whether an article will be popular or not provides important insight when choosing the content to be published. For editors, knowing which articles are the most promising, will allow them to make better decisions during the editorial stage. Content creators can improve their articles and customize them, if they know how they will appeal to their audience. Finally, this prior knowledge can also be utilized for more technical purposes, e.g. making the most promising content readily available for mobile devices, which are heavily affected by connectivity issues and network strength fluctuations.

Online content presents great variety, from blog and news articles to social media posts to videos. Being able to predict the most prominent content provides publishers with a competitive advantage as they can successfully provide their audience with desirable content. Additionally, given that most companies invest heavily in online marketing [23], detecting the most promising content early allows for efficient and effective planning of advertising campaigns. Popularity prediction can also contribute to the efficiency of content distribution in the net (example of active research by Mehrizi et al. in [30]).

One of the main areas of research for popularity prediction focuses on online articles. Different

[1] `https://www.emerce.nl/`

metrics have been used to measure popularity and a variety of predictors has been employed to allow statistical and machine learning models to predict the popularity of online articles. The approaches to this problem can be broadly separated into two categories, namely post-publication and cold-start prediction. The former category employs early publication information to predict the future or eventual popularity of the content ([54, 50, 20, 1]) while the latter relies solely on features extracted from the content and possible metadata of the involved parties, for example publisher, author or publication time. Cold-start predictions have mainly been addressed in the context of recommender systems (for example in [28, 44, 53, 57]). However this thesis deals with the cold-start popularity prediction of online news articles, following the works of [3, 2, 15].

From the aforementioned approaches, only [15] employed word embeddings to represent the titles and texts of the articles. We aim to expand on that approach by employing different embedding techniques to encode the article information in a meaningful way. The first is to encode the articles using word-level embeddings based on Word2Vector [31] and use their sum, weighted sum or weighted average to represent the full article. The second technique encodes texts in their entirety using Paragraph Vector, also known as Document2Vector [26], an extension of Word2Vector for embedding larger structures of text, like sentences, paragraphs or even documents in their entirety.

As the effectiveness of such encoding methods has not yet been extensively studied in the context of cold-start popularity prediction, the models that will be developed in this thesis will serve as proof of concept for their applicability on this task. For that reason the popularity problem will be tackled both as a classification task (predicting popular or not popular) and a regression task (predicting the exact number of views). The focus will be on simpler models, namely Logistic Regression, Support Vector Machine classification and regression and different Linear Regression models.

More specifically this project aims to answer the question:

Can we predict the popularity of an online article, before its publication, based solely on text-generated features?

Which can be decomposed into the following sub-questions:

- How can we assess whether an article is popular or not?
- Which text encoding method performs the best in predicting an article's popularity?
- Which predictive model is the most appropriate for this task?

This thesis focuses on two main aspects of the popularity prediction problem. The first is the informative encoding of the article texts using the most fitting methods. The second is the selection and tuning of the most appropriate predictive models. Examining these two aspects extensively will provide a sufficient outlook in the applicability of the selected methods on the cold-start popularity prediction problem and will support possible future extensions employing more complex methods.

The structure of the thesis is as follows. Chapter 2 provides the theoretical background on the encoding methods applied in this project and Chapter 3 explains the details behind the methods used for popularity prediction. Chapter 4 provides a detailed overview of the configurations for the methods and experiments. Chapter 5 presents the results of the experiments which are finally discussed in Chapter 6 in which also a conclusion of the whole project is provided.

Chapter 2

Language Representation

One of the major difficulties encountered in Natural Language Processing (NLP) tasks is the representation of the input. Statistical or machine learning models require informative representations of linguistic components like words or sentences in order to produce meaningful results. This is achieved by transforming words to a numeric format that can be mapped to a continuous vector space by a vector space model (VSM) [40]. Numerous methods have been developed to that end, each utilizing a different aspect of the text and encapsulating it into vector form. In this section popular encoding methods are discussed, both on word and on document level.

2.1 Count Vectorization

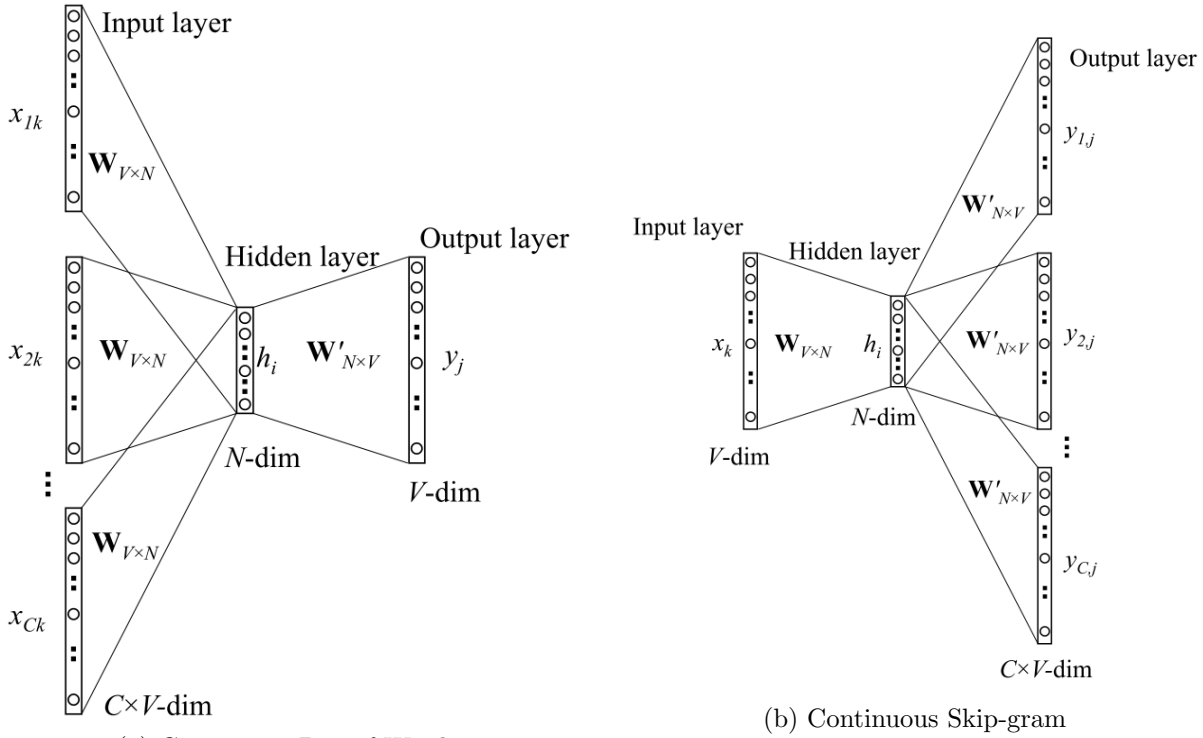
A simple and frequently followed approach in NLP is to encode a word occurring in a text using a vocabulary-long vector, whose elements are all zero, apart from the position of the word in the vocabulary that is one. This method, although very intuitive, can be highly inefficient in big datasets as it leads to increasingly sparser representations as the vocabulary increases. In this case, a complete document would be represented by the counts of each of its words, in a Bag of Words (BoW) manner. An interesting extension uses the term frequency-inverse document frequency (tf-idf) [29, 18] of the words to represent a complete text, thus transitioning from simple counts to informative indexes of the importance of a word for a specific document within a given corpus.

However the representation still has significant disadvantages. Regardless of the meaning of the numeric used, words are still encoded into sparse representations and the BoW approach disregards any syntactic and semantic structure of the text, thus failing to encapsulate important information. Nevertheless BoW approaches are still popular and are applied not only on text-related tasks but also on image recognition and classification [49, 33, 35, 41].

2.2 Word Embeddings

Word embeddings were introduced as a more efficient alternative for word encoding. Instead of using sparse representations, the words are embedded into a continuous vector space, designed to encapsulate semantic relationships of words. Word embeddings can be broadly grouped into co-occurrence counts models and context-predicting models [25]. Count models form the initial word embeddings using raw co-occurrence counts and then move on to perform a series of transformations on them, for

example re-weighting co-occurrence counts to improve informativeness and dimensionality reduction for smoothing. Prediction models however directly adjust the vector weights to maximize the probability of predicting a word's context, without first collecting context vectors. This approach causes words occurring in similar contexts to have similar vectors. Another significant difference between the two is that prediction models train vectors in a supervised manner while count models do so, mostly, in an unsupervised manner. The two styles have been extensively evaluated by by Lavelli et. al in [25] and Baroni et al. in [4]. In this thesis context-predicting methods will be used and thus will be further described in this section.



(a) Continuous Bag of Words

(b) Continuous Skip-gram

Figure 2.1: Architecture illustration of Word2Vec as introduced in [31], taken from [43]

2.2.1 Word2Vector

Word2Vector is an implementation of the algorithms proposed by Mikolov et al. in [31]. It was developed by Mikolov and his team during their employment at Google. It is a tool that provides an efficient alternative to its contemporary methods in generating word embeddings, based on context-predicting methods. It consists of two architectures for word representation, the Continuous Bag of Words (CBOW) and the Continuous Skip-gram (CSG), both seen in figure 2.1. The idea behind these two architectures was to reduce the computational complexity by removing the non-linear hidden layer that was used in Neural Network Language Models (NNLM) [5].

Continuous Bag of Words

The CBOW architecture's aim is to predict a target word w_t from its surrounding C context words w_I. A window is defined to select words preceding and following the target word. The context words are transformed to one-hot encoded vectors $x_k \in \mathbb{R}^V$, where V is the length of the vocabulary. These vectors constitute the network's input. They are averaged and then multiplied with a weight matrix $W \in \mathbb{R}^{V \times N}$, where N is the selected dimensionality of the embeddings, to produce the hidden layer output $h \in \mathbb{R}^N$:

$$h = \frac{1}{C} \sum_{i=1}^{C} W_{V \times N}^T x_i \tag{2.1}$$

The rows w_i of the weight matrix are the word embeddings v_{w_i} of the words in the vocabulary. Then to calculate the scores u_j for each word, the hidden layer output is multiplied with w'_j, where w'_j is the j-th column of the second weight matrix $W' \in \mathbb{R}^{N \times V}$.

$$u_j = {w'}_j^T h \tag{2.2}$$

Finally the scores are passed through a softmax function to produce the network output, the word posterior distribution.

$$p(w_t|w_I) = \frac{\exp(u_j)}{\sum_{j'=1}^{V} \exp(u_{j'})} \tag{2.3}$$

The purpose of training this network is to find the weights that maximize the likelihood that the target output word w_t occurs within the given word context w_I. The loss function to achieve this is based on the log probabilities of the softmax:

$$E = u_{j^*} - \log \sum_{j'=1}^{V} exp(u_{j'}) \tag{2.4}$$

where $E = -\log p(w_t|w_I)$ is the loss function to minimize and j^* is the target word in the output layer. The training process is carried out using stochastic gradient descent and back-propagation. The weights of the hidden-to-output matrix will be adjusted, based on the partial derivatives of the loss function on the weight matrix W':

$$\begin{aligned} {w'}_{ij}^{new} &= {w'}_{ij}^{old} - \eta \frac{\partial E}{\partial {w'}_{ij}^{old}} \\ &= {w'}_{ij}^{old} - \eta \frac{\partial E}{\partial u_j} \frac{\partial u_j}{\partial {w'}_{ij}^{old}} \\ &= {w'}_{ij}^{old} - \eta \cdot (y_j - t_j) \cdot h_i \end{aligned} \tag{2.5}$$

where $\eta \in [0, 1]$ is the learning rate and t_j is 1 if $j^* = j$ and 0 otherwise. Having updated the values for W', the first weight matrix constituting the word embeddings is updated. First the derivative of

E to the hidden layer is calculated:

$$\begin{aligned}\frac{\partial E}{\partial h_i} &= \sum_{j'=1}^{V} \frac{\partial E}{\partial u_j}\frac{\partial u_j}{\partial h_i} \\ &= \sum_{j'=1}^{V} (y_j - t_j) \cdot w'_{ij} := EH_i \end{aligned} \tag{2.6}$$

where H_i is the i-th hidden unit in the hidden layer and $EH_i \in \mathbb{R}^N$ is the sum of the output vectors of all words in the vocabulary, weighted by their prediction error: $(y_j - t_j)$. Then following the chain rule, the derivative of E for each element of W can be calculated:

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial h_i}\frac{\partial h_i}{\partial w_{ki}} = EH_i \cdot x_k = xEH^T \tag{2.7}$$

Finally the embeddings are updated. Since only one element of x is non-zero, it is excluded and only the embeddings corresponding to the context words $v_{w_{I,c}}$ are updated:

$$v^{new}_{w_{I,c}} = v^{old}_{w_{I,c}} - \frac{1}{C} \cdot \eta \cdot EH^T \tag{2.8}$$

where $v_{w_{I,c}}$ is the row of W that had a non-zero derivative and thus is the only one updated at that step.

Continuous Skip-gram

The continuous skip-gram architecture aims to predict context words w_I based on a target word w_t. In essence CSG is the opposite of CBOW. In this case, since there is one input word, the input-to-hidden layer practically copies the row (embedding) corresponding to this word from the weight matrix W. The forward propagation is carried out as in CBOW, the difference being that the output is C word posterior probabilities, one for each of the context words. Each of the outputs is calculated using the same W' matrix:

$$y_{c,j} = p(w_{c,j} = w_{O,c}|w_t) = \frac{\exp(u_{c,j})}{\sum_{j'=1}^{V}\exp(u_{c,j'})} \tag{2.9}$$

The loss function along with the update function for the hidden-to-output weight matrix is adjusted to accommodate the C different outputs:

$$\begin{aligned} E &= -\log p(w_{O,1}, w_{O,2}, ..., w_{O,C}|w_t) \\ &= -\log \prod_{c=1}^{C} \frac{\exp(u_{c,j})}{\sum_{j'=1}^{V}\exp(u_{c,j'})} \\ &= -\sum_{c=1}^{C}(u_{j^*_c} + C \cdot \log \sum_{j'=1}^{V}\exp(u_{c,j'})) \end{aligned} \tag{2.10}$$

$$w'^{(new)}_{ij} = w'^{(old)}_{ij} - \eta \cdot \sum_{c=1}^{C}(y_{c,j} - t_{c,j}) \cdot h_{c,i} \tag{2.11}$$

The update function for the input-to-hidden matrix remains the same as in (2.8).

