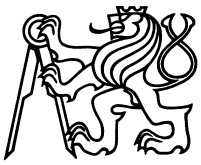
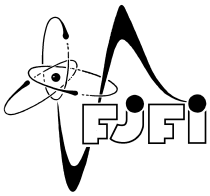


Czech Technical University in Prague
Faculty of Nuclear Sciences and Physical Engineering



Optimalization of distances for multi-instance clustering

Optimalizace vzdálenosti pro multi-instanční shlukovací problémy

Master's thesis

Author: **Marek Dědič**

Supervisor: **Ing. Tomáš Pevný, Ph.D.**

Academic year: 2019/2020

- Thesis assignment -

- Thesis assignment -

Acknowledgment:
I would like to thank Dr. Tomáš Pevný for his expert guidance, insightful observations, innovative ideas and approaches and his attitude of always trying to push the envelope.

Author's declaration:
I declare that this research project is entirely my own work and I have listed all the used sources in the bibliography.

Prague, January 6th 2020 Marek Dědič

Název práce:

Optimalizace vzdálenosti pro multi-instanční shlukovací problémy

Autor: Bc. Marek Dědič

Obor: Matematická informatika

Druh práce: Diplomová práce

Vedoucí práce: Ing. Tomáš Pevný, Ph.D., Fakulta elektrotechnická, České vysoké učení technické v Praze, Avast Software s.r.o.

Abstrakt: Klastrová analýza je jedním z typických problémů náležících do skupiny algoritmů učení bez učitele. Jednou z hlavních voleb při návrhu jakéhokoliv klastrovacího algoritmu je výběr správné vzdálenostní funkce. V této práci je představen přístup k výběru vzdálenostní fuknce pomocí strojového učení. Tento přístup staví na multi-instančním učení, přístupu nabízejícím vysoký výpočetní výkon a velkou expresivní sílu pro popis dat s vnitřní strukturou pomocí hierarchických modelů. Tři metody stavěné na multi-instančním učení jsou představeny spolu s předchozími pracemi, na kterých staví. Jedna z metod spadá do skupiny algoritmů učení bez učitele zatímco dvě metody spadají do skupiny algoritmů učení s učitelem. Metody jsou teoreticky popsány a experimentálně vyhodnoceny na veřejně dostupných datasetech pro multi-instanční učení a na korporátním datasetu dat z oblasti síťové bezpečnosti. Výsledky jsou následně zhodnoceny a metody porovnány.

Klíčová slova: strojové učení, multi-instanční učení, shluková analýza, contractive predictive coding, triplet loss, manet loss, neuronové sítě, vícevrstvý perceptron

Title:

Optimalization of distances for multi-instance clustering

Author: Bc. Marek Dědič

Abstract: Clustering is a prime example of a problem typically associated with unsupervised learning. One of the key design choices when using any clustering algorithm is to choose the right distance metric. In this work, an approach for using machine learning to learn the metric is introduced. The approach build on multi-instance learning, an approach offering high computational performance and a strong expressive power for describing data with an inherent structure using hierarchical models. Three methods building on multi-instance learning are presented together with the prior art they build upon. One of the methods is unsupervised while two are supervised. The methods are theoretically discussed and experimentally evaluated on publicly available datasets for multi-instance learning, as well as a corporate dataset of network security data. The results are then discussed and the methods compared.

Keywords: machine learning, multi-instance learning, clustering, contrastive predictive coding, triplet loss, magnet loss, neural networks, multi-layer perceptron

Contents

Introduction

Clustering is a prime example of a problem typically associated with unsupervised learning. One of the key design choices when using any clustering algorithm is to choose the right distance metric. Traditionally, one of the standard distance metrics such as the Euclidean, Manhattan or Mahalanobis distance would be used, using the designer's knowledge of the domain. For complex tasks, however, it is difficult to design a metric that is well-suited to the problem at hand. One of the possible solutions to this problem is using distance metric learning, that is using machine learning to construct a distance metric under which the desired clustering can be done easily. This automates the process of creating a problem-specific metric.

Multi-instance learning is a well-known approach to supervised machine learning on structured data. Multi-instance learning alleviates the need for having a label for each instance in the training set, instead relying on learning how to represent whole bags of instances. The feature extractor and the classifier are then trained in an end-to-end fashion. This has the advantage of being able to exploit the inherent structure of the data when designing the machine learning model.

