



**FACULTY  
OF INFORMATION  
TECHNOLOGY  
CTU IN PRAGUE**

## ASSIGNMENT OF MASTER'S THESIS

**Title:** Recommending related images to articles  
**Student:** Bc. Matouš Pištora  
**Supervisor:** doc. Ing. Pavel Kordík, Ph.D.  
**Study Programme:** Informatics  
**Study Branch:** Knowledge Engineering  
**Department:** Department of Theoretical Computer Science  
**Validity:** Until the end of summer semester 2018/19

### Instructions

Survey state of the art algorithms in image processing and text mining. Focus on modern deep learning methods and possibilities to obtain high quality neural embedding for both images and text. Design and implement a system capable of recommending images related to article based on text of the article. Extend the system to support multiple languages or domains (news, sport, hobby, etc.). Test the performance of the recommender system on images and articles supplied by a publishing house.

### References

Will be provided by the supervisor.

doc. Ing. Jan Janoušek, Ph.D.  
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.  
Dean

Prague January 30, 2018



CZECH TECHNICAL UNIVERSITY IN PRAGUE  
FACULTY OF INFORMATION TECHNOLOGY  
DEPARTMENT OF THEORETICAL COMPUTER SCIENCE



Master's thesis

## Recommending related images to articles

*Bc. Matouš Pištora*

Supervisor: doc. Ing. Pavel Kordík, Ph.D.

February 15, 2019



---

## **Acknowledgements**

I would like to thank my supervisor doc. Ing. Pavel Kordík, Ph.D. for his guidance in writing my thesis and my family and friends for their support.



---

# Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the “Work”), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work for non-profit purposes only, in any way that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on February 15, 2019

.....

Czech Technical University in Prague

Faculty of Information Technology

© 2019 Matouš Pištora. All rights reserved.

*This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).*

### **Citation of this thesis**

Pištora, Matouš. *Recommending related images to articles*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.



---

# Abstrakt

Tato diplomová práce se soustřeďuje na nejnovější algoritmy zpracování obrazu a vytěžování textu včetně metod hlubokého učení a neuronových sítí. Je navržen systém, který je schopen na základě textu novinového článku navrhnout obrázky související s jeho obsahem. Součástí systému jsou moderní algoritmy na vytěžování informací z textu a obrázků, které byly testovány společně s regresními algoritmy. Tento systém je rozšířen na více jazyků.

**Klíčová slova** strojové učení, učení s učitelem, hluboké učení, zpracování přirozeného jazyka, vnoření slov, počítačové vidění, zpracování obrazu

---

# Abstract

This diploma thesis focuses on the analysis of state-of-the-art algorithms in image processing and text mining including modern deep learning and neural networks. A system capable of recommending images related to an article based on the text of the article has been designed and implemented with the use of supervised learning. Multiple image and text feature algorithms have been evaluated along with numerous regression algorithms. The system was extended to multiple languages and domains.

**Keywords** machine learning, supervised learning, deep learning, natural language processing, word embedding, computer vision, image processing

---

# Contents

<b>Introduction</b>	<b>1</b>
<b>1 Analysis</b>	<b>3</b>
1.1 Recommendation . . . . .	3
1.2 Text embedding . . . . .	18
1.3 Image embedding . . . . .	34
<b>2 Dataset and data analysis</b>	<b>49</b>
2.1 Reuters dataset . . . . .	49
2.2 Aktualne dataset . . . . .	50
2.3 Text embedding extraction . . . . .	50
2.4 Image embedding extraction . . . . .	52
2.5 Visualization . . . . .	53
<b>3 Experiments</b>	<b>59</b>
3.1 Methodology . . . . .	59
3.2 Experiment 1 – Regression model tuning . . . . .	61
3.3 Experiment 2 – Feature extraction models evaluation . . . . .	79
3.4 Experiment 3 – Application on multiple domains . . . . .	81
3.5 Experiment 4 – Dataset by a Czech publishing house . . . . .	84
3.6 Results . . . . .	87
<b>Conclusion</b>	<b>89</b>
<b>Bibliography</b>	<b>91</b>
<b>A Recommendations examples</b>	<b>99</b>
<b>B Glossary</b>	<b>103</b>



---

## List of Figures

1.1	P-norms illustration. . . . .	8
1.2	Bias-variance tradeoff. . . . .	9
1.3	Multilayer perceptron. . . . .	14
1.4	Neural network activation functions. . . . .	16
1.5	CBOw and Skip-gram architecture . . . . .	21
1.6	GloVe and word2vec learning times. . . . .	24
1.7	ConceptNet node example. . . . .	27
1.8	Long range dependency in translation. . . . .	30
1.9	Short range dependency in translation. . . . .	30
1.10	Context in BERT and other models. . . . .	31
1.11	BERT input representation. . . . .	32
1.12	LeNet5 architecture. . . . .	34
1.13	Convolutional layer. . . . .	35
1.14	AlexNet architecture. . . . .	36
1.15	Overlapping pooling. . . . .	37
1.16	Inception module. . . . .	38
1.17	GoogLeNet architecture. . . . .	39
1.18	Factorization of convolutional layer. . . . .	40
1.19	Asymmetric factorization of convolutional layer. . . . .	41
1.20	Inception module A. . . . .	41
1.21	Inception module B. . . . .	42
1.22	Inception module C. . . . .	42
1.23	ResNet architecture. . . . .	44
1.24	Residual block. . . . .	45
1.25	Residual activations v2. . . . .	46
1.26	ResNeXt block. . . . .	47
1.27	Depthwise decomposition of a convolutional layer. . . . .	47
1.28	Comparison of CNNs. . . . .	48
2.1	PCA and t-SNE. . . . .	54

2.2	Clustering methods. . . . .	55
2.3	Word2vec clusters on text features. . . . .	57
2.4	Word2vec clusters on image features. . . . .	58
3.1	Recommender design. . . . .	62
3.2	KNN n neighbors. . . . .	63
3.3	KNN metric. . . . .	64
3.4	KNN accuracy metric. . . . .	64
3.5	KNN weights. . . . .	65
3.6	KNN scaling. . . . .	66
3.7	KNN PCA kernel. . . . .	67
3.8	KNN PCA number of components. . . . .	68
3.9	ElasticNet L1 ratio. . . . .	69
3.10	ElasticNet alpha. . . . .	70
3.11	ElasticNet maximum no. of iterations. . . . .	71
3.12	ElasticNet PCA. . . . .	71
3.13	MLP hidden layer count accuracy. . . . .	73
3.14	MLP hidden layer count time. . . . .	74
3.15	MLP activation function accuracy. . . . .	74
3.16	MLP activation function time. . . . .	75
3.17	MLP initial learning rate. . . . .	76
3.18	Random forest. . . . .	77
3.19	Prediction results. . . . .	78
3.20	$k$ and top-k accuracy relation. . . . .	78
3.21	Prediction results on t-SNE projection. . . . .	79
3.22	Prediction results on t-SNE projection 2. . . . .	79
3.23	Document embedding extraction. . . . .	80
3.24	BERT document embedding extraction. . . . .	81
3.25	Features comparison with KNN model. . . . .	82
3.26	Features comparison with ElasticNet model. . . . .	83
3.27	Performance on different domains. . . . .	85
3.28	Czech text feature extraction models. . . . .	86
3.29	Aktualne domains. . . . .	87
A.1	Recommendation for an article about a bus fire. . . . .	100
A.2	Recommendation for an article about politics. . . . .	101
A.3	Recommendation for an article about sports. . . . .	102

---

## List of Tables

1.1	Bag of words embedding. . . . .	18
1.2	Word co-occurrence probabilities . . . . .	22
1.3	Comparison of deep-learning models. . . . .	33
2.1	Reuters article categories. . . . .	50
2.2	Aktualne article categories. . . . .	50
2.3	Feature models . . . . .	53
3.1	KNN hyperparameters. . . . .	63
3.2	KNN optimized parameters. . . . .	67
3.3	ElasticNet hyperparameters. . . . .	69
3.4	ElasticNet optimized parameters. . . . .	72
3.5	MLP hyperparameters. . . . .	72
3.6	ElasticNet optimized parameters. . . . .	76
3.7	Experiment 1 results. . . . .	77





---

# Introduction

Moore's law states that the number of transistors in integrated circuits doubles every year. It has been true for the past two decades, and as a result the increase of hardware capabilities opened new doors for computationally intensive algorithms. In the last few years, there has been a huge advancement in the field of deep convolutional neural networks, designed for image classification and image feature extraction. In the field of machine translations and natural language processing, the progress has also been significant. The advancement in both fields can be utilized in common tasks, such as helping editors of publishing houses to choose an image for an article by recommending related images.

The objective of this thesis is to study some of the newest algorithms in both text mining and image feature embedding and combine the obtained knowledge into one system capable of recommending images related to articles based on their content, with the help of a regression model. The focus of the text and image feature extraction should be on neural networks. The thesis will test the system on multiple domains, such as news and sports, and extend the system on another language.

The theoretical part of the thesis focuses on state-of-the-art algorithms in image processing and text mining. It is divided into parts about word vector embeddings extracted from text, image feature embeddings, and various algorithms such as regression ones.

In the second chapter, we will concentrate on obtaining a dataset in order to train the recommendation model with supervised learning. We will discuss the options of extracting image and text features studied in the previous chapter.

The third chapter involves experiments that will be conducted on the recommending system. Multiple hyperparameters of the regression models will be tuned in order to obtain the best performance. The model will be tested on multiple domains and extended to multiple languages.



---

# Analysis

In this chapter, I will write about how to approach the task of recommending images to articles and the background to the practical part of the thesis.

One part of the assignment of the thesis is to design and implement a system capable of recommending images related to an article based on its text. The system will consist of two stages. In the first one, the text will be processed using a feature extraction model to infer a vector embedding. In the second stage, the vector embedding will be the input of a trained recommendation model and the output will be a set of images related to the article from a given database of images. The model will be trained on a training dataset of articles and pictures. The pictures will be processed with a feature extraction model to obtain the information about its content. The whole model design can be seen in Figure 3.1.

This approach introduces three main problems:

1. Recommendation model using text and image embeddings.
2. Extraction of the text embeddings.
3. Extraction of the image embeddings.

The following sections describe and analyze possible solutions to each of the problems.

## 1.1 Recommendation

This section briefly describes the algorithms and techniques used in the practical part of the thesis and lays the theoretical foundations for the following sections.

### 1.1.1 Data preprocessing

In data mining there are many different preprocessing techniques including instance selection, normalization, transformation, feature extraction, selection

or projection. Their goal is to make the data complete, consistent and more reliable in order to increase the performance on the machine learning task. The dataset obtained for the practical task is fortunately of high quality, so only a few techniques will be used in the implementation.

#### 1.1.1.1 Standardization

Standardization, which is also called the Z-score normalization [17], is the process of unifying an attribute's distribution by transforming the numeric variables to have a mean equal to 0 and standard deviation of 1. For value  $v$  of attribute  $A$  with mean  $\bar{A}$  and standard deviation  $\sigma_A$  the normalized value  $\hat{v}$  is equal to:

$$\hat{v} = \frac{v - \bar{A}}{\sigma_A} \quad (1.1)$$

If the mean  $\bar{A}$  and standard deviation  $\sigma_A$  of the attribute are not known or available, sample mean and standard deviation are used:

$$\bar{A} = \frac{1}{n} \sum_{i=1}^n v_i \quad (1.2)$$

$$\sigma_A = + \sqrt{\frac{1}{n} \sum_{i=1}^n (v_i - \bar{A})^2} \quad (1.3)$$

#### 1.1.1.2 Normalization

Normalization, also called Min-Max normalization, is the process of unifying an attribute's range by transformation of attribute  $A$  to that specific range in  $[min, max]$ , which is usually  $[0, 1]$  or  $[-1, 1]$ . The equation for value  $v$  of attribute  $A$  and its normalized value  $\hat{v}$  is:

$$\hat{v} = \frac{v - \min(A)}{\max(A) - \min(A)} (max - min) + min \quad (1.4)$$

#### 1.1.1.3 Principal component analysis

Principal component analysis (PCA) is a popular dimensionality reduction and feature extraction method, though it was originally a technique to transform a set of possibly linearly correlated attributes into a set of linearly uncorrelated principal components. The term was coined in 1933 in [25], but originates in the work of Person in 1901 [44]. The techniques transform  $n$  observations with  $p$  variables to a set of  $\min(n - 1, p)$  components with an orthogonal transformation, where the new set of components has the largest variance [7].

The definition of PCA from [67] is the following: For a dataset  $X = x_i$  where  $i = 1, 2, \dots, N$  and  $x_i$  is a vector with dimension  $D$ , the PCA can transform this set into  $M$  dimensional subspace where  $M < D$ . The projection is denoted by  $\mathbf{y} = \mathbf{A}\mathbf{x}$ , where  $\mathbf{A} = [\mathbf{u}_1^T, \dots, \mathbf{u}_M^T]$  and  $\mathbf{u}_k^T \mathbf{u}_k = 1$  for  $k = 1, 2, \dots, M$ . The goal is to maximize the variance of  $\{\mathbf{y}_i\}$  which is the trace of covariance matrix of the same  $\{\mathbf{y}_i\}$ . This gives us the equations:

$$\mathbf{A}^* = \arg \max_{\mathbf{A}} \text{tr}(\mathbf{S}_y) \quad (1.5)$$

$$\mathbf{S}_y = \frac{1}{N} \sum_{i=1}^N (\mathbf{y}_i - \bar{\mathbf{y}}) (\mathbf{y}_i - \bar{\mathbf{y}})^T \quad (1.6)$$

$$\bar{\mathbf{y}} = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i \quad (1.7)$$

Let  $\mathbf{S}_x$  be the covariance matrix of  $\{\mathbf{x}_i\}$ . Since  $\text{tr}(\mathbf{S}_y) = \text{tr}(\mathbf{A}\mathbf{S}_x\mathbf{A}^T)$ , then by using the Lagrangian multiplier  $\lambda$  and taking the derivative, we get the following:

$$\mathbf{S}_x \mathbf{u}_k = \lambda_k \mathbf{u}_k \quad (1.8)$$

It means that  $\mathbf{u}_k$  is the eigenvector of  $\mathbf{S}_x$ . Now  $\mathbf{x}_i$  can be represented by the equation:

$$\mathbf{x}_i = \sum_{k=1}^D (\mathbf{x}_i^T \mathbf{u}_k) \mathbf{u}_k \quad (1.9)$$

It can also be approximated by  $\tilde{\mathbf{x}}_i$  with a reduced dimension  $M$ :

$$\tilde{\mathbf{x}}_i = \sum_{k=1}^M (\mathbf{x}_i^T \mathbf{u}_k) \mathbf{u}_k \quad (1.10)$$

where  $\mathbf{u}_k$  is the eigenvector of  $\mathbf{S}_x$  corresponding to the  $k$ th largest value, thus maximizing the variance of the transformed  $\tilde{\mathbf{x}}_i$ .

The Kernel PCA variation can also be used with a non-linear projection  $\phi(\mathbf{x})$ . Its main advantage is that it is able to find a projection in which the data is linearly separable and as such allows the use of linear regression models on nonlinear problems. Because using the plain Kernel PCA would be extremely inefficient for large dimensions, the kernel method is used [7].

Some of the kernel functions used in Kernel PCA are:

- **Linear** (which produces the same as PCA) kernel with free parameter  $c$ :

$$K(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T \mathbf{y} + c \quad (1.11)$$

- **Polynomial** kernel with degree  $d$  and free parameter  $c$ :

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^\top \mathbf{y} + c)^d \quad (1.12)$$

- **Gaussian radial basis function** kernel:

$$K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2}\right) \quad (1.13)$$

- **Cosine** kernel with the degree  $\theta$  between the two vectors:

$$K(\mathbf{x}, \mathbf{y}) = \|\mathbf{x}\| \|\mathbf{y}\| \cos \theta \quad (1.14)$$

#### 1.1.1.4 T-distributed Stochastic Neighbor Embedding

T-distributed Stochastic Neighbor Embedding (t-SNE) is a feature projection technique designed for dimensionality reduction of high-dimensional datasets for the purpose of visualization. The algorithm works by computing the probability distribution of data points in the original high-dimensional space so that point close to each other i.e. neighbors are likely to be chosen by the distribution and far away points have a low probability of being chosen. In the next step a similar probability distribution in two (or how many is desired) dimensions is estimated by optimizing the Kullback-Leibler divergence  $D$  between these two distributions  $P$  and  $Q$ :

$$D_{\text{KL}}(P\|Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx \quad (1.15)$$

Gradient descent is used for the optimization. Because the computation speed of this algorithm is high for high-dimensional and large datasets, it is recommended by the author to combine the two techniques and for example reduce the original feature space to 30 dimensions using PCA first and then use the t-SNE algorithm.

#### 1.1.2 Distance measures

Various machine learning algorithms use some sort of distance or similarity measures, for example the K Nearest Neighbors model. A selection of distances is defined in this subsection with [14] as the source.

A distance (or dissimilarity) is a function  $d : X \times X \rightarrow \mathbb{R}$  on set  $X$  that for all  $x, y \in X$  holds:

1.  $d(x, y) \geq 0$  (*non-negativity*)
2.  $d(x, y) = d(y, x)$  (*symmetry*)
3.  $d(x, x) = 0$  (*reflexivity*)

A metric is a function  $d : X \times X \rightarrow \mathbb{R}$  on set  $X$  that for all  $x, y, z \in X$  holds:

1.  $d(x, y) \geq 0$  (*non-negativity*)
2.  $d(x, y) = d(y, x)$  (*symmetry*)
3.  $d(x, y) = 0$  if and only if  $x = y$  (*identity of indiscernibles*)
4.  $d(x, y) \leq d(x, z) + d(z, y)$  (*triangle inequality*)

Given these definitions, we can define the following metrics:

- $l_p$  **metric** for  $p, 1 \leq p \leq \infty$  is a norm metric on  $\mathbb{R}^n$  defined as:

$$\|x - y\|_p = \left( \sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}} \quad (1.16)$$

where for  $p = \infty$  the  $\|x - y\|_\infty = \max_{1 \leq i \leq n} |x_i - y_i|$ . If  $0 < p < 1$  the function loses its triangle inequality property and becomes a distance. It is also one of the so-called Minkowski metrics and in the experiment section are be called simply as Minkowski. The  $l_p$  metrics are illustrated in Figure 1.1.

- **Euclidean metric** is a special case of  $l_p$  metric where  $p = 2$ :

$$\|x - y\|_2 = \sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2} \quad (1.17)$$

- **Squared Euclidean metric** is a similar to euclidean, but is faster to compute.:

$$\|x - y\|_2^2 = (x_1 - y_1)^2 + \cdots + (x_n - y_n)^2 \quad (1.18)$$

- **Manhattan metric** is a special case of  $l_p$  distance where  $p = 1$ :

$$\|x - y\|_1 = |x_1 - y_1| + |x_2 - y_2| + \cdots + |x_n - y_n| \quad (1.19)$$

- **Chebyshev metric** is a special case of  $l_p$  metric where  $p = \infty$ :

$$\|x - y\|_\infty = \max \{|x_1 - y_1|, |x_2 - y_2|, \dots, |x_n - y_n|\} \quad (1.20)$$

- **Cosine distance** which is  $1 - \text{cosine similarity}$  is not a metric and is defined by:

$$1 - \frac{\langle x, y \rangle}{\|x\|_2 \cdot \|y\|_2} = 1 - \cos \phi \quad (1.21)$$

In the case  $\bar{x} = \bar{y} = 0$ , the cosine distance equals to correlation distance.

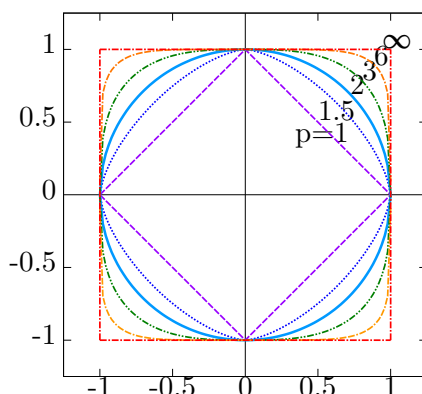


Figure 1.1: Illustration of the unit circles of selected  $p$ -norms in two dimensions. Source: [48].

### 1.1.3 Regression

In machine learning the algorithms divide into several types. During *supervised learning* the algorithms build a model from a set of data containing both the input and the desired output. This type of algorithms include *classification*, which aims to predict the category the input observation belongs to, and *regression*, which is the process of estimating the relationship among input and output variables. Regression will be the method chosen for our task, and some of its algorithms used in the practical part are described in the following text.

Another type of algorithms is *unsupervised learning* which tries to find a structure within the input data without a set of desired outputs. It will be used in data analysis for clustering. *Reinforcement learning* is the area of machine learning where the learning agents do not have a set of desired output data but are trained to maximize a certain reward function, for example the artificial intelligence in games.

There are many different criteria regarding the choice of the correct algorithm, such as bias-variance tradeoff (seen in Figure 1.2). It is the balance between the model successfully capturing the structures within the training data while still being able to generalize to new input data. Bias means the difference between the model average prediction and the correct values. Models with high bias simplify the data too much, which is called *underfitting*. On the other hand, high variance models fit the data too well and fail to generalize (perform well on new data). This is called *overfitting*.

Regression algorithms can also be categorized based on whether they are *parametric* or *nonparametric* i.e. whether the model has a set of parameters that has to be learned in the training phase to successfully predict the output or not. They should not be confused with *hyperparameters*, which are parameters set before the training starts, such as the type of distance function,



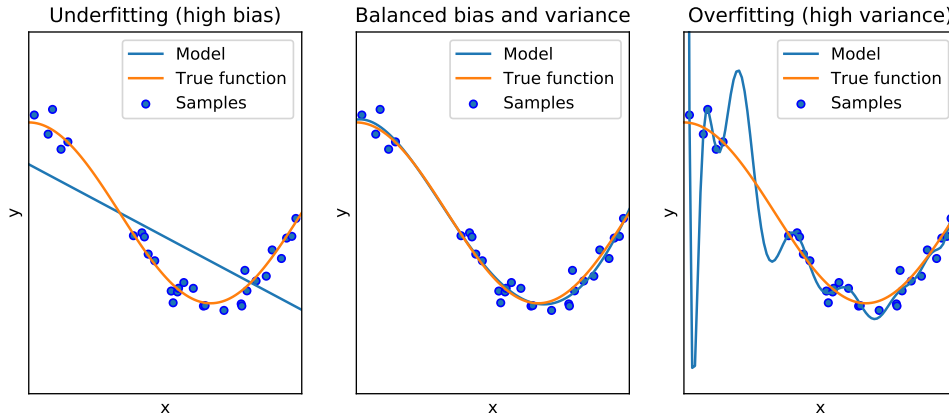


Figure 1.2: Illustration of the bias-variance tradeoff and the effects of over- and under-fitting.

optimization algorithm or number of hidden layers in a neural network.

Algorithms are also characterized by whether they are linear or nonlinear, i.e. whether they are able to solve only a linear problem, where the output is a linear combination of input variables, or also a nonlinear problem, where the model predicts the input using nonlinear combination of its parameters. The analogy in classification problems is whether the model can find a hyperplane separating the two classes or whether the separation between the classes is a general hypersurface. Nonlinear problems are generally harder to solve than linear ones.

The regression problem in this thesis is probably a nonlinear problem. Given the input is features extracted from a text and the output is features describing an image, I expect there is no linear relationship between a text and a picture that is to some degree related to it.

Our problem also presents the obstacle of a so-called *multioutput* regression. Majority of regression problems predict only one or a few variables, for example the price of a house or the mileage of a car, but our task is to predict hundreds of features characterizing an image.

### 1.1.3.1 K Nearest Neighbors

K Nearest Neighbors algorithm (KNN) is a basic nonparametric regression method [2]. Given a set of samples  $(x_1, y_1), \dots, (x_n, y_n) \in \mathbb{R}^d \times \mathbb{R}$ , a fixed parameter  $k$ , set of indices  $N_k(x)$  of  $k$  nearest neighbors to  $x$ , we get the regression estimation:

$$\hat{f}(x) = \frac{1}{k} \sum_{i \in N_k(x)} y_i * w_i \quad (1.22)$$

The pseudocode for this algorithm with a given distance function  $D$  (for example the euclidean function), weights  $w$  (usually uniform ones or the inverse of distances) and a set of samples  $(x_1, y_1), \dots, (x_n, y_n)$  can be seen in Figure 1. The weights are usually uniform (i.e. 1s) or inverse of the distance from  $x$  ( $1/d$ ).

---

**Algorithm 1:** Pseudocode of naive KNN regression.

---

**Input:** Set of observations  $\{x_i, y_i\}$ , input observation  $x$   
**Result:** Prediction  $\hat{y}$   
**for**  $i \leftarrow 1, n$  **do**  
  |  $d_i \leftarrow D(x, x_i)$   
**end**  
 $I \leftarrow$  first  $k$  items of  $\text{Argsort}(d)$   
 $\hat{y} \leftarrow \text{WeightedAverage}(y_{i \in I}, w_{i \in I})$   
**return**  $\hat{y}$

---

### 1.1.3.2 Linear regression

Linear model is a type of parametric model which makes a prediction based on the linear combination of the input, learned weights  $\theta$  and a bias  $\theta_0$  [18]:

$$\hat{y} = \theta_0 1 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n + \epsilon = \mathbf{x}_i^\top \boldsymbol{\theta} + \theta_0 \quad (1.23)$$

This model is trained on a training set to minimize the *root mean square error* (RMSE) between prediction  $\hat{y}$  and the actual value  $y$ . Because minimizing a function is the same as minimizing its square value, we can use the *mean square error* (MSE) which is computationally less intensive:

$$\begin{aligned} \text{MSE}(Y, \hat{Y}) &= \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2 \\ \text{RMSE}(Y, \hat{Y}) &= \sqrt{\frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2} = \sqrt{\text{MSE}(Y, \hat{Y})} \end{aligned}$$

### 1.1.3.3 Regularized regression

If a regression model is prone to overfitting, we can introduce regularization to the optimization function. The ridge regression model introduces a  $l_2$  regularization term which is the sum of squares of the weights. The optimization function is now:

$$J(\theta) = \text{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^n \theta_i^2 \quad (1.24)$$

On the other hand, the *Least absolute shrinkage and selection operator* regression algorithm called simply Lasso introduces a  $l_1$  regularization:

$$J(\theta) = \text{MSE}(\theta) + \alpha \sum_{i=1}^n |\theta_i| \quad (1.25)$$

A combination of these regularization techniques is called the Elastic Net and introduces the parameter  $r$ ,  $0 \leq r \leq 1$ , which is the ratio between  $l_1$  and  $l_2$  regularization:

$$J(\theta) = \text{MSE}(\theta) + r\alpha \sum_{i=1}^n |\theta_i| + \frac{1-r}{2}\alpha \sum_{i=1}^n \theta_i^2 \quad (1.26)$$

Both Lasso and Elastic nets are prone to reduce some of features' weights to zero, given a large  $l_1$  regularization penalty. Thus a Ridge regression or Elastic net with low  $r$  are usually preferred. In 2014 [73], it was proven that Elastic net can be reduced to a linear Support vector machine.