Negative Sampling

The CSG architecture of Word2Vector trains embeddings so as to maximize the probability of a context words w_I given a target word w_t, namely $P(w_I|w_t)$. But this process takes into account the probabilities of all the other words in the vocabulary, leading to a high number of computations. Mikolov et al. [32] introduced negative sampling, a simplified version of Noise Contrastive Estimation (NCE)[16], to optimize the training process. In that case only the probability $P(w_I|w_t)$ is maximized while the probability of a selected number of negative samples is minimized.

Following [43], using negative sampling produces the following error function:

$$E = -\log \sigma({v'}_{w_O}^T h) - \sum_{w_j \in W_{neg}} \log \sigma(-{v'}_{w_j}^T h) \tag{2.12}$$

where σ represents the softmax function, W_{neg} is the set of negative sample words, ${v'}_{w_\cdot}$ is the embedding of a word $w_\cdot$ and h is the input-to-hidden output. Negative Sampling can also be applied in the CBOW architecture, given the appropriate modifications.

Sub-sampling Frequent Words

Frequent words in big datasets usually have a negative impact on the prediction task of language models. Therefore techniques like stop-word removal is a widely used pre-processing step. Another approach to the same issue in the case of Word2Vector is the sub-sampling of frequent words. As presented in [32], sub-sampling is used to counter the unavoidable imbalance between frequent and infrequent words in a dataset. The probability of a word being sub-sampled is determined by:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \tag{2.13}$$

where $f(w_i)$ is the frequency of word w_i and t the chosen threshold, which is usually 10^{-5}. Words with frequency higher than the chosen threshold t will be sub-sampled and the frequency ordering will be preserved. Even though the formula was decided heuristically, it has been shown to improve both learning times and accuracy of rare-word embeddings [32].

2.3 Averaging Word Embeddings

Given a successful embedding of words, the next step is to encode complete text structures, for example a paragraph. One of the most simplistic yet quite broadly used approach is that of summing or averaging word embeddings in a BoW manner. This approach is mainly used as a baseline in different cases such as in [22, 10, 55, 46, 17].

Using averaging, a paragraph embedding t_i would be calculated by averaging all of its word embeddings $w_j, j = 1...N_i$, where N_i is the number of words in the paragraph:

$$t_i = \frac{1}{N_i} \sum_{j=1}^{N_i} w_j \tag{2.14}$$

Although intuitive and simple to compute, such paragraph embeddings fail to capture important aspects of the text. As in any BoW approach, the syntactic and semantic structure from a text

is lost leading to loss of important information. Therefore other embedding techniques have been developed in an attempt to properly represent the meaning of a text structure larger than a word.

2.4 Document2Vector

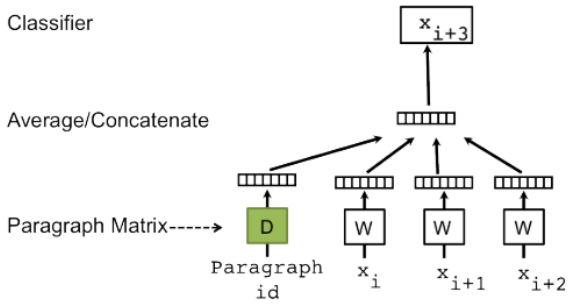
Many sophisticated methods have been developed to generate elaborate document embeddings, like Skip-through Vectors [21], Sent2Vector [34] and Paragraph Vector [26] which is the one applied in this thesis. Le and Mikolov [26] further explored the embeddings of texts and developed the Paragraph Vector, a method to encode texts of variable length, like sentences, paragraphs or even documents in their entirety. In their work Le and Mikolov apply their method on paragraph level, encoding each paragraph of a document separately. Paragraph Vector consists of two architectures, the Distributed Memory model of Paragraph Vector (PV-DM) and the Distributed Bag of Words model of Paragraph Vector (PV-DBOW) which follow the principles of the Bag of Words and Skip-gram models respectively, as seen in [31].

PV-DM, seen in figure 2.2a, is similar to Word2Vec's CBOW model, the main difference being that alongside the words' weight matrix W, a second weight matrix $D \in \mathbb{R}^{UxM}$ is used to encode the paragraphs. U is the number of paragraphs in the dataset and M is the selected dimensionality of the paragraph embeddings. Thus every paragraph is mapped to a row d_j of the paragraph weight matrix D, which are unique, real-valued vectors and represent the paragraph embeddings t_j. The input of the network is a concatenation of the context words' and the paragraph's embeddings. Word embeddings are shared across all paragraphs in the dataset but paragraph embeddings are shared only for context generated from the same paragraph and not across the whole dataset. The embeddings of the paragraphs are randomly initialized at the beginning of the training process, like the word embedding are, and their final values are determined during the training process.

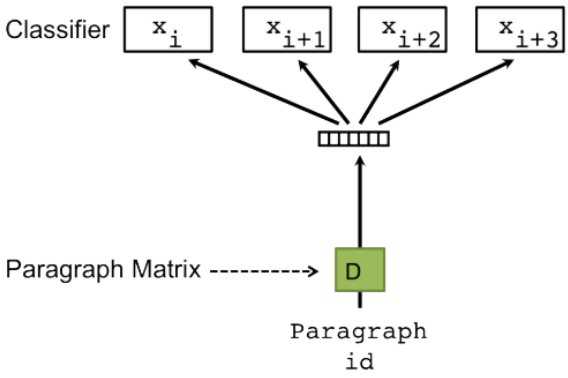
The training is carried out like in Word2Vec's CBOW, as explained in Section 2.2.1. The difference is that in addition to the word matrix update, the paragraph embeddings are also updated using back propagation, similar to equation 2.8:

$$t_{d_{I,c}}^{new} = t_{d_{I,c}}^{old} - \frac{1}{C} \cdot \eta \cdot EH^T \tag{2.15}$$

where $t_{d_{I,c}}$ is the row of D that had a non-zero derivative and therefore is the only one updated at that step. Thus, even if the paragraph matrix is large, the updates on the matrix are sparse, allowing the training process to maintain time and calculation efficiency.



(a) Paragraph Vector - Distributed Memory. Input consists of a concatenation among the word embeddings and the embedding of paragraph they belong to.



(b) Paragraph Vector - Distributed Bag of Words. Input is simply the embedding of the paragraph at question.

Figure 2.2: Architecture illustration of Paragraph vector as introduced in [26], taken from [10]

PV-DBOW, seen in figure 2.2b, follows the notion of the Skip-gram model and aims to maximize the probability distribution over the words that belong to a paragraph given its embedding. The main advantage of this approach, as stated by the creators, is that it allows for faster training as only the weights of the output layer need to be stored during the process, contrary to both output layer weights weights and word embeddings in the case of PV-DM.

Le and Mikolov utilize both architectures and combine the generated paragraph embeddings for their experiments.

Chapter 3

Models

In this section the models used to tackle the popularity prediction problem are discussed. Popularity prediction is handled both as a classification and as a regression task and therefore the following subsections describe models of both categories.

3.1 Classification

In machine learning, classification is the problem of correctly identifying the category an instance belongs to, based on previously observed instances whose category is known. This choice is made based on a set of explanatory variables called features. These features can be categorical (country of residence), ordinal (clothing size) or numerical (weight, number of page views). Another way to decide the category an instance belongs to is by measuring its distance from or similarity to previously observed instances. The algorithms that implement classification are called classifiers.

3.1.1 Logistic Regression

Logistic regression is a statistical method used to model a binary dependent variable, using one or more independent variables. The term was first coined by Berkson in [6]. Given that the two classes of the dependent variable are 0 (negative) and 1 (positive), logistic regression yields the probability that an instance belongs to the positive class. This probability is given using a logistic/sigmoid function. Classification is carried out by placing a cutoff point on the yielded probability to decide which class an instance is more likely to belong to. The exact value of the cutoff point can be decided based on the cost of false positives against false negatives, but for a balanced classification task a value of 0.5 is usually chosen.

In the multivariate case, a linear approximation of the independent variables is used and the regression function has the following form:

$$\begin{aligned} \beta^T X &= \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n, \\ h_\beta(x) &= \frac{1}{1 + e^{-\beta^T X}} \end{aligned} \tag{3.1}$$

In order to find the best set of parameters β, a cost function needs to be optimized. Simple Mean Squared Error (MSE) would not be the optimal option in this scenario as the exponent makes

equation 3.1 non-convex, leading to many local optima. Thus a global optimal solution would not be guaranteed using methods like Stochastic Gradient Descent (SGD). Therefore the Logistic Loss is used, to formulate a convex cost function that guarantees an optimal solution:

$$J(\beta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\beta(x_i), y_i) \tag{3.2}$$

where m is the number of instances used in this update step, $h_\beta(x_i)$ is the predicted class for instance i, y_i is the actual class of instance i and Cost is defined in equation 3.3:

$$\begin{aligned} Cost(h_\beta(x_i), y_i) &= \begin{cases} -\log(h_\beta(x_i)), & \text{if } y_i = 1 \\ -\log(1 - h_\beta(x_i)), & \text{if } y_i = 0 \end{cases} \Rightarrow \\ Cost(h_\beta(x_i), y_i) &= -y_i \log(h_\beta(x_i)) - (1 - y_i)\log(1 - h_\beta(x_i)) \end{aligned} \tag{3.3}$$

Finally, after substituting equation 3.3 in equation 3.2:

$$J(\beta) = \frac{1}{m}\sum_{i=1}^{m}[-y_i \log(h_\beta(x_i)) - (1 - y_i)\log(1 - h_\beta(x_i))] \tag{3.4}$$

The now convex equation 3.4 is the final cost function that is minimized with SGD, in order to find the optimal coefficients β. The derivative of 3.4 is:

$$\frac{\partial J}{\partial \beta} = \frac{1}{m}\sum_{i=1}^{m}[(h_\beta(x_i) - y_i)x_i] \tag{3.5}$$

and thus an update step of the coefficients can be calculated as follows:

$$\begin{aligned} \beta^{t+1} &= \beta^t - \alpha\frac{\partial J}{\partial \beta} \Rightarrow \\ \beta^{t+1} &= \beta^t - \alpha\frac{1}{m}\sum_{i=1}^{m}[(h_\beta(x_i) - y_i)x_i] \end{aligned} \tag{3.6}$$

3.1.2 Support Vector Machines

Support Vector Machines (SVMs) [9] are supervised models that, based on a training set, classify instances into one of two classes. The purpose of SVMs is to linearly separate the two classes with the largest possible gap between them, by calculating the maximum-margin hyper-plane. This hyper-plane is calculated using only the most difficult-to-discern cases among the instances of the two classes, namely the support vectors, as seen in figure 3.1[1].

[1]Larhmam (`https://commons.wikimedia.org/wiki/File:SVM_margin.png`), `https://creativecommons.org/licenses/by-sa/4.0/legalcode`

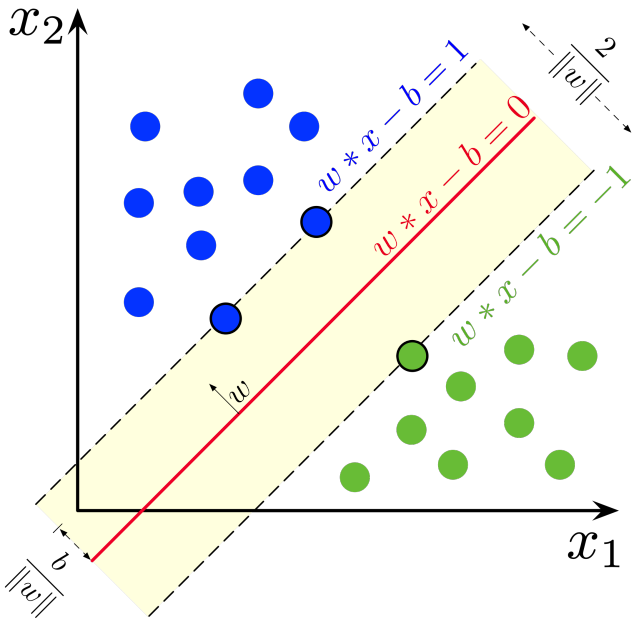


Figure 3.1: Support Vector Machine

Given that the problem is linearly separable, a plane in the form of $wx - b = 0$ requires that the two classes are clearly separated by a margin, spreading out on either side of the plane. The boundaries of the margin are given by the equations:

$$wx_i - b = \begin{cases} 1, \text{instances of class labeled 1} \\ -1, \text{instances of class labeled -1} \end{cases} \tag{3.7}$$

To constrain instances from landing in the margin, the decision boundary is defined as:

$$wx_i - b = \begin{cases} \geq 1, \text{for } y_i = 1 \\ \leq -1, \text{for } y_i = -1 \end{cases} \quad \Rightarrow y_i(wx_i - b) \geq 1 \tag{3.8}$$

Finally the purpose is to maximize the margin, which is equal to $\frac{2}{||w||}$, or in other words, to minimize $||w||$ and to accomplish this, $\frac{1}{2}||w||^2$ is minimized, subject to equation 3.8. This leads to the following function in Lagrangian form:

$$L = \frac{1}{2}||w||^2 + \sum_{i=1}^{n} \lambda_i (1 - y_i(wx_i - b)) \tag{3.9}$$

However many problems are not linearly separable. In order to accommodate for that, SVMs use two methods: the soft-margin variant and the kernel trick. With the soft-margin the SVM allows for a degree of misclassification among the instances. Thus by introducing a penalty factor ξ_i for misclassified instances the constraint in equation 3.8 becomes:

$$y_i(wx_i - b) \geq 1 - \xi_i \tag{3.10}$$

and the function to minimize becomes:

$$L = \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\xi_i + \sum_{i=1}^{n}\lambda_i(1 - \xi_i - y_i(wx_i - b)) \tag{3.11}$$

where C is the factor that determines the trade-off between maximum margin around the hyper-plane and allowed level of misclassification.

The kernel trick is a more elaborate alternative, as it maps the instances in a higher-dimensional space where they can be linearly separated. This mapping is achieved by applying kernel functions on the data. Complicated, non-linear hyper-planes with optimal margins can be calculated in this new feature space. The most often used functions include the polynomial and the radial-basis function kernels.

The aforementioned methods refer to binary classification tasks. However SVM functionality can be extended to multi-class classification tasks. There are two approaches that rely on training multiple binary classifiers to that end. The first is called one-versus-all (OVA), where each class is discerned from all the others, and the second is called one-versus-one (OVO), where all classes are differentiated in a pair-wise fashion.

3.2 Regression

Regression analysis represents a group of statistical methods that aim to model and analyze the relations between at least one independent variable and a dependent variable, also referred to as predictor. In its most common application, regression analysis evaluates the average value of the dependent value given that all other independent variables are fixed. This is carried out by an estimated function of the independent variables called regression function. The different methods that have been developed can be organized into two categories, parametric and non-parametric. It can be used for prediction and forecasting but also for variable selection, as it explores the relation between each independent variable and the predictor.

