Optimal Transport in Text Classification

Research Case Studies closing report

presented at the Department IV
of Trier University

by

Bharath Kumar Nagaraju, Chakravenu Gandham, Martin Jaehde

(s4bhnaga,s4chgand,s4majaeh)@uni-trier.de

Supervisor: Prof. Dr. Volker Schulz

Trier, March 27, 2023

Contents

1 Introduction

Optimal transport is a mathematical framework for measuring the distance between two probability distributions. The main idea behind optimal transport is to find the most efficient way of transforming one distribution into the other, subject to certain constraints.

This can be thought of as a transportation problem: It is a common occurrence that children have the great idea to build their sandcastles in the grass instead of the sandbox. Then their parents face the task of restoring the initial situation in the most efficient manner possible. This problem is the perfect approach to introduce the theory of optimal transportation. The pile and the hole in the sandbox have the same volume which can be normalized [2]. Hence, they can be modeled by probability measures μ and ν defined respectively on some measure spaces X and Y. The energy or the cost consumed consumed by moving the pile of sand from X to Y is measured by a non-negative cost function

$$c : X \times Y \to \mathbb{R}_0^+ \cup \{+\infty\}.[2]$$

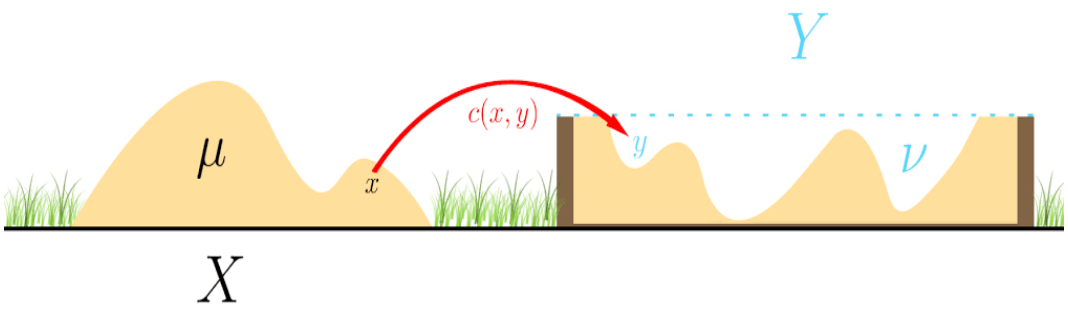

Figure 1.1: How to move pile X to the shape of pile Y with minimal effort? I.e. given a cost function c(x,y) of moving a grain $x \in X$ to the position $y \in Y$, what is the optimal displacement of $x \in X$ to $y \in Y$? ref: [2]

In the same way, optimal transport seeks to find the most efficient way of transforming one distribution into another. For example, imagine that you have a distribution of gray pixels in one image and a distribution of colored pixels in another image. The goal of optimal transport would be to find the most efficient way of transforming the gray distribution into the colored distribution, such that the overall transformation is as smooth and efficient as possible.

Optimal transport has many applications in fields such as computer vision, economics, and machine learning. In computer vision, it can be used to align images or to compute a distance between two images. In economics, it can be used to model transportation costs or to analyze market equilibrium. In machine learning, it has become increasingly popular for measuring the distance between probability distributions, particularly in the context of generative modeling and domain adaptation.

The problem of optimal transport can be formulated as a linear programming problem, and there are several algorithms for solving it, including the simplex method and the interior point method. The Wasserstein distance, which is a particular instance of optimal transport distance, has become increasingly popular in machine learning and statistics as a way of measuring the distance between probability distributions.

Overall, optimal transport is a powerful mathematical tool that has a wide range of applications in many different fields. Its ability to measure the distance between probability distributions makes it particularly useful in situations where traditional distance metrics, such as Euclidean distance, may not be appropriate.

2 Mathematical Foundations of Optimal Transport

2.1 Optimal Transport between histograms and discrete measures

Definition 1 [12] : A probability vector (also known as histogram) a is a vector with positive entries that sum to one.

$$a \in I\!R_+^n, \text{ such that } \sum_{i=1}^{n} a_i = 1$$

The probability simplex $\sum_n$ is the set of all histograms:

$$\sum_n := \left\{ a \in I\!R_+^n : \sum_{i=1}^{n} a_i = 1 \right\}$$

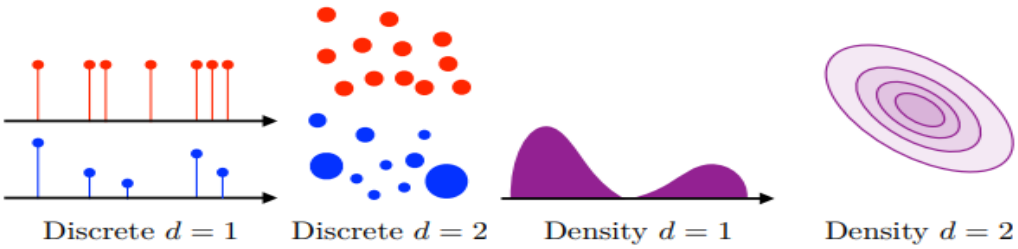

Figure 2.1: Schematic display of discrete distributions $\alpha = \sum_{i=1}^{n} a_i \delta_{x_i}$ (red corresponds to empirical uniform distribution a $a_i = 1/n$, and blue to arbitrary distributions) and densities $d\alpha(x) = \rho_\alpha(x)dx$ (in purple), in both one and two dimensions. Discrete distributions in one-dimension are displayed as stem plots (with length equal to a_i) and in two dimensions using point clouds (in which case their radius might be equal to a_i or, for a more visually accurate representation, their area) ref [12] .

Definition 2 [12] : A discrete measure α represents weighted points in space X. Namely,

$$\alpha = \sum_{i=1}^{n} a_i \delta_{x_i} \tag{2.1}$$

where a is vector of weights and $x_1 \ldots\ldots x_n \in X$ are the locations. As usual, δ_x is the Dirac's function, which equals 1 at location x and 0 everywhere else.