#### 1.1.3.4 Support vector regression

Support vector regression (SVR) is similar to its classification counterpart Support vector machine (SVM) and is easier to visualize. The task of a SVM model for a linear problem is to construct a hyperplane in the feature space to separate samples into two classes while maximizing the distance of the samples from the separating hyperplane. The hyperplane solves for a set of samples  $x$  and weights  $w$  the equation:

$$\mathbf{w}^T \mathbf{x} + b = 0 \quad (1.27)$$

Given an observation  $x$  and weights  $\mathbf{w}$  the prediction of class  $y$  is  $\hat{y}$ :

$$\hat{y} = \begin{cases} 0 & \text{if } \mathbf{w}^T \mathbf{x} + b < 0 \\ 1 & \text{if } \mathbf{w}^T \mathbf{x} + b \geq 0 \end{cases} \quad (1.28)$$

We can either not allow misclassification of a prediction by setting a hard margin for the classifier or allow it with a soft margin. The hard margin with given  $y_i \in (-1, 1)$  denoting the two positive and negative classes and a threshold parameter  $\epsilon$  is the constraint:

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq \epsilon, \quad i = \{1 \dots m\} \quad (1.29)$$

while minimizing:

$$\frac{1}{2} \mathbf{w}^T \mathbf{w} \quad (1.30)$$

For non-linearly separable problems we introduce a soft margin with parameter  $\alpha^{(i)} \geq 0$  and a hyperparameter  $C$ . The optimization function becomes:

$$\frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^m \alpha_i \quad (1.31)$$

under the constraint:

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq \epsilon - \alpha_i \quad (1.32)$$

The way to make a regression SVR out of SVM is simply to minimize the same function under a similar constraint for hard margin:

$$-\epsilon \leq y_i - \mathbf{w}^T \mathbf{x}_i + b \leq \epsilon \quad (1.33)$$

And again the same optimization function with similar constraint:

$$-\epsilon - \alpha_i \leq y_i - \mathbf{w}^T \mathbf{x}_i + b \leq \epsilon + \alpha_i \quad (1.34)$$

There are numerous improvements of the algorithm, such as constructing a dual problem solved by the efficient quadratic programming. The other important trick is the use of a kernel function and the kernel trick using kernels described in Subsection 1.1.1.3. This allows the model to solve even non-linear problems.

The SVR and kernel SVR are unfortunately not very suitable for multi-output classification, where there is more than one variable on the output and as such will be skipped in the experiment section.

### 1.1.3.5 Decision trees and random forests

Decision trees is another non-parametric supervised method used for classification or regression. Its basic idea is to recursively break the dataset into smaller subdivisions according to a set of learned decision rules. Among its many advantages are non-linearity, simplicity, clear interpretation and good scalability. Because they are prone to overfitting, an ensemble model of trees can be constructed by training multiple decision trees each on a random sub-sample of the training dataset. Such an ensemble could create a model where trees are correlated to each other because they all choose decision rules based on a few strong predictors. To overcome it a technique *random forest* is used, where each tree uses only a random subset of the features. An extension to *random trees* called extremely randomized trees or *extratrees* can also be used. The difference from Random forests is that during learning, the individual trees learn on the whole training dataset. Also during the training of a tree, we do not compute an ideal cut-point to split the feature space. Instead, a number of cut-points is generated randomly and one that yields the highest score is selected.

### 1.1.3.6 Multilayer perceptrons and artificial neural networks

The popular deep neural networks used in this thesis to extract text and image features all originated in *perceptrons*. Perceptron [53] is a simple mathematical function inspired by the human neuron. It is a simple linear model where each unit takes vector of values  $\mathbf{x}$  and weights  $\mathbf{w}$  on the input and using an activation function  $f$  calculates the output:

$$\mathbf{y} = f(\mathbf{w}^T \mathbf{x}) \quad (1.35)$$

The activation function  $f$  was originally the heaviside step function:

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \quad (1.36)$$

This model can be learned by updating the weights during iterating over a training set. Given a desired output  $y$  and an actual output  $\hat{y}$ , for each sample  $j$  update the weight  $w$  to a new weight  $\hat{w}$  with learning rate  $r$ :

$$\hat{w}_i = w_i + r(y_j - \hat{y}_j)x_{i,j} \quad (1.37)$$

By stacking multiple perceptrons in multiple layers we obtain the *multi-layer perceptron* (MLP) seen in Figure 1.3. It consist of an input layer, one or more hidden layers and one output layer. By adding multiple hidden layers the model can learn to solve problems that are not linearly separable such as the XOR problem. In the classical multilayer perceptron all nodes are fully connected, which means that each neuron has on its input the outputs of all neurons from the previous layer. Such a layer is also called a dense layer. Other typical types of layers include convolutional, recurrent and pooling layers.

The multilayer perceptron model usually learns by what is called the gradient descent backpropagation. It was popularized by [54] in the 1980's. The idea is to initialize the weights randomly and learn the correct weights by optimizing an error function while iterating over the training dataset. It is done by using the iterative gradient descent method for finding the local minimum of a error function. Given a predicted output value  $\hat{y}$  and actual value  $y$  from the training set, we can compute an error function  $E$  measuring how different the prediction was from the truth. The activation function  $f$  has to be differentiable. It is done by iteratively decreasing the error function in the direction of the negative gradient, because theoretically the function reaches its minimum the "fastest". A weight  $w_{ij}$  of node  $o_j$  with learning rate  $\eta$  is changed to new weight  $\hat{w}_{ij}$ :

$$\hat{w}_{ij} = w_{ij} + \Delta w_{ij} = w_{ij} - \eta * \frac{\partial E_{total}}{\partial w_{ij}} \quad (1.38)$$

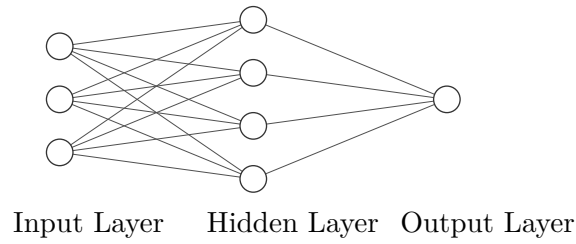


Figure 1.3: Illustration of a Multilayer perceptron model with one hidden layer. Created with [33].

The partial derivative of  $E$  in respect to weight  $w_{ij}$  can be calculated using the chain of derivatives with respect to the node input  $net_{o_j}$  and output  $out_{o_j}$  (for example a logistic function) for an output node  $o_j$ :

$$\frac{\partial E_{total}}{\partial w_{ij}} = \frac{\partial E_{total}}{\partial out_{o_j}} * \frac{\partial out_{o_j}}{\partial net_{o_j}} * \frac{\partial net_{o_j}}{\partial w_{ij}} \quad (1.39)$$

For a hidden node  $h_j$  we need to take into account the error on all of its outputs  $E_{total} = \sum_k E_{o_k}$ :

$$\frac{\partial E_{total}}{\partial w_{ij}} = \sum_k \frac{\partial E_{o_k}}{\partial out_{o_j}} * \frac{\partial out_{o_j}}{\partial net_{o_j}} * \frac{\partial net_{o_j}}{\partial w_{ij}} \quad (1.40)$$

Modern neural networks do not use the classical gradient descent method, because computing the gradient across a large training dataset is computa-

tionally inefficient or even unfeasible. Instead, they use modern algorithms such as stochastic gradient descent where the loss is not calculated across all training data but across only a small subsample. This does not guarantee that the descent will be in the “fastest” direction or even that it will be a descent at all. Nevertheless, this method is working if random subsampling is used. On large datasets with large models it performs much better than normal gradient descent.

### 1.1.3.7 Activation functions

Neurons can have a variety of different activation functions. Some of the popular ones are following (also pictured in Figure 1.4):

- **Identity**

$$f(x) = x \quad (1.41)$$

- **Heaviside** also called binary step

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases} \quad (1.42)$$

- **Logistic** also called sigmoid

$$f(x) = \frac{1}{1 + e^{-x}} \quad (1.43)$$

- **Hyperbolic tangent** also called TanH

$$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})} \quad (1.44)$$

- **Rectified linear unit** also called ReLU

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \quad (1.45)$$

- **Softmax**

$$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^J e^{x_j}} \quad \text{for } i = 1, \dots, J \quad (1.46)$$

The Softmax function is a special type of function on a whole layer, which takes a vector of outputs and normalizes it into a probability distribution with the range  $[0, 1]$  and  $\sum_i x_i = 1$ . This is a very important function as it is used as a final layer of classification networks.

Interestingly, the widely popular ReLU function is not differentiable at 0 and therefore it cannot be theoretically used as an activation function with backpropagating gradient descent. Practically, it performs well because the weights typically do not arrive at their local minima. Software implementations typically replace it by its one-sided derivative and because the calculation is subject to numerical errors anyway, the gradient descent works.

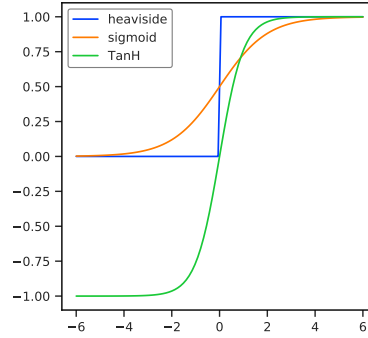


Figure 1.4: Examples of nonlinear activation functions used in neural networks.

### 1.1.3.8 Losses

The error function for optimization in neural networks, also called loss, is usually MSE. Among popular loss functions for predicted output  $\hat{y}$  and desired output  $y$  belong the following ones:

- **Root mean square error**, also known as RMSE

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2} \quad (1.47)$$

- **Mean square error**, also known as MSE, which is similar to RMSE but computationally less intensive

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (1.48)$$

- **Mean average error**, also known as MAE

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i| \quad (1.49)$$

- **Mean square logarithmic error**, also known as MSLE

$$\text{MSLE} = \frac{1}{n} \sum_{i=1}^n (\log(y_i + 1) - \log(\hat{y}_i + 1))^2 \quad (1.50)$$

### 1.1.3.9 Clustering

Clustering is an example of unsupervised machine learning where the objective is to split a set of observations into groups called clusters, where the instances are more similar to each other than to observations in other clusters.



**K-Means clustering** [34] is an iterative method. In the initialization step  $k$ , observations are selected as the foundation of new clusters. Then a number of iterations is performed until the algorithm converges or the maximum number of iterations is reached. The iteration step consists of two parts. In the first part, each observation is assigned to the cluster whose mean has the least euclidean distance from the observation. The next step consists of computing the new mean of the cluster (also called the centroid). The algorithm does not guarantee to find the global optimum and when using a distance other than the euclidean it does not guarantee to converge (in its original variation).

**Agglomerative hierarchical clustering** is a method that works by constructing a hierarchical clustering tree. It is initialized by assigning each data point into its own cluster. In the next step, two clusters with the smallest distance are found and combined into one. This step is repeated until only one cluster remains. The tree is then cut horizontally to get a desired number of clusters.

There is a number of different ways to compute the distance between two clusters. One called single-linkage takes the minimum of distance between all possible points between the two clusters, whereas in complete-linkage the maximum of such distances is used. A variant called Ward-linkage is a method where we find two clusters whose merging would minimize the increase in within-cluster distance which is the weighted squared distance between cluster centers.

**Birch clustering** [72] (Balanced iterative reducing and clustering using hierarchies) is a hierarchical clustering algorithm designed to perform over large datasets with the help of a clustering feature tree, which is a kind of tree that tries to preserve the clustering nature of the data. It works by operating on lower levels with agglomerative micro-clustering and the constructed CF tree and on higher level with macro-clustering integrating other clustering methods.

**Spectral clustering** [41] is an efficient method for clustering based on the k-means algorithm. A similarity graph of the objects is created using either  $n$  nearest neighbors or neighbors within distance  $\epsilon$ . Then we compute the eigenvectors of its Laplacian matrix and the smallest ones will serve as the object's new features. A normal k-means algorithm is then used to compute the clusters. This problem can also be reformulated as a weighted kernel k-means algorithm and vice versa.

---

Sentence 1	John likes to watch movies. Mary likes movies too.
Sentence 2	John also likes to watch football games.
Vocabulary	[John, likes, to, watch, movies, Mary, too, also, football, games]
BOW of “movies”	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
BOW of sentence 1	[1, 2, 1, 1, 2, 1, 1, 0, 0, 0]
BOW of sentence 2	[1, 1, 1, 1, 0, 0, 0, 1, 1, 1]

---

Table 1.1: Example of a bag of words embeddings.

## 1.2 Text embedding

Text embedding is a very common task in NLP as it is used for document classification, text summarization, machine translation, sentiment analysis, question answering, etc. Usually the text is parsed into tokens, e.g. words and the model is trained to infer the vector representation of the tokens. The models can learn the word embedding for example from a matrix representation of the corpora or by iterating on a local context window, where the vector representation of a word is learned while predicting other words within the context window. Modern word embedding methods are explained in this section.

### 1.2.1 Baseline models

One of the most basic text embedding is based on the frequency of words and is called the Bag of Words (BOW) or also count vector or one-hot encoding. For a given list of vocabulary  $V$ , the vector representing a certain is a vector  $v$  with the length  $|V|$ , which has all 0s except on the index of the word in the vocabulary. A vector embedding of a query, sentence or a document can be a similar vector where each index can indicate the presence or frequency of the word in the document such as in Table 1.1. The resulting vector has dimensionality equal to the size of the vocabulary which is often very high. Such a vector representation is also very sparse and impractical. Another disadvantage of this approach is that it completely disregards the document syntax.

One of the methods to enhance this approach is the Latent semantic analysis or LSA (explained in full in [11]). A document-term matrix  $X$  is constructed from the corpus, where the element  $X_{t,d}$  is the frequency of the term  $t$  in document  $d$ . Next, the document-term matrix is factorized. Every rectangular matrix  $X$  can be decomposed using singular value decomposition (SVD) into the product of three matrices  $U\Sigma V$ , where  $U$  and  $V$  are orthogonal matrices and  $\Sigma$  is a diagonal matrix. Using only  $k$  largest singular values of  $\Sigma$  gives us  $X_k$  which is the rank  $k$  approximation of  $X$  with the smallest error

and correspondingly matrices  $U_k$  and  $V_k$ . This gives us a lower dimensional representation of term  $t_i$  which is the  $i$ -th row of  $U_k$ .

In more advanced schemes, the frequency can be substituted for example with TF-IDF (Term frequency – inverse document frequency). For a term  $t$  and a document  $d$  in a corpus  $D$ , the TF-IDF weights are usually defined as:

$$\text{tfidf}(t, d, D) = f(t, d) * \log\left(\frac{|D|}{f(t, D)}\right) \quad (1.51)$$

where  $f(t, d)$  is the number of times term  $t$  appears in document  $d$ ,  $|D|$  is the size of the corpus  $D$  and  $f(t, D)$  is the number of times the term  $t$  appears in the corpus  $D$ . The TD-IDF weights are also often used to weigh other text embeddings that do not take word frequency into account.

This is only one of the variants of matrix factorization methods. Others can use a term-term co-occurrence matrix, such as Hyperspace Analogue to Language [36], where rows correspond to the words, and columns of the matrix correspond to the frequency of the word in a context window of another given word. The usual problem of these methods is that the most common words such as “and” or “the” carrying little meaning contribute too much to the model. A normalizing transformation of the matrix based on correlation or entropy of the words can be used to overcome this effect.

### 1.2.2 word2vec

Word2vec is a model introduced in [39] by a research group led by Tomáš Mikolov at Google. It is an unsupervised model used to produce a distributed representation of a word in a continuous vector space with hundreds of dimensions. One of its main features is that the vector representations of semantically similar words, i.e. words appearing in similar context, are in close proximity to each other. Furthermore, the continuous vector space allows for simple vector arithmetic such as summing two vectors, which can produce a meaningful result.

$$\begin{aligned} \text{czech} + \text{currency} &\approx \text{koruna} \\ \text{Vietnam} + \text{capital} &\approx \text{Hanoi} \\ \text{France} - \text{Paris} &\approx \text{Italy} - \text{Rome} \\ \text{copper} - \text{Cu} &\approx \text{zinc} - \text{Zn} \\ \text{king} + \text{man} - \text{woman} &\approx \text{queen} \end{aligned}$$

Another key feature is its small computational complexity and therefore the ability to learn from a huge corpora of text containing billions of words with millions of words in its vocabulary, whereas other previous architectures could only use hundreds of millions of words for learning before becoming computationally unfeasible.

The algorithm is based on a Neural Probabilistic Language Model [6] and is an extension to the Skip-gram model introduced by the same team previously in [38]. It treats words as atomic units which is on one hand simplistic and robust, on the other hand it lacks the notion of similarity between words and treats homonyms the same. Therefore both meanings of the word “bat”, an animal and a wooden club, have the same vector representation in spite of their vast difference. Still, this architecture can outperform other complex models.

### 1.2.2.1 Architecture

The architecture comes in two flavours: Continuous Bag-of-Words model (CBOW) and continuous Skip-gram model. The former one takes the context of current word as the basis for prediction, whereas the latter uses the current word to predict its context. With the increasing size of the context, the models produce better vector embedding at the cost of decreasing the speed of the algorithm. Although they seem like the same architecture mirrored as seen in Figure 1.5, there are several key differences.

The CBOW is a simple feed-forward neural network. It uses one hidden layer to predict the current word using its context. First we build a vocabulary  $V$  with all the words in the corpora. In the input layer with size  $|V|$ , each of the  $N$  surrounding words ( $N/2$  previous words and  $N/2$  following words) is encoded as a one-hot vector (similarly to a simple BOW model). The input layer is then projected to a hidden layer with size  $D$  (this size is a hyperparameter). It is finally projected to the output layer with size  $V$  with a Softmax activation function (in Figure 1.53) and the training criterion is the correct classification of the current word. The hidden layer has no activation function, but most importantly, the weights are used as the embedding of the target word after their training. Note that all the context words use the same hidden layer weights and the order of the words does not influence the output.

The continuous Skip-gram model is also a feed-forward neural network with input layer the size of  $|V|$ , a hidden layer and an output layer with size  $|V|$ . It uses the current word to predict words in a certain window around it. Since distant words are usually less related, they are given less weight by sampling less of distant words in the training. For a set maximum distance  $C$ , a random number  $R$  where  $1 \leq R \leq C$  is chosen and  $R$  previous and  $R$  following words are predicted.

$$\log p(w_c|w_j) \tag{1.52}$$

$$p(i|j) = \frac{\exp(\hat{w}_i \top w_j)}{\sum_{w \in V} \exp(\hat{w} \top w_j)} \tag{1.53}$$

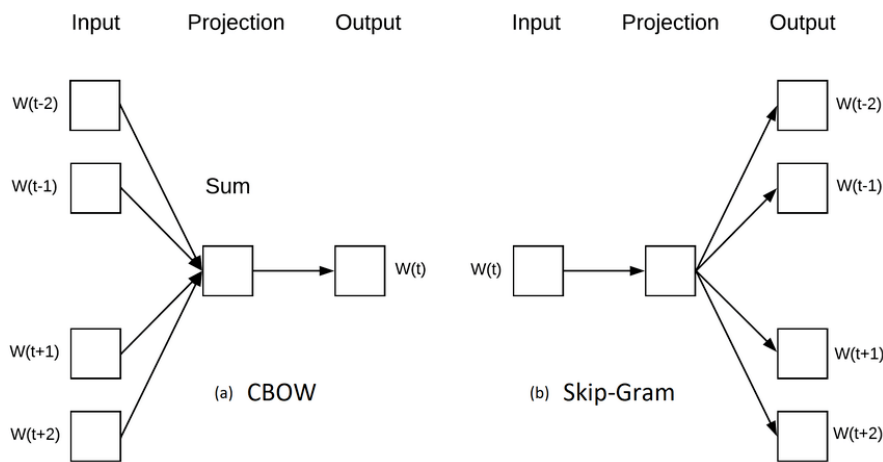


Figure 1.5: Continuous Bag-of-Words and Skip-gram architectures. Source: [39].

### 1.2.2.2 Optimization

Both models are then further optimized for computational speed and accuracy. The most computationally demanding part of the calculation is the Softmax activation function in Figure 1.53. This architecture introduces a Hierarchical Softmax, which instead of evaluating  $|V|$  output nodes uses a binary tree representation to evaluate only about  $\log_2 |V|$  nodes. Another technique is called Negative sampling. Instead of updating all the hidden neuron weights, only a few of the “negative” word weights are updated apart from the expected output word. A “negative” word means in this context a word that should not be predicted, i.e. a false prediction. According to the paper [39], only 2 to 5 “negative” words can be used for large datasets increasing the training speed by a few orders of magnitude.

The computational speed is also increased by subsampling frequent words. Some words as “the”, “a” or “in” carry less information and thus the training of each word is skipped with the inverse probability of its frequency when the frequency is over a certain threshold. Accuracy is also increased by introducing phrases. Some groups of words are treated as a single token instead of splitting them into multiple tokens in order to preserve their meaning. For example “New York Times” or “Steve Jobs” are treated as a single phrase because together they carry a very different meaning than their individual parts. According to the paper, the CBOW performs slightly better on syntactic tasks and the Skip-gram performs significantly better on semantic tasks, as a result the latter is usually the model of choice.

## 1. ANALYSIS

---

Probability and Ratio	k = solid	k = gas	k = water	k = fashion
$P(k \text{steam})$	$2.2 \times 10^{-5}$	$7.8 \times 10^{-4}$	$2.2 \times 10^{-3}$	$1.8 \times 10^{-5}$
$P(k \text{ice})$	$1.9 \times 10^{-4}$	$6.6 \times 10^{-5}$	$3.0 \times 10^{-3}$	$1.7 \times 10^{-5}$
$P(k \text{ice})/P(k \text{steam})$	8.9	$8.5 \times 10^{-2}$	1.36	0.96

Table 1.2: Word co-occurrence probabilities. Adapted from [46].

### 1.2.3 GloVe

Another popular method for learning vector space representations, which builds also upon the Word2Vec method, emerged just a year later in 2015. It is called GloVe for Global Vectors and it was introduced in [46]. It is another unsupervised statistical model that combines the advantages of global matrix factorization described in Subsection 1.2.1 and the local context window method popularized by the Word2Vec model in the previous Subsection 1.2.2.

Before Word2Vec, there was a trend of learning larger and larger neural networks such as in [10] to learn the word representation from the full context of the word. The Word2Vec algorithm used only a single-layer neural network and had a great success showing that even relatively simple architecture can cope with current NLP tasks. More algorithms improving this concept soon emerged, such as the vLBL model in [40]. The authors of the GloVe algorithm especially valued its capacity to learn linguistic pattern such as linear relationships between the vector embeddings of the words, because it also meant that the dimensions of the embedding also carried some meaning unlike in common matrix factorization methods.

#### 1.2.3.1 Architecture

Unlike the Word2Vec model, GloVe takes advantage of the statistical information of the learning corpus and builds upon the word-word co-occurrence matrix while also taking into account the word context. In other words, the elements of the matrix represent how many times a certain word appeared in the text next to another word within a certain window. However, the main principle of GloVe is not to use the co-occurrences themselves but their ratios because they carry more meaning. This is described in Table 1.2 on words “steam” and “ice”. The table shows that the probability of them occurring in the context of another word that is either common to both of them (“water”) or unrelated to both of them (“fashion”) is relatively the same, so the ratio is close to 1. But a word such as “gas” would appear only in the context of “steam” and not “ice”. Such probabilities are easily computed from the co-occurrence matrix.

Formally, let  $X$  be the word-word co-occurrence matrix with elements  $X_{ij}$

denoting how many times the word  $j$  appeared in the context of the word  $i$ .  $X_i = \sum_k X_{ik}$  is then the number of times the word  $i$  appeared in the proximity of any word and  $P_{ij} = P(j|i) = X_{ij}/X_i$  is the probability the word  $i$  appears in the context of word  $j$ . Now let  $w$  be the two word vectors and  $\tilde{w}$  a separate context word vector. From this we can finally derive the following equation:

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ij}}{P_{jk}} \quad (1.54)$$

At this point, by using a number of properties we want to achieve, we can derive the final GloVe equation. All details will not be described, but they can be seen in the original paper [46]. One of the properties is that the function should be a simple arithmetic function (unlike using neural networks) in order not to obfuscate its function. The authors therefore took the simple difference between the two word vectors. Another property is a linear relation between the words and their context word so a dot product between them was used. This gives the equation:

$$F(\text{dot}(w_i - w_j, \tilde{w}_k)) = \frac{P_{ij}}{P_{jk}} \quad (1.55)$$

Another property is homomorphism between the groups  $(\mathbb{R}, +)$  and  $(\mathbb{R} > 0, \times)$ :

$$F\left((w_i - w_j)^T \tilde{w}_k\right) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)} \quad (1.56)$$

The solution to those equations is  $F = \exp$  or:

$$w_i^T \tilde{w}_k = \log(P_{ik}) = \log(X_{ik}) - \log(X_i) \quad (1.57)$$

Next, we added two biases: one for  $\tilde{w}_k$  and the second one for  $w_i$  which absorbs the term  $\log(X_i)$  independent of  $k$ . The addition of bias accounts for the varied occurrences of different words. Because the least frequent co-occurrences can amount to noise and the most frequent co-occurrences with little meaning would dominate the final function, a weight function is introduced. The final equation with the weight function is as follows:

$$\sum_{ij} \text{weight}(X_{ij})(\text{dot}(w_i, \tilde{w}_j) + b_i + \tilde{b}_k - \log(X_{ij}))^2$$

$$\text{weight}(x) = \min(1, (x/x_{max})^\alpha)$$

where  $x_{max}$  is set to 100 and  $\alpha$  is set to 3/4, both set empirically.

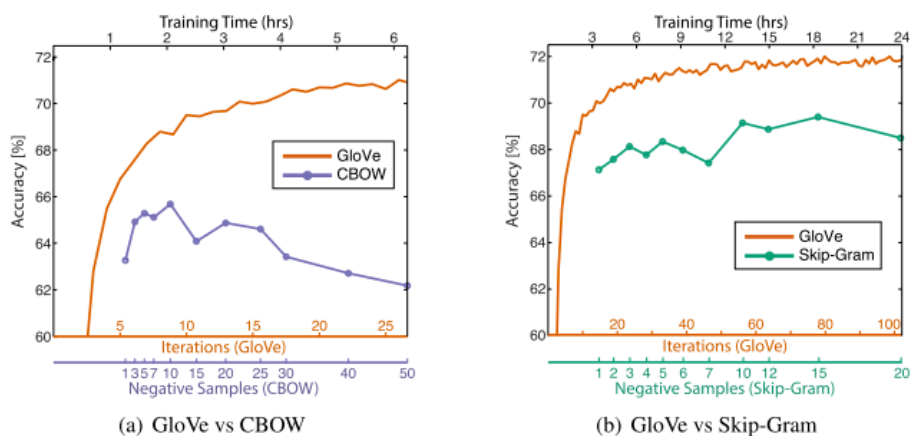


Figure 1.6: Comparison of learning time of GloVe and word2vec.