In the scope of this thesis regression models will be employed to face popularity prediction as a regression problem, by predicting the number of views an article has accumulated. Starting from simple solutions like linear regression and leading up to support vector regression, we explore which model can best fit the data and predict their popularity.

3.2.1 Linear Regression

Linear regression models are used to estimate a linear approximation between a dependent variable and one (simple linear regression) or more (multiple linear regression) independent variables. It is assumed that there is a linear relationship between the dependent and independent variables. To model this relationship a function of the following form is estimated:

$$y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + ... + \beta_p x_{i,p} + \epsilon_i \tag{3.12}$$

where y_i is the dependent variable, $x_{i,j}$ are the independent variables as observed in the data and ϵ_i is the error variable, a random variable that adds noise to the relation between the variables.

Different methods have been developed to estimate the parameters of a linear model. Often Ordinary Least Squares (OLS) estimation is used, either as is or with slight variations by penalizing

the cost function, examples being ridge and lasso regression. The term Least Squares is credited to Carl Friedrich Gauss [8] and first published by Adrien-Marie Legendre [48].

In ordinary Least Squares the aim is to estimate the best parameters β by minimizing the sum of squares of the residuals:

$$L(\beta) = \sum_{i=1}^{n} [y_i - f(x_i, \beta)]^2 \tag{3.13}$$

where y_i is the true value of instance i and $f(x_i, \beta)$ is the linear function of input instance x_i with parameters $\beta \in \mathbb{R}^m$, with m being the number of features for each instance i. Since equation 3.13 is convex, a global minimum can be found with zero gradient.

Although simple and intuitive, the OLS approach is sensitive to outliers that can heavily influence the residuals and can not handle cases where multicollinearity occurs. To counter these issues penalized versions of the OLS have been developed, namely ridge and lasso regressions. Both variations minimize a penalized version of OLS cost function. Ridge regression [52, 37], also known as Tikhonov regularization or weight decay, is used to apply regularization on ill-posed problems. Its main aim is to counter the problem of multicollinearity that is encountered in models that make use of a large number of parameters. LASSO [51] is an acronym for Least Absolute Shrinkage and Selection Operator and is a method applied for variable selection and regularization. Its aim is to boost the accuracy of the models it is applied on.

3.2.2 Support Vector Regression

Support vector machines can also be used for regression, called Support Vector Regression (SVR) [12]. The main principles of SVMs are maintained, namely support vectors and maximal margin. However given the continuous nature of the target values, a precise prediction would be impossible. In essence the aim of an SVR model is to fit all the prediction errors within a specific margin and thus estimating a hyper-plane. Similar to SVM the aim is to minimize the margin $\frac{1}{2}||w||^2$, subject to $-\epsilon \leq y_i - wx_i - b \leq +\epsilon$, where $\pm\epsilon$ represents the limits within which all predictions should fall. Usually slack variables are included to accommodate cases where the hard-margin problem is infeasible.

Chapter 4

Experimental Design

In order to answer the research questions posed in the introduction a number of approaches and factors need to be examined through different experiments. The scenarios and factors we examine are described in this section. First, details on the acquisition of the dataset are given along with a short exploratory analysis, the data preprocessing and dataset enrichment. Second, a description of the different data encoding techniques is provided. Finally, the predictive models and their configurations are presented in detail.

4.1 Dataset

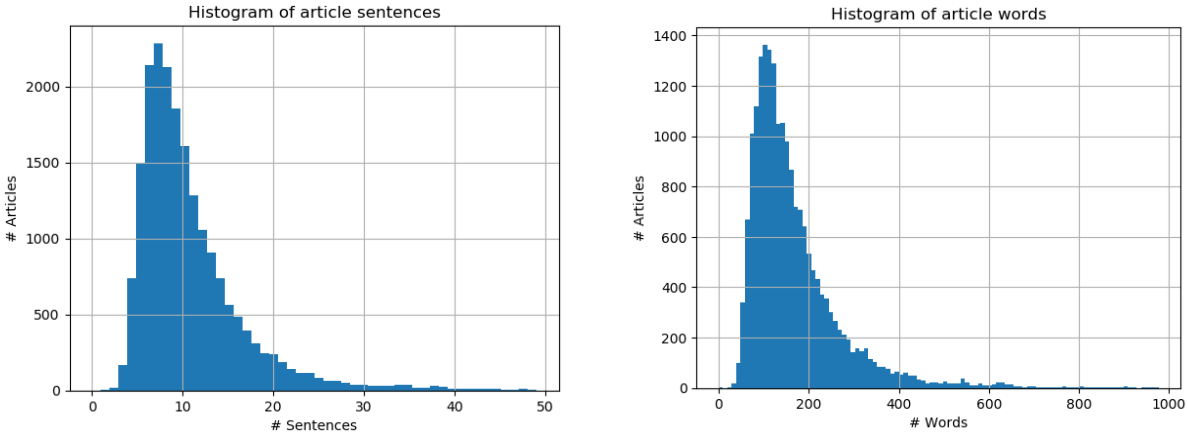
In this subsection the details of the data acquisition, preprocessing and enrichment are presented. Additionally a brief exploratory analysis on the data is provided.

4.1.1 Acquisition and Analysis

The dataset for this project is comprised of articles from the Emerce website, one of the most visited sites for media, marketing and e-business news articles in the Netherlands. The dataset was scraped from the website using a custom web spider developed with Scrapy[1] and includes the article's title and the text, the author, channel/sub-channel and view count, as displayed on the articles' page. The articles are in Dutch, span a period from the 27th of February 2015 until the 6th of June 2019 and were scraped on the 21st of October 2019, to obtain a stabilized view of viewer traffic. In total 20263 articles are scraped and 20056 remained after removing instances with missing values, for example articles that do not include any text but only media, like an informative poster.

The intended audience of the Emerce website is mainly CEOs and administrative personnel from the business world, therefore most of the articles are short and concise. The dataset contains around 3.65 million words in total. The vast majority of the articles consists of no more than 336 words, centered mainly around 181 words, as seen in figure 4.1b. However there is a small number of articles that reach more than 1000 words, with the largest reaching 5602 words. The number of sentences follows a similar pattern (figure 4.1a).

[1] `https://scrapy.org/`



(a) Distribution of sentences in the articles.

(b) Distribution of words in the articles.

Figure 4.1: Histograms of sentence and word counts in the articles of the Emerce dataset. The word count is organized in 100 bins and the sentence count in 50 bins.

Text generated features will be used as independent variables, more specifically embeddings. The dependent variable used to represent popularity is the number of views. This measurement is obtained along with the articles from the Emerce website, as the view count of each article is publicly available. Article views range from 43 to over 320 thousand. An informative overview of the article view distribution can be seen in figure 4.2. Only 66 articles in the dataset have a View count of over 100 thousand and are thus excluded for better legibility of the graph. A full graph is provided in the Appendix.

View count has been previously used as an interpretation of popularity in [19, 2] and although it is rare for a news portal to publicly share this information, in the case of Emerce, it is available. The main downside in using the publicly available view count is the lack of precision. Up until before one thousand views, the exact view count is displayed (e.g. 827 views). However for view counts over one thousand the number is abbreviated in a decimal form in the magnitude of the thousands with precision on the first decimal, meaning that an article with 12983 views will be displayed as '12.9K'.

As seen in figure 4.2 the view count follows a bimodal distribution. This shape will be used to separate the classes for the classification task. After preliminary tests, the best split is in two classes, non-popular or regular and popular articles. The threshold used for the separation of classes is 19000 views. It also becomes apparent that the two classes are highly unbalanced. This issue will be handled in different, subtle ways, through the predictive models, as the different sampling techniques are out of the scope of this thesis. Additionally for the regression task the target values will first be converted into natural logarithmic form. This will allow for faster training and easier hyper-parameter optimization of the regressors and suffer less from outliers.

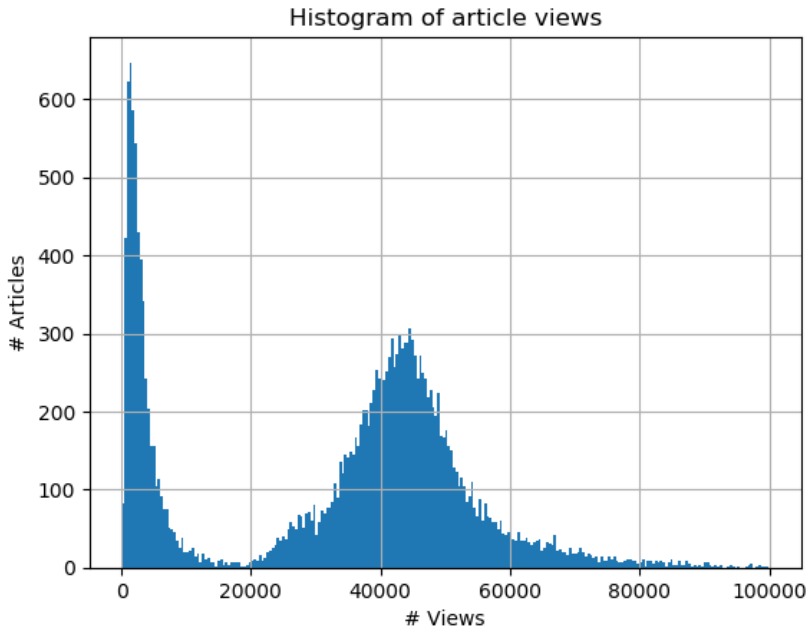


Figure 4.2: Histogram of view counts in the dataset. A natural shape can be seen with a very sharp and narrow normal distribution on the left with a right tail and a wide normal distribution on the right, also with a right tail. The two shapes will represent the two classes of the classification task.

4.1.2 Preprocessing

In NLP, the datasets are usually preprocessed in order to produce informative inputs. The following preprocessing steps are applied in this thesis to prepare the data for encoding. First all punctuation marks, digits and hyperlinks are removed. Given the formal nature of news articles, extensive and varied use of punctuation is not expected and therefore punctuation marks are not included in the vocabulary due to lack of informative strength. Digits and numbers in general could provide some insight or allow the detection of patterns but given the wide range of topics in the dataset, it is considered counter-productive to include numerical characters to the vocabulary. Hyperlinks are removed again due to lack of informativeness and great variation that can be observed for the same hyperlink, possibly deteriorating the quality of the vocabulary.

Another proprocessing step is tokenization to separate word structures in the text, using the *word_tokenize* functionality of Natural Language Toolkit (NLTK) [7]. Furthermore NLTK's list of Dutch stop words is used for stop word removal, to exclude high frequency words that do not contribute significantly to generating informative representations.

4.1.3 Dataset Enrichment

Although the Emerce dataset consists of around 20000 usable articles, the language structures observed in it are expected to be relatively limited compared to the Dutch language's capacity and expressibility, since we deal only with news articles of specific topics. Therefore we enrich the dataset with articles from the Dutch Wikipedia and examine whether the produced embeddings improve prediction quality or not.

The Wikipedia articles were acquired via the Wikipedia dumps[2]. The .xml form of only the articles' text snapshot of the 20th of September 2019[3] was downloaded, so as to save space and processing efforts in extracting the texts. The articles were extracted from the .xml files using the attardi-wikiextractor[4].

The Dutch Wikipedia contains almost two million articles. Since the size is almost a hundred times greater than our Emerce dataset, mixing the two datasets should be done with care. Only a small number of Wikipedia articles will be used to enrich our dataset, so as not to dominate over the Emerce articles. The ratios that will be experimented with are 1:1 and 2:1, in favour of the Emerce dataset. Namely the first case will have the full Emerce dataset and an approximately equal number of Wikipedia articles while the second will only have approximately half as many Wikipedia articles.

4.1.4 Data Encoding

The first stage of the prediction problem pertains the encoding of the data. The different methods used in this thesis are described in detail in this section.

Tf-idf and BoW

The initial encoding of the articles is aimed in forming a comparison base for the embedding techniques that we explore in this project and thus the simple Bag of Words approach is used, in combination with tf-idf as described in Section 2.1. To obtain a vectorized form of the articles, NLTK's *CountVectorizer* is used with a 1500 features in its vocabulary. The words are stemmed and stop words are removed using NLTK's Snowball stemmer [38] for Dutch and NLTK's list of stop Dutch words respectively. Only unigrams are used to form the count vectors, since preliminary tests demonstrated lower overall performance and higher tendency to predict the dominant class when bigrams and/or trigrams were employed. The next step is responsible for the transformation from counts to tf-idf indexes. This is achieved by using Scikit-learn's [36] *TfidfTransformer*, with L2 normalization and smoothing to avoid zero-occurrence words.

Embeddings

The experiments revolve around document-level embeddings. Two approaches are taken, averaged word embeddings to represent a document using Word2Vector and complete document embeddings using Document2Vector. Initially custom-made implementations were employed to generate word and document embeddings. However the performance of the implementations was sub-optimal with regard to processing speed and overall flexibility and embedding quality. Thus gensim (Generate Similar) [42] was selected for the generation of embeddings, since it provides highly efficient, configurable tools both for word- and document-level embeddings.

For the generation of word-level embeddings gensim's *Word2Vec* model is used. The parameters used to train models are inspired by Mikolov's implementation[5] and other popular, pre-trained embeddings [6] and both CSG and CBOW architectures are used. The embedding dimension is 300,

[2] `https://dumps.wikimedia.org/`

[3] `https://ftp.acc.umu.se/mirror/wikimedia.org/dumps/nlwiki/20190920/`

[4] `https://github.com/attardi/wikiextractor`

[5] `https://code.google.com/archive/p/word2vec/`

[6] `http://vectors.nlpl.eu/repository/`, `https://nlp.stanford.edu/projects/glove/`

negative sampling is set to 5 words and the window size is 5 for CSG and 10 for CBOW. Only the number of epochs varies between 5, 10, 20 and 50 epochs. The generated word embeddings are averaged over the number of words for each text to form document embeddings.

For the document-level embeddings gensim's *Doc2Vec* model is used. Different sets of embeddings are generated and the best sets are decided based on rudimental visual inspection and mainly on their performance on the prediction task. Inspiration for the parameters of the embeddings is taken from the original paper on Paragraph Vector by Le and Mikolov [26] and the follow-up work by Lau and Baldwin [24]. Lau and Baldwin propose different parameter settings and their experiments demonstrate the superiority of PV-DBOW, contrary to the results of Le and Mikolov that support PV-DM. For this reason both model architectures are experimented with. Table 4.1 presents the scope of parameter values that will be explored.