2.2 Monge's optimal transportation problem

The Monge problem, also known as the Monge-Kantorovich problem, is one of the foundational problems in optimal transport theory. It was first introduced by **Gaspard Monge** in 1781, and later extended by **Leonid Kantorovich** in the 1940s.

Here is a case for Monge problem: In this we are given two probability distributions, what is the most efficient way of transforming one distribution into the other? More specifically, given two measures μ and ν on two metric spaces X and Y, respectively, the Monge problem seeks to find a transport plan T that maps μ to ν, while minimizing the total cost of transportation.

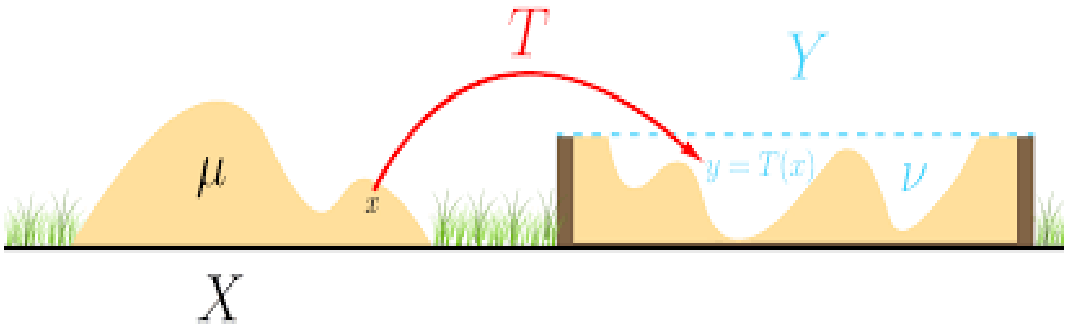

Figure 2.2: Moving pile of sand from X to Y ref: [2]

To understand Monge problem more easily let us consider a case in which there are two uniform probability vectors **e** and **f** of equal size n. Assume $\boldsymbol{C_{ij}}$ be the cost or energy consumed matrix of moving bin i of **e** to bin j of **f** . e and f are uniform and equals 1/n, will concerned about weight/mass transportation later. Let us take the permutation σ for minimum cost of transportation.

$$\min_{\sigma \in \text{Perm}(n)} \frac{1}{n} \sum_{i=1}^{n} C_{i,\sigma(i)} \tag{2.2}$$

Even though bins are uniform and equals 1/n not necessary the case in optimization problem, to keep natural generalization to weight transportation. This issue is known as the **assignment problem** [12].

The solution is not unique, consider the following example of transporting $x_i \rightarrow y_j$, for $i, j \in \{1, 2\}$.

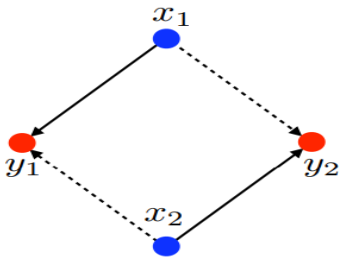

Figure 2.3: blue dots from measure α and red dots from measure β are pairwise equidistant. Hence, either matching $\sigma = (1, 2)$ (full line) or $\sigma = (2, 1)$ (dotted line) is optimal. Let us move on to a more general case: probability vectors/discrete distributions with different masses. Since we cannot transport a bin of some mass to another bin with different mass, we have a new restriction to take care of. Namely, if α ,β are discrete measures ref: [12]

$$\alpha = \sum a_i \delta_{x_i} \qquad \beta = \sum b_i \delta_{y_i}$$

We have to find a function T

$$T : x_1, ..., x_n \rightarrow y_1, ..., y_m, \text{ such that } b_j = \sum_{i:T(x_i)=y_j} a_i$$

This Transport function(T) is called **push forward operator** and the mass conservation condition is:

$$T_{\#}\alpha = \beta$$

The cost of transporting a unit of mass from point x in X to point y in Y is given by a cost function c(x,y), which measures the transportation cost between x and y.

$$\min_T \left\{ \sum_i c(x_i, T(x_i)) : \ T_{\#}\alpha = \beta \right\} \tag{2.3}$$

Over the set of all measurable maps T such that

$$T_{\#}\alpha = \beta$$

In the case where the cost function c(x,y) is a distance metric, the Monge problem reduces to the classical transportation problem, which can be solved using linear programming techniques. However, in many applications of optimal transport, the cost function is not a distance metric, and the Monge problem is much more challenging to solve.

2.3 Kantorovich Relaxation

What if the probability distributions don't match up? Let us consider a final generalization, which supports to solve this kind of issues by allowing the mass at any point x_i to be dispatched across several locales. This inflexibility is known as **Kantorovich relaxation**.

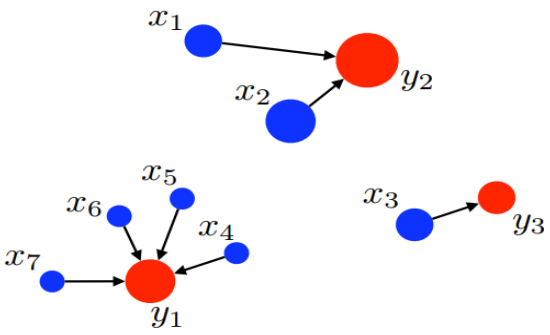

Figure 2.4: Monge chart ref: [12]

A Monge chart can associate the blue measure α_i to the red measure β_j. The weights α_i are displayed proportionally to the area of the fragment marked at each position. The mapping then's similar that $T(x_1) = T(x_2) = y_2, T(x_3) = y_3$, whereas for $4 \leq i \leq 7$ we have $T(x_i) = y_1$.

In place of a permutation σ or a chart T, a coupling matrix $P \in R^{n \times m}$ is sought. $P_{i,j}$ describes the quantum of mass flowing from bin i to another bin j. Note that, to be permissible, the matrix P must satisfy that the sum of the amount of weight going out should be equals to the total weight of that position. Also, the sum of total weight transported to a position must equal the total weight/mass of the target position.