### 1.2.3.2 Comparison to Word2vec

Although the GloVe model is based on different assumptions and uses different methods for optimization, they surprisingly perform quite similarly and, as the authors of the paper show, are mathematically closely related. The word2vec model uses a softmax function to estimate the probability with which a word  $i$  appears in the context of a word  $j$ . This actually means that it computes the cross-entropy between the actual and predicted distributions of word  $i$  in context of word  $j$  weighted by their co-occurrence. Hence, the optimization function differs only in the loss function, where word2vec uses the cross-entropy and GloVe uses log mean squared error. The main difference is however the training method where word2vec learning complexity scales with the size of the corpus  $|C|$ , while GloVe scales with the number of non-zero elements of the matrix  $X$ , which is at most the same complexity as in word2vec. The training times from the papers can be seen in Figure 1.6.

### 1.2.4 FastText

FastText is an open-source algorithm to extract text embeddings released by the Facebook AI Research team in 2016. It is another model that improves the word2vec models. The main feature of this model is that it does not ignore the word morphology by assigning each word a distinct vector. Instead, it learns on character n-grams and as such can even produce word representation for words out of the vocabulary. Some methods implementing character n-grams of even the sole characters have been used in the past. Most similar to the FastText algorithm are [58] and [69], but this method learned only on a paraphrase pair corpus, while FastText learns on any corpus.



### 1.2.4.1 Architecture

The main idea is to split the words into n-grams (with added boundary symbols  $\langle$  and  $\rangle$ ). For  $n = 3$  the word *where* will be represented as trigrams  $\langle \mathbf{wh}, \mathbf{we}, \mathbf{her}, \mathbf{ere}$  and  $\mathbf{re} \rangle$ . Note that the trigram  $\langle \mathbf{her} \rangle$  for the word *her* is different than the trigram  $\mathbf{her}$  for the word *where*. For the training of the models, all possible n-grams for  $3 \leq n \leq 6$  are used, although different sets of  $n$  can be used.

The original word2vec model uses a softmax function to define the probability of a context word and maximizes the average log probability over given context, as seen in Equations 1.53 and 1.52. That is not a suitable solution for this model, because it predicts only one word at a time. Instead the problem is reformulated to a set of independent binary classification tasks. For the given word  $t$  and the context word  $c$ , their vector representations  $w_t$  and  $w_c$  and a set of randomly sampled words  $N$  we can obtain so-called negative log-likelihood:

$$\log \left( 1 + e^{-s(w_t, w_c)} \right) + \sum_{w_n \in N} \log \left( 1 + e^{-s(w_t, w_n)} \right) \quad (1.58)$$

This function uses a custom scoring function  $s$ , which for a given set of n-grams  $G_t$  appearing in the word  $t$  is a sum of dot products between two vectors  $w_g$  and  $w_c$ :

$$s(w_t, w_c) = \sum_{g \in G_t} w_g^T w_c \quad (1.59)$$

In order to be memory efficient, the model uses a hashing function to map n-grams to integers. The given word embedding is then produced by taking its index in the word dictionary and the set of its hashed n-grams.

To optimize the loss function, the model uses the stochastic gradient descent variation called the Hogwild [50] algorithm. By using this algorithm, the model can learn in parallel. It is still approximately  $1.5\times$  slower than the base word2vec model, but performs generally better than both word2vec Skip-gram and CBOW on syntactic tasks and similarly to Skip-gram on semantic tasks.

### 1.2.5 Conceptnet Numberbatch

ConceptNet [60] is not a model with the sole purpose of obtaining word embeddings from a given corpus, although it is one of its uses. It is a semantic network of knowledge about word meanings, i.e. a knowledge graph where nodes are words or phrases and the edges are labelled with relationships of the connecting nodes. The purpose of such network is to represent the general human knowledge and allow NLP applications to better understand the meaning behind words. It was built using many sources including other sister

## 1. ANALYSIS

---

projects, Wikitionary, Open Multilingual WordNet, DBpedia, and others. It should be noted that the graph is multilingual.

# en machine learning

An English term in ConceptNet 5.6

Sources: Open Mind Common Sense contributors, DBpedia 2015, JMDict 1.07, OpenCyc 2012, English Wiktionary, and French Wiktionary  
View this term in the API

[Documentation](#) [FAQ](#) [Chat](#) [Blog](#)

Synonyms	Related terms	Types of machine learning	Links to other sites
<ul style="list-style-type: none"><li><a href="#">cs</a> strojové učení →</li><li><a href="#">ja</a> 機械学習 (n) →</li><li><a href="#">fr</a> apprentissage statistique (n) →</li><li><a href="#">de</a> maschinelles lernen →</li><li><a href="#">es</a> aprendizaje automático →</li><li><a href="#">et</a> masinõppimine →</li><li><a href="#">fi</a> koneoppiminen →</li><li><a href="#">fr</a> apprentissage automatique →</li><li><a href="#">hu</a> gépi tanulás →</li><li><a href="#">is</a> vélrænt nám →</li><li><a href="#">it</a> apprendimento automatico →</li><li><a href="#">ja</a> 機械学習 →</li><li><a href="#">pt</a> aprendizado de máquina →</li><li><a href="#">pt</a> aprendizagem automática →</li><li><a href="#">sv</a> maskinlärning →</li></ul>	<ul style="list-style-type: none"><li><a href="#">en</a> algorithm →</li><li><a href="#">cs</a> strojové učení (n) →</li><li><a href="#">en</a> automated machine learning (n) →</li><li><a href="#">en</a> learn →</li><li><a href="#">fr</a> apprentissage statistique →</li><li><a href="#">en</a> bootstrap aggregating (n) →</li><li><a href="#">en</a> data science (n) →</li><li><a href="#">en</a> deep learning (n) →</li><li><a href="#">en</a> ground truthing (n) →</li><li><a href="#">en</a> tron →</li><li><a href="#">es</a> aprendizaje automático (n) →</li><li><a href="#">fi</a> koneoppiminen (n) →</li><li><a href="#">is</a> vélrænt nám (n) →</li><li><a href="#">ja</a> 機械学習 (n) →</li><li><a href="#">zh</a> 機器學習 (n) →</li><li><a href="#">en</a> artificial neural network →</li><li><a href="#">en</a> data compression →</li></ul>	<ul style="list-style-type: none"><li><a href="#">en</a> discriminative weight learning (n) →</li><li><a href="#">en</a> a neural network →</li><li><a href="#">en</a> generative weight learning (n) →</li><li><a href="#">en</a> machine rule induction (n) →</li></ul>	<ul style="list-style-type: none"><li><a href="#">sw.opencyc.org</a> MachineLearning →</li><li><a href="#">umbel.org</a> MachineLearning →</li><li><a href="#">en.wiktionary.org</a> machine_learning →</li><li><a href="#">fr.wiktionary.org</a> machine_learning →</li></ul>
<b>Context of this term</b> <ul style="list-style-type: none"><li><a href="#">en</a> artificial intelligence →</li><li><a href="#">fr</a> intelligence artificielle →</li></ul>	<b>Terms with this context</b> <ul style="list-style-type: none"><li><a href="#">en</a> ensemble (n) →</li></ul>	<b>machine learning is a type of...</b> <ul style="list-style-type: none"><li><a href="#">en</a> computer activity (n) →</li></ul>	<b>In the genre of machine learning</b> <ul style="list-style-type: none"><li><a href="#">en</a> weka (n) →</li></ul>


 ConceptNet 5 is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License. If you use it in research, please cite this AAAI paper. See Copying and Sharing ConceptNet for more details.

Figure 1.7: Example of a ConceptNet node available at conceptnet.io. [60]

This network can also be used to create a model called Conceptnet Numberbatch (to distinguish between the knowledge network and the embedding model itself). In the previous versions of the model, the embeddings were derived from the knowledge graph by singular value decomposition (briefly explained in Subsection 1.2.3). Recent advancements in the field allowed for the use of a combination of the ConceptNet and a distributional semantics model (such as word2vec or GloVe) using a variation of a retrofitting technique introduced in [15]. Such feature space model performs better on word-relatedness tasks and analogy tasks, for example SAT-style analogy questions.

### 1.2.5.1 Word embedding retrieval

The word embeddings can be obtained from the knowledge graph in a number of ways. The most relevant to ConceptNet is the Holographic Embedding method [42], which is a method to learn compositional vector space representations of entire knowledge graphs using circular correlation of the graph. However it has not yet been successfully implemented on ConceptNet.

Another method is to construct a term-term matrix, where elements of the matrix represent the weighted sum of edges between the two term nodes. Before constructing this matrix the graph is pruned by discarding all nodes with fewer than 3 connections. This matrix is similar to the co-occurrence matrix of word2vec and GloVe with the difference that context of the word is not inferred by the proximity in a corpus but by relationships of the terms in the knowledge graph. The word embeddings are calculated from the matrix using positive pointwise mutual information:

$$\text{PPMI}(w, c) = \max \left( \log_2 \frac{P(w, c)}{P(w)P(c)}, 0 \right) \quad (1.60)$$

The matrix is then truncated to 300 dimensions using singular value decomposition. The concept of using PPMI is similar to using ratios of word co-occurrences in GloVe. Pruned nodes can be reconstructed using the weighed average of its neighbors.

The method used to construct Conceptnet numberbatch from the knowledge graph is another method called Retrofitting introduced in [15]. It uses a knowledge graph to adjust a pre-existing matrix of word embeddings, such as from word2vec or GloVe. The new vectors  $q_i$  are retrofitted to  $\hat{q}_i$  with the objective of staying close to their neighbors with edges  $E$  in the knowledge graph. The objective function is:

$$\Psi(Q) = \sum_{i=1}^n \left[ \alpha_i |q_i - \hat{q}_i|^2 + \sum_{(i,j) \in E} \beta_{ij} |q_i - q_j|^2 \right] \quad (1.61)$$

The  $\alpha$  and  $\beta$  parameters are used to control the relative weights of the associations, and they can be inferred from the edge weights of the knowledge

graph. This method benefits from the multilingual property of the graph as it can learn more about the words from their translations to other languages. One last step is that the authors subtracted the mean of vectors and normalized them again to unit vectors in order to lower the influence of highly connected nodes such as “person”, “work” or “say”.

This approach can also benefit from ensemble retrofitting of multiple matrices. The authors applied the technique on both word2vec and GloVe matrix, concatenated them by column and again reduced their dimensionality to 300 with the use of truncated singular value decomposition. Again, the words not present in vocabularies of word2vec and GloVe were reconstructed using the average of vectors of their neighboring nodes.

### 1.2.6 BERT

In recent years there has been a great advance in using deep learning to solve NLP tasks. Many different models have been introduced and it remains to be seen which one of them stands the test of time and turns out to be the most influential of them all. One of the most promising seems to be the BERT [13] which builds upon many influential state-of-the-art ideas.

#### 1.2.6.1 Contextual language models

The previous shallow embeddings, such as word2vec, GloVe or FastText, suffered from the fact that every word was described by a single vector embedding which tried to capture its meaning in all contexts together. However, this approach is imperfect because words can have different meaning depending on the circumstances. In the sentences “Bat is the only mammal capable of sustained flight” and “I will hit the ball with my bat” the word “bat” carries completely different meanings.

One of the papers that popularized modern contextual language models is the ELMo model [47]. Contextual language models were previously employed to learn special Residual Neural Networks (RNNs) called Long Short-term Memory models (LSTM) [24] where the objective was to predict the next word in a given sentence. Nevertheless, the drawback of RNNs is that they remember only one state at the time and form the so-called long range dependencies pictured in Figure 1.8. In the task of machine translation and sentence generation, it becomes a big problem because at the end of the sentence, the RNN has to remember every previous word and their semantic and syntactic relationship in its memory, which turned out to be a major flaw. The ELMo model solves this problem with two techniques.

The language model still tries to predict the next word in a sequence, but the final vector is derived from all the successive hidden states of the language model. In this case the language model does not have to carry all the information about the beginning of the sentence to the end.

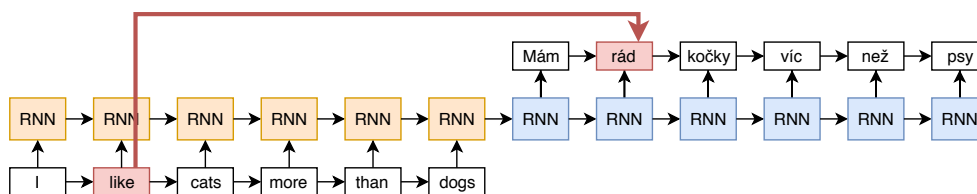


Figure 1.8: Example of long range dependency of words in RNN translators.

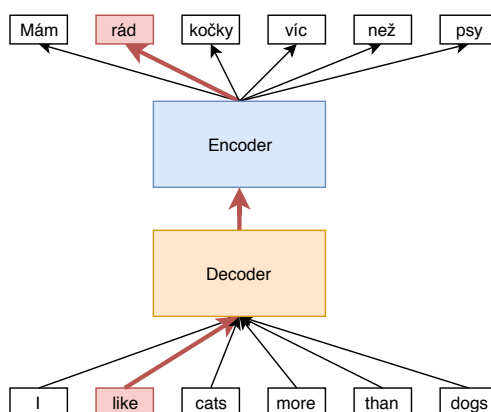


Figure 1.9: Example of short range dependency of words in translation with Transformers.

The second technique is an improvement of the first. Even though humans usually read text from left to right, doing so in a language model does not necessarily contain all the information available about the sequence. The ELMo model therefore utilizes the so-called bi-directional language models, which not only try to guess the next word in a sequence, but also learn on reversed sequences and try to predict all previous words. This improves its ability to learn the meaning of words even at the beginning of the sentence.

### 1.2.6.2 Attention

The bi-directional language models used in ELMo have one drawback: they do not use the previous and the following context at the same time, but in separate individual models. BERT takes the bi-directional language models one step further and uses what could be called omni-directional language model, where the model takes all the tokens in the sequence into account at the same time (seen in Figure 1.10). This is not possible with LSTMs, because the notion of a “next” word vanishes in this situation, and so the BERT model uses multiple Transformer encoder blocks [66]. The Transformer does not use a recurrent network to remember its previous states, but works directly with all the inputs with what is called attention using multiple attention heads.

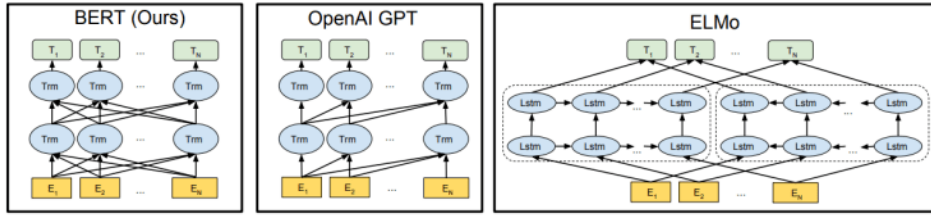


Figure 1.10: Illustration of differences in pre-training model architectures of BERT, OpenAI GPT and ELMo. Source: [13].

The Transformer is a model developed for machine language translation; it uses stacks of encoders and decoders. Its major advantage is its lack of long range dependencies (pictured in Figure 1.9). A decoder was the intuitive part of the transformer to use in language model learning as its task is to predict a next token in sequence, which is exactly what directional context models use. This approach was used in an OpenAI model [49]. Because the decoder works only in one direction, BERT uses the encoder part of the transformer which introduces a whole new omni-directional approach. The encoder looks at the whole sequence. It also enables the fine-tuning of the model, which was popularized in NLP by [27] and applied to Transformers in the OpenAI model [49].

### 1.2.6.3 Learning

With the omni-directional all-context approach, the training task of predicting the next word in a sequence breaks down, because in its multi-layered set of encoders, the token embedding would be influenced by itself via other heads. BERT solves this problem by masking the target tokens. During the training, 15% of tokens are chosen at random and replaced by the token [MASK] and the task is to guess what the correct word in the position should be. This works reasonably well, but the training is only happening on the token where the mask is. To solve this problem, instead of iteratively replacing all the tokens, 10% of the chosen tokens are replaced not by a mask, but a completely different token. 10% are left as the original words, forcing the model to learn whether the token given makes sense in its context for all the tokens all the time.

A second part of the learning phase is introduced to aid the model in various two-sentence NLP tasks including answering. The training consists of feeding the model with two consecutive sentences differentiated by introduction of additional segment embedding showing the classifier which tokens belong to which sentence. The model is given two sentences A and B and the model's task is to learn whether the sentence B is the sentence that follows A or not. The sentence B is the following sentence with a 50% chance and 50% it is a random sentence. Apart from the sequence embedding supplied at the

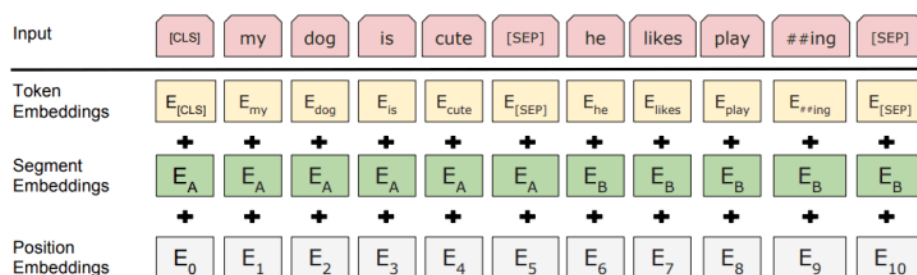


Figure 1.11: Representation of BERT model input with token, segmentation and positional embeddings. Source: [13].

input, the model also uses position embeddings, which are just indices of all tokens as seen in Figure 1.11.

#### 1.2.6.4 Embedding

For classification tasks, the output has to ultimately be a single vector. There are multiple ways to do it, such as the average or max of all individual tokens, but the researchers used a different idea. At the beginning of each input sequence of tokens is a special token [CLS]. At the end of the model the vector representing this token is fed to a classifier (such as a single-layer softmax). This results in a better performance than using the average or max pooling.

Although fine-tuning BERT to a specific task can improve performance, it can be used only as a feature extractor and still performs at a very good level.



	Base model	Downstream tasks	Downstream model	Fine-tuning
ELMo [47]	two-layer biLSTM	feature-based	task-specific	none
ULMFiT [27]	AWD-LSTM	model-based	task-agnostic	all layers; with various training tricks
OpenAI GPT [49]	Transformer decoder	model-based	task-agnostic	pre-trained layers + top task layer(s)
BERT [13]	Transformer encoder	model-based	task-agnostic	pre-trained layers + top task layer(s)

Table 1.3: Comparison of deep-learning models and their architecture design. Adapted from [68].

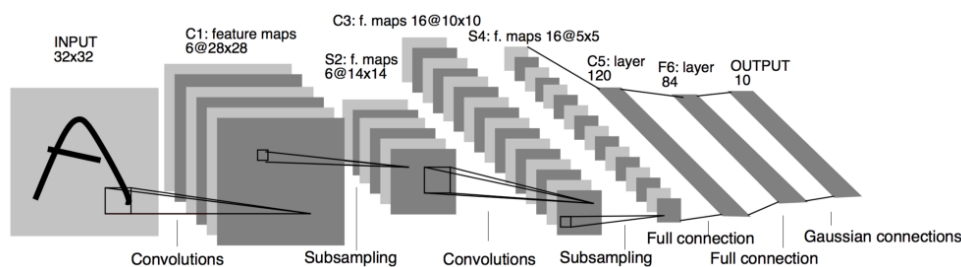


Figure 1.12: Architecture of the LeNet 5 neural network. Source: [32]

### 1.3 Image embedding

Feature extraction from images plays a big role in machine learning as it is used for many modern applications as optical character recognition, face recognition and many more applications in computer vision. Before the popularity of deep convolutional neural networks (CNNs), the main technique was using hand-picked features, such as SIFT [35] or SURF [5]. In the last decade, large neural networks and their training became achievable thanks to the increase of hardware capabilities. Before we delve into deep convolutional neural networks, let us have a look at their history.

#### 1.3.1 AlexNet

The pioneer of convolutional neural networks in image classification is LeNet5 [32] published in 1998 (pictured in Figure 1.12). Its purpose was to automatically classify hand-written digits on bank cheques digitized to  $32 \times 32$  pixel images. From our point of view, it is a simple 7 layer neural network, but at the time the hardware was the main constraint and significantly limited the size of neural networks. Its key features were convolutional layers (shown in Figure 1.13), where the input of a neuron is not from all neurons in the previous layer as in a fully-connected dense layer, but only from a window of a much smaller size, called filters. This enables the neural network to learn not only local features, but also global features in the deeper layers. This architecture achieved just 0.7% error rate on classification of the MNIST dataset of handwritten digits.

Another revolution came with the advance in hardware and training of neural networks on GPUs. In 2012, a neural network called AlexNet introduced in [31] won the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [56]. The competition evaluates algorithms for object detection, localization and classification of images and videos at large scale using a subset of the ImageNet dataset consisting of 150 000 photos collected from Flickr and other search engines and their hand-picked labels of 1000 categories (one instance can have one or multiple labels). The AlexNet won

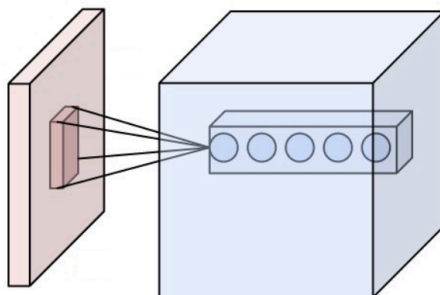


Figure 1.13: Example of a convolutional layer in neural network. It takes only a part of previous layer on the input in contrast to a dense layer. [3]

with 16.4% top-5 accuracy error in classification task by a large margin of 9.8% and similarly overwhelming result in the localization task. Its novel ideas included non-saturating ReLU activation, local response normalization and overlapping pooling. To reduce overfitting, the neural network also uses a dropout function for neurons and a dataset augmentation.

The architecture of AlexNet consists of five convolutional layers with overlapping pooling on layers 1, 2 and 5 and local response normalization on layers 1 and 2, and three fully connected dense layers with a dropout function on the first and the second dense layers with probability 0.5 during training, depicted in Figure 1.14. The most important innovation was the non-saturating ReLU activation function (Figure 1.4):

$$f(x) = \max(0, x) \quad (1.62)$$

where the standard activation at the time was a saturating *tanh* function

$$f(x) = \tanh(x) \quad (1.63)$$

or a similar one. A saturating activation function means the gradient closes to zero with large absolute values [20]. Gradients close to zero will update the weights by a very small amount and can eventually even stop learning, which is called the vanishing gradient problem, described fully in [23]. The ReLU activation function keeps the gradient large enough to avoid the problem and even make the learning times faster by up to six times according to the paper [31].

Even though the ReLU activation does not require a normalization to avoid saturating, the AlexNet uses the Local Response Normalization to aid generalization. In short, this technique implements a localized inhibition of neurons based on activities in its neighbourhood in order to create a competition for big activities in neurons. One of the ways to avoid overfitting, which is a problem of convolutional neural networks, is to use overlapping pooling layers (pictured in Figure 1.15). Traditionally, the pooling layer consisted of

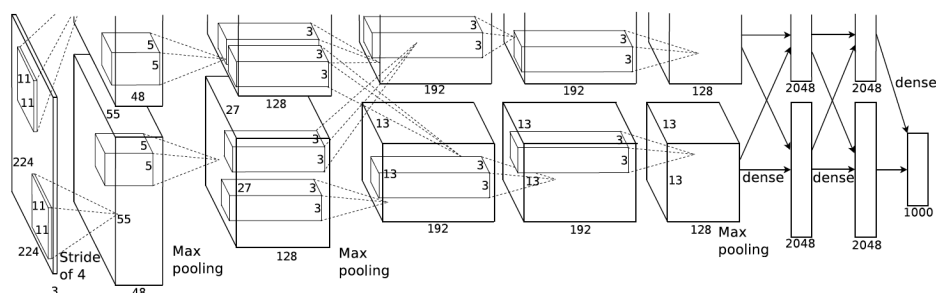


Figure 1.14: Depiction of AlexNet neural network on two GPUs with connections only of some layers. [31]

a grid of pooling units with size  $s \times s$  spaced  $s$  pixels apart. In the overlapping pooling layers, the units are spaced  $t$ ,  $t < s$  pixels apart resulting in the units overlapping by  $s - t$  pixels. Apart from being less likely to overfit, it also increases accuracy by a small amount. The main feature to reduce overfitting was augmenting the dataset by training on subsamples with size  $224 \times 224$  of the original  $256 \times 256$  pixel images and their horizontal reflections. This increased the training set 2048 times. In the testing phase, only the corner and center images with their horizontal reflections was used, and their softmax layers were averaged to produce the final result leading to yet another increase in accuracy. A significant boost to the speed of training was achieved by using a dropout function in some layers, which sets the output of a neuron to 0 with a set probability, in our case 0.5, and therefore does not contribute to learning and can be omitted in that phase. In the test phase, all neurons are scaled by a factor of 0.5 too to approximate the inputs during the learning phase.

### 1.3.2 InceptionNet

#### 1.3.2.1 Version 1 - GoogLeNet

The first version of InceptionNet [63], also called the GoogLeNet, came in 2014, two years after AlexNet, and won the annual ImageNet competition. Its basic idea is to go even deeper with layers as stated by the authors and their reference to the famous internet meme [30] from the movie Inception. The network implemented a lot of new ideas from other state-of-the-art architectures.

One of the biggest constraints of deep neural networks was their size and computational demands. The AlexNet had over  $60M$  parameters and used multiple top-of-the-line GPUs to train for 3 weeks. That is why one of the main ideas of InceptionNet was to reduce the size of the network and the number of its parameters to allow for deepening of the architecture. The

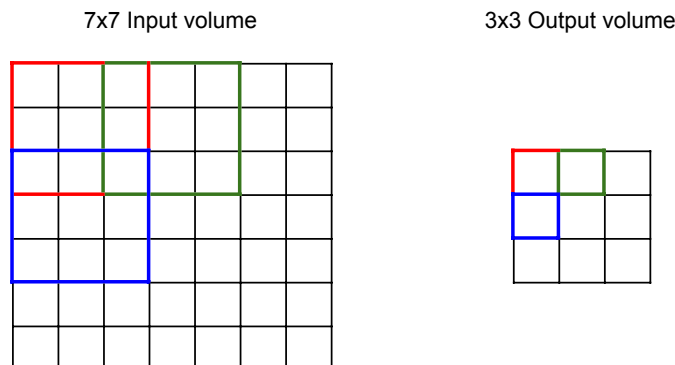


Figure 1.15: Depiction of overlapping pooling layer with size 3 and stride 2.

architecture of InceptionNet therefore used  $1 \times 1$  convolutional layers before its other convolutional layers, because a  $1 \times 1$  layer still reduces the dimensionality of the variables. For example, for an input with 256 depth to apply  $5 \times 5$  convolution to get output with also 256 depth, the number of parameters is  $256 \times 5 \times 5 \times 256$ . If we apply a  $1 \times 1$  convolution before the second one to reduce the depth from 256 to 64, the number of parameters is now  $256 \times 1 \times 1 \times 64 + 64 \times 5 \times 5 \times 256$ . The reduction of parameters is  $(256 \times 256 \times 25) / (256 \times 64 \times 26) \approx 3.85$  times, which also means a 3.86 increase in speed while retaining accuracy.