Table 4.1: Parameter search space for PV-DM and PV-DBOW. Some parameter values are explored more extensively for PV-DM while others are investigated in more detail for PV-DBOW, based on preliminary performance tests.

Parameter	Values
Embedding dimension	200, 300
Training epochs	100, 250, 300, 400
Window size	5, 15
Negative samples	5
Down-sampling threshold	$1e-6$

4.2 Models

In this subsection the parameters of the predictive models used in the experiments are presented along with the methods employed to find the best performing parameter sets. All parameter tuning processes described in the following parts are carried out using 5-fold cross validation to increase the generalization capability of the models, by avoiding issues like over-fitting and selection bias [14]. We are employing 5-fold cross validation by using sklearn's *GridSearchCV* that allows for cross validation over a grid search.

More particularly the *GridSearchCV* method receives an estimator, a set of parameters to be tested, the evaluation metric of choice and the number of folds for n-fold cross validation. When it is fit to a dataset, is splits the data internally in n different ways using *StratifiedKFold* for classifiers and simple *KFold* for any other kind of estimator. Then the differently parametrized models are trained over the different splits and their performance is compared using the selected evaluation metric. Once the best parameter set is found, a new estimator is trained using that set over the whole dataset that was provided. We employ this method by first splitting the complete dataset into train and test subsets and fitting it only on the former. The best estimator that the method returns is then tested on the test set and that performance is reported.

For classification the dataset is split in 70% train and 30% test subsets. This amounts to 14039 and 6017 articles respectively. We want to focus on the generalization power and reliability of the models and therefore a more challenging set-up for our data is adopted. For the regression tasks, given the difficulty of accurately predicting the number of views over a great range of values, the

data is split in 85% training and 15% test subsets. This translates into 17048 and 3008 articles respectively.

The evaluation metrics used are weighted F1-score for classification and R^2 for regression. The weighted F1-score calculates the metrics for each label and averages them based on the true instances of each label, in order to account for label imbalance.

4.2.1 Classification

For logistic regression the *LogisticRegression* model from sklearn is used. To optimize its performance we experiment with the parameters *solver* and *class_weight*. For the solvers, the choice is between SAG [27], SAGA [11] and LIBLINEAR [13], based on the recommendations given in the sklearn documentation regarding dataset size and dimensionality. *Class_weight* is used to penalize the misclassified examples of each class. By default all classes have equal weights. In this case however we attempt to counter the problem of the imbalanced dataset by penalizing the minority class more. Therefore the weights of the two classes (labeled 0 for not popular and 1 for popular) range from [0:0.8, 1:0.2] to [0:0.2, 1:08] in steps of 0.1.

The *SVC* model from sklearn is used for SVM classification. To optimize its performance a log-scale grid search over diverse values of C and *gamma* is carried out to find the optimal set of parameters. For edge cases a second grid search takes place, where the scope is adjusted accordingly. The parameter values we experiment with are presented in table 4.2. The kernel function used in all cases is the radial basis function, as preparatory analysis showed clearly superior performance to the other options (linear, polynomial and sigmoid). These parameter values are used for all different encoding methods. If any results lead us to exceed these ranges, it will be explicitly mentioned in the results section.

Table 4.2: Parameter search space for SVM classification. **Bold** values represent the first stage of grid search, applied to all cases. Normal values represent the second stage grid search, performed only in the case of an edge value being selected as the best parameter.

Parameter	Values
C	$1e^{-6}, 1e^{-4}, \mathbf{1e^{-2}}, \mathbf{1}, \mathbf{1e^{2}}, 1e^{4}, 1e^{6}$
Gamma	$1e^{-4}, 1e^{-3}, \mathbf{1e^{-2}}, \mathbf{1e^{-1}}, \mathbf{1}, 5, 10, 20, 50$

Additionally we utilize the *class_weight* parameter and set it to "balanced", which adjusts the weights of the classes inversely proportional to the class frequency in the data. This aims to ameliorate the effects of class imbalance during SVM training.

4.2.2 Regression

SKlearn's *Ridge* model is used for Linear Regression, given the high number of parameters we utilize from the embedded articles. Thus we aim to mitigate the issue of possible multicollinearity between the different dimensions of the embeddings. We let the model choose the best *solver* and optimize for the *alpha* parameter among the values $[1e^{-6}, 1e^{-4}, 1e^{-2}, 1]$.

For regression with SVMs the *SVR* model from sklearn is used. The optimization is performed with grid search over *C*, *gamma* and *epsilon*. The values we experiment with are listed in table 4.3.

Table 4.3: Parameter search space for SVM regression. **Bold** values represent the first stage of grid search, applied to all cases. Normal values represent the second stage grid search, performed only in the case of an edge value being selected as the best parameter.

Parameter	Values
C	$1e^{-6}, 1e^{-4}, \mathbf{1e^{-2}}, \mathbf{1}, \mathbf{1e^{2}}, 1e^{4}, 1e^{6}$
Gamma	$1e^{-4}, 1e^{-3}, \mathbf{1e^{-2}}, \mathbf{1e^{-1}}, \mathbf{1}, 5, 10, 20, 50$
Epsilon	$\mathbf{1e^{-2}}, \mathbf{1e^{-1}}, 0.5, \mathbf{1}, 5$

Similarly to SVM classification, further grid search is performed when an edge parameter value is chosen as optimal.

4.2.3 Combined Models

Given the difficulty of accurately predicting the view count through regression, we also experiment with a combination of models. First a binary classifier decides whether an article is popular or not and then each of the two classes are used to train a regressor, whose goal is to predict the view count of the articles in that class specifically. With this attempt we aim to improve the performance of the regressors, since they will be trained on subsets of the data that exhibit smaller spreads of values and will allow them to predict these values more accurately.

To implement this approach we will use the best performing models of classification and regression, as will be shown by our aforementioned experiments. More specifically, the best encoding and classification model will be chosen to carry out the initial distinction of the articles. It will not be trained again but rather the saved model for the classification task will be used. For regression, two new, different regressors will be trained and optimized, specifically for each class. Again in the case of the regressors we will rely heavily on the outcome of the previous models and will experiment only with the most prominent ones.

Chapter 5

Results

In this Chapter the results of the experiments described in Chapter 4 are presented. We have experimented extensively with different parameters both for encoding and predictive models. The results shown here demonstrate the performance of different model combinations.

Regarding the encoding we have three different cases, tf-idf vectorization, averaged Word2Vector word embeddings and Document2Vector document embeddings. A single case for tf-idf is applied, to serve as a baseline. For Word2Vector we employ both CGS and CBOW and train a total of 24 models. Similarly for Document2Vector we employ both PV-DM and PV-DBOW, training a total of 52 different encodings. These numbers also include embeddings trained the enriched datasets. With regard to the predictive models we experiment with Logistic Regression, SVM classification, Linear Regression and SVM Regression.

As our experiments are extensive, we have only included the most prominent or interesting results in the main body of the text. For a complete picture of the results and for reference for our more general remarks, we refer to the complete experiment results in the Appendix.

5.1 Classification Results

This subsection is dedicated to the results of the binary classification approach to the problem of popularity prediction.

5.1.1 Logistic Regression

The results of Logistic Regression based on each encoding method will be presented here. We first present the results for each encoding individually and at the end provide the best performing encoding method for Logistic Regression.

Tf-idf vectorization

First we report on the performance of Logistic Regression that utilizes our baseline encoding, the tf-idf vectorized texts. The parameters of the best predictive model are LIBLINEAR for solver and class weights 0.6 for non-popular and 0.4 for popular and this results in an f1-score of 0.78. The class-specific accuracy is presented in the confusion matrix of figure 5.1. The popular class is quite

accurately predicted while in the case of the non-popular class the accuracy is only slightly better than 50%.

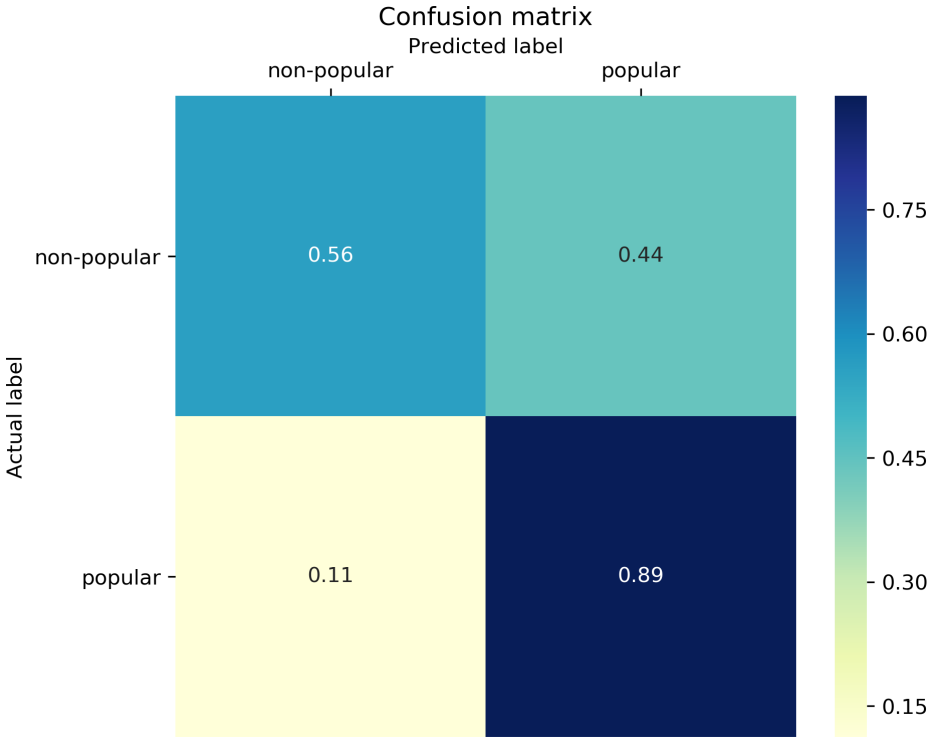


Figure 5.1: Confusion matrix of the best performing Logistic Regression model based on tf-idf vectorized documents.

Word2Vector averaged embeddings

We will now examine the performance of Logistic Regression with averaged Word2Vector embeddings as input. The best performing encoding is produced by the CBOW architecture with a vector dimension of 300, a window size of 10 words, trained for 50 epochs and based on the two-to-one ratio enriched dataset of Emerce and Wikipedia articles. The parameters of the best Logistic Regression model are SAGA for solver and class weights 0.6 for non-popular and 0.4 for popular. The combination of the given encoding and predicting models achieve an f1-score of 0.76.

The best performing models per dataset and architecture along with the overall worst performing models per architecture can be seen in table 5.1. Regarding the encoding options, the first important observation is that the embeddings that are trained longer perform better. Only one of the 6 best performing models were trained for 20 epochs while the remaining 5 were all trained for the maximum of 50 epochs. Furthermore training the same models on the enriched datasets leads to marginally different results with relative increase in accuracy being, at best, 0.4% in the case of CBOW. In total the best performing encodings of CSG and CBOW lead to a 1.14% and 3.2% relative increase in f1-score respectively compared to the worst performing ones.

Table 5.1: Results of Logistic Regression models using Word2Vector averaged embeddings. In **bold** are the best performing models and in *italics* the worst performing models.

Embedding Parameters					**Classifier Parameters**		**F1-Score**
Model	Dataset	Dimension	Window	Epochs	Solver	class weight	
CSG	E	300	5	50	sag	0: 0.6, 1: 0.4	0.757
CSG	E=W	300	5	20	sag	0: 0.6, 1: 0.4	0.756
CSG	E=2W	300	5	50	sag	0: 0.6, 1: 0.4	**0.757**
CSG	E	300	5	5	liblinear	0: 0.6, 1: 0.4	*0.748*
CBOW	E	300	10	50	saga	0: 0.6, 1: 0.4	0.757
CBOW	E=W	300	10	50	sag	0: 0.6, 1: 0.4	0.754
CBOW	E=2W	300	10	50	saga	0: 0.6, 1: 0.4	**0.76**
CBOW	E=W	300	10	5	sag	0: 0.6, 1: 0.4	*0.736*

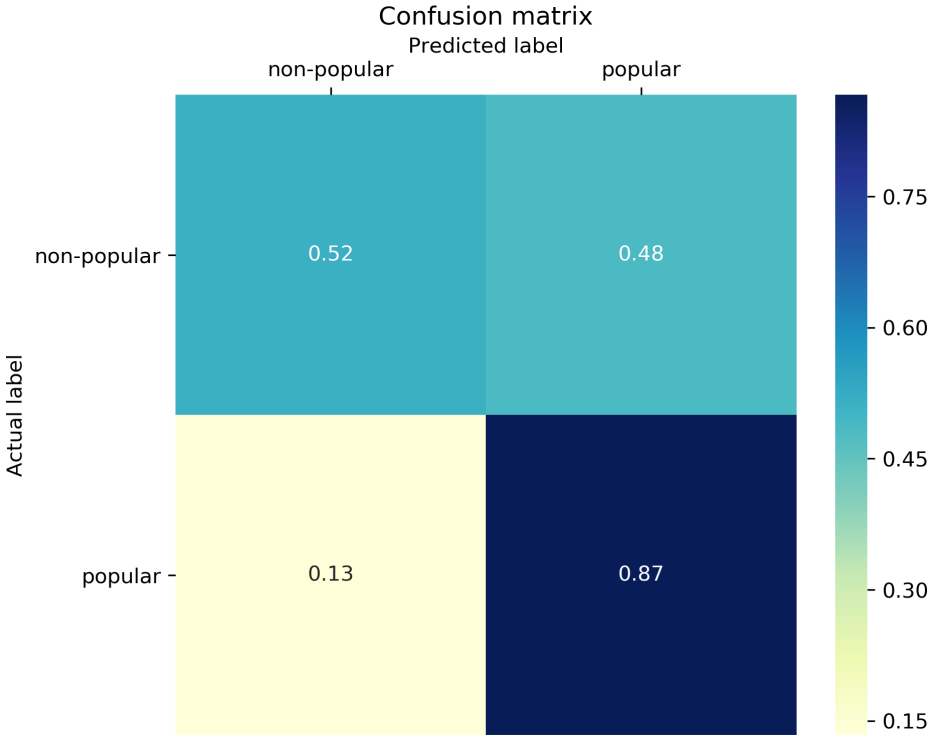


Figure 5.2: Confusion matrix of the best performing Logistic Regression model based on Word2Vector averaged embeddings.