Mathematically, the set of permissible couplings is as shown in (2.4)

$$U(a, b) := \left\{ P \in \mathbb{R}_{+}^{n \times m} : P\mathbb{1}_m = a,\ P^T\mathbb{1}_n = b \right\} \tag{2.4}$$

Kantorovich's optimal transport problem now reads as in (2.5)

$$L_C(a, b) = \min_{P \in U(a,b)} \langle C, P \rangle = \sum_{i,j} C_{i,j} P_{i,j} \tag{2.5}$$

This is a linear program, and as is generally the case with similar programs, its optimal results aren't inescapably unique.

Kantorovich problem between arbitrary measures:

The optimal transport plan itself (moreover as a coupling P or a Monge chart 2.3 T when it exists) has set up numerous operations in DataSciences, and in particular image processing. It has, for case, been used for discrepancy equalization [3].

For the sake of absoluteness, leave the optimal transport equations for arbitrary measures. The delineations are principally the same as for discrete or separate measures.

An arbitrary measure α on (X, d), being d a distance, can be assessed by integrating it with $f(x)$ a continuous function (2.6) :

$$\int_X f(x)d\alpha(x) \tag{2.6}$$

The set of all measures for which the equation (2.6) is finite is known as the set of Radon measures $M(X)$.

The **push forward measure** :

Below you can see an illustration 2.3 of a coupling π between two arbitrary measures. By mentally integrating π on the X-axis (horizontal axis), one can note that the integral coincides with β. Analogously with α through the integration on the y-axis (vertical).

$$\text{Let } \alpha \in \mathcal{M}(X),\ b \in \mathcal{M}(Y). \text{Then } T_{\#}\alpha = \beta \text{ if}$$

$$\forall h \in \mathcal{C}(Y) \int_Y h(y)d\beta(x) = \int_X h(T(x))d\alpha(x) \tag{2.7}$$

Monge problem for arbitrary measures:

$$\min_T \left\{ \int c(x, T(x))d\alpha(x) :\ T_{\#}\alpha = \beta \right\}$$

permissible couplings of arbitrary measures

$$u(\alpha, \beta) \coloneqq \left\{ \pi \in \mathcal{M}_+^1(X \times Y) : T_{\#}(\alpha) = \beta \right\}$$

$$P_X(x, y) = x \ \& \ P_Y(x, y) = y$$

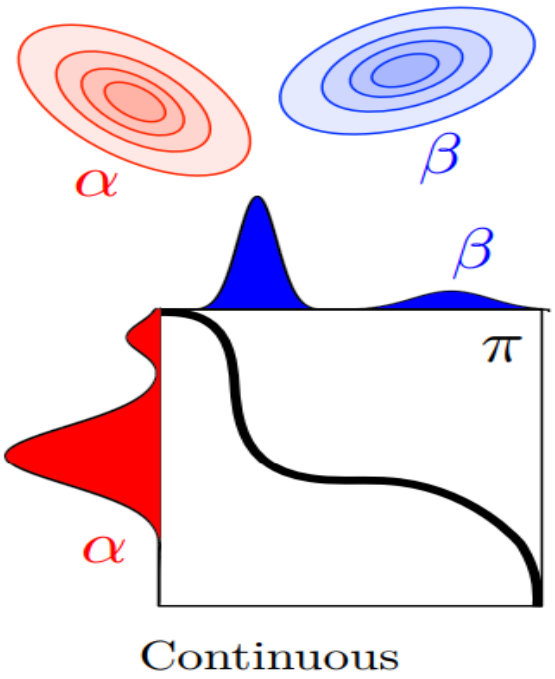

Figure 2.5: Schematic viewed of input measures (α, β) and couplings $U(\alpha, \beta)$ for Kantorovich OT, ref: [12] .

$$\mathcal{L}_{\mathcal{C}}(\alpha, \beta) := \min_{\pi \in u(\alpha,\beta)} \int_{X \times Y} C(x, y) d\pi(x, y) \tag{2.8}$$

If X=Y and c(x,y)=d(x,y), **Wasserstein distance for arbitrary measures** is given as (2.9) :

$$\mathcal{W}_p(\alpha, \beta) = \mathcal{L}_d p(\alpha, \beta)^{1/p} \tag{2.9}$$

2.4 Sinkhorn's Theorem and Sinkhorn-Knopp Algorithm

Here we are going to bandy about Sinkhorn's work this substantially grounded on paper [6].The Sinkhorn's theorem states that every square matrix with positive elements can be transformed into a doubly stochastic matrix D_1AD_2.Consider two diagonal matrices with all positive main diagonals and unique up to a constant factor, namely D_1 and D_2 .But there is problem with matrix which is having all non-negative square matrix that is both row-normalized and column-normalized is called as doubly stochastic matrix [17] .The equation (2.10) shows a non-negative square matrix.

$$\mathcal{A} \in \mathcal{R}^{\mathcal{N}\times\mathcal{N}} \sum_{i=1}^{\mathcal{N}} a_{ij} = 1 \,\forall j \in [1,\mathcal{N}]. \text{ Also, } \sum_{j=1}^{\mathcal{N}} a_{ij} = 1 \,\forall i \in [1,\mathcal{N}]. \tag{2.10}$$

Sinkhorn's theorem, the considered matrix should be a square matrix and contain non negative element. We can observe afterwards that these properties are satisfied by many of the problems that can be actually framed to calculate a corresponding doubly stochastic matrix. Sum of non-negative integers into one has many direct interpretations that shows Sinkhorn's algorithm most powerful, like for example, probabilities, or, a soft sorting indices.

This theorem is a efficient algorithm to calculate an approximation of doubly stochastic matrix that has linear convergence. This theorem just need to iteratively normalize the matrix along columns and rows.

As mentioned above it requires to iterate, as it has a complexity of $\boldsymbol{O}(\boldsymbol{N}^2)$ if applied for constant times at every step. Here if we take normalization as matrix multiplication, this algorithm is fully differentiable. If we have **N** - starting points and **N** - destinations. Also, distance matrix $\boldsymbol{M} \in \mathcal{R}^{\boldsymbol{N}\times\boldsymbol{N}}$ where entry m_{ij} shows the distance, or in another word, the transportation cost of items from base point **i** to target **j**.