This reduction of parameters allowed the architecture not only to go deeper, but also wider, which was the main purpose of the Inception module. The idea is that the information in pictures is distributed both locally and globally depending on the photo. Traditionally, the local features were extracted first and iteratively the global features were extracted from the previous local features by stacking layers upon layers. The computational complexity rises exponentially with number of layers so the authors used the approach to go wider and introduced the Inception module. The idea is to stack the layers not vertically but horizontally and concatenate their outputs (pictured in Figure 1.16). The next layer then has the option to choose either the local or more global features by learning the weights. A max pooling layer is also added because empirically it increased the performance. The naive implementation was complemented with the addition of  $1 \times 1$  convolutional layers to reduce its parameter count.

The architecture stacked 9 Inception modules resulting in 22 layer deep architecture (or 29 including pooling layers). To further reduce the number of parameters, the final layer before softmax was replaced by a global average

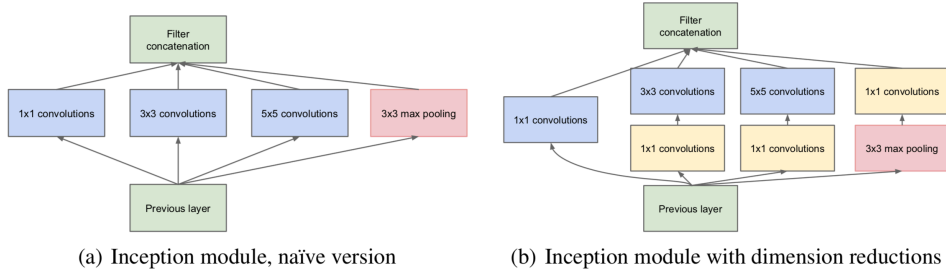


Figure 1.16: Depiction of two inception modules – naïve version and improved version with less parameters. Source: [30].

pooling layer, which is a normal pooling layer with the maximum size. Not only does this reduce the number of parameters but also increases the accuracy. To prevent overfitting and speed up the learning, all the layers used the ReLU activation function. Because deep neural networks are hard to learn as the weights propagate to the start very slowly, the auxiliary classifiers were introduced. The loss was computed from the average of the output classifier and the loss of auxiliary classifiers reduced by 70%. The usual softmax function was used for classification and also a dropout was used on the layers. The overall architecture can be seen in Figure 1.17.

### 1.3.2.2 Version 2

The next iteration of the InceptionNet came a year later and was the second neural network to outperform human experts at the recognition task of the ImageNet competition, setting a new record. There is some confusion about the versions of the architecture, but in this section, we will refer to the one described in [28]. It introduced one main innovation: the batch normalization. Let us denote a batch of input samples  $\mathcal{B} = \{x_{1\dots m}\}$  and learned parameters  $\gamma, \beta$  representing scale and shift. The equations for batch mean  $\mu_{\mathcal{B}}$ , variance  $\sigma_{\mathcal{B}}^2$ , normalized input  $\hat{x}_i$  and output  $y_i$  are:

$$\mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^m x_i \quad (1.64)$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad (1.65)$$

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad (1.66)$$

$$y_i = \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad (1.67)$$

### 1.3. Image embedding

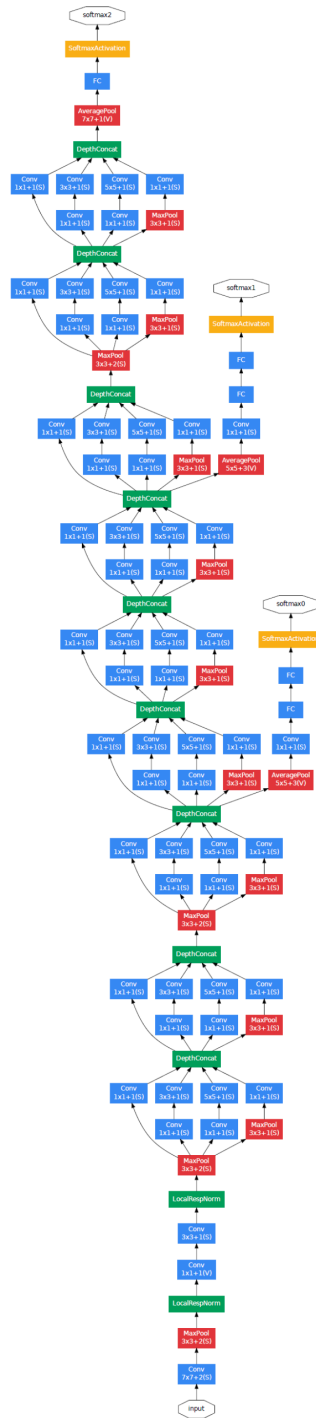


Figure 1.17: Architecture of GoogLeNet with multiple Inception modules and two auxiliary classifiers. Source: [63].

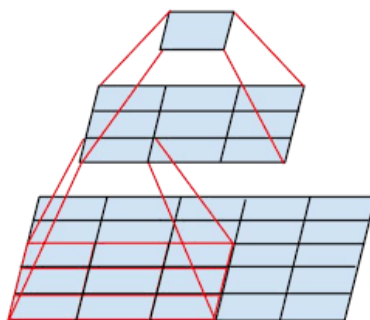


Figure 1.18: Factorization of  $5 \times 5$  convolutional layer into two  $3 \times 3$  convolutional layers. This reduces number of parameters by 64%. Source: [64].

With the use of the popularized ReLU activation function came the need for careful learning rate initialization and subsequent reduction. Because of sensitivity of ReLU to the learning rate, it can be advantageous to have it relatively the same over the course of training. This is done with batch normalization and the result is that higher initial and overall learning rate can be used, speeding up the training drastically while also increasing the accuracy. Likewise, the overfitting of the network is reduced with the result that dropout can be limited or completely removed. Apart from batch normalization, this paper also mentioned a convolutional unit factorization which will be described in the following subsection.

### 1.3.2.3 Version 3

The third version from 2016 introduced in [64] did not win the annual ImageNet competition, but came second after Microsoft's ResNet, which will be described in Subsection 1.3.4. The main gist of the paper is to optimize the network and remove the so-called representational bottleneck. Such a bottleneck is present when a layer significantly reduces the dimensions, constricting the information flow. The network was redesigned to gently decrease the representational size and balance its width and depth. Both width and depth are also increased to produce a higher quality network, while retaining its computational complexity by introducing the following features.

Inspired by the VGGNet [59], the network's computational intensity was decreased by the use of convolution layer factorization or decomposition. The idea is that a convolutional unit of size  $5 \times 5$  has the same extent as two  $3 \times 3$  units stacked on top of each other as shown in Figure 1.18 while reducing the number of parameters by a quarter ( $3 \times 3 + 3 \times 3 < 5 \times 5$ ). This idea was taken one step further by factorizing into asymmetric convolutions shown in Figure 1.19. The  $3 \times 3$  unit can be further factorized into two asymmetrical  $3 \times 1$  and  $1 \times 3$  units, reducing number of parameters by a third. With these techniques, the Inception module split into three variants.



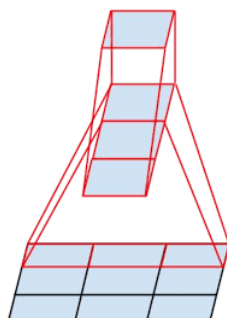


Figure 1.19: Factorization of  $3 \times 3$  convolutional layer into  $1 \times 3$  and  $3 \times 1$  convolutional layers. This reduces number of parameters by  $1/3$ . Source: [64].

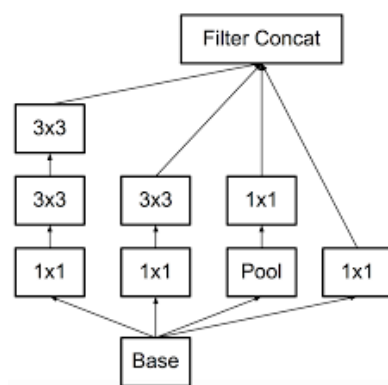


Figure 1.20: Inception module A with a factorized  $5 \times 5$  convolutional layer. Source: [64].

- Module A, which is the standard module with the  $5 \times 5$  unit replaced by two  $3 \times 3$  units (depicted in Figure 1.20).
- Module B, which is a wide module with two  $1 \times 7$  and two  $7 \times 1$  units and with only one  $1 \times 7$  and one  $7 \times 1$  units in parallel (depicted in Figure 1.21).
- Module C, which is a module promoting high dimensional representations and has combinations of  $3 \times 3$ ,  $1 \times 3$  and  $3 \times 1$  units (depicted in Figure 1.22).

The representational bottleneck from feature downscaling by a max pooling layer followed by a convolutional layer cannot be solved by switching the two layers as it would become computationally too expensive. The authors solved it by another inception module which combined side-by-side pooling and  $3 \times 3$  units with larger strides.

The number of auxiliary classifiers decreased from two to one as its purpose of allowing deep networks to learn was no longer valid as they could learn

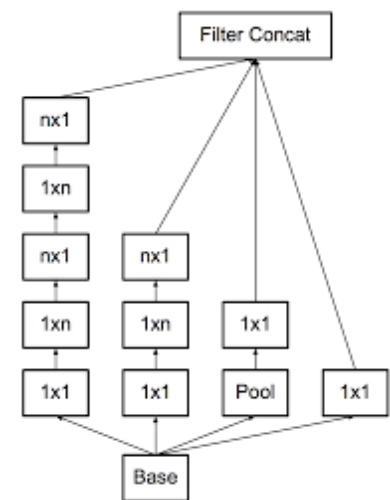


Figure 1.21: Inception module B with two asymmetrically factorized  $7 \times 7$  convolutional layers. Source: [64].

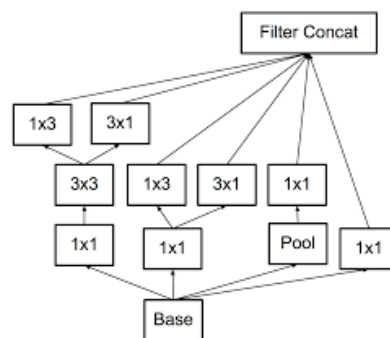


Figure 1.22: Inception module C promoting high dimensional representations. Source: [64].

without it. Instead, it became a sort of a regularizer (with the help of dropout and batch normalization introduced in previous section).

To help the training process a technique called label smoothing was introduced, which was similar to the localized inhibition in AlexNet. This produces new labels  $\hat{L}$  with the equation:

$$\hat{L} = (1 - \epsilon) * L + \epsilon/|L| \quad (1.68)$$

A new optimization technique called root mean square propagation (RMSProp) gradient descent was also used, which helped stabilize the learning error with keeping record of its running mean.

#### 1.3.2.4 Version 4

The paper that introduced the fourth version [62] also designed a version called Inception-ResNet, which is, as the name suggests, a combination of Inception and ResNet architectures. Both the ResNet and Inception-ResNet will be described later.

The modifications to the plain Inception architecture introduced in this paper were quite simple. The first one is the introduction of new beginning layers called the stem, which is a simple stack of convolutional and pooling layers designed to reduce the size of the input. This enabled the use larger images, i.e. images containing more information. The second one is the unification and simplification of the architecture as well as its optimization which allowed for faster and better results.

#### 1.3.3 VGGNet

The runner-up of the ImageNet competition in 2014 was VGGNet [59] (named after Oxford's Visual geometry group), whose paper studies the effect of the network depth on its accuracy. The architecture is a straightforward stacking of layers of  $3 \times 3$  convolutions and max pool on top of each other. It was the first one to come up with the concept of convolutional unit's factorization. Depth of different versions of the architecture ranged from 11 to 19 layers and had top-5 error rate from 10.4% to 8.8%, which was achieved surprisingly not on the deepest version, but rather on the 16 layer version. This showed the limit of increasing the accuracy with layer depth in the current architecture. The other innovation this model used was its multi-scale training and multi-scale testing. This technique tackled the issue of scale of the objects in the picture by training on different versions of the same picture with different scales. Better results were also achieved by using the ensemble of different versions of the models. The architecture of the full model can be seen in Figure 1.23.

#### 1.3.4 ResNet

The idea of increasing depth of neural networks is simple, but with the addition of new layers to an existing model, two possible scenarios come to mind.

1. The additional depth does not increase performance. The original layers are working as in the original net and the new layers will act as identity function.
2. The additional deep layers allow for better approximation of the input and further reduce the error.

The authors of ResNet [21] actually ran experiments with the result that some of the deeper nets had worse accuracy than their shallower counterparts.

# 1. ANALYSIS

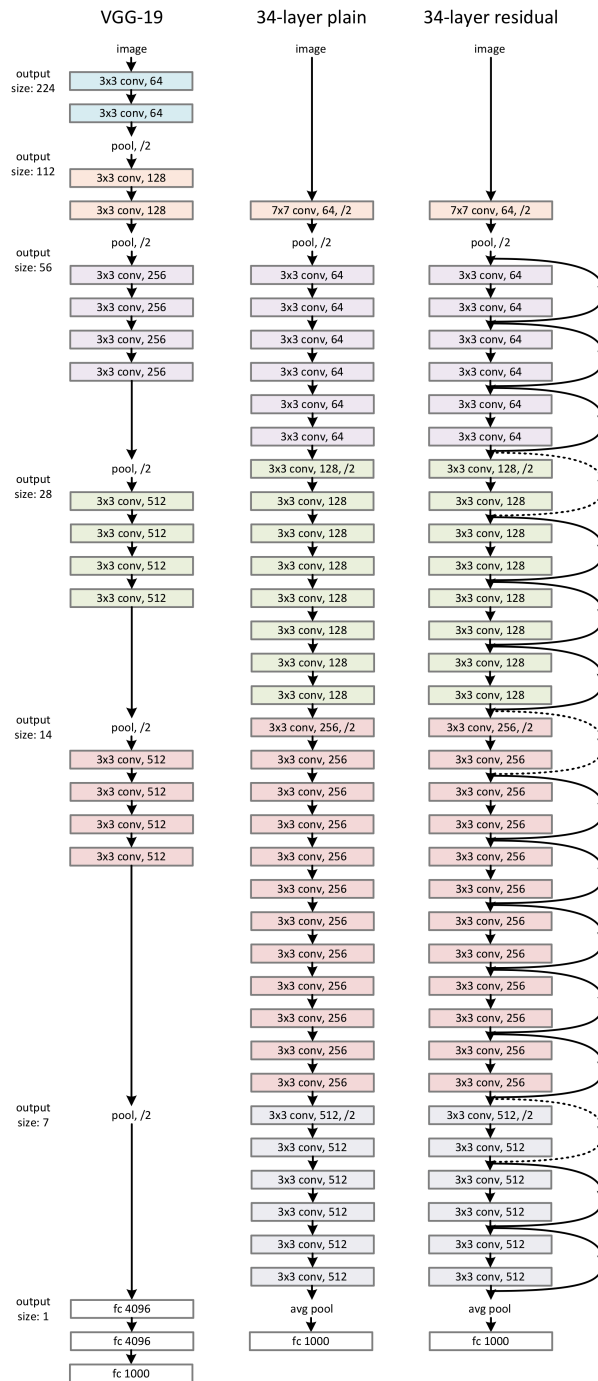


Figure 1.23: Architecture of ResNet network with residual connections in comparison with a plain ResNet network and a VGGNet 19 networks. Source: [59].

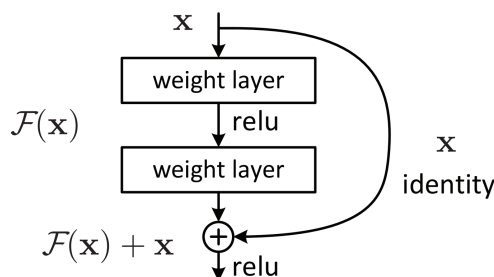


Figure 1.24: Residual block – building block of residual learning. Source: [59].

This is supposedly not because of overfitting or vanishing/exploding gradient, but because the accuracy gets saturated and degrades. The authors of the ResNet resolved the problem with the introduction of a residual block (seen in Figure 1.24). The basis of this block is replacing the mapping of a few layers  $H(x) = F(x)$  by a residual mapping  $H(x) = F(x) + x$ . This is called deep residual learning framework and was extensively tested by the authors and proved to be very effective. Neural networks with depth ranging from 18 to 152 layers were tested and still the neural net performance increased with increasing depth. Networks with residual connections also converge faster than their plain counterparts. Authors also tested a projection mapping instead of a simple identity mapping by adding weights to the identity, which was implemented by a  $1 \times 1$  convolution layer. Projection units did increase the performance, but generally the increase was attributed to the increase of parameters of the model. The ResNet won the ImageNet competition in 2015 and paved the way for further deepening of CNNs.

#### 1.3.4.1 ResNet – Version 2

The following year [22], the authors further improved upon the architecture by optimizing the order of ReLU activation, weighing and batch normalization in the residual unit (shown in Figure 1.25). The original ResNet module did not have the identity connection on the output, but rather on the input of the unit. As a result, ReLU activation function was applied before its addition to the output. The authors experimented with various designs but settled on a version where no additional function was applied on the identity connection. This enabled the authors to train networks with depth up to 1001 layers further pushing the limits and performance of CNNs.

#### 1.3.4.2 Combinations with InceptionNet

Following the success of both ResNet and InceptionNet with vastly different design features, the next logical step was their combination. The team of InceptionNet introduced an Inception-ResNet architecture in the Inception-

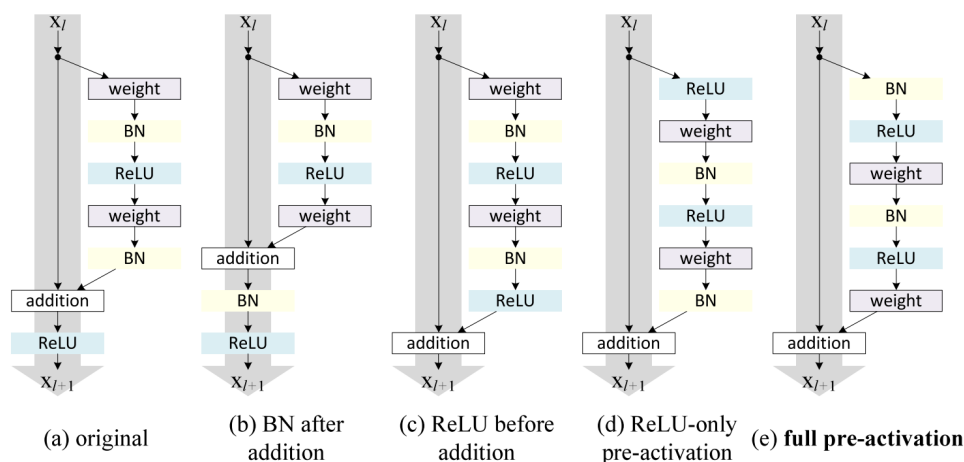


Figure 1.25: Permutations of functions of a new residual block. Source: [59].

v4 paper [62], where the Inception module evolved into the Inception-ResNet module with the addition of identity. This resulted in two versions, Inception-ResNet-v1, which is Inception-v3 with residual connections, and Inception-ResNet-v2, which is Inception-v4 with residual connections. Both these designs converged quicker than their original counterparts and had slightly better accuracy. They did however apply ReLU after the residual connection as in ResNet-v1, because the findings of the second version were not published at the time.

A variation of ResNet called ResNeXt [71] improved upon the ResNet architecture by making it wider (as in InceptionNet) with the addition of multiple parallel ResNet units with one identity connection together (shown in Figure 1.26). The difference from the Inception module is that all the submodules share the same design. The authors experimented with adding more layers and more units to the same module, while retaining the number of parameters, effectively testing the question of depth vs width. The conclusion was that wider networks perform better than ultra deep networks with the same number of parameters.

### 1.3.5 MobileNet

The motivation behind MobileNet introduced in [26] was to reduce the computational intensity of CNNs, because the number of parameters and multiply-add operations of CNNs could make it inefficient for practical use. The authors introduced three main ideas. The first one was decomposition of convolutions not width- and height-wise as in previous designs, but depth-wise followed by a point-wise convolution. The decomposition is pictured in Figure 1.27. This reduces the number of multiply-add operations and parameters around 8 times, while keeping the accuracy comparable.

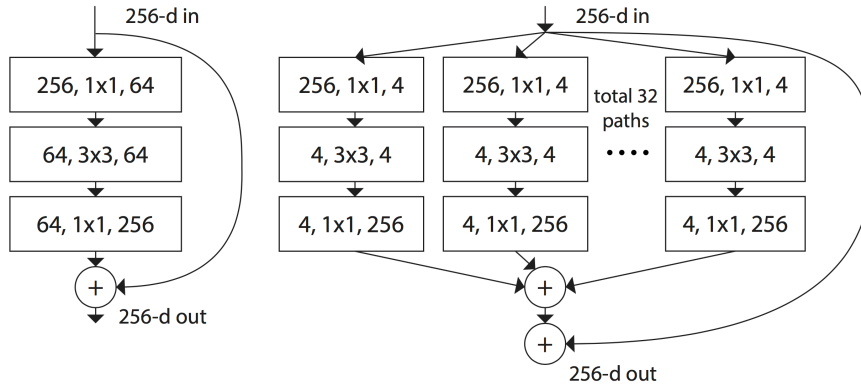


Figure 1.26: Comparison of a basic and a ResNeXt wide residual blocks. Source: [71].

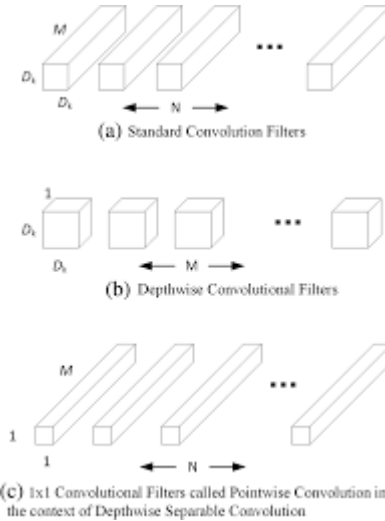


Figure 1.27: Depthwise decomposition of a convolutional layer. Source: [26].

The following modifications are width and depth multipliers. The width multiplier  $\alpha$  is in range  $[0, 1]$  and scales the depth of the output of depth-wise convolutions and input of point-wise convolutions. With smaller  $\alpha$ , computational cost and number of parameters is reduced at greater scale than accuracy. The last improvement is a resolution multiplier  $\rho$  in range  $[0, 1]$ , which reduces the size of the image on the input of the network. Again, the accuracy drops smoothly with smaller  $\rho$ , while computational complexity drops faster. A newer version of this architecture [57] again introduced residual connections to the architecture, along with a few other improvements pushing its accuracy to be on par with ResNet-101 with less than 10% of its parameters. Comparison of accuracy, learning speed and number of parameters of various neural networks including MobileNet can be seen in Figure 1.28.

## 1. ANALYSIS

---



---

## Dataset and data analysis

This chapter will deal with the method of obtaining the data and with the visualization of the relations between articles and their pictures.

### 2.1 Reuters dataset

In order to teach a model to predict an image for an article, we need to obtain a large dataset of articles with their corresponding images. Some datasets which contain hundreds of thousands of articles with their headlines, categories [65], social network feedback [16], etc., are publicly available. There are also databases with images, such as the previously mentioned ImageNet database [12] containing more than 14 million images in over 20 thousand categories that is widely used for model training for image classification and localization. But, to my knowledge, there is no publicly available dataset containing both images and articles or text characterizing them <sup>1</sup>.

In view of the lack of a suitable database, I created a database of articles from six domains (world news, business news, sports, health, technology, and entertainment), each with one image per article. All the articles are scraped from the Reuters website [52], from the Archive<sup>2</sup> subsection in January 2019.

The scraping was done by crawling the available pages of each category using the Python Requests library and downloading the text (including the headline) and the first image along with metadata about the date, author, etc., which were ultimately not used in the experiments, but may prove useful in a future work. The images were also scaled down to 480 pixels image to further reduce their sizes. This reduction should not affect the feature extraction because the models used in the paper reduce the image size less than  $300 \times 300$  pixels anyway.

---

<sup>1</sup>Apart from image annotation datasets, but the annotations are too short and too dissimilar to news articles.

<sup>2</sup><https://www.reuters.com/news/archive/>

The number of articles from each domain was limited by the number of articles available on the website, which is around 32 500 latest articles for the largest categories. However some of them do not contain any images and therefore cannot be used. When multiple images were present, only the first one was downloaded. In the end, the database contains 102 493 unique articles, with 0.5 GB of text data and 2.5 GB of image data. How many articles are in each category can be seen in Table 2.1.

Category	Count	Published in
World news	21 263	2017–2019
Business news	19 976	2016–2019
Sports news	27 699	2015–2019
Health news	12 250	2011–2019
Technology news	24 373	2012–2019
Entertainment news	2 201	2008–2019

Table 2.1: Article categories, their count and period in which they were published of the Reuters Archive dataset.

## 2.2 Aktualne dataset

Aktuálně.cz is a Czech news website of the Czech publishing house Economia. I obtained a similar dataset as the previous Reuters Archive dataset of the text of the articles and their pictures. The articles are split into two categories, “Domáci” and “Zahraničí”, and are of course in the Czech language. It means that a slightly different approach to extracting text embeddings will be used. All images are in a uniform  $640 \times 360$  pixels format. The counts of articles of each article are in Table 2.2.

Category	Count	Published in
Domáci	28 560	2014–2019
Zahraniční	28 417	2004–2019

Table 2.2: Article categories, their counts and period in which they were published of the Aktualne dataset.

## 2.3 Text embedding extraction

The majority of the text feature extraction methods in the previous chapter worked on separate words and created only word embeddings. Creating sentence or document embeddings might seem like another difficult task but [4]

showed that simply averaging all the individual word embeddings into one provides a strong baseline for other algorithms and thus will be used. The authors also show that using TF-IDF as weights improves the performance, which will be also tested in our experiment, as well as using element-wise maximum instead of average of the individual vectors.