Moving on to the Logistic Regression models themselves, the first observation is that the ideal class weight is 0.6 for non-popular and 0.4 for popular articles, as it is selected by every model. The best performing solver is SAG, selected by 12 of the models, followed by LIBLINEAR with 7 and SAGA with 5. The class-specific accuracy of the best performing model can be seen in the confusion matrix in figure 5.2. We can see that high accuracy is achieved in the majority class however the best Logistic Regression model predicts correctly around 50% of the minority class.

Document2Vector embeddings

Next we address the performance of Logistic regression based on Document2Vector-generated input. The best performing encoding is produced by the PV-DBOW architecture with a vector dimension of 300, a window size of 15 words, trained for 400 epochs and based on the one-to-one ratio enriched dataset of Emerce and Wikipedia articles. The parameters of the best Logistic Regression model are LIBLINEAR for solver and class weights 0.6 for non-popular and 0.4 for popular. The combination of the aforementioned encoding and predicting models achieves an f1-score of 0.777.

The best performing models per dataset and architecture along with the overall worst performing models per architecture can be seen in table 5.2. Regarding the encoding options, we observe again that longer training for embeddings is related to higher performance. Ceteris paribus, all embeddings trained for 100 epochs perform the worst compared to all their respective alternatives. Using selectively trained embeddings on the enriched dataset leads to better performing predictive models in both architectures. The impact of the enriched dataset becomes more apparent in the case of PV-DBOW with a 13.9% relative improvement in performance between the best performing pure and enriched dataset configurations. In the case of PV-DM the relative improvement in performance is just 2.9%. Compared to the worst performing encodings of each architecture we have a relative improvement of 19% for PV-DBOW and 11.6% for PV-DM.

Table 5.2: Results of Logistic Regression models using Document2Vector embeddings. In **bold** are the best performing models and in *italics* the worst performing models.

Embedding Parameters					**Classifier Parameters**		**F1-Score**
Architecture	Dataset	Dimension	Window	Epochs	Solver	class weight	
PV-DBOW	E	200	5	400	liblinear	0: 0.6, 1: 0.4	0.682
	E=W	300	15	400	saga	0: 0.6, 1: 0.4	**0.777**
	E=2W	300	15	300	liblinear	0: 0.6, 1: 0.4	0.775
	E	300	15	100	sag	0: 0.6, 1: 0.4	*0.653*
PV-DM	E	300	5	400	saga	0: 0.6, 1: 0.4	0.746
	E=W	300	5	400	liblinear	0: 0.6, 1: 0.4	**0.764**
	E=2W	300	5	300	liblinear	0: 0.6, 1: 0.4	0.762
	E	200	5	100	liblinear	0: 0.6, 1: 0.4	*0.685*

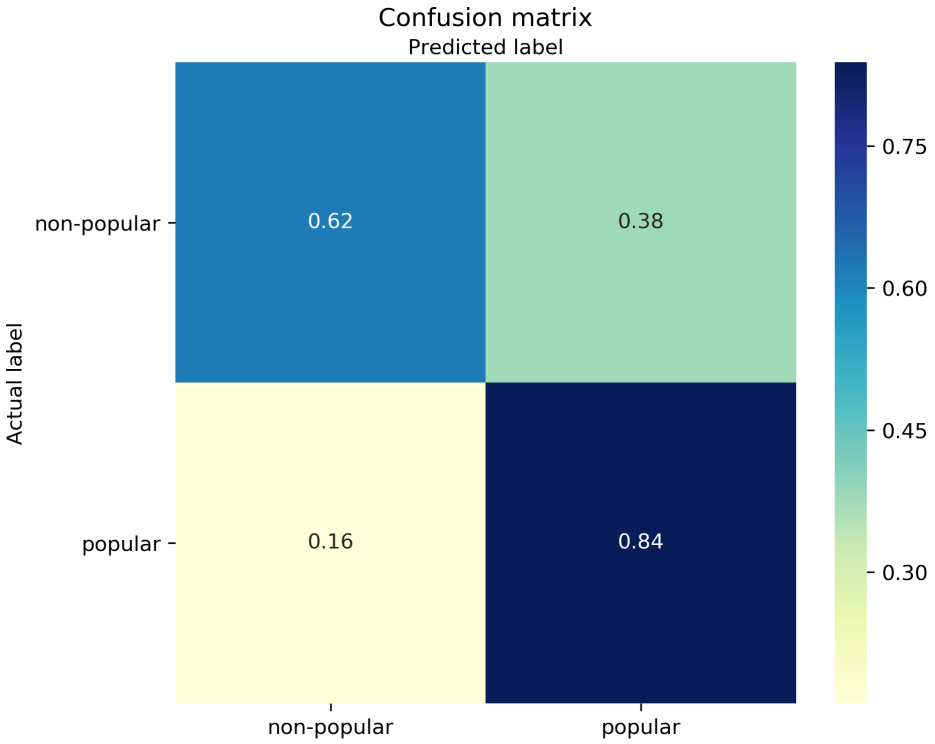


Figure 5.3: Confusion matrix of the best performing Logistic Regression model based on Document2Vector embeddings.

Regarding the predictive models, we observe again that the ideal class weight is 0.6 for non-popular and 0.4 for popular articles, as it is selected by almost every model. In this case 6 models select a different ratio during the hyper-parameter optimization. The choice of solvers during the optimization step shows that each solver is selected equally, namely SAGA and LIBLINEAR 17 times each and SAG 18 times. The class-specific accuracy of the best performing model can be seen in the confusion matrix in figure 5.3. Overall we can see that the predictions are more accurate for the majority class. For the minority class we observe that the misclassification percentage is more than double compared to the majority class but still correctly classifies more than 60% of the instances.

Best performing Encoding for Logistic Regression

The best encoding for Logistic Regression are the Document2Vector-based embeddings. Both architectures perform very well when applied with their best parameters with PV-DBOW showing a marginally better performance. Although the produced f1-score is lower than the one generated with the tf-idf-based encoding, the document embeddings exhibit much better class-wise accuracy, as we discovered by investigating the confusion matrices.

5.1.2 SVM Classification

In this subsection the results of SVM classification based on each encoding method will be presented. We first present the results for each encoding individually and at the end provide the best performing

encoding method for SVM classification.

Tf-idf vectorization

The first results come from investigating the performance of SVM classification that utilizes our baseline encoding, the tf-idf vectorized texts. The parameters of the best SVM are C=1 and gamma=1 and produced an f1-score of 0.78. The class-specific accuracy is presented in the confusion matrix of figure 5.4. The instances of the popular class are correctly predicted with more than 80% accuracy. For the non-popular class the accuracy is almost 70%.

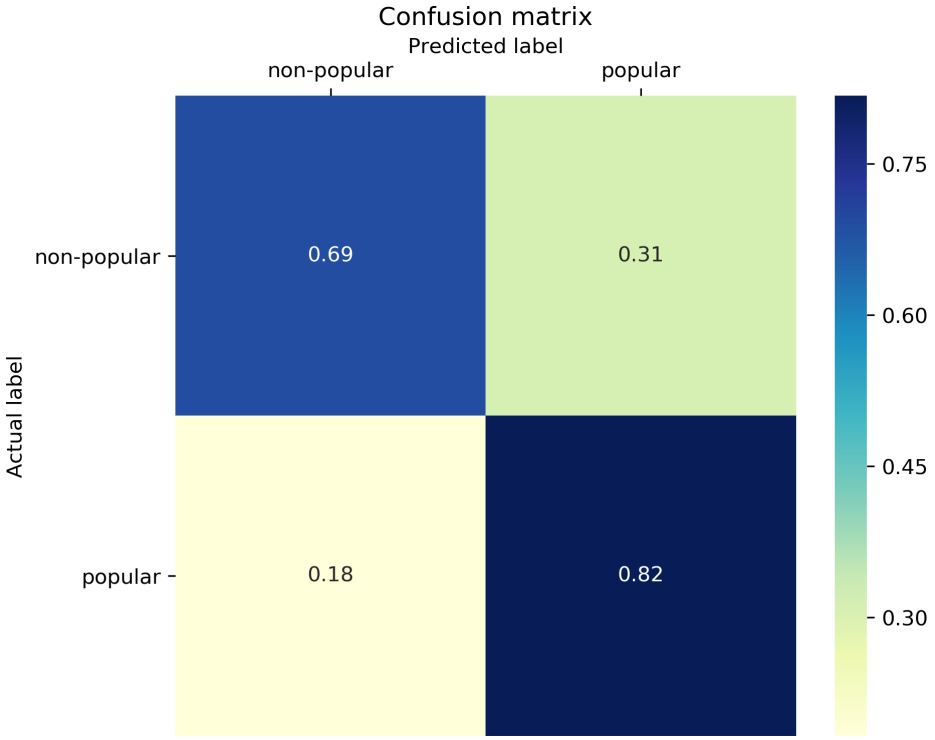


Figure 5.4: Confusion matrix of the best performing SVM classification model based on tf-idf vectorized documents.

Word2Vector averaged embeddings

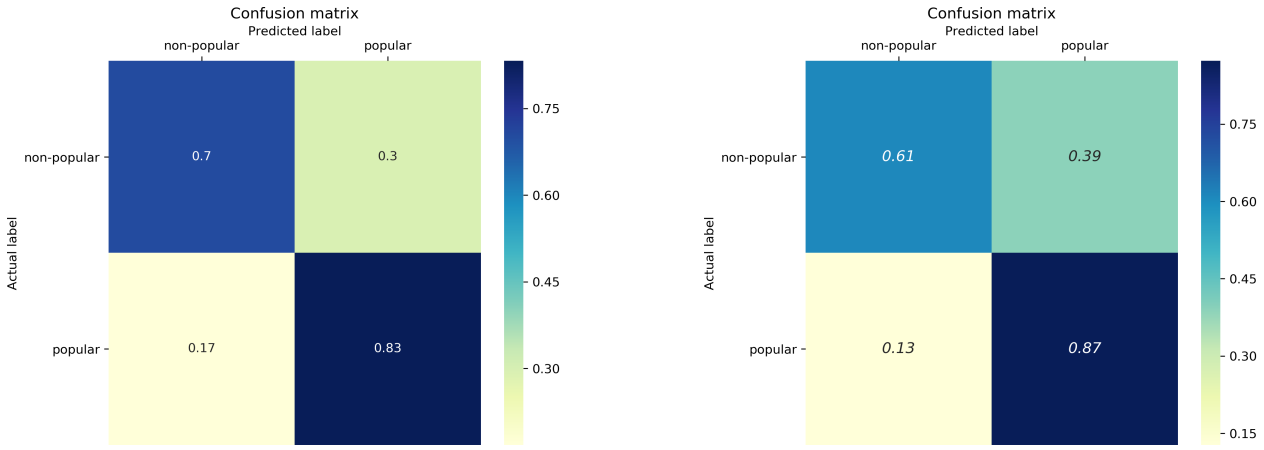
Next we present the results of SVM classification based on Word2Vector averaged embeddings. The best performing encoding is produced by the CSG architecture with a vector dimension of 300, a window size of 5 words, trained for 50 epochs and using only the Emerce dataset. The best SVM parameters are $C = 1$ and $gamma = 1$ and gave an f1-score of 0.799.

An overview of the best and worst performing models is presented in table 5.3. The best performing embeddings are decisively the ones trained for 50 epochs, as they were chosen by every set-up with only one exception. This is further supported by the fact that the embeddings that are trained for 5 epochs displayed the worst performance for both Word2Vector architectures. Enriching the dataset for the training stage of the embeddings did not produce consistently better performing

encodings. For CSG the performance is marginally worse while in CBOW a slight improvement can be observed between the Emerce-only and the enriched datasets. Compared to the worst performing models a relative improvement of 2.8% in performance is achieved in CSG and 6.1% in CBOW.

Table 5.3: Results of SVM classification models using Word2Vector averaged embeddings. In **bold** are the best performing models and in *italics* the worst performing models.

Embedding Parameters					**Classifier Parameters**		**F1-Score**
Model	Dataset	Dimension	Window	Epochs	C	gamma	
CSG	E	300	5	50	1	1	**0.799**
	E=W	300	5	20	1	1	0.793
	E=2W	300	5	50	1	1	0.794
	E	300	5	5	10000	0.01	*0.777*
CBOW	E	300	10	50	1	0.1	0.789
	E=W	300	10	50	1	0.1	**0.794**
	E=2W	300	10	50	1	0.1	0.793
	E=W	300	10	5	1000	0.1	*0.749*



(a) CSG architecture, f1-score=0.799

(b) CBOW architecture, f1-score=0.794

Figure 5.5: Comparison of confusion matrices of the best performing Word2Vector embeddings per architecture.

Regarding the predictive models, we can observe that the ideal value for C is 1, as it is selected by all best performing models. The best gamma value for SVMs trained on CSG embeddings is 1 and on CBOW embeddings is 0.1. The worst performing models of each architecture require very high values of C, showing the need to decrease the margin of the SVM and thus reducing misclassifications on the train set. However the accuracy of the models does not improve considerably as the value C increases. Gamma demonstrates a stable behaviour per architecture with the sole exception being the worst performing model for CSG where its value drops to 0.01. The class-specific accuracy of the best performing model can be seen in the confusion matrix in figure 5.5a. The majority class is predicted correctly with 83% accuracy and the minority class reaches 70% accuracy. Next to it in

figure 5.5b we show the confusion matrix of the second best performing model, based on CBOW. We can see that the marginal difference in f1-score translates into great differences in class-specific accuracy. The accuracy of the majority, popular class increases by 4% but the minority, non-popular class experiences a 9% drop in accuracy. The comparison in figure 5.5 supports our choice of best encoding and predictive model.

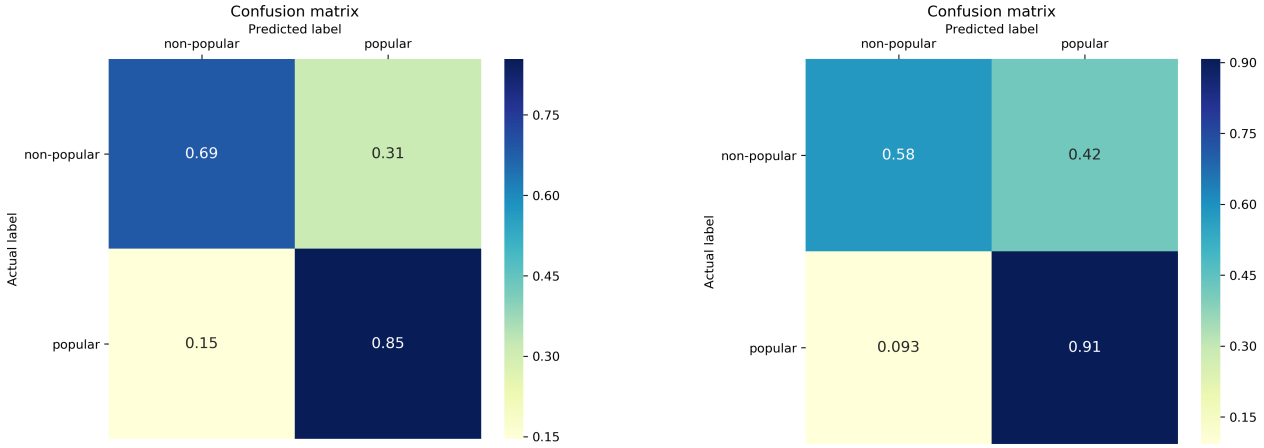
Document2Vector embeddings

The last results of SVM classification are based on Document2Vector-generated input. The best performing embeddings are produced by the PV-DM architecture with a vector dimension of 300, a window size of 5 words, trained for 300 epochs and based on the pure Emerce dataset. The parameters of the best SVM model are $C = 1$ and $gamma = 0.1$. The combination of the given encoding and predictive models gave an f1-score of 0.811.