By using Kantorovich relaxation as discussed 2.3 , lets represent **A** as a joint probability matrix, we can define its entropy, the marginal probability entropy, and KL-divergence between two transportation matrix. As follows:

$$H(A) = \sum_{ij} a_{ij} log_{ij} \tag{2.11}$$

$$H(r) = \sum_{i=1}^{N} \left(\sum_{j=1}^{N} a_{ij} \right) log_{ij} \sum_{j=1}^{N} a_{ij} \tag{2.12}$$

$$H(c) = \sum_{j=1}^{N} \left(\sum_{i=1}^{N} a_{ij} \right) log_{ij} \sum_{i=1}^{N} a_{ij} \tag{2.13}$$

$$\mathcal{D}_{KL}(A\|B) = \sum_{ij} a_{ij} log \frac{a_{ij}}{b_{ij}} \tag{2.14}$$

These distributions are having plain probability, so the joint probabilities inequality is present :

$$H(A) \leq H(r) + H(c) \tag{2.15}$$

From the equation (2.15) shows inequality, as when r is independent from c, the equality holds then distributions changes.

Objective of OT becomes:

$$\mathcal{P} = arg \min_{A \in U(r,c)} \langle A, M \rangle - \frac{1}{\lambda} H(A) \tag{2.16}$$

It is known in the field of OT that the solution to this kind of *argmin* problem is a rescaling of the matrix K= - λ M [12] , which could be expressed as $\tilde{\boldsymbol{K}} = \boldsymbol{Diag}(\boldsymbol{\mu})\boldsymbol{K}\boldsymbol{Diag}(\boldsymbol{\nu})$, and with the following constraints:

$$u \circ (\tilde{K}.\nu) = r \tag{2.17}$$

$$(u.\tilde{K}) \circ \nu = c \tag{2.18}$$

Sinkhorn iteration is used to solve this king of re-scaling issues. Even though in equations (2.17) and (2.18) we are not using Sinkhorn algorithm exactly, we have set of constraints regularizing r,c, but can be resolved by mapping the row-marginal to r and column-marginal to c respectively.

The doubly stochastic matrix obtained through the Sinkhorn algorithm can be used to solve optimization problems such as optimal transport and entropy regularization [1]. Sinkhorn's algorithm has found applications in various fields such as computer vision, machine learning [15], geometric domains [18] and computational neuroscience, among others [13].

3 Text Classification with Optimal Transport

In the last chapter, we have seen the mathematical basics of Optimal Transport. Now the question arises in which application areas OT can be used. In this work, we focus on the application in text classification. Therefore, this chapter is dealing with the way of usage of OT in a text classification scenario. It is mainly based on the paper [8].

The general idea is to transform a document into a probability distribution of its words. We can then use the distance value, which we will be the minimal cost of the Optimal Transport problem, for classification using the K-nearest-neighbor (KNN) principle. To be able to use Optimal Transport in this case we additionally first embed words into a vector space and then use the Euclidean distance. Because of this, we will first introduce the Word2Vec Framework which was presented in [10]. After that, we then show in more detail how KNN can be used with Optimal Transport to approach a text classification task.

3.1 Word2Vec Framework

There are several strategies to work with data in textual format. One very common strategy is to use word embeddings, i.e. to transform words into high dimensional vectors while maintaining semantic and syntactic relations. This enables us to use well known techniques from mathematics to analyze the data. The Word2Vec framework presents two neural network architectures which are used to do the word embedding: Continuous Bag-of Words Model (CBOW) and Continuous Skip-Gram Model. Here we will just give a brief overview over the two models. For more details, take a look into [10]. Typically the dimension of the vectors lie between 50 and 300. The vector representations resulting out of these models are very powerful and if trained on enough data they can learn different semantic relations in the sense that e.g. $vector("King") - vector("Man") + vector("woman") \approx vector("Queen")$ or $vector("Paris") - vector("France") + vector("Italy") \approx vector("Rome")$[10]. Further, semantically related words lie close to each other in the embedded vector space, illustrated in figure 3.1.

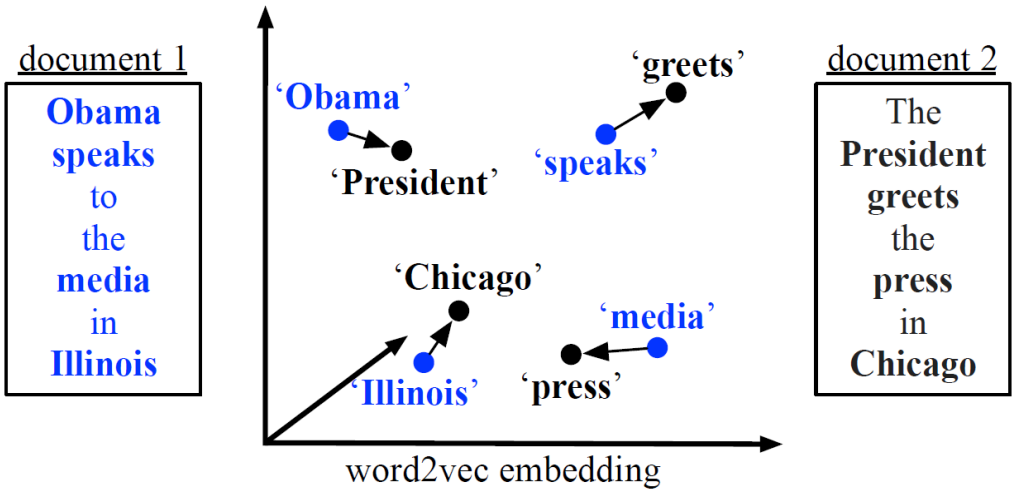

Figure 3.1: Illustration of the vector space in which the words are embedded with the help of Word2Vec. [8]

3.1.1 Models Architecture

Both models work in a self-supervised way, i.e. in training there are no labels given but they can be derived from the training data. Each of them consists of three layers: an input, a projection and an output layer. In both models we consider a window of words $\tilde{w} := (w_{t-n_{window}}, ..., w_{t-1}, ..., w_{t+1}, w_{t+n_{window}})$ around each word w_t, i.e n_{window} words before and after w_t. The length of the window n_{window} can be given as parameter to the training of the model. All words are encoded using One-Hot-Encoding. That means, if we have n_{voc} different words in the dataset, each word is given as n_{voc}-dimensional vector with the value 1 at the index of the word and 0 everywhere else. Because n_{voc} can become very large, it can be useful to restrict the number of words in the vocabulary by only taking the words into account with the highest frequency in the training data.