### 2.3.1 word2vec

There are two approaches to using word2vec. One is to use the dataset available and train a new word embedding model and the other is to download and use a pretrained model. Training a model specifically to your task is generally the better idea but because Google released a model [19] pretrained on a publicly unavailable corpus of Google News containing 100 billion words, training your own model is not advised. Training models is time consuming even on high-performance hardware not available to me, and most importantly, larger models generally perform better than specialized smaller models. For the Czech language, a similar albeit a much smaller model trained on Wikipedia was used [61]. According to the authors, the CBOW version of word2vec performed much better than Skip-gram and as a result and therefore that version was used.

A python open-source vector space modelling toolkit Gensim [51] was used for the extraction. It provides an interface for easy preprocessing of the text (which is just simple removal of punctuation, lowercasing and splitting on whitespaces) and vector embedding extraction using an api for pretrained model downloading. Word embeddings were obtained and they were either averaged, averaged with TF-IDF weighing or element-wise maximum of the vectors was extracted.

### 2.3.2 GloVe

Extraction of GloVe features was very similar to word2vec features. A publicly available model by the authors of the algorithm was used [46] for the English language. It was trained on Wikipedia dump and the English Giga-word dataset [43] together containing 6 billion words. For the Czech language, the pretrained model by the same authors [61] as with word2vec. Again the words embeddings were averaged to obtain document embeddings, because it was the best performing in the experiments. This approach is used also for FastText and Conceptnet numberbatch.

### 2.3.3 FastText

Obtaining FastText embedding is the same as in previous cases. A pretrained model is publicly available from the authors [8] trained on 16 billion tokens from Wikipedia, UMBC webbase corpus and statmt.org news dataset was

used. The authors also released pretrained models for other 157 languages trained on their Wikipedia and common crawl including a model for the Czech language used in this thesis.

### 2.3.4 Conceptnet numberbatch

Again, a pretrained model from the authors [60] trained on ConceptNet graph and both word2vec and GloVe previously mentioned embeddings was used. Unfortunately, only an English language version is available.

### 2.3.5 BERT

Feature extraction with BERT is different from the previous models. Because it produces embeddings of a whole sentence, the problem of producing document embeddings seemingly disappears. It does not because its width is limited to 512 tokens, which fits at most some paragraphs but not the whole article. Two different approaches were used to extract document embeddings.

In the first approach, the maximum number of tokens from the beginning of the article was used. It contains the headline and the lead paragraph, both trying to capture the meaning of the whole article. The information from the rest of the article is however missing. The second approach is to split the article by sentences, obtain embeddings for each sentence, and then again average the obtained vectors. This approach is more similar to the one used in previous models.

An open-source interface released on GitHub was used [70] with a pretrained model by Google [13]. A smaller model using uncased tokens was used due to the computational intensity of the larger model. Finetuning was not performed again because of computational intensity, but given the quality of the pre-trained model, it should not be an issue. This model produces vectors with length of 768, which is more than 2.5 times larger than all previous models using vectors of length 300.

## 2.4 Image embedding extraction

Extracting image features is pretty straightforward. All the authors also released a pretrained version of the model. Finetuning was not performed because the high quality pretrained model is sufficient for our purposes. A Python implementation of Tensorflow [1] with a wrapper [55] was used, which conveniently includes all the pretrained models on its official GitHub repository<sup>3</sup>. The versions of pretrained models are: Inception-v4, ResNet-v2 with 152 layers, MobileNet-v2 version 1.4 with 224 layers and VGGNet with 19 layers. The pictures are automatically rescaled and preprocessed and produce vector embeddings with 1000 or 1001 size.

<sup>3</sup><https://github.com/tensorflow/models/tree/master/research/slim>

## 2.5 Visualization

To better understand and evaluate the data we obtained, some data visualization techniques can be used. In this section, I would like to visualize the relations between the text and image features and better understand the data.

### 2.5.1 Feature projection

Because data with hundreds of dimensions cannot be meaningfully visualized in two dimensions, two methods for feature projection were used: PCA and t-SNE. These methods project the data from a high-dimensional space to a feature space with only a few dimensions, in our case only two in order to produce easily readable graphs.

These methods were applied on the word2vec features and the result can be seen in Figure 2.1. PCA failed to separate the articles into any clusters and only produced a few outliers. On the other hand, the t-SNE method successfully separated the dataset into meaningful clusters and thus is a far superior method, which will be used in the following sections.

### 2.5.2 Application on datasets

In this section a smaller dataset is used with unique articles from the World News category and one unique image per article. It contains only 4 000 articles in order to speed up the computation time and to prevent graph overplotting and preserve graph clarity. From this dataset, we extracted text and image features listed in Table 2.3.

Type	Model	Feature dimensions
Text features	word2vec	300
	word2vec weighted	300
	GloVe	300
	FastText	300
	ConceptNet	300
	BERT	768
Image features	InceptionNet v4	1001
	ResNet v2	1001
	VGGNet	1000
	MobileNet	1001

Table 2.3: Feature models, their type and dimensionality.

Next we determine which clustering model works best on our data. The ideal model should mark isolated groups as separate clusters and differentiate the rest as best as it can. The clustering model choice is not critical as it is

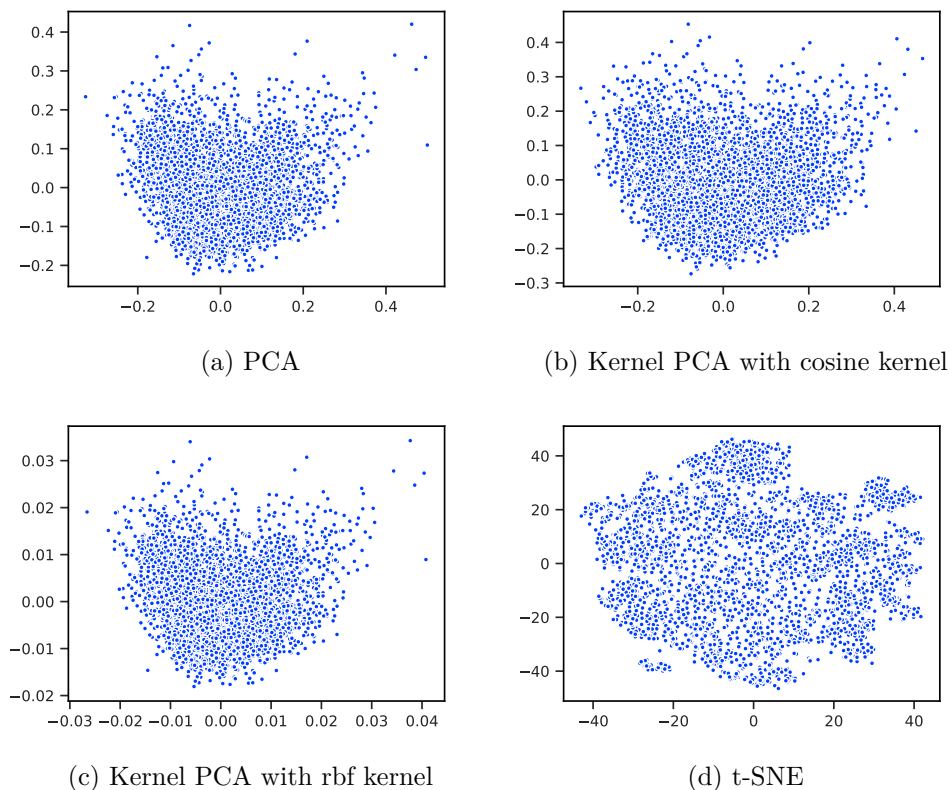


Figure 2.1: Comparison of Kernel PCA with different kernels and t-SNE as feature projection methods

only a visualization technique. The methods were used on a t-SNE projection of word2vec text features set to form 6 clusters. The result can be seen in Figure 2.2 where each cluster is assigned a distinct color. The Spectral clustering method described in subsection 1.1.3.9 works best on separating isolated groups of vectors and so it will be the method of our choice. Note that the borders between clusters are not clearly defined, because the clusters were calculated in the original multidimensional space (where they are probably clearly defined) and only then were projected to two dimensions making the clusters to mix a little bit.

We can remember the color assigned to each article obtained in the previous step, apply t-SNE to other features and plot the results with the remembered colors. This can help us visualize if articles close to each other in one feature space are also neighbors in another feature space. That would indicate that there is some form of correlation between the feature spaces, which would be highly useful for prediction. This approach is demonstrated on Figure 2.3, where in the top left corner is a plot with colored clusters of

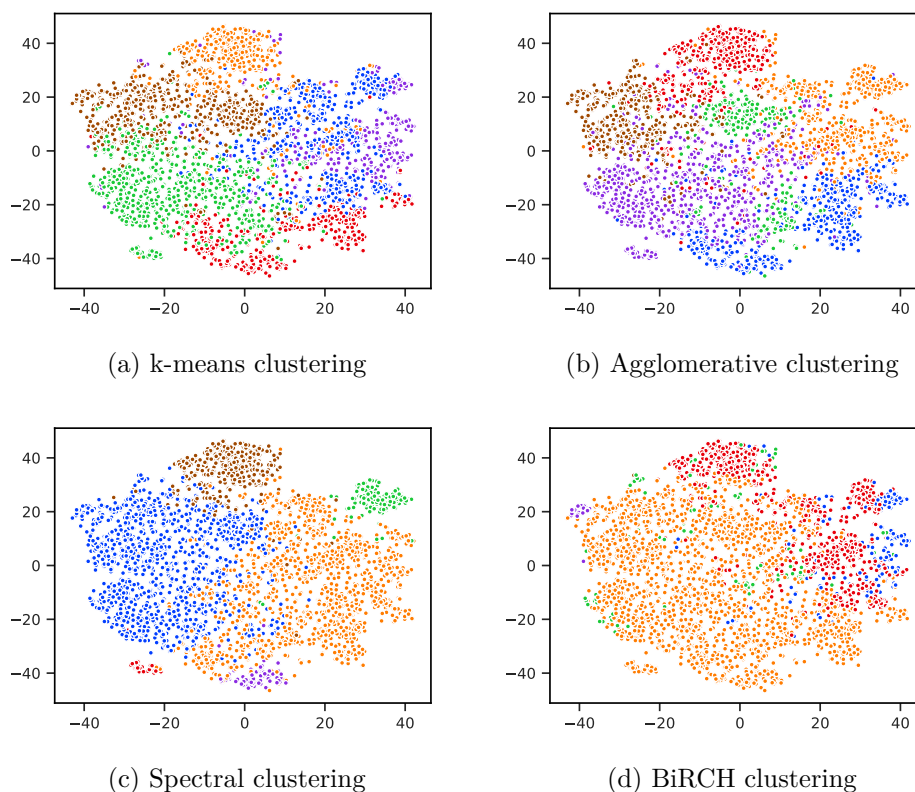


Figure 2.2: Clustering methods applied on word2vec text features and projected with t-SNE, colored by their assigned cluster.

the t-SNE projection of word2vec text features and the same cluster colors applied on a similar projection of other features. It can be clearly seen that in all feature spaces, the articles can be clustered similarly.

Lastly we can use the same obtained colors on the t-SNE projections of our image feature spaces. If the colors are to be ignored for a while, we can observe that the image feature space is much less separable and the t-SNE does not produce isolated groups of pictures. This can be explained in two ways: either the feature extraction models are not capable of extracting enough information from the data or the models are correct but the data itself is either highly varied or too similar. Based on my observations, I think the latter is true, as the images in the world news dataset often show one or multiple persons, usually politicians, which means that for a model trained to classify varied classes of objects, animals or plants, to differentiate is a difficult task.

We can further analyze the colors of clusters obtained in the previous steps. The clusters are mixed together indicating that there are no clear separated

## 2. DATASET AND DATA ANALYSIS

---

clusters in both text and image features or that the dimensionality reduction is not capable of preserving such structures. It means there are no groups of articles and pictures in this dataset that are clearly different from the others. However, the colors are not completely random indicating that there is some correlation between the feature spaces. The goal of this section was therefore achieved, even though the effect is to a lesser extent than anticipated.



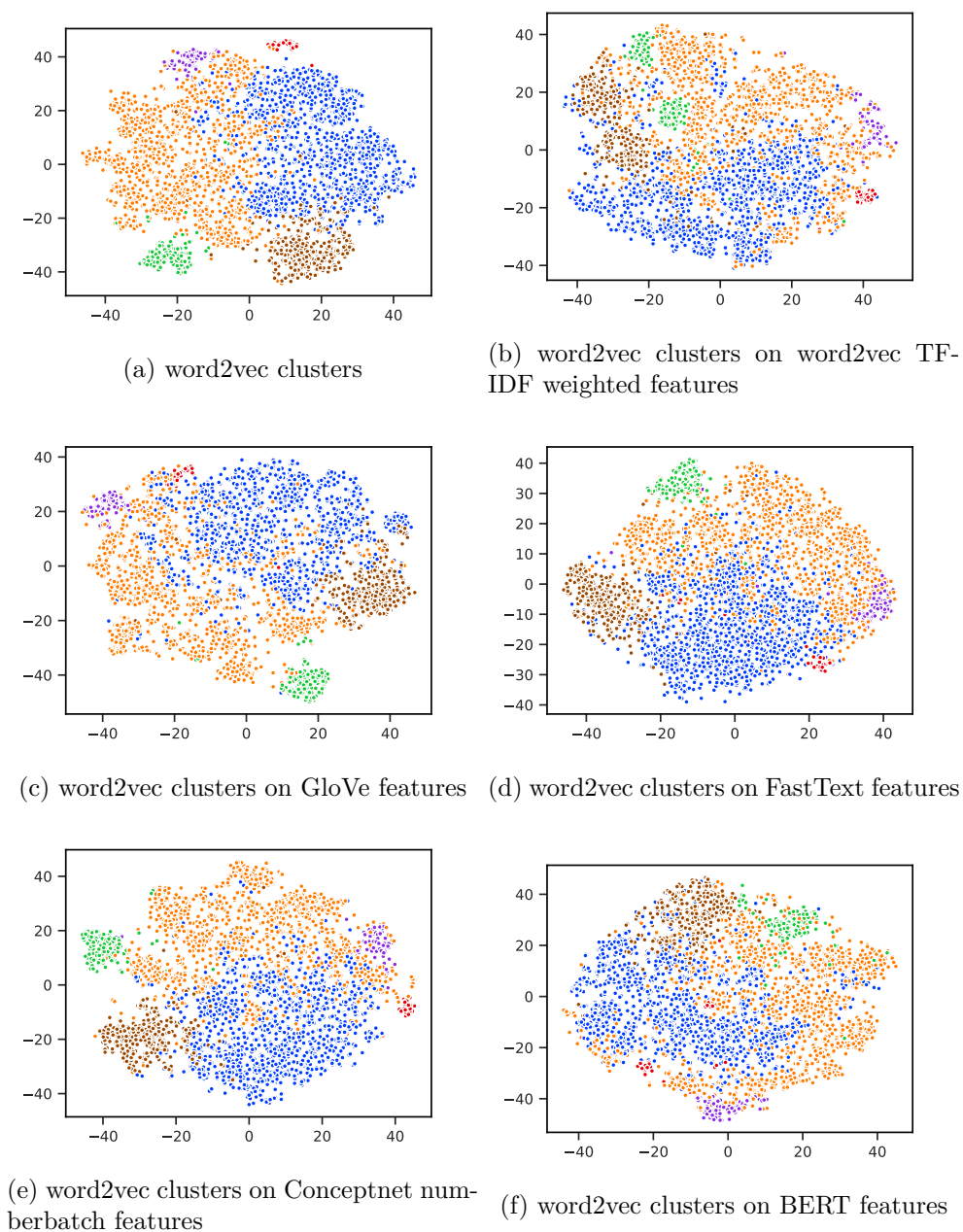
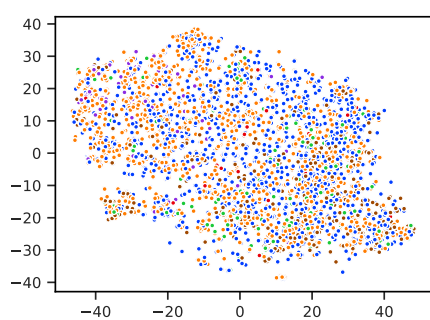
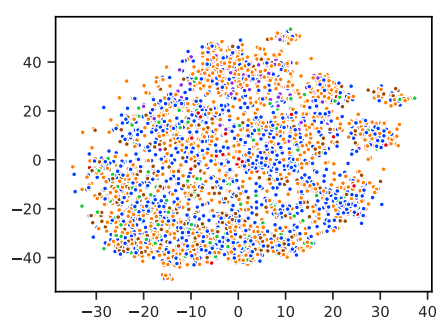


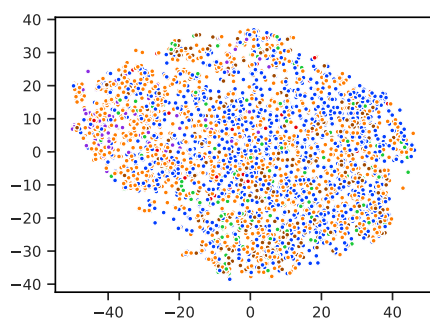
Figure 2.3: Word2vec clusters applied on text feature t-SNE projections.



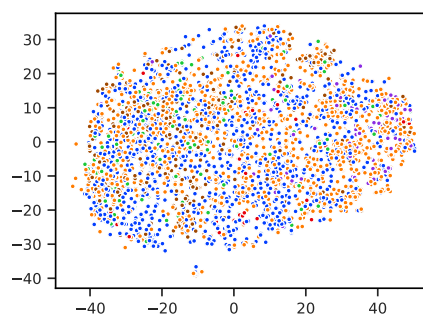
(a) word2vec clusters



(b) word2vec clusters on ResNet features



(c) word2vec clusters on VGGNet features



(d) word2vec clusters on MobileNet features

Figure 2.4: Word2vec clusters applied on image feature t-SNE projections.

---

# Experiments

To show how an algorithms can predict an image likely to be related to a given article, this chapter presents a series of experiments conducted in order to design the best recommender system.

In the previous chapter, we described how the data for these experiments was obtained. This one proposes a design of the recommender system, including a metric for evaluating the algorithms. In the experiments, the feature extraction models are evaluated along with the regression algorithms for transformation between text and image feature space.

## 3.1 Methodology

In order to recommend relevant images to articles, we need to train a model to recognize which images are relevant to the given article and which are not. The recommendation of images is in some ways similar to item-based top-n recommendation, for example queries on web search. Both tasks have the goal of identifying a set of  $N$  items that will be of interest to the user. We can therefore use a similar metric to measure its success.

### 3.1.1 Metric

The usual metrics used in the item recommendation field are called *precision@k* (precision at rank  $k$ ) and *recall@k* (recall at rank  $k$ ). According to [37], we can define them as follows: Let the response to a query be a ranked list of items  $R_q = (d_1, d_2, d_3, \dots, d_m)$  and a list of relevant items  $D_q$ ; then an item's relevance  $r_i$  to the query is defined as:

$$r_i = \begin{cases} 1 & d_i \in D_q \\ 0 & \text{otherwise} \end{cases} \quad (3.1)$$

### 3. EXPERIMENTS

---

With a rank  $k$  representing number of items from the top of list  $D_q$ , we define the metrics precision and recall at rank  $k$  for a given query as:

$$\text{precision}(k) = \frac{1}{k} \sum_{i=1}^k r_i \quad (3.2)$$

$$\text{recall}(k) = \frac{1}{|D_q|} \sum_{i=1}^k r_i \quad (3.3)$$

In plain words, precision@k means the fraction of relevant items of retrieved items, and recall@k is the fraction of relevant items of all relevant items.

In order to use these metrics, we have to define what the “relevance” of images to the original article is. As it is highly subjective to judge what picture will be suitable for an article, we will define only one relevant image, i. e. the article’s original image  $d_o$ , so  $D_q = \{d_p\}$ . Given that only one item is relevant and we always recommend  $k$  images, the precision@k and recall@k metrics reduce to 0 in the case the original image was not among the recommended, in other words  $1/k$  and precision@k and 1 for recall@k. Thus we will use the metric *accuracy@k*, also known as *top-k accuracy* defined as:

$$\text{accuracy}(k) = \begin{cases} 1 & d_o \in D_q \\ 0 & \text{otherwise} \end{cases} \quad (3.4)$$

This will be the metric used to evaluate performance of the models.

#### 3.1.2 Item recommendation

Nevertheless, we also have to define which images will be considered recommended and which will not, i.e. we have to define the set  $D_q$ . In our case, a regression model takes a text feature vector on the input and learns to predict the corresponding image feature vector by minimizing the loss function, which is usually the mean squared error function in Equation 1.48. Given the predicted image feature vector  $\hat{Y}$ , we will define the  $k$  recommended images as nearest neighbors of the vector  $\hat{Y}$  using a distance metric  $D$  and a function  $KNN$  returning  $k$  nearest neighbors as:

$$\text{recommended}_i\text{tems}(\hat{Y}, k) = KNN(\hat{Y}, k, D) \quad (3.5)$$

To provide the most rudimentary baseline with the size of test set used in Experiment 1 of 3 761 samples, we can simply use random choice of images. The top-1 accuracy would then be  $\frac{1}{3761} \approx 0.026\%$  and top-20 accuracy  $\frac{20}{3761} \approx 0.532\%$ .

### 3.1.3 Linearity of the problem

One of the key information is whether the feature space of our problem can be solved by linear regression. In practice it means whether linear regression models, which are significantly faster, can be used without decreasing the accuracy or nonlinear models have to be used. To my knowledge, the linearity of the feature space cannot be determined either analytically nor numerically in a reasonable computation time. However, since our feature space has high dimensionality (from 300 to 1 000) only two orders below the number of samples (around 18 000), we can assume that linear regression models are going to perform adequately on the dataset<sup>4</sup>. It will be also tested in Experiment 1 by transforming the feature spaces to linearly separable spaces using kernel PCA and Multilayer Perceptron with multiple hidden layers.

### 3.1.4 Recommendation system design

The design of the recommender system is simple. A regression model is first trained on a training dataset with supervised learning. It is then applied on texts of new articles and recommends images from a given image bank. The whole system design is pictured in Figure 3.1.

## 3.2 Experiment 1 – Regression model tuning

In this experiment, various regression algorithms are analyzed. First, data normalization or scaling is investigated, then hyperparameters of the regression algorithms are tuned, and finally the optimized algorithms are compared and the best are chosen for the following experiments.

In the following graphs top-1, top-5, top-10 and top-20, accuracy metrics are depicted in blue, orange, green, and red color unless specified otherwise.

### 3.2.1 Dataset

This experiment was run on the world news category with non-unique images removed. It resulted in a dataset of 18 803 articles and images. The text feature extraction model used is the word2vec model, which produces 300 dimensional vectors, and the image feature model is the InceptionNet model producing 1 001 dimensional vectors. The results of all experiments are the mean of KFold cross-validation with  $K = 5$ . It means the size of the training set is 80% of the sample set or 15 042 samples and the size of the test set is 20% of the sample set or 3 761 samples.

---

<sup>4</sup>A dataset with  $N$  observations is linearly separable in  $N - 1$  dimensions.

### 3. EXPERIMENTS

---

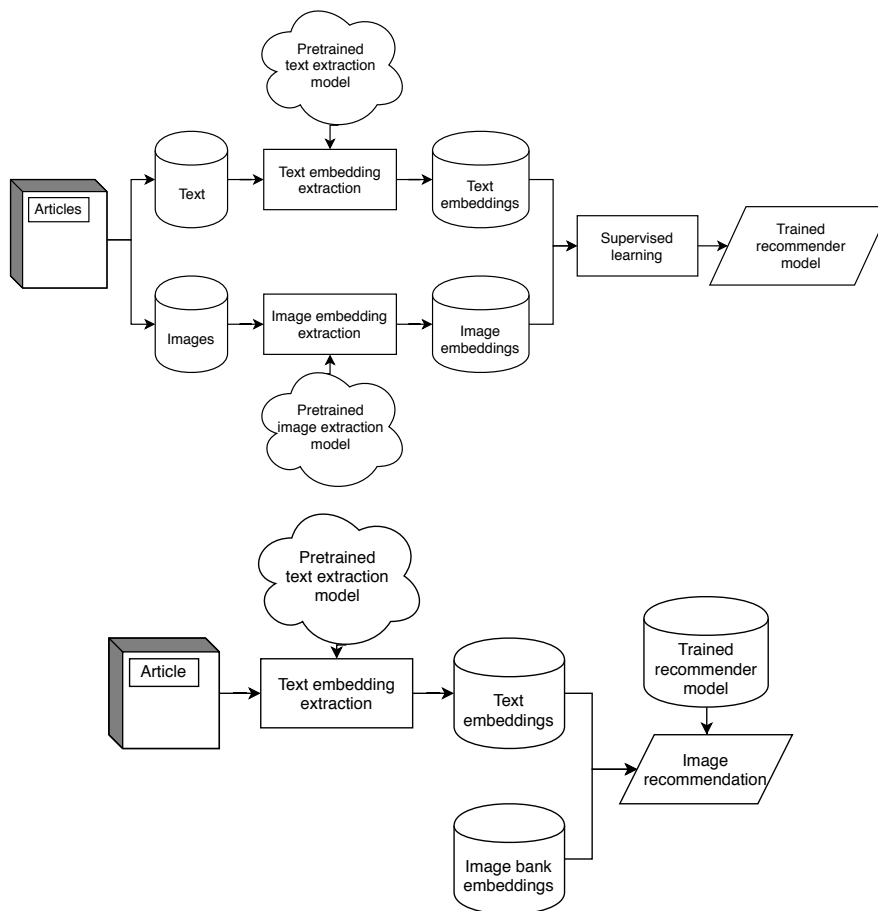


Figure 3.1: Training of the recommender and its application on new articles.

#### 3.2.2 K Nearest Neighbors

Not only does KNN algorithm have a few parameters for optimization but also the data preprocessing phase can be optimized. The feature preprocessing techniques and the hyperparameters optimized are in Table 3.1.

Note that if the dataset has not been split between training and testing sets, in the case of an KNN model the accuracy would be perfect because the nearest image for any given article would be the article's image itself.

##### 3.2.2.1 Number of neighbors

In the first part of this experiment, number of neighbors  $n$  was set as  $n \in \{1, 3, 5, 10\}$  and in the view of the results in Figure 3.2, the best was  $n = 1$  with 2.48% top-1 accuracy with the Cosine distance. The results were reproduced with multiple distances to show that the result is independent on the choice of distance measure. It is rather unsurprising, as with  $n = 1$  the prediction is

### 3.2. Experiment 1 – Regression model tuning

Hyperparameter	Explanation
preprocessing	feature scaling and standardization
dimensionality reduction	different PCA techniques and parameters
number of neighbors	number of nearest neighbors
metric	distance metric
weights	weight function used in prediction

Table 3.1: Hyperparameters of KNN model optimized in Experiment 1.