The best performing models per dataset and architecture along with the overall worst performing models per architecture can be seen in table 5.4. The pattern of longer training for embeddings leading to better performing predictive models is generally observed in this case as well, although continuously higher number of epochs does not always lead to better predictive models. Selectively training embeddings on the enriched dataset leads consistently to better performance for the PV-DBOW architecture. In its case we observe a relative improvement in performance of 17.2% between the best performing pure and enriched dataset configurations. In the case of PV-DM the embeddings trained on the pure dataset lead to better performing SVMs with the best of them performing 3.6% better than the best enriched dataset embeddings. When comparing the best to the worst performing models we see a 28.7% relative improvement in PV-DBOW and a 10% for PV-DM.

Table 5.4: Results of SVM classification models using Document2Vector embeddings. In **bold** are the best performing models and in *italics* the worst performing models.

Embedding Parameters					**Classifier Parameters**		**F1-Score**
Model	Dataset	Dimension	Window	Epochs	C	gamma	
PV-DBOW	E	200	5	400	1	0.1	0.689
	E=W	200	5	300	1	0.1	0.804
	E=2W	200	15	300	1	0.1	**0.808**
	E	300	5	100	1	0.1	*0.628*
PV-DM	E	300	5	300	1	0.1	**0.811**
	E=W	200	5	300	1	0.01	0.783
	E=2W	300	5	300	1	0.01	0.781
	E=W	200	15	300	1	0.1	*0.738*



(a) PV-DM encoding, f1-score=0.811

(b) PV-DBOW encoding, f1-score=0.808

Figure 5.6: Comparison of confusion matrices of the best performing Document2Vector encodings per architecture.

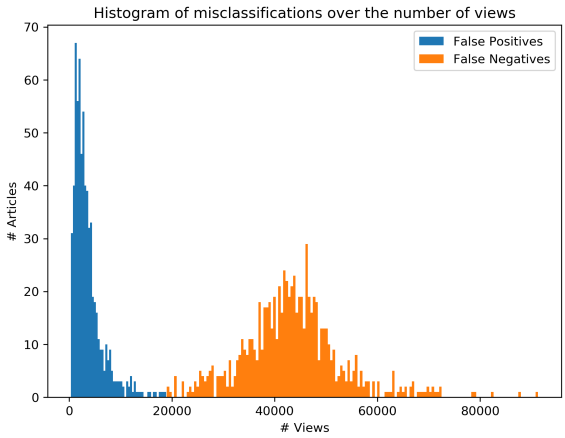
Looking into the SVM configuration in more detail, there is a noteworthy consistency in the selection of parameters. All the best performing models select C=1 and almost all of them gamma=0.1 with two exceptions in PV-DM that select gamma=0.01. Even in the case of the worst performing models the parameter selection is still consistent. The class-specific accuracy of the best predictive model can be seen in the confusion matrix in figure 5.6a. The results show approximately 70% accuracy in the non-popular class and 85% in the popular class. This is also another case where f1-score is not wholly representative of a model's performance. The confusion matrix in figure 5.6b shows the second best performing SVM, which utilizes PV-DBOW-generated embeddings. Although it performs predictions in the popular class with great accuracy, in the order of 91%, it achieves only 58% accuracy in the non-popular class.

Best performing Encoding for SVM classification

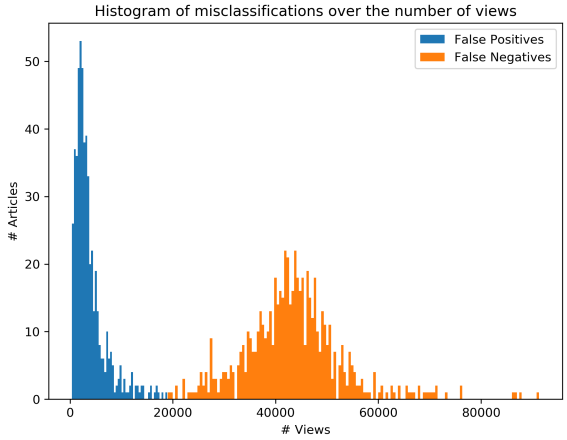
With regard to f1-score, the best performing encoding for SVM classification are the embeddings generated by the PV-DM architecture of Document2Vector. However it is important to point out that this performance is very similar to that of the SVM that is based on Word2Vector averaged embeddings generated from the CSG architecture. The difference in f1-score stems clearly from the size of the two classes, which weights the 4% accuracy increase of the popular class more than the 1% accuracy decrease of the non-popular class. Choosing the best of the two depends on which misclassifications are considered more important in the particular setting that is being examined.

5.1.3 Misclassifications

In the two histograms of figure 5.7 we can see the distribution of the misclassified instances over their original view count for the best performing Logistic Regression and SVM. This allows us to investigate the reasons of the misclassifications and estimate whether there is a pattern in the errors of our model or a tendency to misclassify specific instances.



(a) Misclassifications of best Logistic Regression model based on Document2Vector. Out of 6017 articles, 1434 are misclassified.



(b) Misclassifications of best SVM model based on Document2Vector. Out of 6017 articles, 1166 are misclassified.

Figure 5.7: Histograms of false positives and false negatives of the best classifiers per predictive model. On the left is the best Logistic Regression model and on the right the best SVM model. For both histograms the false positives are organized in 50 bins and the false negatives in 150 bins. Misclassifications over 100000 are not shown for clarity.

The histograms are generated from the performance of the classifiers on the same test set which numbers 6017 articles. False positives are the instances that are predicted as popular while they are non-popular and false negatives are the opposite, predicted non-popular while they are popular. Both kinds of misclassifications follow the distribution of their class, as we saw in the graph of the view distribution in section 4.1. It is not only the articles near the threshold that are misclassified and no other pattern is observed. This happens in the misclassifications of both the best Logistic Regression model and the best SVM model.

5.2 Regression Results

This Section lists the results of the regression approach to the popularity prediction problem. Generally the performance of the regression models was poor, with small variations between the different encoding and predictive models. Therefore we present only a representative overview of the best performing models.

We first establish a baseline for regression using Ridge Regression and the tf-idf vectorized texts. The best model has a value of $\alpha = 1$ and achieves an R^2 score of 0.247. The best performing Word2Vector averaged embeddings in Ridge Regression come from the CSG architecture with a dimension of 300, a window size of 5 words and trained for 50 epochs only on the Emerce dataset. The regressor has a value of $\alpha = 1$ and produced an R^2 score of 0.238. Regarding the Document2Vector embeddings, the best performing model comes from the PV-DBOW architecture with a dimension of 300, a window size of 15 and trained for 400 epochs on the two-to-one enriched dataset of Emerce and Wikipedia articles to achieve $R^2 = 0.267$. Thus Ridge Regression performs the best when it uses Document2Vector embeddings as input.

We then move on to SVM regression. First with tf-idf vectorized texts, the optimal set of parameters for the SVM are $C = 100$, $gamma = 1$ and $epsilon = 0.1$, obtains $R^2 = 0.311$. For Word2Vector embeddings, the best performing one is generated by the CSG architecture and has a dimension of 300, a window size of 5 and trained for 10 epochs on the one-to-one enriched dataset. The optimal SVM parameter set is $C = 1$, $gamma = 1$, $epsilon = 0.5$ and achieves a score of $R^2 = 0.354$. Finally the PV-DM architecture of Document2Vector produces the best performing document embedding with a vector dimension of 200, a window size of 5, trained for 400 epochs and using the pure Emerce dataset. The parameters of the regressor are $C = 1$, $gamma = 0.01$, $epsilon = 0.5$ and this results in a score of $R^2 = 0.368$. SVM regression performs the best using Document2Vector embeddings as input.

A point of interest regarding the two architectures of Document2Vector is that although PV-DM leads to the best results in SVM regression models, PV-DBOW leads to a similar performance. However that performance is achieved only when PV-DBOW trains on the enriched dataset, like it does in Ridge Regression, allowing the predictive model to go from less than 0.1 R^2 score to 0.339. An overview of the results can be seen in table 5.5.

Table 5.5: Best performing regression models per encoding method. In **bold** are the best performing models.

Predictive Model	Encoding	$\boldsymbol{R^2}$**-score**
Ridge Regression	tf-idf	0.247
	Word2Vector - CSG	0.238
	Document2Vector	**0.267**
SVM Regression	tf-idf	0.311
	Word2Vector	0.354
	Document2Vector	**0.368**

Since R^2 is not, strictly speaking, an accuracy metric, a visual representation of the best performing regressor is shown in figure 5.8, to assist in interpreting the actual effectiveness of the model. The first important finding is that great errors occur in the view prediction of the less popular articles. The cluster of instances above the diagonal to the left of the graph demonstrates that most errors in prediction are overestimations. The worst of the overestimations are errors in the scale of 4 logarithmic units. Underestimations are not as numerous, although they too reach a scale of 3 logarithmic units, as we can see from the lower tail of the cluster at the right. We also see a uniform spread with regard to the range of the errors and the actual view counts, without a particular pattern.

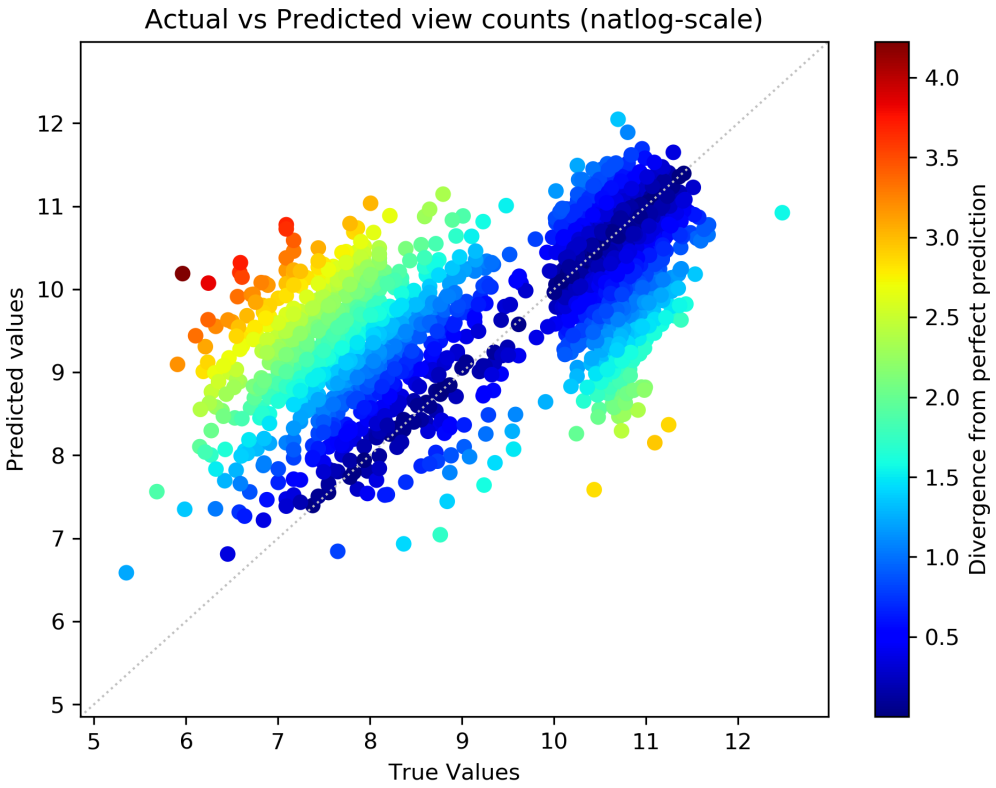


Figure 5.8: Plot of true view count (x-axis) against predicted view count (y-axis) of the best performing SVM regression model based on Document2Vector embeddings. Views are in natural logarithmic scale.

5.3 Combined Model Results

For the Combined model we first apply the best performing classifier, the SVM trained on the embeddings of the PV-DM architecture, as described in Section 5.1. The results are as expected, since the dataset and the data split are the same. We first separate the dataset into train and test, as described in Section 4.2. Then each subset is separated into non-popular and popular articles, in order to train the two new regressors. The performance of the regression is worse than before, based both on R^2 score (achieved less than 0.1) and visual inspection of the true versus predicted view counts of different models.

5.4 Embeddings

After presenting the performance of the predictive models, we present a brief inspection of the embeddings, to provide some insight into their quality. The following tables 5.6 and 5.7 show some examples of article similarity, based on their embeddings and cosine similarity in the vector space. We present and argue on the basis of classification since it is the most promising of the two approaches to the popularity prediction problem. This shows that features that contribute to an article's popularity

are not as successfully encapsulated in the generated embeddings.

Table 5.6: Most similar articles to 'NOS staakt ondersteuning Smart TV app voor veel toestellen' based on their Word2Vector averaged embeddings. The class of each article is also shown.

Articles	Title	Label
Reference	NOS staakt ondersteuning Smart TV app voor veel toestellen	Non-Popular
Most similar	IFA: 'Volgende Android TV stuk gebruikersvriendelijker'	Popular
	Consumentenbond: apps op slimme televisies snel verouderd	Popular
	Asus lanceert smartphone met tv-tuner	Non-Popular
	Ziggo GO straks ook voor Apple TV en Android TV	Non-Popular
	Consumentenbond: Fabrikanten van Android-smartphones stoppen te snel met updaten	Popular
	KPN zendt EK-wedstrijden in 4K uit	Popular
	Nederlandse omroepen nog niet op nieuwe Apple TV	Popular
	KPN trekt stekker uit online tv dienst Play	Popular
	Consumentenbond: miljoenen Nederlanders hebben onveilig Androidtoestel	Popular

Table 5.7: Most similar articles to 'NOS staakt ondersteuning Smart TV app voor veel toestellen' based on their Document2Vector embeddings. The class of each article is also shown.