While in the CBOW-model in training the window of words $\tilde{w}$ is given as input to the model to predict w_t, in the Skip-Gram model it is the other way around. In the projection layer of the CBOW model and in the output layer of the Skip-gram model, shared weight matrices are used for each word. After training the trained weight matrix of the projection layer then encodes the word embedding in both models. The structure of these models is illustrated in figure 3.2.

3.2 KNN with Optimal Transport Distance Measure

We now consider, that we have a vocabulary Voc with a size of n_{voc} words, whereas a word is given in vector representation $w_i \in \mathbb{R}^{n_{dim}}$, e.g. by the Word2Vec embedding.

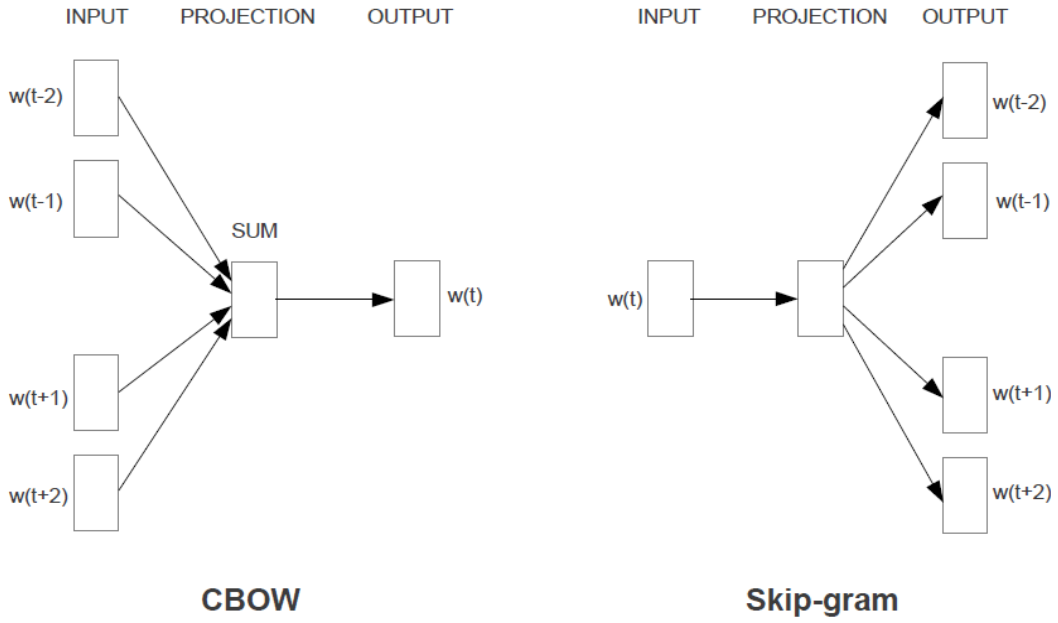

Figure 3.2: Model architectures of CBOW and Skip-Gram.[10]

Because they don't add semantic expressiveness, we assume there are no stop words in the vocabulary. A document is then given in normalized bag-of-words (nBOW) representation. This means that a document d is given as n_{voc}-dimensional vector, with $d_i = \frac{c_i}{\sum_{j=1}^{n_{voc}} c_j}$. The value c_i denotes the absolute frequency of the i-th word in the document d. Because it obviously holds $\sum_{j=1}^{n_{voc}} d_j = 1$, a document d can be seen as a kind of probability distribution. Therefore, it makes sense to consider this as an Optimal Transport problem.

But first we take a general look onto K-nearest-neighbor classification (KNN) and after that we define the Word Mover's Distance and explain how that can be used together with KNN for text classification.

3.2.1 K-Nearest Neighbour Classification (KNN)

KNN is a supervised classification method, i.e. labels are given for the training data. Shortly, one can say that unseen/new data is classified with the help of the k most similar data points in the training data set. Let's look into it in more detail.

To determine how similar two data points (in our case documents) are, we need a score for the similarity. Therefore, in this section we assume that a distance measure for documents is given, $dist : \mathbb{R}^{n_{voc}} \times \mathbb{R}^{n_{voc}} \to \mathbb{R}_0^+$. The lower the value the more similar two documents are. In the training phase, the training data is just stored together with its

labels. We denote the training documents as $\mathcal{X}$ and the labels as y. When we want to predict the label of a new document $\hat{d}$, we calculate the distance of $\hat{d}$ to all documents in the training set $\mathcal{X}$ and further on only consider the k documents $(d_{i_1}, ..., d_{i_k})$ with the lowest distance value $dist(\hat{d}, d)$. The label $\hat{y}$ of $\hat{d}$ is then predicted as the class with the highest frequency among $(y_{i_1}, ..., y_{i_k})$. The value k is given as a parameter. While a low value for k makes the model sensitive to noise, a higher value might consider too many examples from a different class. Therefore, k has to be optimized during training.

As we have seen, besides the parameter k also the distance measure $dist$ is essential to perform KNN classification. That's why in the next section we introduce the Word Mover's Distance, which is a distance measure for documents in nBOW representation based on Optimal Transport.

3.2.2 Word Mover's Distance

The Word Mover's Distance (WMD) was presented by Matt Kusner et al.[8] and defines a distance measure between documents given in a nBOW format. With that, we have all ingredients to perform text classification using KNN.