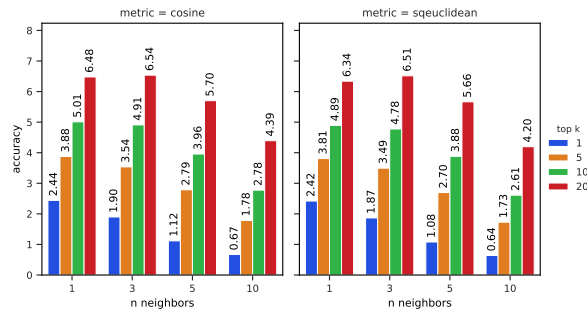


Figure 3.2: Bar graph of the effect of number of neighbors on KNN model accuracy (in percent).

the output for the neighbor in the training set and it is not uncommon it is the closest point to the correct prediction (in a high dimensional space). With  $n > 1$  the predicted results features are averaged from the neighbors resulting in values further from the expected features.

#### 3.2.2.2 Metrics

Our KNN implementation [45] supports lots of metrics but for the purpose of our algorithm we will try only the Minkowski metrics and the Cosine distance (often used in NLP tasks) because other metrics are either for boolean values or highly specialized. The results of the experiment in Figure 3.3 show that the Cosine distance is the best performing metric with 2.37% top-1 accuracy followed closely by the Squared Euclidean metric with 2.29%. The Cosine distance is more accurate and also faster and as such will be used in the following experiments. Note that because our data is almost centered, the Correlation distance is very similar to Cosine distance. Also the Euclidean and Squared Euclidean distances have equal results (because squaring is a monotonic function and preserves order), so the squared one will be used throughout our experiments as the faster function out of the two.

According to our metric definition in Equation 3.5, the recommended items are  $k$  neighbors from the predicted item. It means that we can also

### 3. EXPERIMENTS

---

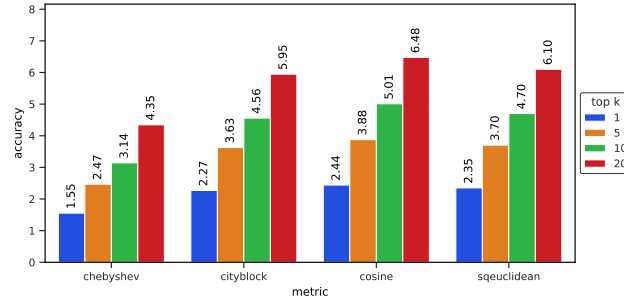


Figure 3.3: Bar graph of the effect of distance metric on KNN model accuracy (in percent).

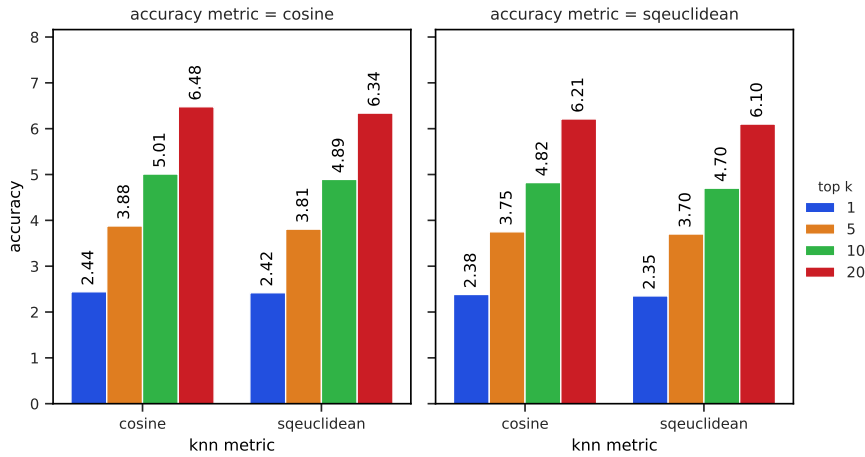


Figure 3.4: Bar graph of the effect of distance metric of KNN model and of accuracy neighbor function on accuracy.

optimize the nearest neighbor algorithm of the recommendation model. In the view of the previous experiment, the measured metrics will be the Cosine and Squared Euclidean distances. The graph in Figure 3.4 confirms the previous result that the cosine distance is superior distance in both the recommendation nearest neighbor model and the regression model itself, although by only a small margin.

#### 3.2.2.3 Weights

After measuring the effect of the weight function (results in Figure 3.5), it is apparent that uniform weights are more accurate than distance weights. It is also faster as no weight calculations actually take place. Note that when  $n = 1$ , the results are the same because there is only one neighbor in both cases.



## 3.2. Experiment 1 – Regression model tuning

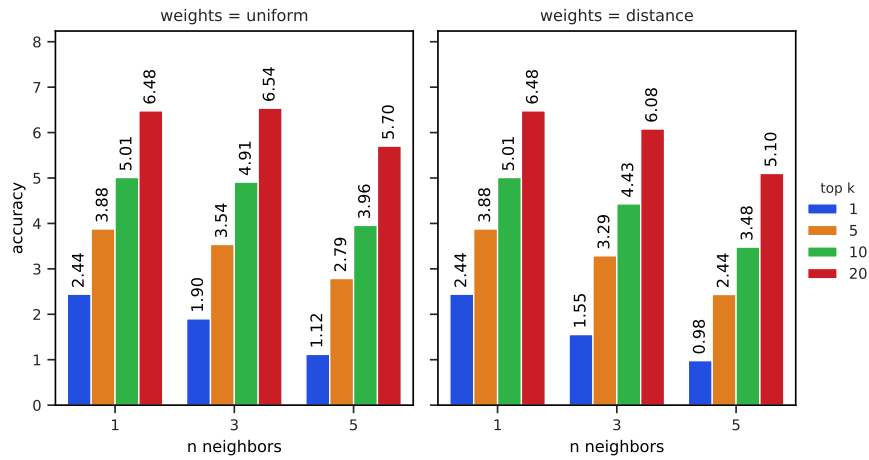


Figure 3.5: Bar graph of the effect of weight function on KNN model accuracy (in percent).

### 3.2.2.4 Scaling

Then the effect of feature scaling and standardization was examined. The results in Figure 3.6 show that while rescaling the features to the range (0, 1) has negative effect on the result, the standardization to zero mean and unit variance provides a small but measurable advantage. Given that standardization is required for Kernel Ridge Regression and recommended for MLP Regression, it will be used with KNN as well.

### 3.2.2.5 PCA

In the following part of the experiment, the effect of PCA with multiple kernels is examined. The results will hint at whether our problem is linear or non-linear. Because of the complexity of kernel PCA, the training set was reduced to only 8 000 samples with 2 000 test samples. The number of components is the original dimensionality to not remove any variance from the feature space. Results are graphed in Figure 3.7. We can see that both the linear PCA and RBF kernel PCA are superior to not using PCA. Rather surprising is that linear PCA performed better than the kernel PCA. It is an indication that linear regression models are powerful enough for our problem. Also the time complexity of the kernel PCA on large datasets makes it impractical and will not be used in the following experiments.

The correct number of components the PCA can be optimized on both the text features and image features. The results graphed in Figure 3.8 show that it is more accurate not to reduce the text features dimensions at all and to reduce the image features dimension from 1001 to between 250 and 500 which would roughly equal in dimensions with text features. It provides a nearly

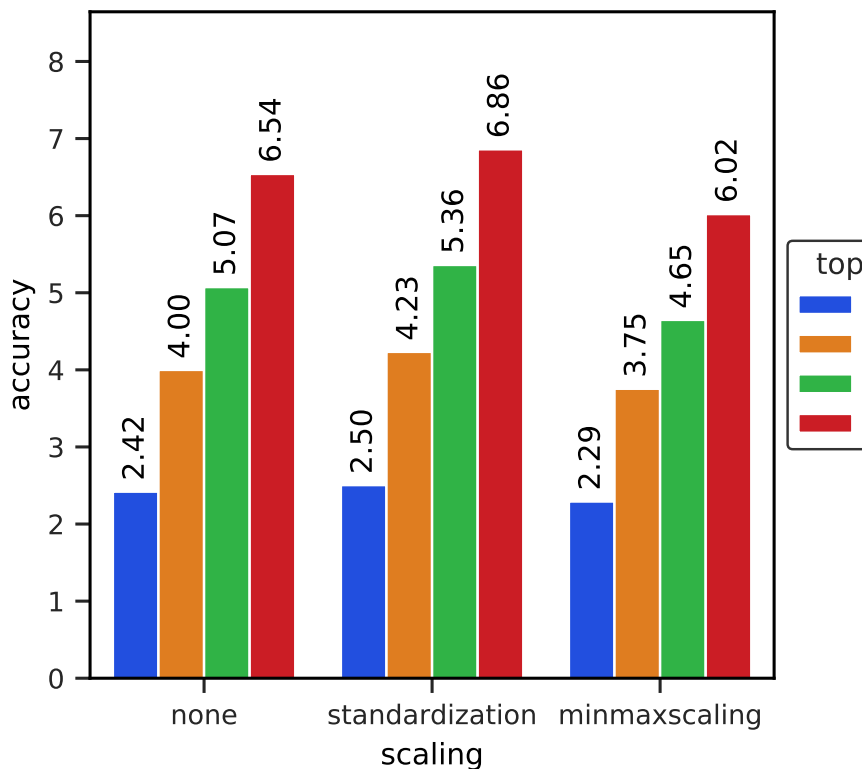


Figure 3.6: Bar graph of the effect of feature scaling on KNN model accuracy (in percent).

60% boost in computation time of the KNN regression model which is now around 4.5 seconds without a significant reduction in accuracy.

### 3.2.2.6 Results

The optimal model parameters and preprocessing techniques are shown in the Table 3.2. We have achieved the accuracy of 2.5% top-1 accuracy and 6.88% top-20 accuracy (in Figure 3.6), which is almost 100 times better than the 0.026% top-1 accuracy of random image selection.

The drawback of KNN is that it does not generalize well and thus may not be the optimal regression model despite its accuracy. It is also a so-called lazy learner, which means the majority of computation time is in the testing phase. This means queries in real application could take a long time, well over couple of seconds for large datasets and feature dimensions, making it somewhat impractical.

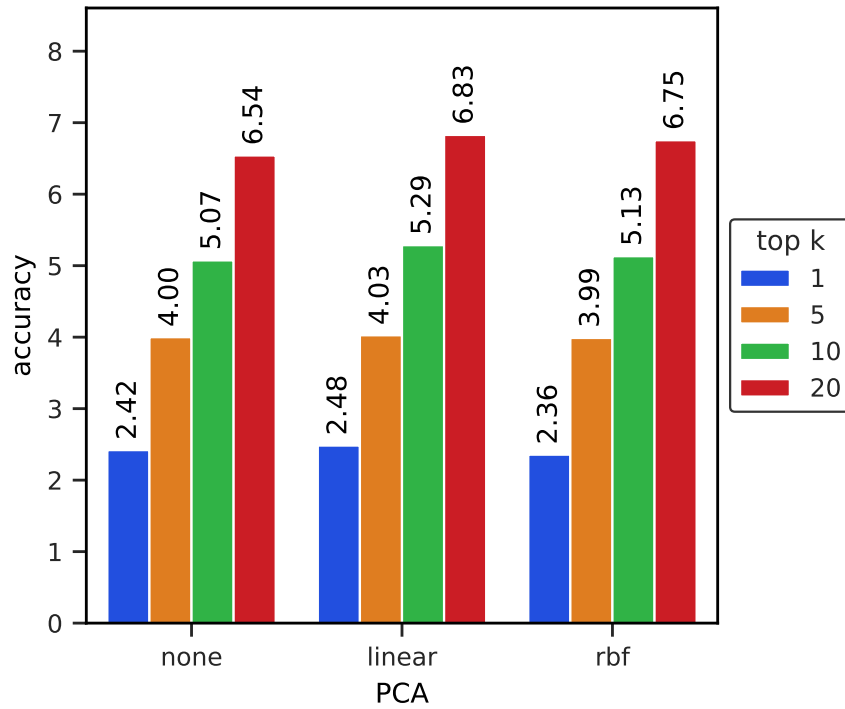


Figure 3.7: Bar graph of the effect of PCA kernel on KNN model accuracy (in percent).

Hyperparameter	Optimized value
preprocessing	standardization
dimensionality reduction	linear PCA with 300 components
number of neighbors	1
metric	Cosine distance
weights	uniform

Table 3.2: Optimized hyperparameters of KNN model.

### 3. EXPERIMENTS

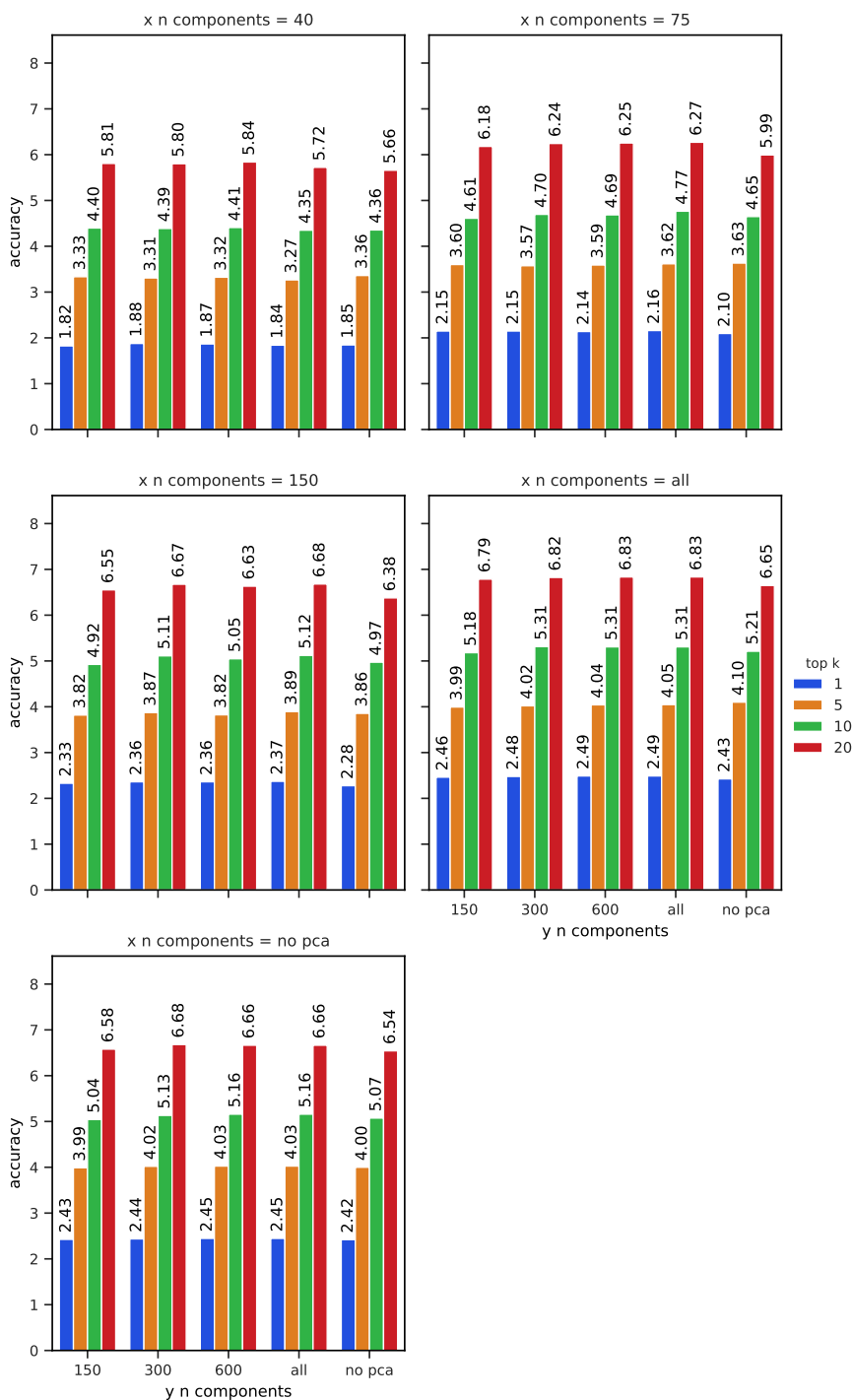


Figure 3.8: Bar graph of the effect of number of components of linear PCA on KNN model accuracy (in percent).

### 3.2. Experiment 1 – Regression model tuning

Hyperparameter	Explanation
alpha	penalty terms weight
L1 ratio	ratio between L1 and L2 penalization
maximum no. of iterations	maximum number of iterations of gradient descent
preprocessing	use of standardization and/or PCA

Table 3.3: Hyperparameters of ElasticNet model optimized in Experiment 1.

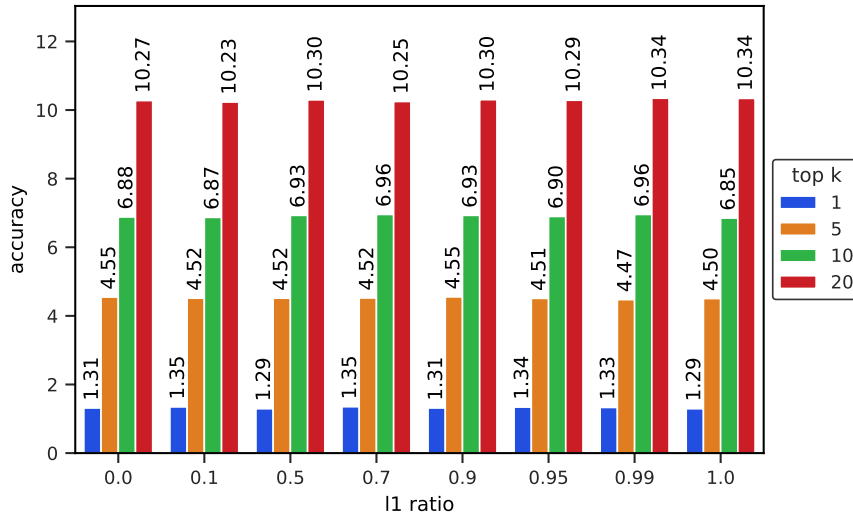


Figure 3.9: Bar graph of the effect of l1 rat parameter on ElasticNet model accuracy (in percent).

### 3.2.3 ElasticNet Regression

ElasticNet has two parameters we can optimize. The first one is the parameter  $\alpha$ , which sets the weights of the penalties. Note that if  $\alpha = 0$ , then the model reduces to the ordinary least squares model. The second parameter is the ratio between the L1 and L2 penalization, called simply the L1 ratio. Because of the model coefficient penalization, the data is standardized.

#### 3.2.3.1 L1 ratio

First, the L1 ratio will be optimized with a fixed  $\alpha = 0.001$ . The results graphed in Figure 3.9 are inconclusive about the best L1 ratio, as it does not show a clear trend in the results. As such L1 ratio of 0.5 will be used because it is the default parameter in the implementation and should provide adequate results.

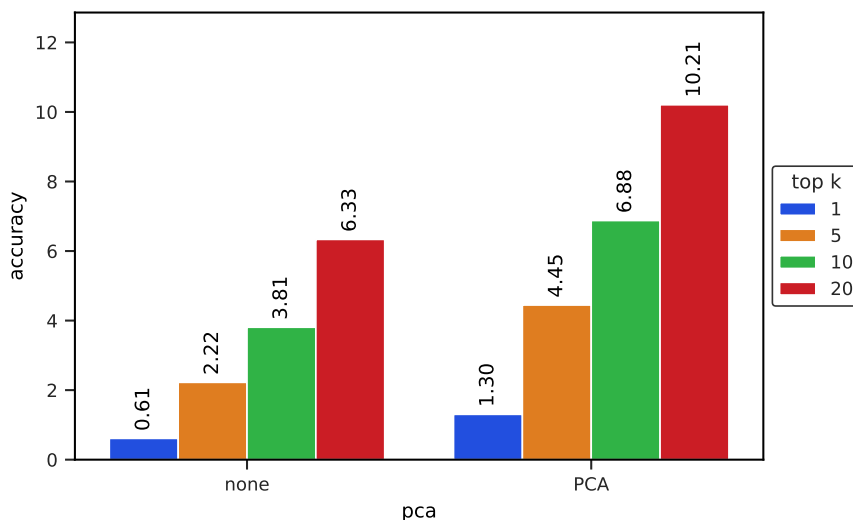


Figure 3.10: Bar graph of the effect of alpha parameter on ElasticNet model accuracy (in percent).

### 3.2.3.2 Alpha

Next, the alpha parameter is optimized with alpha values from 0 to 1. The results can be seen in Figure 3.10; the best performing alpha is less than 0.001. We will set the argument to 0.0001 in the following experiments as it was also the best value according to a Lasso model fit with least angle regression algorithm with Akaike and Bayes intercept criterion (more about the algorithm in [74]).

### 3.2.3.3 Iteration limit

Since the implementation of ElasticNet we use is optimized with an iterative gradient descent algorithm, we can control the maximum number of iterations it computes in order to prevent the algorithm from overfitting. As we can see in Figure 3.11, the model tends to overfit a little bit after 100 iterations.

### 3.2.3.4 PCA

Although both standardization and PCA have been used so far, only standardization is theoretically necessary (otherwise features with greater magnitude would gain control over the whole model). In this experiment, we compare the model with and without the use of PCA. The result can be seen in Figure 3.12 and shows that the model performs very poorly without PCA.

### 3.2. Experiment 1 – Regression model tuning

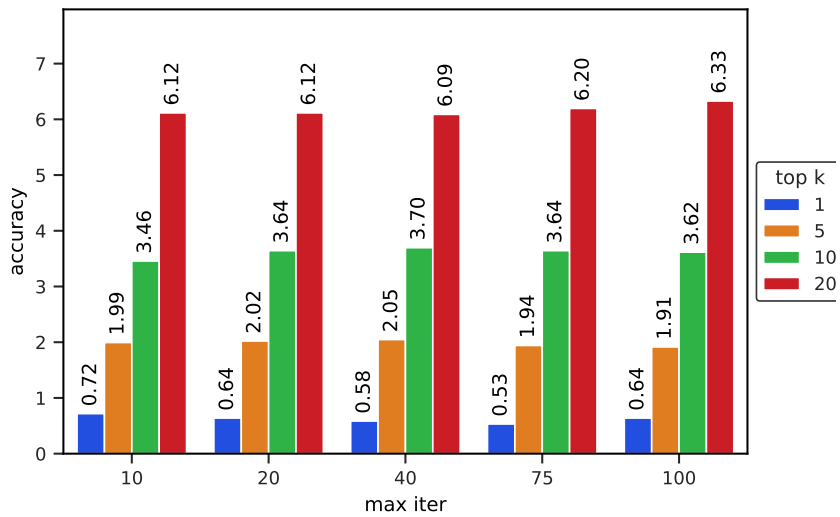


Figure 3.11: Bar graph of the effect of maximum number of iterations on ElasticNet model accuracy (in percent).

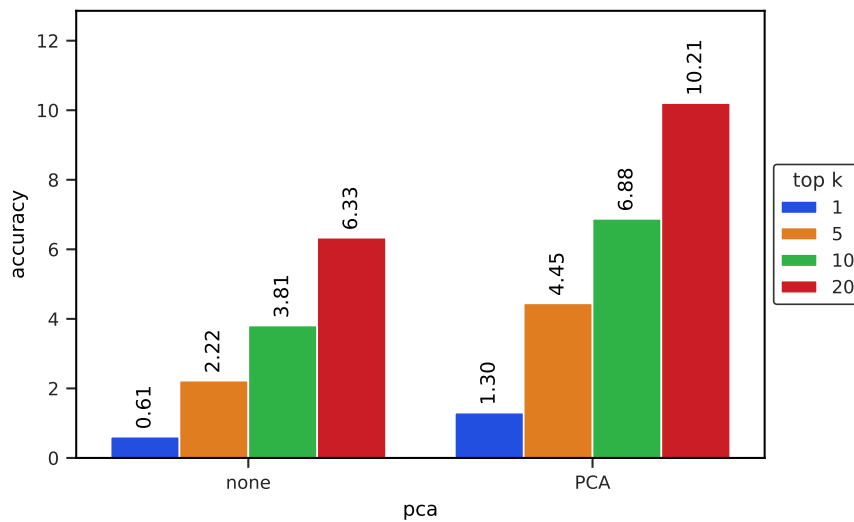


Figure 3.12: Bar graph of the effect of PCA on ElasticNet model accuracy (in percent).

### 3. EXPERIMENTS

---

Hyperparameter	Value
alpha	0.0001
L1 ratio	0.5
maximum of iterations	50
preprocessing	standardization and PCA

Table 3.4: Optimized hyperparameters of ElasticNet model.

Hyperparameter	Explanation
hidden layers	size and number of the hidden layers
maximum no. of iterations	maximum number of iterations during training
activation	activation function on hidden layers

Table 3.5: Hyperparameters of Multilayer Perceptron model optimized in Experiment 1.

#### 3.2.3.5 Results

The optimal model parameters and preprocessing techniques are shown in Table 3.4. We have achieved the accuracy of 1.54% top-1 accuracy and 10.28% top-20 accuracy (in Figure 3.11). Compared to the KNN model it performed only about half as good in top-1 accuracy, but performed better in top-20 accuracy. It shows that the ElasticNet model fits the problem less than KNN, i.e. generalizes better but performs a little worse.

#### 3.2.4 Multilayer Perceptron Regression

Multilayer Perceptrons can be optimized by choosing the right activation function, loss function, regularization, dropout, topology, and many other things. Note that Multilayer Perceptron with multiple hidden layers can learn to separate data that is not linearly separable, which is why the topology will be one of the parameters we will optimize. We will try the logistic sigmoid function, the hyperbolic tan function, and the rectified linear unit one as activation functions. All data is standardized and preprocessed with PCA with 300 components beforehand (training MLP without PCA performed very poorly).

##### 3.2.4.1 Hidden layer sizes and overfitting

The first hyperparameter to optimize is the net's topology. All layers are dense layers with L2 regularization and 300 units. Because neural nets are prone to overfitting even with regularization, the topologies were tested together along with number of iterations. The important property of multiple hidden layers is that MLP with more than one hidden layer can solve non-linear problems.



## 3.2. Experiment 1 – Regression model tuning

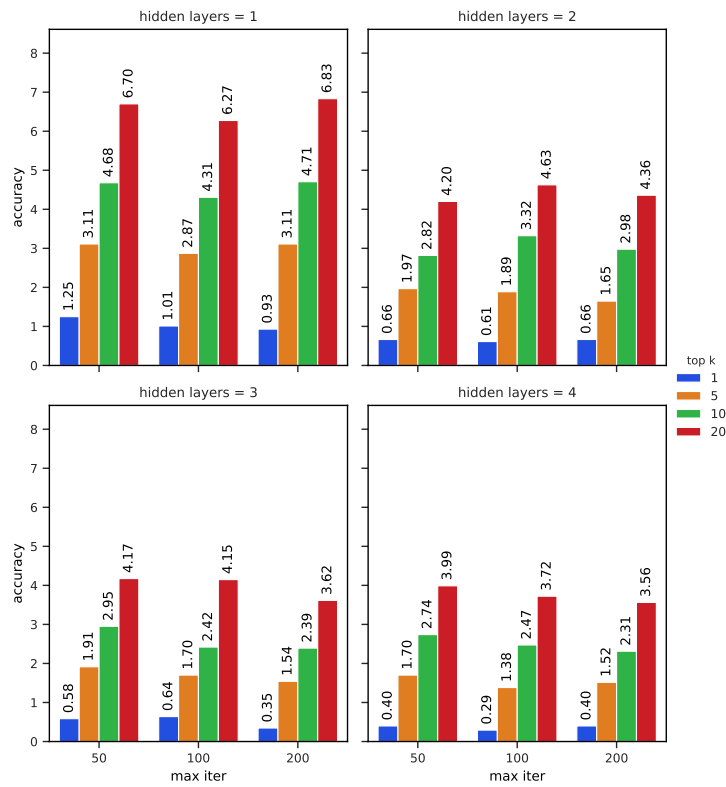


Figure 3.13: Bar graphs of the effect of the hidden layer size on Multilayer Perceptron model accuracy (in percent).