Articles	Title	Label
Reference	NOS staakt ondersteuning Smart TV app voor veel toestellen	Non-Popular
Most similar	Nederlandse omroepen nog niet op nieuwe Apple TV	Popular
	Consumentenbond: Fabrikanten van Android-smartphones stoppen te snel met updaten	Popular
	IFA: 'Volgende Android TV stuk gebruikersvriendelijker'	Popular
	KPN trekt stekker uit online tv dienst Play	Popular
	Consumentenbond: apps op slimme televisies snel verouderd	Popular
	KPN zendt EK-wedstrijden in 4K uit	Popular
	Ziggo GO straks ook voor Apple TV en Android TV	Non-Popular
	Consumentenbond: miljoenen Nederlanders hebben onveilig Androidtoestel	Popular
	Asus lanceert smartphone met tv-tuner	Non-Popular

We can see that the embeddings in both cases provide articles that are indeed similar to the one we reference, at least regarding the topic. However the similarity they achieve does not encapsulate popularity well. 'NOS staakt ondersteuning Smart TV app voor veel toestellen' is an article that belongs to the non-popular class but only 2 out of the nine most similar articles are also from the same class. The rest belong to the popular class.

Chapter 6

Discussion and Conclusion

In this chapter we will discuss the results presented in the previous chapter with regard to the research question raised in the introduction of this thesis. We will also provide our insights in directing possible future research by outlining the main strengths and weaknesses of the methods we applied. Finally we will conclude the thesis.

6.1 Discussion

6.1.1 Research Questions

First we address the research questions with regard to our findings.

How can we assess whether an article is popular or not?

For this project, popularity of an article is interpreted as the number of views an article receives. However the number of views, which stems directly from the click rate of an article, might not constitute the most informative index of popularity. This perspective was investigated in the work of Agarwal et al. in [1], that promoted the idea of a more informative index based on different post-read actions. Furthermore in the work of Yangjie and Yao [56] it is shown that the articles that people read, share and comment on can be very different and highly topic-dependent. Therefore using the view count as the sole measurement of popularity, especially for a broad scope of article topics, might have been adequate for the research scope of this thesis but we estimate it to be sub-optimal for a business product that should be robust and applicable over a broad scope of articles.

Which text encoding method performs the best in predicting an article's popularity?

For the encoding models themselves, we can see that our selected embedding techniques can perform adequately well in a relatively narrow, binary classification task. Even the most simplistic method, that of tf-idf, performs well, regardless of its shortcomings in class-wise accuracy, which is an inherent problem of the dataset. Regarding the regression task, the encodings do not achieve considerable accuracy and additionally the errors cover a great range, both in sense of instances being wrongly predicted and of magnitude of errors. Nevertheless there are differences among encoding methods, which shows that more sophisticated encoding methods can obtain better results as they encapsulate more of the underlying elements that can influence popularity.

All in all Document2Vector embeddings lead to better performing predictive models. In classification the models utilizing these embeddings reach better f1-scores and have a more balanced and consistent class-wise accuracy. In regression the ones reaching the highest R^2 score are again the models that use the Document2Vector embeddings, even though the score is still low. There are cases where the distinction is not as clear between Word2Vector and Document2Vector, as we saw in section 5.1, but the latter performs the best over all tasks and predictive models.

Following our closer inspection of the embeddings and their relations in subsection 5.4, we see that both Word2Vector and Document2Vector generate embeddings with a good degree of similarity between articles regarding their topic but not as much regarding their class. This is a good indicator that the main reason the accuracy of our models could not improve further is the encoding method we employ and its limitations. This statement is also supported by the distribution of misclassifications for both Logistic Regression and SVMs and could also explain the poor performance of the regressors. This shows that some information regarding popularity is not perfectly encapsulated in the embeddings and thus our predictive models make wrong predictions for articles from the whole scope of view counts.

It is also important to discuss the differences in performance of the two Document2Vector architectures. PV-DM performs better than PV-DBOW when the pure Emerce dataset is used and that becomes even more apparent from the regression results. A model that uses Emerce-only PV-DBOW embeddings barely performs at all, but when it utilizes embeddings trained on the enriched dataset, it achieves an R^2 score comparable to that of the best performing model. That fact shows us that the approach of PV-DBOW in training document embeddings requires one important thing, more data. It requires more data than PV-DM, as even the two-to-one enriched dataset generates document embeddings that lead to higher performing models. The difference in model performance is significant and consistent in both classification and regression. PV-DM also benefits from the enriched datasets but in its case the results are not as consistent. Nevertheless, the use of an extended dataset affects positively the performance of Document2Vector embeddings.

Another important observation is the number of training epochs. This observation is not as obvious, but in almost all cases PV-DBOW embeddings that are trained longer lead to better performing predictive models. This is observed over different model combinations, not exclusively in PV-DBOW, and could thus be a general remark for the embedding method, that in principle Document2Vector requires more training epochs to generate informative encodings of texts.

We have two final remarks on the embeddings. First, it should be mentioned that each encoding method comes at a cost. Tf-idf leads to worse performing models but the process of generating the encodings is faster than the Word2Vector and Document2Vector that lead to beter results. Although we have not documented time costs of each method, it is a general observation that stems from our experience throughout this project. Second, it is more difficult to encode new instances when using Word2Vector and even more so Document2Vector. Elaborate techniques are present for the former but the latter requires careful handling, as its vocabulary does not consist of words but of text structures and in our case complete documents. It should be noted that gensim's *Doc2Vec* model did not possess a method to extend the vocabulary on a trained model and continue training. This is an important point for the further development of this system.

Which predictive model is the most appropriate for this task?

First, we have to mention that binary classification models achieve adequately good results while even the best regression models underperform. We estimate that the reason behind this difference is that our features, meaning the encoding methods we utilize, are not informative enough to accurately predict the number of views an article will gather. Our observations are in line with the work of Bandari et al. in [3]. Using different features they encounter the same effect, where classification is performing well while the performance of regression is lacking. This could be the result of the embeddings failing to encapsulate features that represent popularity or that the models are not fit to extract and utilize such information from the embeddings.

Focusing on classification, both Logistic Regression and SVMs are performing well, especially when they utilize informative encodings. The main flaw of all classifiers is the class-wise accuracy imbalance, that stems from the imbalance that is inherent in the dataset. We attempted to counter this issue by penalizing the misclassifications of the minority class more than those of the majority class. This is accomplished by utilizing the available functionality of the Sklearn models. Their final results show a clear superiority of SVM in that front. Although the quality of the encodings also influences performance, the best SVM models have significantly improved class-wise accuracy. This is evident as the best of the high-performing Logistic Regression models achieves 62% and 84% accuracy for the non-popular and popular classes while the worst of the high-performing SVMs achieves 69% and 82% respectively. Therefore, from the two classification methods we tested, SVMs are the best option, regardless of possible extra time costs for training.

For regression, although the overall performance is low, we should point out our observations. First of all using SVM regression with a high number of features that can not be interpreted separately (although various attempts have been made such as in [47, 45]) and, as such, can not be meaningfully prioritized, is not efficient. Second, the SVM regressors require considerably more time to train and optimize and it is possible that different dimensions of the input are highly correlated or even not informative at all, deteriorating the performance of the regressor.

Last, the performance of the hybrid model is the most surprising. We expected the regression models to perform better, as the range of values they had to estimate did not exhibit such great differences. However the regressors performed even worse and essentially could not fit the data at all. This could be an indication that R^2 is not suitable for such a regression task and comparison and thus when the regressors had to be optimized on a narrower range of values, the optimization process failed.

6.1.2 Future Work

Popularity prediction on the internet is affected by a multitude of factors. We attempted to narrow down the scope and only examine the content of the news articles and we did so in a compact and abstract way by employing embedding techniques. Improved performance could be achieved by including more interpretable features of an article or by taking into consideration meta-data surrounding its publication. Additionally, we interpreted popularity as the number of views an article received, which is not the most representative and definitely not the only measurement of popularity. Employing more elaborate measurements of popularity is sure to benefit any predictive model.

Regarding the predictive models, testing different or multiple performance metrics is a valid first step in improving performance in a robust way, as we saw how similar scores can translate into

significantly different behaviour in a model. After establishing a reliable metric, experimenting with more elaborate models is a valid route to improve overall performance. We saw significant and consistent improvement as the complexity of the models increased. More experiments could also answer the question whether the performance of the predictive models is caused by the restrictions they have themselves or by the restrictions of the embedding methods.

Lastly, broadening the scope of application is also of high interest, both from a research and a business perspective. A popularity prediction model that utilizes content from multiple sources can prove to be more stable and reliable. However that route should be investigated with caution as word and document embeddings have been proven to be highly domain dependent.

6.2 Conclusion

In this thesis we set out to predict the popularity of online news articles before publication. We focused only on the main text of the articles and encoded it using embedding techniques. Our aim was to investigate the performance of these techniques for cold-start popularity prediction and therefore we applied both classification and regression. The former performed adequately well and demonstrated specific shortcomings while the latter performed poorly. That serves to show that the embedding techniques we utilized are more appropriate for classification and less so for regression, at least in combination with the predictive models we experimented with. More steps can be taken to produce better performing systems and we outlined the main points of interest, based on our research and experience.

Bibliography

[1] Deepak Agarwal, Bee-Chung Chen, and Xuanhui Wang. Multi-faceted ranking of news articles using post-read actions. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 694–703, New York, NY, USA, 2012. ACM.

[2] Ioannis Arapakis, B. Barla Cambazoglu, and Mounia Lalmas. On the feasibility of predicting news popularity at cold start. In *Social Informatics: 6th International Conference, SocInfo 2014, Barcelona, Spain, November 11-13, 2014. Proceedings*, pages 290–299, Cham, 2014. Springer International Publishing.

[3] Roja Bandari, Sitaram Asur, and Bernardo Huberman. The pulse of news in social media: Forecasting popularity. *ICWSM 2012 - Proceedings of the 6th International AAAI Conference on Weblogs and Social Media*, 2012.

[4] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[5] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.

[6] Joseph Berkson. Application of the logistic function to bio-assay. *Journal of the American Statistical Association*, 39(227):357–365, 1944.

[7] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st edition, 2009.

[8] Otto Bretscher. *Linear Algebra With Applications (3rd ed.)*. NJ: Prentice Hall, Upper Saddle River, 2013.

[9] Corinna Cortes and Vladimir Vapnik. Support-vector networks. In *Machine Learning*, pages 273–297, 1995.

[10] Andrew M. Dai, Christopher Olah, and Quoc V. Le. Document embedding with paragraph vectors. *CoRR*, abs/1507.07998, 2015.

[11] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1646–1654. Curran Associates, Inc., 2014.

[12] Harris Drucker, Christopher J. C. Burges, Linda Kaufman, Alex J. Smola, and Vladimir Vapnik. Support vector regression machines. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 155–161. MIT Press, 1997.

[13] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[14] C. Cawley Gavin and L. C. Talbot Nicola. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11:2079–2107, 2010.

[15] X. Guan, Q. Peng, Y. Li, and Z. Zhu. Hierarchical neural network for online news popularity prediction. In *2017 Chinese Automation Congress (CAC)*, pages 3005–3009, Oct 2017.

[16] Michael U. Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, Feb 2012.

[17] Mark Hughes, Irene Li, Spyros Kotoulas, and Toyotaro Suzumura. Medical text classification using convolutional neural networks. *CoRR*, abs/1704.06841, 2017.

[18] Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[19] Yaser Keneshloo, Shuguang Wang, Eui-Hong (Sam) Han, and Naren Ramakrishnan. *Predicting the Popularity of News Articles*, pages 441–449. Society for Industrial and Applied Mathematics, 2016.

[20] S. Kim, S. Kim, and H. Cho. Predicting the virtual temperature of web-blog articles as a measurement tool for online popularity. In *2011 IEEE 11th International Conference on Computer and Information Technology*, pages 449–454, Aug 2011.

[21] Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3294–3302. Curran Associates, Inc., 2015.

[22] Alexandre Klementiev, Ivan Titov, and Binod Bhattarai. Inducing crosslingual distributed representations of words. In *Proceedings of COLING 2012*, pages 1459–1474, Mumbai, India, December 2012.

[23] Kristen Herhold. How Businesses Invest in Digital Marketing in 2018. `https://themanifest.com/digital-marketing/how-businesses-invest-digital-marketing`, 2018.

[24] Jey Han Lau and Timothy Baldwin. An empirical evaluation of doc2vec with practical insights into document embedding generation. *CoRR*, abs/1607.05368, 2016.

[25] Alberto Lavelli, Fabrizio Sebastiani, and Roberto Zanoli. Distributional term representations: An experimental comparison. In *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management*, CIKM '04, page 615–624, New York, NY, USA, 2004. Association for Computing Machinery.

[26] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

[27] Nicolas Le Roux, Mark Schmidt, and Francis Bach. A Stochastic Gradient Method with an Exponential Convergence Rate for Finite Training Sets. In *NIPS'12 - 26th Annual Conference on Neural Information Processing Systems (2012)*, Lake Tahoe, United States, 2012.

[28] Blerina Lika, Kostas Kolomvatsos, and Stathes Hadjiefthymiades. Facing the cold start problem in recommender systems. *Expert Systems with Applications*, 41(4, Part 2):2065 – 2073, 2014.

[29] H. P. Luhn. A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of Research and Development*, 1(4):309–317, Oct 1957.

[30] S. Mehrizi, A. Tsakmalis, S. Chatzinotas, and B. Ottersten. Content popularity estimation in edge-caching networks from bayesian inference perspective. In *2019 16th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pages 1–6, Jan 2019.

[31] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, Workshop Track Proceedings*, 2013.

[32] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[33] Emmanuel Okafor, Pornntiwa Pawara, Faik Karaaba, Olarik Surinta, Valeriu Cordeanu, Lamber Schomaker, and Marco Wiering. Comparative study between deep learning and bag of visual words for wild-animal recognition. In *Symposium Series on Computational Intelligence (IEEE-SSCI), Athens*, 2016.

[34] Matteo Pagliardini, Prakhar Gupta, and Martin Jaggi. Unsupervised learning of sentence embeddings using compositional n-gram features. *CoRR*, abs/1703.02507, 2017.

[35] Pornntiwa Pawara, Emmanuel Okafor, Olarik Surinta, Lamber Schomaker, and Marco Wiering. Comparing local descriptors and bags of visual words to deep convolutional neural networks for plant recognition. In *International Conference on Pattern Recognition Applications and Methods*, 2017.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[37] David L. Phillips. A technique for the numerical solution of certain integral equations of the first kind. *J. ACM*, 9(1):84–97, January 1962.