The idea behind the WMD is that we consider two documents d, d' as probability distributions over the vocabulary space, i.e. $d \hat{=} \alpha$ and $d' \hat{=} \beta$ in the notation of section 2.3. The probability of each word is given by the normalized frequency of this word in the document as described in at the beginning of section 3.2. The space of probability distributions is given by the space of documents while the cost matrix is filled with the pairwise Euclidean distances of the word vectors:

$$c_{i,j} = \sqrt{\sum_{k=1}^{n_{dim}} (v_k - w_k)^2},$$

where v, w are word vectors. We use the Kantorovich formulation of OT, i.e. words can be mapped in total or even in parts to a word of the other document. The distance measure is then defined as the cost to transform the first distribution to the second one in an optimal way (see equation 2.8):

$$dist(d, d') := \mathcal{L}_C(\alpha, \beta).$$

At the top of figure 3.3 this is illustrated. The distance of two Documents D_1 and D_2 to the document D_0 is calculated. The arrows signify the flow of the words from one document to another document, calculated with help of the OT matrix. Values next to the arrows describe the corresponding distance values. The added distance values sum up to an overall distance value between the documents. One can see that corresponding to that formulation, D_1 is closer to D_0 than D_2, as the distance value is lower. In these examples we have a 1-on-1 correspondence between the words, but what happens if

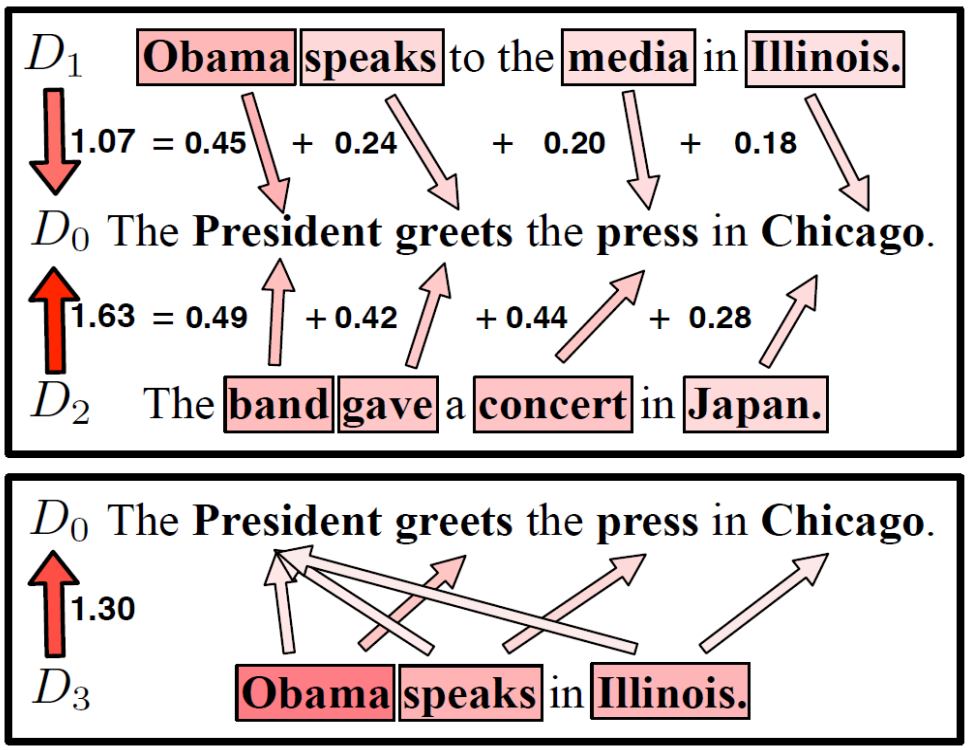

Figure 3.3: Exemplary distance calculation for three documents.[8]

we have a different number of non-stop words in the documents? Like in Kantorovich formulation, it is possible that several words are mapped onto other words in parts. Therefore, in this case several words will be mapped in parts to the same word of the other document. This is illustrated at the bottom of figure 3.3.

As we now have defined the WMD, we are able to do text classification by performing K-nearest-neighbor classification as in section 3.2.1 with the WMD as regarding distance measure.

4 Experiments

We evaluated the method on two datasets and compared the performance to other well known techniques for text classification, i.e SVM and Naive Bayes. First, we will describe the two datasets. After that, it will be discussed how each method was implemented. In the end, we then compare and discuss the results of the experiments.

4.1 Datasets

The case study explores the application of optimal transport in measuring textual similarity and utilizes movie synopses and the Ford data which contains job descriptions.

The first dataset is the **Internet Movie Database (IMDb)** [9] , which contains a large collection of movie synopses. IMDB dataset having 50K movie reviews for natural language processing or Text analytics. This consists a set of 25,000 highly polar movie reviews for training and 25,000 for testing. So, predict the number of positive and negative reviews.

The **ford** dataset [16] consists of job description, breaking down the job description into smaller, yet important parts is crucial during the recruitment process. This approach can benefit CRM-based companies with their downstream services, which are instrumental in determining a candidate's skill score and fit score. Additionally, this strategy can be used to create effective summaries. Given the preference for concise information, it is essential to condense the entire job description into a shorter format that still conveys the necessary context.

4.2 Implementation

4.2.1 KNN with Optimal Transport

The experiments were implemented in Python and mainly followed the strategy described in section 3.2. For preprocessing and the creation of the word2vec-model, the open-source library *Gensim* [14] was used. After that, we used the $scikit-learn$ package [11] to build and use a KNN-classifier.

We first did a preprocessing step to remove stop words (in the Ford dataset "/"-symbols were removed additionally) and to transform every document into a vector of its words. After that, we have build a word2vec-model based on the CBOW structure presented in section 3.1.1. We trained the model on the regarding dataset. The parameters used to build the model can be read from table 4.2.1. The vector length was fixed to 50 because the datasets are not that large. Therefore, a length of 50 is enough to capture most of the complexity of semantics in the dataset, and results in a faster computation.

Datasets	vector length	window length	minimum frequency	vocabulary size
IMDB	50	5	10	19724
Ford	50	5	1	19450

Table 4.1: Parameters used for the build up of the word2vec model and resulting vocabulary size.