The results can be seen in Figure 3.13 with computation time in Figure 3.14. They show that a simple MLP with one hidden layer and number of iterations limited to 50 (which was the least number of iterations tested) is the best performing one.

### 3.2.4.2 Activation function

The three activation functions tested were the hyperbolic tan function and the the rectified linear unit function. Only one hidden layer with size of 300 nodes was used with 50 iterations of learning in order to test them. The results are in Figure 3.15 and in Figure 3.16. In terms of speed, the ReLU function is the fastest one. However, in terms of accuracy, the model with this configuration could not learn to predict the results and the accuracy is only slightly better than random baseline. In the following experiments, the configuration will be tuned to achieve hopefully better results.

### 3. EXPERIMENTS

---

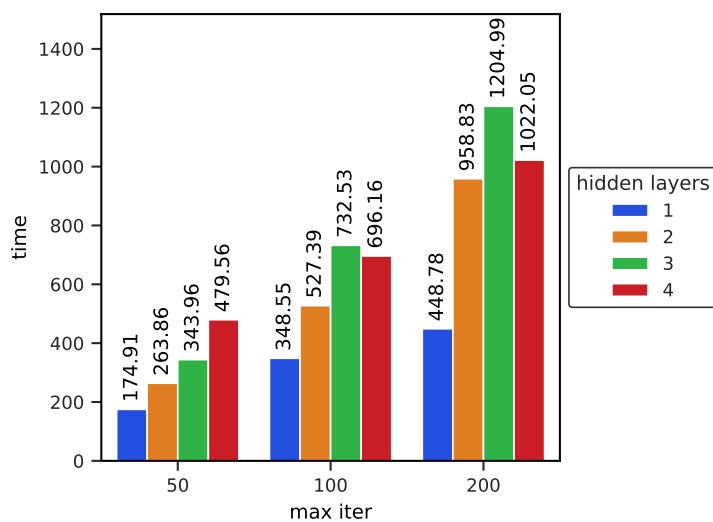


Figure 3.14: Bar graphs of the effect of the hidden layer size on Multilayer Perceptron model speed (in seconds).

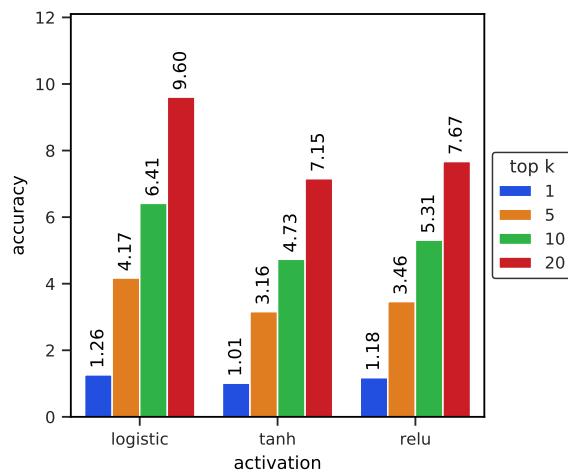


Figure 3.15: Bar graph of the effect of the hidden layer activation function on Multilayer Perceptron model accuracy (in percent).

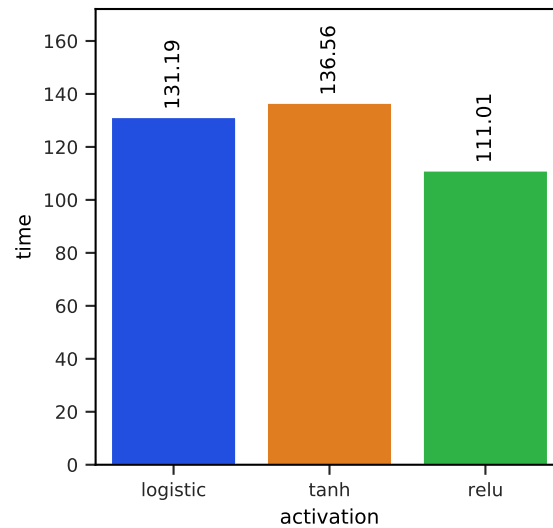


Figure 3.16: Bar graph of the effect of the hidden layer activation function on Multilayer Perceptron model speed (in seconds).

### 3.2.4.3 Learning rate

In this experiment, the solver of the optimization problem is the Adam solver [29]. It is an extension to stochastic gradient descent, which maintains an adaptive learning rate for each of the parameter (improving performance on sparse gradients). They are calculated from their moving averages from the gradient and from the square gradient. The weights also decay over time. Even though the learning rate adapts throughout the training, correct initial learning rate can have an impact on the performance. Results of this experiment are in Figure 3.17 and show that all initial learning rates perform very similarly, therefore the initial learning rate of 0.001 will be used as it performed the best by a small margin.

### 3.2.4.4 Results

Surprisingly the Multilayer Perceptron could not achieve the results as good as the previous models even after its optimization, although the results were similar to an Elastic Net (which is not surprising considering they use almost the same regularization and had same number of parameters, but different function). The best top-1 accuracy it achieved was 1.26% and top-20 accuracy was 9.60% with optimized parameters listed in Table 3.6.

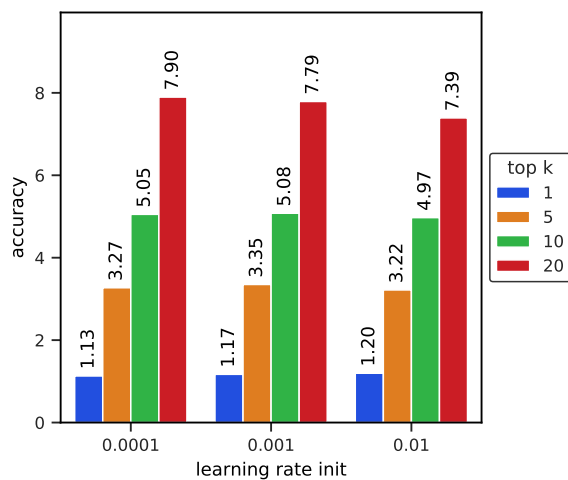


Figure 3.17: Bar graph of the effect of initial learning rate on Multilayer Perceptron model accuracy (in percent).

Hyperparameter	Value
activation	logistic sigmoid
hidden layers	1
hidden layer units	300
maximum no. of iterations	50
preprocessing	standardization and PCA

Table 3.6: Optimized hyperparameters of ElasticNet model.

### 3.2.5 Random forests

A rather popular technique in machine learning is ensemble learning. It is a technique where multiple trained models are used for prediction in order to overcome the flaws of each of the individual models. For example the majority of winners of the ImageNet competitions used multiple trained CNNs and averaged their predictions.

A popular variant of ensemble methods called Random forest is used in this experiment along with its variant called Extremely randomized trees. Because this family ensemble model relies on a large number of trained trees, it is heavily memory dependant, especially with a large dataset. This causes memory issues and forced us to use a model with only 100 trees with minimum 5 samples on an internal node (to prevent further splitting). The results can be seen in 3.18 and show that they do not achieve results as good as the previous models. Extratrees model was performing better with more than 10 times faster learning speed than Random Forest, showing that randomizing is

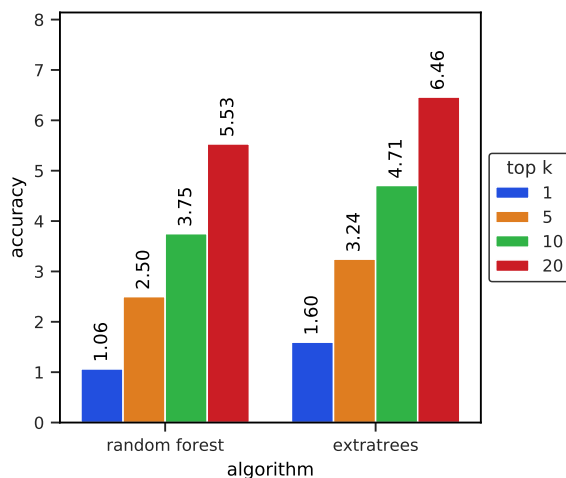


Figure 3.18: Bar graph with the results of a Random forest and an Extremely randomized trees models.

Model	Top-1 accuracy	Top-20 accuracy
K Nearest neighbors	2.5%	6.88%
Elastic Net	1.54%	10.28%
Multilayer Perceptron	1.26%	9.60%
Random Forests	1.60%	9.46%

Table 3.7: Results of optimized .

sometimes better than exact optimization.

### 3.2.6 Result

Results of experiment 1 can be seen in Table 3.7 and Figure 3.19. The K Nearest Neighbor regressor performed the best in top-1 accuracy with 2.5% and the ElasticNet regressor was the best in top-20 accuracy with 10.28%.

The results can also be visualized on the t-SNE projection used in Chapter Data 2. They are pictured in Figure 3.21 on InceptionNet features and in Figure 3.22 on word2vec features. We can see that successful predictions of the model are not grouped but are scattered throughout the whole dataset. We can also plot models accuracy depending on the  $k$  in top- $k$ . The plot can be seen in Figure 3.20 and show that after a certain threshold of parameter  $k$ , the ElasticNet regressor is the best model we obtained.

### 3. EXPERIMENTS

---

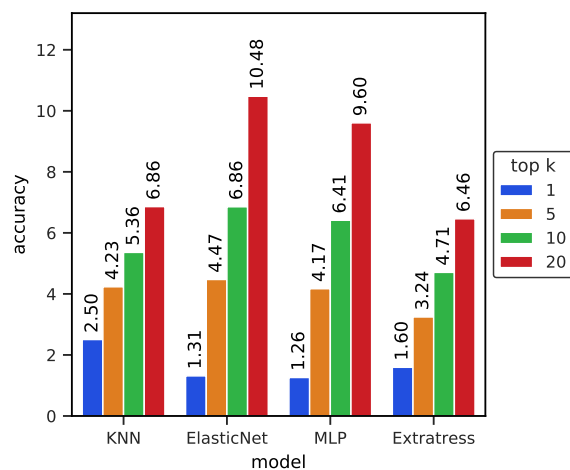


Figure 3.19: Bar graph comparing the accuracy of optimized regression models.

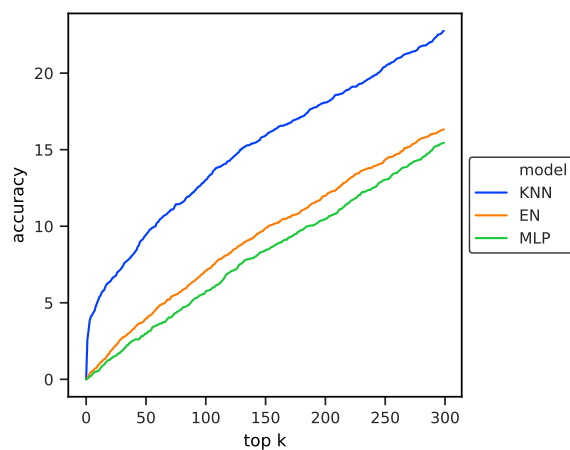


Figure 3.20: Dependency of top-k accuracy based on  $k$  for different regression models.

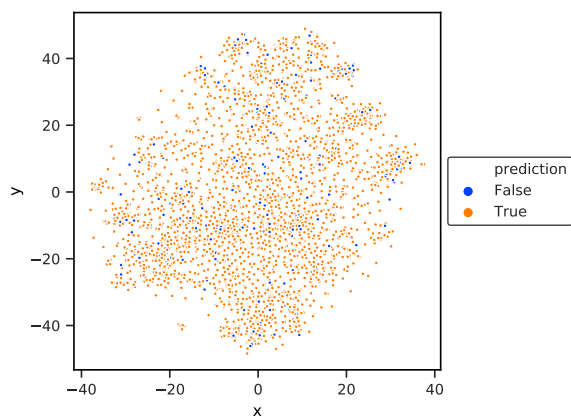


Figure 3.21: Plot of t-SNE projection of InceptionNet features with observations colored by their prediction success.

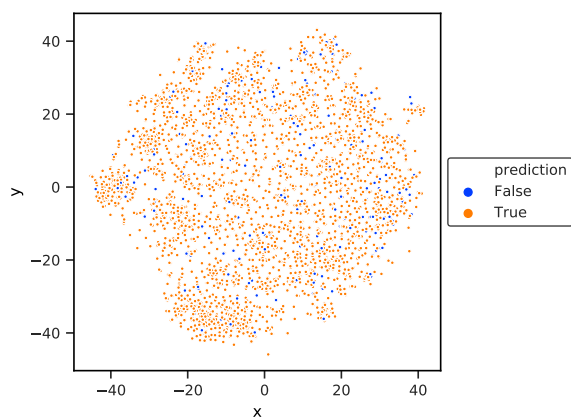


Figure 3.22: Plot of t-SNE projection of word2vec features with observations colored by their prediction success.

### 3.3 Experiment 2 – Feature extraction models evaluation

In the previous experiment, we tuned the regression algorithms only on the word2vec text features and InceptionNet image features. This experiment focuses on whether the word2vec feature extraction model performs the best out of the text feature models and whether the InceptionNet excels among image feature extraction models. The best performing models from the previous experiments in 3.7 were the KNN model with 2.5% top-1 accuracy and ElasticNet with 10.28% top-20 accuracy. In this experiment, these models were trained on different text features and image features specified in Table 2.3.

### 3. EXPERIMENTS

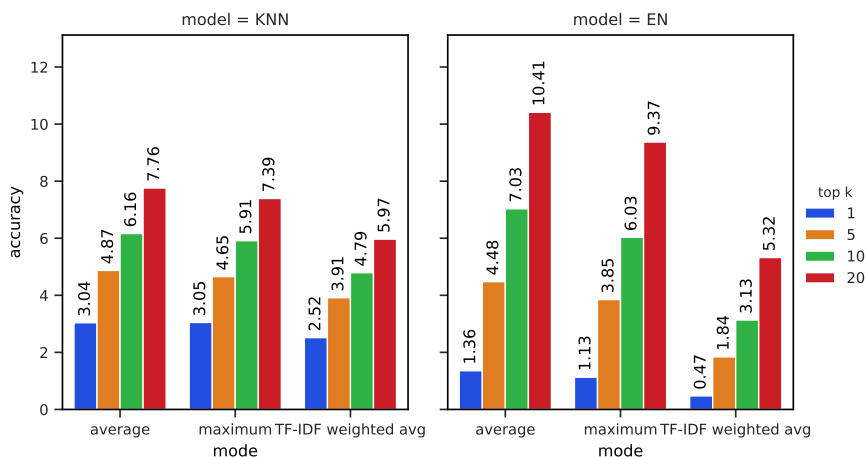


Figure 3.23: Bar graph comparing different ways of document embedding extraction from word embeddings.

#### 3.3.1 Sentence embedding

First, we will test the best way to get document embeddings from word embeddings. They can be obtained in the following ways:

1. Averaging word vectors.
2. Maximum of word vectors element-wise.
3. Averaging word vectors weighted by TF-IDF weights.

The results can be seen in Figure 3.23 and show that simply averaging the word vectors brings the best results. This is rather surprising because according to the research done beforehand, the TF-IDF weighted embeddings should have been the most accurate but it is perhaps too strong and penalizes infrequent words too much.

A similar test was done for BERT model where the embedding was extracted either by averaging sentence vectors or using as much as possible of the beginning of the article. The results are presented in Figure 3.24

#### 3.3.2 Results

The results comparing all feature extraction methods can be seen in Figure 3.25 and in Figure 3.26 and show that among the image feature extraction methods MobileNet is the winner. It achieved consistently the highest both top-1 and top-20 accuracy when used with different text feature extraction methods. Among the text feature extraction models, GloVe is also consistently the best performing model and thus will be used in the next experiment. Compared to the 3.06% top-1 accuracy and 7.81% top-20 accuracy of



### 3.4. Experiment 3 – Application on multiple domains

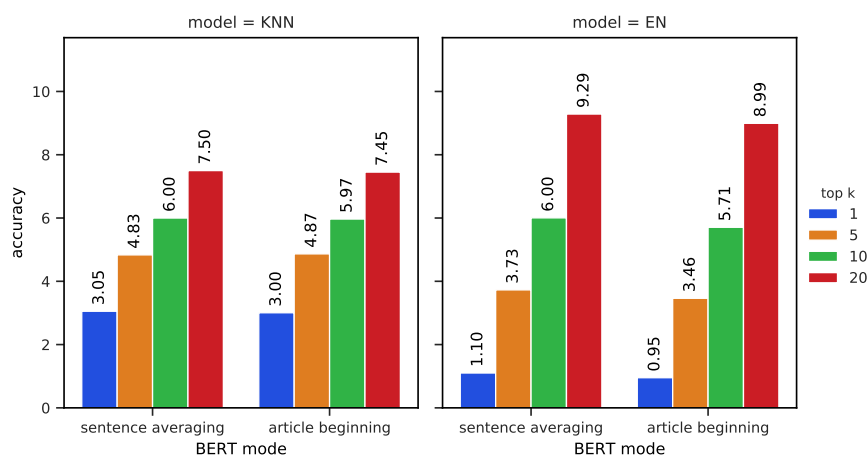


Figure 3.24: Bar graph comparing different ways of document embedding extraction from word embeddings using BERT model.

word2vec and InceptionNet features, we achieved 4.29% top-1 and 9.92% top-20 accuracy with the GloVe and MobileNet embeddings with the KNN model and 1.86% top-1 and 13.34% top-20 accuracy with the ElasticNet model.

On the whole, all the feature extraction algorithms have similar performance. This result is not surprising considering all these algorithms were designed to perform a similar task.

The state-of-the-art model BERT is unexpectedly the worst performing model. It is the only embedding technique that takes into account the context of the word, but this information is perhaps not utilized fully in our case and could hinder the performance.

### 3.4 Experiment 3 – Application on multiple domains

Based on the previous experiments, we use the best performing regression algorithms and feature extraction models and apply them on other domains in the whole dataset – business news, sports, health, technology. The entertainment news category was excluded due to the low number of samples. We hope to achieve similar or better results as in previous experiments and thus to prove that the whole methodology is coherent.

In order for the data to be comparable, the number of samples in all categories has to be equal. All categories had image duplicates removed and randomly subsampled to 9000 samples, which is the number of samples in the smallest category – health news. The models used are the KNN and ElasticNet models with the GloVe text features and MobileNet image features according to the results of previous experiments.

### 3. EXPERIMENTS

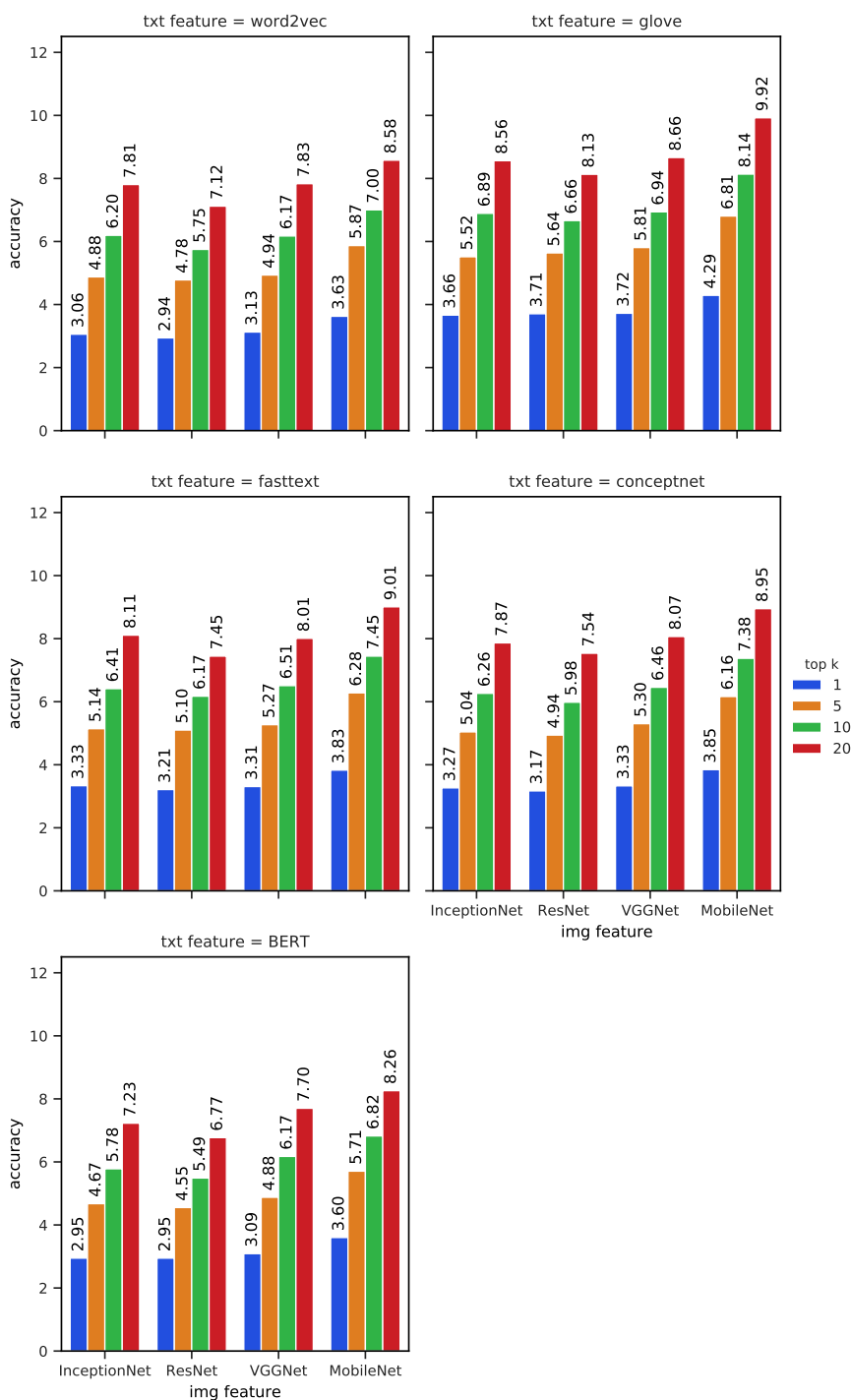


Figure 3.25: Bar graph with comparison of the image and text features with KNN model.

### 3.4. Experiment 3 – Application on multiple domains

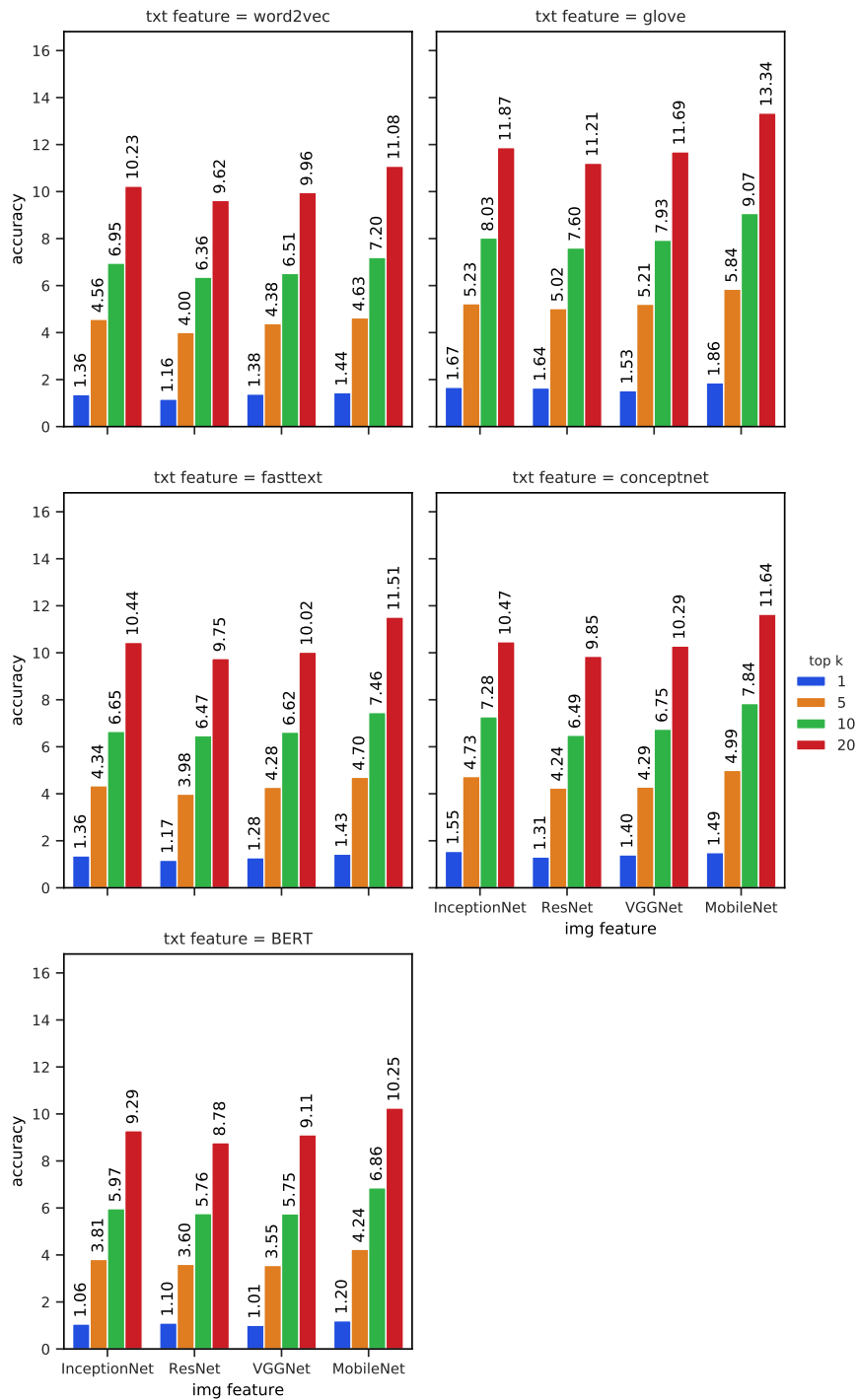


Figure 3.26: Bar graph with comparison of the image and text features with ElasticNet model.