[38] Martin Porter and Richard Boulton. `https://snowballstem.org/`. 25-11-2019.

[39] Ivens Portugal, Paulo Alencar, and Donald Cowan. The use of machine learning algorithms in recommender systems: A systematic review. *Expert Systems with Applications*, 97:205 – 227, 2018.

[40] Vijay V. Raghavan and S. K. M. Wong. A critical analysis of vector space model for information retrieval. *Journal of the American Society for Information Science*, 37(5):279–287, 1986.

[41] Rindra Rantoson and Adrien Bartoli. A 3d deformable model-based framework for the retrieval of near-isometric flattenable objects using bag-of-visual-words. *Computer Vision and Image Understanding*, 167:89 – 108, 2018.

[42] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010.

[43] Xin Rong. word2vec parameter learning explained. *CoRR*, abs/1411.2738, 2014.

[44] Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '02, pages 253–260, New York, NY, USA, 2002.

[45] Jamin Shin, Andrea Madotto, and Pascale Fung. Interpreting word embeddings with eigenvector analysis. In *NIPS 2018 Workshop IRASL*, 2018.

[46] Lai Siwei, Xu Liheng, Liu Kang, and Zhao Jun. Recurrent convolutional neural networks for text classification. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[47] Daniel Smilkov, Nikhil Thorat, Charles Nicholson, Emily Reif, Fernanda B. Viégas, and Martin Wattenberg. Embedding projector: Interactive visualization and interpretation of embeddings. In *30th Conference on Neural Information Processing Systems*, 2016.

[48] Stephen M. Stigler. Gauss and the invention of least squares. *Ann. Statist.*, 9(3):465–474, 05 1981.

[49] Olarik Surinta, Mahir F. Karaaba, Tusar K. Mishra, Lamber Schomaker, and Marco Wiering. Recognizing handwritten characters with local descriptors and bags of visual words. In *16th International Conference on Engineering Applications of Neural Networks (EANN)*, 2015.

[50] Alexandru Tatar, Jérémie Leguay, Panayotis Antoniadis, Arnaud Limbourg, Marcelo Dias de Amorim, and Serge Fdida. Predicting the popularity of online articles based on user comments. In *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*, WIMS '11, pages 67:1–67:8, New York, NY, USA, 2011. ACM.

[51] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[52] A.N. Tikhonov, A. Goncharsky, V.V. Stepanov, and A.G. Yagola. *Numerical Methods for the Solution of Ill-Posed Problems*. Mathematics and Its Applications. Springer Netherlands, 1995.

[53] Michele Trevisiol, Luca Maria Aiello, Rossano Schifanella, and Alejandro Jaimes. Cold-start news recommendation with domain-dependent browse graph. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, pages 81–88, New York, NY, USA, 2014.

[54] Manos Tsagkias, Wouter Weerkamp, and Maarten de Rijke. Predicting the volume of comments on online news stories. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, pages 1765–1768, New York, NY, USA, 2009.

[55] Lyndon White, Roberto Togneri, Wei Liu, and Mohammed Bennamoun. How well sentence embeddings capture meaning. In *Proceedings of the 20th Australasian Document Computing Symposium*, ADCS '15, pages 9:1–9:8, New York, NY, USA, 2015. ACM.

[56] Yangjie Yao and Aixin Sun. Are most-viewed news articles most-shared? In Rafael E. Banchs, Fabrizio Silvestri, Tie-Yan Liu, Min Zhang, Sheng Gao, and Jun Lang, editors, *Information Retrieval Technology*, pages 404–415, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[57] Zi-Ke Zhang, Chuang Liu, Yi-Cheng Zhang, and Tao Zhou. Solving the cold-start problem in recommender systems with social tags. *EPL (Europhysics Letters)*, 92(2):28002, oct 2010.

Appendix

Graphs

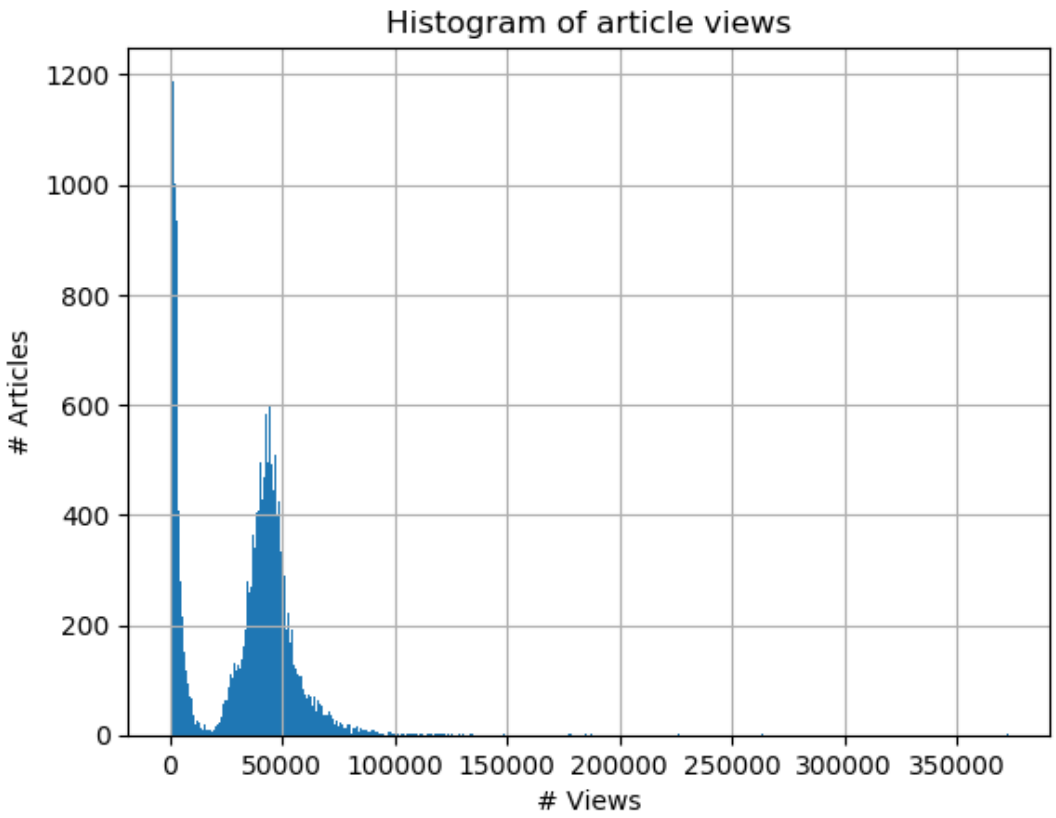


Figure 6.1: Histogram of view counts in the dataset. A natural shape can be seen with a very sharp and narrow normal distribution on the left with a right tail and a wide normal distribution on the right, also with a right tail. The two shapes will represent the two classes on the classification task.

Full experiment results

Table: Complete list of hyper-parameter optimization experiments of Logistic Regression models using the Document2Vector embeddings trained on the Emerce dataset. In **bold** are the best performing models of each architecture.

Embedding Parameters				**Classifier Parameters**		**F1-Score**
Model	Dimension	Window	Epochs	Solver	class weights	
PV-DBOW	200	5	100	liblinear	0: 0.6, 1: 0.4	0.6532
			250	liblinear	0: 0.6, 1: 0.4	0.6795
			400	liblinear	0: 0.6, 1: 0.4	0.6821
		15	100	saga	0: 0.6, 1: 0.4	0.6537
			250	sag	0: 0.6, 1: 0.4	0.6775
			400	sag	0: 0.6, 1: 0.4	0.6757
	300	5	100	sag	0: 0.7, 1: 0.3	0.6556
			250	saga	0: 0.6, 1: 0.4	0.6803
			400	liblinear	0: 0.6, 1: 0.4	**0.683**
		15	100	sag	0: 0.7, 1: 0.3	0.6528
			250	saga	0: 0.6, 1: 0.4	0.6816
			400	saga	0: 0.6, 1: 0.4	0.6784
PV-DM	200	5	100	liblinear	0: 0.6, 1: 0.4	0.6851
			250	saga	0: 0.6, 1: 0.4	0.729
			300	saga	0: 0.6, 1: 0.4	0.7392
			400	saga	0: 0.6, 1: 0.4	0.7422
		15	100	liblinear	0: 0.6, 1: 0.4	0.693
			250	saga	0: 0.6, 1: 0.4	0.7076
			300	liblinear	0: 0.6, 1: 0.4	0.7196
			400	saga	0: 0.6, 1: 0.4	0.725
	300	5	100	liblinear	0: 0.7, 1: 0.3	0.6873
			250	saga	0: 0.6, 1: 0.4	0.7251
			300	sag	0: 0.6, 1: 0.4	0.7371
			400	saga	0: 0.6, 1: 0.4	**0.746**
		15	100	liblinear	0: 0.7, 1: 0.3	0.6913
			250	liblinear	0: 0.6, 1: 0.4	0.7135
			300	saga	0: 0.6, 1: 0.4	0.7074
			400	sag	0: 0.6, 1: 0.4	0.7261

Table: Complete list of hyper-parameter optimization experiments of Logistic Regression models using the Document2Vector embeddings trained on the enriched datasets of Emerce and Wikipedia. In **bold** are the best performing models of each architecture.

Data ratios	**Embedding Parameters**				**Classifier Parameters**		**F1-Score**
	Model	Dimension	Window	Epochs	Solver	class weight	
One-to-One	PV-DBOW	200	5	300	sag	0: 0.6, 1: 0.4	0.7738
				400	sag	0: 0.6, 1: 0.4	0.7738
			15	300	saga	0: 0.6, 1: 0.4	0.7723
				400	liblinear	0: 0.6, 1: 0.4	0.7696
		300	5	300	sag	0: 0.5, 1: 0.5	0.7737
				400	sag	0: 0.5, 1: 0.5	0.7723
			15	300	sag	0: 0.6, 1: 0.4	0.7759
				400	sag	0: 0.6, 1: 0.4	**0.7766**
	PV-DM	200	5	300	liblinear	0: 0.6, 1: 0.4	0.7582
					saga	0: 0.6, 1: 0.4	0.7291
		300	15		liblinear	0: 0.6, 1: 0.4	**0.7576**
					saga	0: 0.6, 1: 0.4	0.7296
Two-to-One	PV-DBOW	200	5	300	liblinear	0: 0.6, 1: 0.4	0.7729
				400	sag	0: 0.6, 1: 0.4	0.7721
			15	300	liblinear	0: 0.6, 1: 0.4	0.7727
				400	sag	0: 0.6, 1: 0.4	0.7636
		300	5	300	liblinear	0: 0.6, 1: 0.4	0.7722
				400	sag	0: 0.5, 1: 0.5	0.7682
			15	300	saga	0: 0.6, 1: 0.4	**0.7747**
				400	sag	0: 0.5, 1: 0.5	0.7696
	PV-DM	200	5	300	sag	0: 0.6, 1: 0.4	0.7567
					saga	0: 0.6, 1: 0.4	0.7232
		300	15		liblinear	0: 0.6, 1: 0.4	**0.7624**
					sag	0: 0.6, 1: 0.4	0.7313

Table: Complete list of hyper-parameter optimization experiments of SVMs using the Word2Vector embeddings trained on the pure and enriched datasets. In **bold** are the best performing models of each architecture.

Embedding Parameters					**Regressor Parameters**			**R^2-Score**
Model	Dataset	Dimension	Window	Epochs	C	gamma	epsilon	
CSG	E	300	5	10	1	1	0.5	0.3381
	E=W	300	5	20	1	1	0.5	**0.354**
	E=2W	300	5	50	1	1	0.5	0.3472
	E=W	300	5	20	100	1	0.1	*0.3158*
CBOW	E	300	10	50	1	0.01	0.5	0.3234
	E=W	300	10	50	1	0.01	0.5	**0.3298**
	E=2W	300	10	50	1	0.01	0.5	0.3275
	E	300	10	5	100	0.01	1	*0.2563*

Table: Complete list of hyper-parameter optimization experiments of SVMs using the Document2Vector embeddings trained on the Emerce dataset. In **bold** are the best performing models of each architecture.

Embedding Parameters				**Classifier Parameters**		**F1-Score**
Model	Dimension	Window	Epochs	C	gamma	
PV-DBOW	200	5	100	100	5	0.6453
			250	1	0.1	0.6693
			400	1	0.1	**0.6931**
		15	100	10000	5	0.6492
			250	0.01	0.01	0.6692
			400	1	0.1	0.6926
	300	5	100	100	1	0.6311
			250	0.01	1	0.6684
			400	1	0.1	0.6876
		15	100	100	5	0.6366
			250	0.01	0.1	0.6664
			400	1	0.1	0.6901
PV-DM	200	5	100	100	1	0.7591
			250	1	0.1	0.7992
			300	1	0.1	0.816
			400	1	0.1	0.7867
		15	100	1	10	0.7816
			250	0.01	0.01	0.7943
			300	1	0.1	0.8035
			400	1	0.1	0.7786
	300	5	100	100	1	0.7592
			250	1	0.1	0.797
			300	1	0.1	**0.8112**
			400	1	0.1	0.7916
		15	100	1	12	0.7802
			250	1	0.1	0.7916
			300	1	0.1	0.781
			400	1	0.1	0.7816

Table: Complete list of hyper-parameter optimization experiments of SVMs using the Document2Vector embeddings trained on the enriched datasets of Emerce and Wikipedia. In **bold** are the best performing models of each architecture.

Data ratios	**Embedding Parameters**				**Classifier Parameters**		**F1-Score**
	Model	Dimension	Window	Epochs	C	gamma	
One-to-One	PV-DBOW	200	5	300	1	0.1	**0.8042**
				400	100	0.01	0.7787
			15	300	1	0.1	0.8015
				400	1	0.1	0.7901
		300	5	300	100	0.01	0.7887
				400	100	0.01	0.7876
			15	300	1	0.1	0.7975
				400	10000	0.01	0.7892
	PV-DM	200	5	300	1	0.01	**0.7828**
					1	0.1	0.7409
		300	15		100	0.01	0.776
					100	0.01	0.7668
Two-to-One	PV-DBOW	200	5	300	1	0.1	0.8066
				400	100	0.01	0.7934
			15	300	1	0.1	**0.8076**
				400	1	0.1	0.7915
		300	5	300	1	0.1	0.804
				400	100	0.01	0.7888
			15	300	1	0.1	0.8037
				400	10000	0.01	0.7931
	PV-DM	200	5	300	100	0.01	0.7711
					1	0.1	0.7603
		300	15		1	0.1	**0.781**
					1	0.1	0.7619