Then the dataset was split into a train and a test set with a test ratio of 0.1.

In contrast to chapter 3.2, we do not implement the documents in normalized bag-of-words representation but as 2-dimensional array of its word vectors (empirical representation). Both representations are equivalent. This follows from the fact that from the empirical view, we can obtain the bag-of-words representation just by listing the relative frequency of each word in the empirical representation. On the other way round, we can just line up the word vectors with a non-zero entry in the bag-of words presentation in the regarding frequency to obtain the empirical representation.

We can then use the *ot.bregman.empirical_sinkhorn2* method from the POT package [5] to compute similarites between documents. The advantage of that function is, that we dynamically compute a cost matrix for each pair of documents instead of computing and storing a cost matrix in the size of $n_{voc} \times n_{voc}$, which would be necessary in the bag-of-words representation. This dynamically created matrix is a lot smaller and therefore leads to speed up in computation. Further, by using the empirical view, we avoid numerical issues resulting from very low frequency values in the bag-of-words representation if a document has got many words. Because with the empirical representation we receive arrays of different length, the arrays are padded with zeros, such that they have the same length again. This is necessary to build the KNeighborsClassifier. In the function for computing the similarity, the padded zeros are then removed again before applying the sinkhorn algorithm.

A validation with 100 test examples revealed that $k = 8$ performed best and $\epsilon := \frac{1}{\lambda} = 25$ in the sinkhorn algorithm gave the best trade-off between computation time and accuracy for both datasets.

4.2.2 Data preparation for SVM and Naive Bayes classification

We used scikit-learn in Python package to implement text classification.

Classifier Building in Scikit-learn: Until now, we have learned about the theoretical background of SVM. Now we will learn about its implementation in Python using scikit-learn [11] .

In the model the building part we used ford [16] and IMDb dataset [9] as mentioned in the section 4.1. Here, we build a model to classify the type. The dataset is available in the kaggle.

Data pre-processing: Removed blank data if any, added new column names for unnamed and converted the data type to string using astype() in pandas library

Prepare Train and Test Data sets: The training data set will be used to fit the model and test data for performing predictions.This can be done through the train_test_split from the sklearn library [11]. The Training Data will have 90% of the data and Test data will have the remaining 10% as we have set the parameter test_size=0.1.

Encoding: Label encode the target variable This is done to transform Categorical data of string type into numerical values.

Word Vectorization: It turns a collection of text documents into numerical feature vectors. We used Term Frequency Inverse Document frequency (TFIDF method).

Data is ready for the further classification Algorithms.

Support Vector Machine (SVM)

It uses kernel for handling nonlinear input spaces. It is used in a variety of applications such as face detection, intrusion detection, classification of emails, news articles and web pages, classification of genes, and handwriting recognition. It offers very high accuracy compared to other logistic regression, and decision trees classifiers.

In this classifier a hyperplane is used as a margin to separate data points. That is why it is also known as a discriminate classifier. SVM finds an optimal hyperplane which helps in classifying new data points.

In n-dimensional space, hyper-plane has (n-1) dimensions.

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots \beta_n X_n = \beta_0 + \sum_{i=1}^{n} \beta_i X_i$$

Remaining data processing is done as mentioned in 4.2.2 We used SVC method from sklearn for SVM classification.After the training, we must provide the testing data to see how well our model predicts.

with ford dataset [16] able to predict with an accuracy of 76.67%

with imdb dataset [9] able to predict with an accuracy of 90.67 %

Summary: Support Vector Machine is a Supervised learning algorithm to solve classification and regression problems for linear and nonlinear problems.Above described the implementation of the SVM algorithm using Python and covered its evaluation using a confusion matrix and classification score (available in the code submitted). I used the Spyder for implementation and visualization.

Naïve Bayes

It's a supervised and also one of the crucial algorithm in machine learning which is used in classification problems. It's derived from Bayes Theorem where we can train high-dimensional datasets.

A probabilistic classifier is a algorithm which learns the probability of every object/item, it's classified group and features. That's why Naïve Bayes is a supervised learning. To understand in a simple way, for example there are many birds with similar features and colors, so it is impossible to identify them based on their features and colors. However, we can make a probability prediction using the Naive Bayes Algorithm.

Bayes Theorem: [7]

Bayes Theory or rule contains the hypothesis and the evidence provided. Here in our case let us consider $P(B)$ is the Hypothesis from given evidence $P(A)$. Its mainly works based on two things $P(B)$, which seeks to find out the probability before the evidence is available, and $P(B|A)$, which finds out the probability after the evidence is available. Below is the equation (4.1)

$$P(B|A) = \frac{P(B) * P(A|B)}{P(A)} \tag{4.1}$$

By using the Bayes Theorem, one can find $P(B|A)$ from $P(A|B)$. In essence, the Bayes Theorem can be used to calculate the probability of a hypothesis based on the evidence provided.

Conditional probability is a type of probability that helps to reduce dependence on a single event. This covers the problems which are having more than one features to consider.It is a subset of probability and can be calculated for multiple events. Specifically, if we have events X and Y, the conditional probability of Y is the probability of Y occurring after X has already occurred, denoted as P(Y|X). The formula for calculating this is:

Predictors: Let X be two features [X1, X2],

Target: Y

posterior probability can be calculated using the formula (4.2) -

$$P\left(\frac{Y}{X_1, X_2}\right) = \frac{P(Y) * P(\frac{X_1, X_2}{Y})}{P(X_1, X_2)} \tag{4.2}$$

Conditional Independence for P(X1, X2/Y=1) (4.3),

$$P\left(\frac{Y}{X_1, X_2}\right) = \frac{P(Y) * P(\frac{X_1}{Y}) * P(\frac{X_2}{Y})}{P(X_1, X_2)} \tag{4.3}$$

The formula shown can be used to calculate the posterior probability when employing the Naïve Bayes Classifier.

In our classification, We used the same data but applied a different fitting model using the MultinomialNB method. We observed accuracy for both data sets.