### 3.4.1 Results

The result of the experiments can be seen in Figure 3.27 and shows that both the KNN and ElasticNet models perform the worst on the World news category. It may be either a matter of chance or because, generally speaking, the images in World News category are harder to predict. The KNN model performed the best on the Technology, Health and Business news, achieving up to 5.92% accuracy. Note that the higher accuracy is achieved also by the size of the testing dataset, which is around twice as small as in the previous experiments. The ElasticNet model achieved accuracy of nearly 25% top-20, i.e. the original picture was in the top 20 suggested images in 437 out of 1800 queries.

## 3.5 Experiment 4 – Dataset by a Czech publishing house

One of the assignments of the thesis is to test the performance of the recommender system on a dataset supplied by a publishing house, in our case the Aktualne dataset. Because we have previously run the experiments only on English language, we need to test the performance of the system with different text embeddings first. We have 4 models available: word2vec CBOW, word2vec Skip-gram, GloVe and FastText.

The results can be seen in Figure 3.28 and show that the FastText model is the most accurate. However, this is probably not because of the model architecture itself, but because the FastText model was trained on the largest corpus. The precise token count is not available, but FastText was trained on both Wikipedia and common crawl, whereas the other models were trained only on Wikipedia. Interestingly, our results confirm the results of the authors of word2vec models that their CBOW variant is performing better than their Skip-gram on Czech language.

The model was also tested on both categories of the Aktualne dataset. Even though both categories have more than 15 000 samples (with duplicate images removed) in both categories, they have been subsampled to 9 000 samples, so that the results are comparable to Experiment 3. The results in Figure 3.29 are similar to results from Reuters dataset in the way that in both of them, the “World news” or “Zahranici” categories are the hardest to predict. Articles in Czech language are harder to predict, because they score only 1.67% and 3.23% top-1 accuracy in the KNN model compared to the average top-1 accuracy of around 5% of articles in English language. This could however be not because of the complexity of the language itself, but the quality of the text embedding model.

### 3.5. Experiment 4 – Dataset by a Czech publishing house

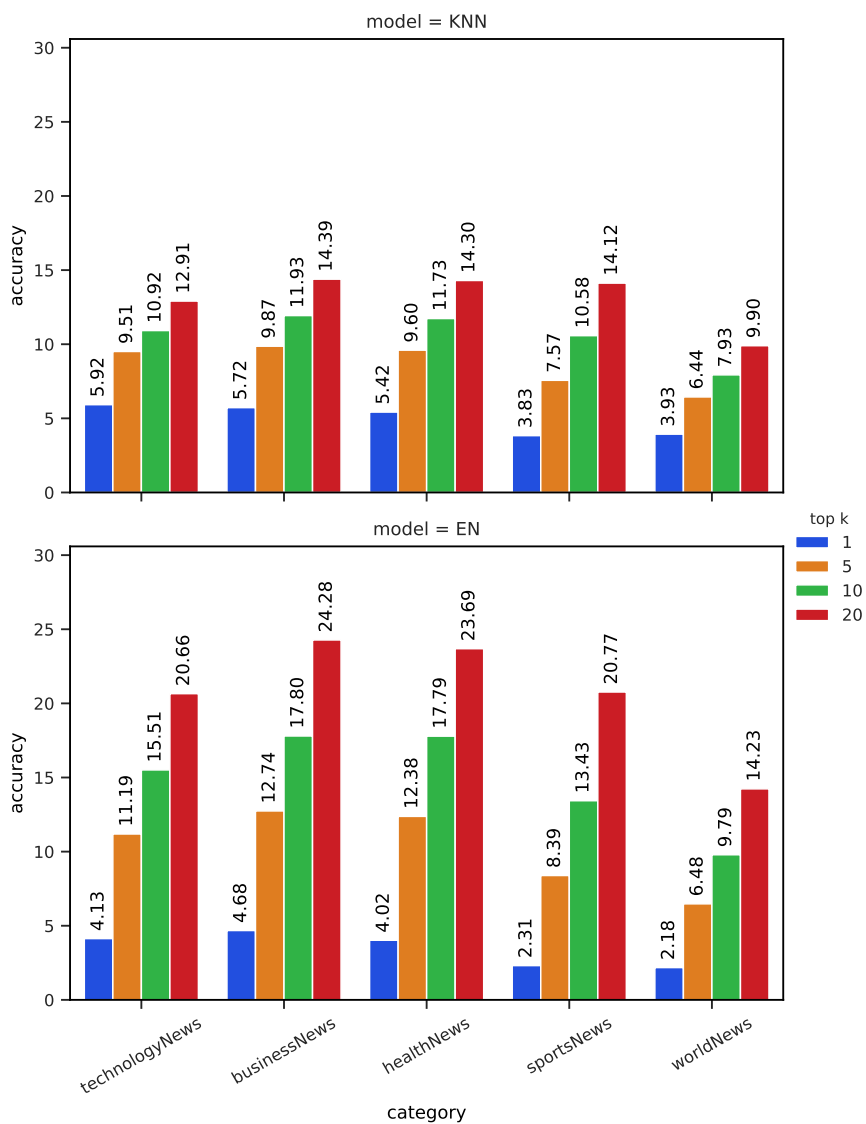


Figure 3.27: Bar graph with application of the algorithms on different domains.

### 3. EXPERIMENTS

---

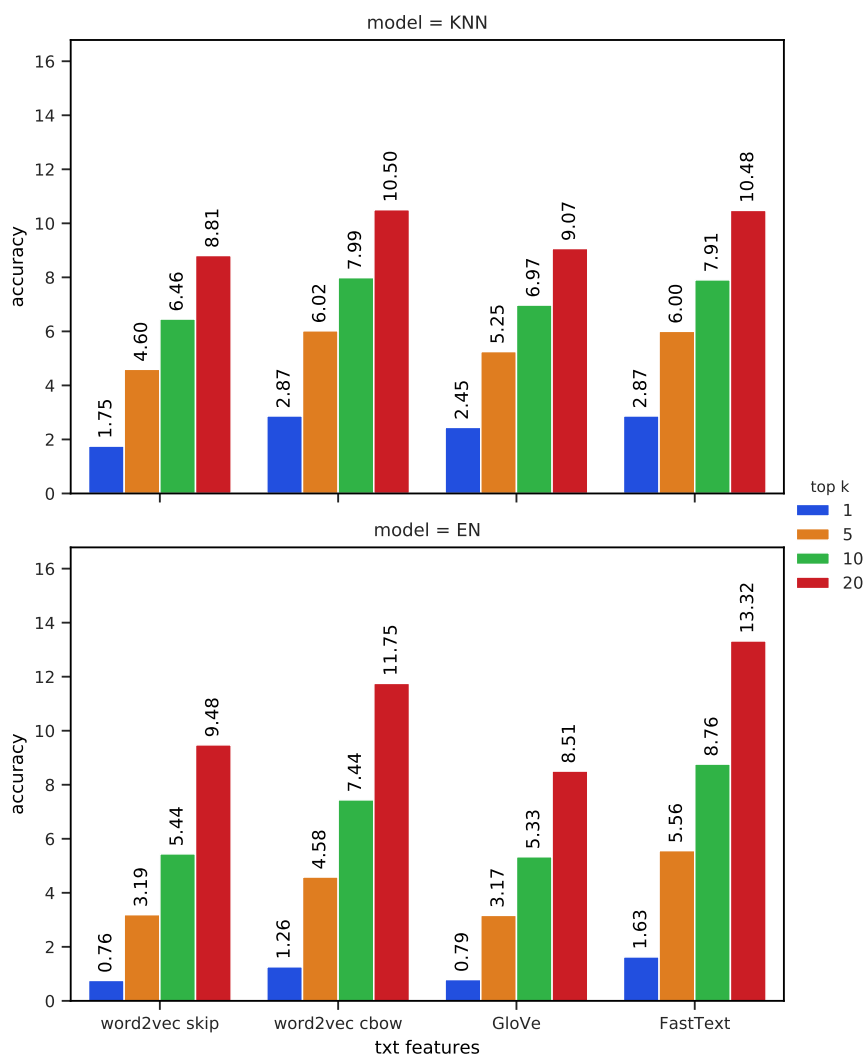


Figure 3.28: Bar graph of performance of text feature extraction models of Czech language.

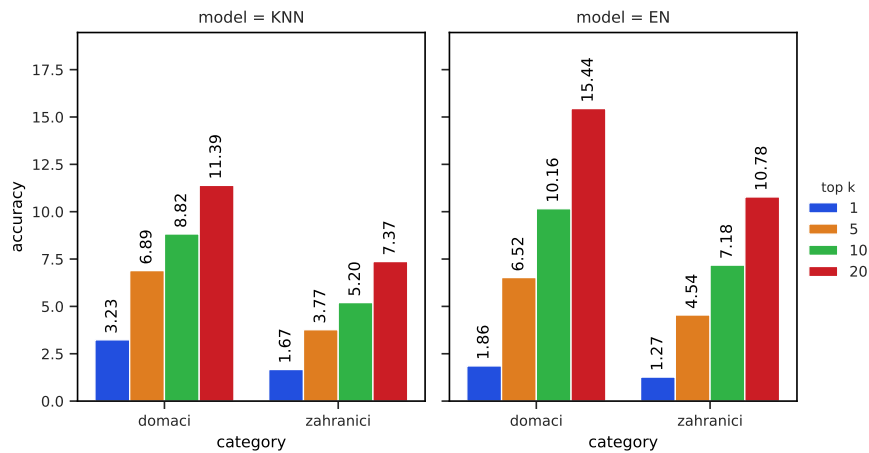


Figure 3.29: Bar graph with application of the algorithms on different domains from Aktualne dataset.

### 3.6 Results

In the chapter Experiments, we have tested numerous text and image feature extraction techniques and optimized a model for recommending images related to the text of an article. The best top-1 accuracy of more than 4% compared to a random model with less than 0.06% accuracy means more than 70 times increase. Using top-20 accuracy, which is a realistic number of images an editor might look at when choosing an image, the best model scored nearly 25% compared to a random model with 1% accuracy. However, the most important fact is that all the recommended images are related to the text and are not only random images. We do not have available data about how humans would cope with the non-trivial task of recommending images related to articles but we assume they would not achieve a very high accuracy either.

The objective of the thesis, i.e. to design and implement a system for image recommendation, has been fulfilled. The system is capable of recommending images across multiple domains in multiple languages. Examples of recommendations of images related to an article can be seen in Appendix A.

The system could be improved in several areas. The regression model is working well but there is still room for amendment. In the articles where one or multiple persons are depicted the model lacks the ability to recognize the person. It means that among the recommended images for an article about a certain personality, images of other people appear as well. This imperfection could be amended with a face recognition algorithm.





---

## Conclusion

The main objective of the thesis was to design a system capable of recommending images related to the text of an article, implement it and extend it to multiple domains or languages. Another aim was to survey state-of-the-art algorithms in image processing and text embedding with the focus on high quality neural embeddings.

In the first part, the theoretical background was introduced with the focus on state-of-the-art deep neural networks for image processing and neural language modeling for text feature embedding. Multiple additional algorithms required for the recommendation model were also researched. In the practical part, we obtained a large dataset for supervised learning of the algorithm and extracted image and text features. A system capable of recommending images was designed, implemented and optimized. The model was evaluated on multiple domains and extended to support an additional language.

All requirements of the assignment of this thesis have been fulfilled. The theoretical part of the thesis achieved the aim of researching state-of-the-art algorithms in the fields of image processing and text mining with the focus on neural networks. The practical part of the thesis achieved the goal of implementing the recommender system and was extensively optimized, tuned and evaluated on several domains and languages.

The model can be successfully used to predict images related to an article. With some adjustments, it can be used in real world application in a publishing house. Personally, I enjoyed the learning process while writing the thesis and intend to learn even more about machine learning, specifically about deep learning with neural networks.



---

## Bibliography

- [1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. “Tensorflow: a system for large-scale machine learning.” In: *OSDI*. Vol. 16. 2016, pp. 265–283.
- [2] Naomi S Altman. “An introduction to kernel and nearest-neighbor non-parametric regression”. In: *The American Statistician* 46.3 (1992), pp. 175–185.
- [3] Aphex34. *Convolutional layer*. 2015. URL: [https://commons.wikimedia.org/wiki/File:Conv\\_layer.png](https://commons.wikimedia.org/wiki/File:Conv_layer.png) (visited on 02/07/2019).
- [4] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. “A simple but tough-to-beat baseline for sentence embeddings”. In: (2016).
- [5] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. “Surf: Speeded up robust features”. In: *European conference on computer vision*. Springer. 2006, pp. 404–417.
- [6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. “A neural probabilistic language model”. In: *Journal of machine learning research* 3.Feb (2003), pp. 1137–1155.
- [7] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.
- [8] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. “Enriching word vectors with subword information”. In: *Transactions of the Association for Computational Linguistics* 5 (2017), pp. 135–146.
- [9] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. “An analysis of deep neural network models for practical applications”. In: *arXiv preprint arXiv:1605.07678* (2016).

- [10] Ronan Collobert and Jason Weston. “A unified architecture for natural language processing: Deep neural networks with multitask learning”. In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 160–167.
- [11] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. “Indexing by latent semantic analysis”. In: *Journal of the American society for information science* 41.6 (1990), pp. 391–407.
- [12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. “Imagenet: A large-scale hierarchical image database”. In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. Ieee. 2009, pp. 248–255.
- [13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).
- [14] Michel Marie Deza and Elena Deza. “Encyclopedia of distances”. In: *Encyclopedia of Distances*. Springer, 2009, pp. 1–583.
- [15] Manaal Faruqui, Jesse Dodge, Sujay K Jauhar, Chris Dyer, Eduard Hovy, and Noah A Smith. “Retrofitting word vectors to semantic lexicons”. In: *arXiv preprint arXiv:1411.4166* (2014).
- [16] Kelwin Fernandes, Pedro Vinagre, and Paulo Cortez. “A proactive intelligent decision support system for predicting the popularity of online news”. In: *Portuguese Conference on Artificial Intelligence*. Springer. 2015, pp. 535–546.
- [17] Salvador García, Julián Luengo, and Francisco Herrera. *Data preprocessing in data mining*. Springer, 2015.
- [18] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems.* ” O’Reilly Media, Inc.”, 2017.
- [19] Google. *word2vec, Tool for computing continuous distributed representations of words*. 2019. URL: <https://code.google.com/archive/p/word2vec/> (visited on 02/07/2019).
- [20] Caglar Gulcehre, Marcin Moczulski, Misha Denil, and Yoshua Bengio. “Noisy activation functions”. In: *International Conference on Machine Learning*. 2016, pp. 3059–3068.
- [21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

- 
- [22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Identity mappings in deep residual networks”. In: *European conference on computer vision*. Springer. 2016, pp. 630–645.
- [23] Sepp Hochreiter. “The vanishing gradient problem during learning recurrent neural nets and problem solutions”. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), pp. 107–116.
- [24] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.
- [25] Harold Hotelling. “Analysis of a complex of statistical variables into principal components.” In: *Journal of educational psychology* 24.6 (1933), p. 417.
- [26] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. “Mobilenets: Efficient convolutional neural networks for mobile vision applications”. In: *arXiv preprint arXiv:1704.04861* (2017).
- [27] Jeremy Howard and Sebastian Ruder. “Universal language model fine-tuning for text classification”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vol. 1. 2018, pp. 328–339.
- [28] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *arXiv preprint arXiv:1502.03167* (2015).
- [29] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).
- [30] *Know your meme: We need to go deeper*. 2013. URL: <https://knowyourmeme.com/memes/we-need-to-go-deeper> (visited on 02/07/2019).
- [31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.
- [32] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.
- [33] Alexander LeNail. “NN-SVG: Publication-Ready Neural Network Architecture Schematics”. In: *Journal of Open Source Software* 4.33 (2019), p. 747. DOI: 10.21105/joss.00747.
- [34] Stuart Lloyd. “Least squares quantization in PCM”. In: *IEEE transactions on information theory* 28.2 (1982), pp. 129–137.

- [35] David G Lowe. “Distinctive image features from scale-invariant keypoints”. In: *International journal of computer vision* 60.2 (2004), pp. 91–110.
- [36] Kevin Lund and Curt Burgess. “Hyperspace analogue to language (HAL): A general model semantic representation.” In: *Brain and Cognition*. Vol. 30. 3. ACADEMIC PRESS INC JNL-COMP SUBSCRIPTIONS 525 B ST, STE 1900, SAN DIEGO, CA .... 1996, pp. 5–5.
- [37] Z. Markov and D.T. Larose. *Data Mining the Web: Uncovering Patterns in Web Content, Structure, and Usage*. Wiley series on methods and applications in data mining. Wiley, 2007, p. 33. ISBN: 9780471666554.
- [38] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).
- [39] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. “Distributed representations of words and phrases and their compositionality”. In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.
- [40] Andriy Mnih and Koray Kavukcuoglu. “Learning word embeddings efficiently with noise-contrastive estimation”. In: *Advances in neural information processing systems*. 2013, pp. 2265–2273.
- [41] Andrew Y Ng, Michael I Jordan, and Yair Weiss. “On spectral clustering: Analysis and an algorithm”. In: *Advances in neural information processing systems*. 2002, pp. 849–856.
- [42] Maximilian Nickel, Lorenzo Rosasco, Tomaso A Poggio, et al. “Holographic Embeddings of Knowledge Graphs.” In: *AAAI*. Vol. 2. 1. 2016, pp. 3–2.
- [43] Robert Parker, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. “English gigaword fifth edition, linguistic data consortium”. In: *Google Scholar* (2011).
- [44] Karl Pearson. “LIII. On lines and planes of closest fit to systems of points in space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572.
- [45] F. Pedregosa, G. Varoquaux, A. Gramfort, et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [46] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. “GloVe: Global Vectors for Word Representation”. In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543.

- 
- [47] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. “Deep contextualized word representations”. In: *arXiv preprint arXiv:1802.05365* (2018).
- [48] Quartl. *Vector p-Norms*. 2011. URL: [https://commons.wikimedia.org/wiki/File:Vector-p-Norms\\_qt11.svg](https://commons.wikimedia.org/wiki/File:Vector-p-Norms_qt11.svg) (visited on 02/07/2019).
- [49] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. “Improving language understanding by generative pre-training”. In: URL [https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/languageunsupervised/language\\_understanding\\_paper.pdf](https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/languageunsupervised/language_understanding_paper.pdf) (2018).
- [50] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. “Hogwild: A lock-free approach to parallelizing stochastic gradient descent”. In: *Advances in neural information processing systems*. 2011, pp. 693–701.
- [51] 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*. ELRA, May 2010, pp. 45–50.
- [52] *Reuters*. URL: <https://www.reuters.com/> (visited on 01/20/2019).
- [53] Frank Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65.6 (1958), p. 386.
- [54] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), p. 533.
- [55] Tom Runia. *TF-FeatureExtraction*. 2018. URL: [https://github.com/tomrunia/TF\\_FeatureExtraction](https://github.com/tomrunia/TF_FeatureExtraction).
- [56] Olga Russakovsky, Jia Deng, Hao Su, et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.
- [57] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. In: *arXiv preprint arXiv:1801.04381* (2018).
- [58] Hinrich Schütze. “Word space”. In: *Advances in neural information processing systems*. 1993, pp. 895–902.
- [59] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).
- [60] Robert Speer, Joshua Chin, and Catherine Havasi. “ConceptNet 5.5: An Open Multilingual Graph of General Knowledge.” In: *AAAI*. 2017, pp. 4444–4451.

- [61] Lukáš Svoboda and Tomáš Brychcín. “New word analogy corpus for exploring embeddings of Czech words”. In: *Computational Linguistics and Intelligent Text Processing*. Springer, Apr. 2016, pp. 103–114. DOI: 10.1007/978-3-319-75477-2.
- [62] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. “Inception-v4, inception-resnet and the impact of residual connections on learning.” In: *AAAI*. Vol. 4. 2017, p. 12.
- [63] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.
- [64] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. “Rethinking the inception architecture for computer vision”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.
- [65] Andrew Thompson. *All the news Dataset*. 2017. URL: <https://www.kaggle.com/snapcrack/all-the-news/home> (visited on 01/20/2019).
- [66] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. “Attention is all you need”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5998–6008.
- [67] Quan Wang. “Kernel principal component analysis and its applications in face recognition and active shape models”. In: *arXiv preprint arXiv:1207.3538* (2012).
- [68] Lilian Weng. *Generalized Language Models*. 2019. URL: <https://lilianweng.github.io/lil-log/2019/01/31/generalized-language-models.html> (visited on 02/07/2019).
- [69] John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. “Character: Embedding words and sentences via character n-grams”. In: *arXiv preprint arXiv:1607.02789* (2016).
- [70] Han Xiao. *bert-as-service*. 2018. URL: <https://github.com/hanxiao/bert-as-service>.
- [71] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. “Aggregated residual transformations for deep neural networks”. In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE. 2017, pp. 5987–5995.
- [72] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. “BIRCH: an efficient data clustering method for very large databases”. In: *ACM Sigmod Record*. Vol. 25. 2. ACM. 1996, pp. 103–114.



- [73] Quan Zhou, Wenlin Chen, Shiji Song, Jacob R Gardner, Kilian Q Weinberger, and Yixin Chen. “A Reduction of the Elastic Net to Support Vector Machines with an Application to GPU Computing.” In: *AAAI*. 2015, pp. 3210–3216.
- [74] Hui Zou, Trevor Hastie, Robert Tibshirani, et al. “On the “degrees of freedom” of the lasso”. In: *The Annals of Statistics* 35.5 (2007), pp. 2173–2192.



## **Recommendations examples**

## A. RECOMMENDATIONS EXAMPLES

Na Orlickoústecku začal hořet za jízdy autobus. Jelo v něm 23 dětí, nikdo se nezranil. Oheň vzplál v motorové části autobusu. Řidič se snažil požár uhasit sám, ale nakonec plameny zlikvidovali až hasiči. Cotkytle ( Orlickoústecko ) - Hasiči v úterý dopoledne likvidovali požár autobusu v Cotkytli na Orlickoústecku. Cestovalo jím 23 dětí na školu v přírodě a hořet začal za jízdy. "Při požáru nebyl nikdo zraněn," uvedla mluvčí krajských hasičů Vendula Horáková. "Děti mířící na školu v přírodě naštěstí neměly cestu do cíle dlouhou. Autobus začal hořet zhruba 300 metrů před cílem. Děti tak do hotelu dorazily pěšky a byly v pořádku," dodala Horáková. Oheň vzplál v motorové části autobusu. Řidič se snažil požár uhasit sám, ale nakonec plameny zlikvidovali až hasiči. Příčinou vzniku požáru byla s největší pravděpodobností technická závada na elektroinstalaci. Škoda byla předběžně vyčíslena na 500 tisíc korun.



Figure A.1: Recommendations for an article about a bus fire from the Aktuálne dataset, category domáci. From top to bottom is the article text, image and recommended images.

---

Kremlin says still waiting for U.S. talks to set up Putin-Trump summit MOSCOW (Reuters) - The Kremlin said on Friday it was still waiting for substantive talks with the United States to set up a summit between Russian President Vladimir Putin and U.S. President Donald Trump. Trump said in March that the two leaders would meet soon, but since then already poor ties between Washington and Moscow have deteriorated further over the conflict in Syria and the poisoning of a former Russian spy in Britain. Kremlin spokesman Dmitry Peskov told reporters on a conference call on Friday that there was still no clarity on a possible meeting between the two leaders and that no further steps had been taken by Washington to arrange it. "We are waiting," Peskov said.



Figure A.2: Recommendations for an article about USA and Russian politics from Reuters Archive dataset, category World News. From top to bottom is the article text, image and recommended images.

## A. RECOMMENDATIONS EXAMPLES

---

Off day' leaves Hamilton seeking his lost rhythm BAKU (Reuters) - Triple Formula One world champion Lewis Hamilton blamed a rare 'off day' for an error-filled qualifying performance on Saturday at the Baku street circuit that hosts Azerbaijan's first grand prix. The Briton, winner of the two previous races in Monaco and Montreal, will start in 10th place after clipping a wall and breaking his car's suspension while Mercedes team mate and rival Nico Rosberg took pole position. Hamilton also ran off the track twice. The champion is nine points adrift of the German after seven races and hoped he might be able to regain the lead in Baku but that looks a tall order now unless misfortune strikes Rosberg. "It just wasn't a good one," Hamilton told reporters. ...



Figure A.3: Recommendations for an article about Formula 1 racing from Reuters Archive dataset, category Sports News. From top to bottom is the article text, image and recommended images.

---

## Glossary

- BERT** Bidirectional encoder representations from transformers
- Birch** Balanced iterative reducing and clustering using hierarchies
- BOW** Bag of words
- CBOW** Continuous bag of words
- CNN** Convolutional neural network
- ELMo** Embeddings from language models
- Extratrees** Extremely randomized forest
- GPU** Graphics processing unit
- ILSVRC** ImageNet Large Scale Visual Recognition Challenge
- KNN**  $k$ -nearest neighbors
- Lasso** Least absolute shrinkage and selection operator
- LSA** Latent semantic analysis
- LSTM** Long short-term memory model
- MAE** Mean absolute error
- MLP** Multilayer perceptron
- MNIST** Modified National Institute of Standards and Technology database
- MSE** Mean square error
- MSLE** Mean square logarithmic error

## B. GLOSSARY

---

- PCA** Principal component analysis
- PPMI** Ppositive pointwise mutual information
- ReLU** Rectified linear unit
- ResNet** Residual network
- RMSE** Root mean square error
- RMSProp** Root mean square propagation
- RNN** Residual neural network
- SIFT** Scale-invariant feature transform
- SURF** Speeded-up robust features
- SVD** Singular value decomposition
- SVM** Support vector machine
- SVR** Support vector regression
- t-SNE** t-distributed stochastic neighbor embedding
- TanH** Hyperbolic tangent
- TF-IDF** Term frequency – inverse document frequency
- VGGNet** Visual geometry group network
- vLBL** Vector log-bilinear model



---

## Contents of USB

<code>readme.md</code> .....	the file with USB contents description
<code>src</code> .....	the directory of source codes
<code>thesis</code> .....	the directory of $\text{\LaTeX}$ source codes of the thesis
<code>images</code> .....	the thesis images directory
<code>plots</code> .....	the thesis plots directory
<code>recommendations</code> .....	the thesis appendix directory
<code>*.tex</code> .....	the $\text{\LaTeX}$ source code files of the thesis
<code>impl</code> .....	the directory of the implemented program
<code>*.ipynb</code> .....	the IPython Notebook source codes
<code>*.py</code> .....	the Python source codes
<code>datasets</code> .....	datasets used in source codes
<code>models</code> .....	models used in source codes
<code>text</code> .....	the thesis text directory
<code>_DP_Pištora_Matouš_2018.pdf</code> ....	the Diploma thesis in PDF format