Using ford dataset [16] able to predict with an accuracy of 68.11 %

Using IMDb dataset [9] able to predict with an accuracy of 85.64 %

4.3 Discussion of Results

We calculated the accuracies of each algorithm and enlisted them in the following table.

Datasets	KNN with WMD	SVM	Naive Bayes
IMDB	0.7032	0.9067	0.8564
Ford	0.6697	0.7667	0.6811

Table 4.2: Accuracies for the different models.

As one can see, the accuracy of the KNN with the OT distance measure is a lot lower than the other two algorithms. Especially for the IMDB dataset, which was a binary classification problem, the performance is a lot worse. On the other hand, for the Ford Classification dataset the performance is worse as well but not that much. Further, the experiments took a lot more time with KNN (about 4 days) than with SVM (about 15 minutes) and Naive Bayes (about 1 minute).

That does not mean OT is not usable for text classification itself. But there are a lot of parameters, which we had to adjust at cost of the performance, such that it is feasible to use the algorithm on our data. So the results should be understood more as a base performance on which we can build on. One example is the parameter ϵ in the sinkhorn algorithm which is defined as $\frac{1}{\lambda}$ (λ introduced in chapter 2.4). For an ϵ equal to 10 , the accuracy on the validation set was about 0.75, but it took twice the time to calculate

the label of each document. Further, it could make sense to use a pretrained word2vec model with more dimensions instead of training our own model, or even to use another word embedding algorithm. For example, the word2vec model trained on the IMDB dataset lists "bad" as the fifth most similar word to "good" even before "excellent". This probably results from the fact that "good" and "bad" are often used in the same context and the dataset is not large and divers enough, such that the learned model can distinguish both words sufficiently.

In chapter 5, we take a brief look on how one could improve the performance.

5 Outlook

In chapter 4.3 we have seen that KNN for text classification implemented with the Word Mover's Distance has some major drawbacks if it is implemented naively. This results mainly from the number of similarity calculations ($n_{train} * n_{test}$), the similarity computation itself and the storage of the cost matrix. Therefore some scientists have already developed and are working on alternatives which lead to a faster and less storage intensive calculation.

M. Kusner et al. for example introduced the Word Centroid Distance ($||Xd - Xd'||$) which lower bounds the WMD and therefore can be used to determine a candidate subset on which we then perform the similarity computations [8]. They also introduced the relaxed WMD where one of the constraints in the definition of the set $U(a, b)$ in chapter 2.3 is removed. This results in a simplification in the sense, that we only need to do k-nearest-neighbor search in the word embedded vector space [8].

Another option of optimizing the strategy presented in this work could be to use other word embeddings than the word2vec framework [10]. One very popular word embedding model is the so called BERT-model presented in [4]. This model uses a deep neural network to transform words into 768-dimensional vectors. The semantic expressiveness of these vectors is very good but could come with a higher cost of similarity computation.

In our work we used the euclidean distance to calculate semantic similarities of the word vectors. This could also be optimized by trying different similarity measures. One popular similarity measure for word vectors is the so-called cosine similarity ($sim(v, w) = \frac{v \cdot w}{||v||||w||}$, where $\cdot$ considers means the scalar product) which considers the angle between vectors.

Bibliography

[1] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in neural information processing systems*, pages 2292–2300, 2013.

[2] Prof.Dr.GeroFriesecke Daniela Vögler. Generalization of kantorovich duality to multi-marginal optimal transportation and applications in economics. *Center for Mathematical Sciences,Technische Universität München Chair of Analysis*, 2019.

[3] Julie Delon. Midway image equalization. *Journal of Mathematical Imaging and Vision, 21(2):119 134,*, 2004.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[5] Rémi Flamary, Nicolas Courty, Alexandre Gramfort, Mokhtar Z. Alaya, Aurélie Boisbunon, Stanislas Chambon, Laetitia Chapel, Adrien Corenflos, Kilian Fatras, Nemo Fournier, Léo Gautheron, Nathalie T.H. Gayraud, Hicham Janati, Alain Rakotomamonjy, Ievgen Redko, Antoine Rolet, Antony Schutz, Vivien Seguy, Danica J. Sutherland, Romain Tavenard, Alexander Tong, and Titouan Vayer. Pot: Python optimal transport. *Journal of Machine Learning Research*, 22(78):1–8, 2021.

[6] Zhengping JIANG. Sinkhorn's theorem, sinkhorn alorithm and applications. 2020.

[7] John D. Kelleher and Brian Tierney. *Data Science An Introduction*. CRC Press, Boca Raton, 2018.

[8] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. *Proceedings of the 32nd International Conference on Machine Learning, volume 37 of Proceedings of Machine Learning Research*, 2015.

[9] N. Lakshmipathi. Sentiment analysis of imdb movie reviews. https://www.kaggle.com/code/lakshmi25npathi/sentiment-analysis-of-imdb-movie-reviews, 2020.

[10] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *In Proceedings of Workshop at ICLR*, 2013.

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

[12] Gabriel Peyre and Marco Cuturi. Computational optimal transport: With applications to data science. *Mathematics*, 2018.

[13] Gabriel Peyré and Marco Cuturi. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

[14] Radim Rehurek and Petr Sojka. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 3(2), 2011.

[15] Alexander Schmitz and Benedikt Wirth. A tutorial on sinkhorn's algorithm for matrix scaling and its applications in machine learning. *arXiv preprint arXiv:1905.10730*, 2019.

[16] Vikramjeet Singh and Gavesh Jain. Ford sentence classification dataset. https://www.kaggle.com/datasets/gaveshjain/ford-sentence-classifiaction-dataset, 2022.

[17] Richard Sinkhorn and Paul Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

[18] Justin Solomon, Fernando De Goes, Gabriel Peyré, Marco Cuturi, Adrian Butscher, Andy Nguyen, Tai Du, and Leonidas Guibas. Convolutional wasserstein distances: Efficient optimal transportation on geometric domains. *ACM Transactions on Graphics (TOG)*, 34(4):66, 2015.