In this work, clustering and multi-instance learning meet. While there have been previous attempts at multi-instance clustering, they have utilized the so-called bag-space paradigm, which is prohibitively computationally expensive for large datasets. Instead, an approach based on the so-called embedded-space paradigm was chosen for this work, as this approach offers a mix of high performance and the ability to construct complex hierarchical models tailored to the specific domain or application. For this approach, there exists no known prior art. In order to fill this gap and to offer a comparison, three different methods will be presented, experimentally evaluated and compared. The most promising of these three methods falls into the category of unsupervised learning, for which there is no need for labelled training data, which would make the method applicable to a broad set of problems.

To utilize and evaluate the proposed method, an industry application was selected – network security. The problem of clustering and classifying network traffic is well known. Specifically, the problem of clustering second-level domains based on the activity clients are exhibiting towards them is tackled. Even an experienced network analyst can only investigate a limited amount of domains and having a clustering on the space of domains would allow them to only investigate a few of the domains in a cluster and then being able to label any current or future domain falling in the cluster as either legitimate or malicious. Moreover, with the recent spread of encrypted traffic over the HTTPS protocol, and the gradual roll-out of encryption to core protocols such as HTTP/3 (formerly known as QUIC) and Encrypted SNI, the amount of information available to the analyst is shrinking. If this work is successful, it could be in the future extended to cover the problem of classifying partially or fully encrypted network traffic.

The work is structured as follows: Chapter 1 introduces multi-instance learning, presents two formalisms used to mathematically describe it, presents and compares the three main paradigms used for multi-instance learning and provides a survey of the *state of the art* with respect to multi-instance learning. Chapter 2 presents clustering as a machine learning task, presents its application to multi-instance learning and provides metrics for evaluating the performance of the proposed methods. Chapter 3 is the

core of this work. Three methods for multi-instance clustering are presented, each building on an original method in other domains of machine learning. The three methods are explained and theoretically compared. Chapter 4 compares the methods experimentally on publicly available dataset. The author of this work considers it important to provide a replicable experiment which evaluates the methods in a way that is not constrained by the unavailability of proprietary datasets. Finally, chapter 5 evaluates the three methods on a corporate dataset of network traffic.

Chapter 1

An overview of Multi-instance learning

Multi-instance learning is an approach to machine learning first described by Dietterich et al., 1997. In its original form, multi-instance learning (often referred to just as **MIL**) was used for **supervised learning**, that is to solve problems for which there is already a dataset of known examples and their corresponding labels at the ready. An example of that is in the work Amores, 2013. The other class of problems solved is **unsupervised learning**, where no labels are available in the training phase. Among these is for example Ying Chen et al., 2012.

The multi-instance learning paradigm is a type of representation learning on data with some internal structure. While classical machine learning algorithms typically operate on samples represented by a vector of numbers each, MIL relaxes this requirement by representing a sample as a **bag** of an arbitrary number of objects. This bag is then treated as one sample – e.g. in classification, there is a label for the whole bag.

In this chapter, two formalisms for describing multi-instance learning in the context of classification are presented. The algebraic formalism largely consisting of a translation of Dědič, 2017, where it was first introduced. The probabilistic formalism build upon the ideas from Muandet et al., 2012. Finally, a review of methods for MIL is presented.

1.1 Multisets

In order to properly describe multi-instance learning using the algebraic formalism, a few uncommon definitions are needed.

A multiset is a set allowing repeated elements. Alternatively, it can be seen as an unordered tuple. Knuth, 1968 formally defines the concept as follows:

Definition 1.1. Let $\mathbb{A}$ be a set and $m : \mathbb{A} \to \mathbb{N} \setminus \{0\}$. The tuple $(\mathbb{A}, m)$ is called a **multiset** over the set $\mathbb{A}$. For an $a \in \mathbb{A}$, the number $m(a)$ is called the **multiplicity** (i.e. the number of occurrences) of a. If $a \in \mathbb{A}$, then a is called an element of $(\mathbb{A}, m)$ and denoted as

$$a \in (\mathbb{A}, m)$$

Obviously, the concept of a multiset is a generalization of the concept of a set and as such, it is usually written as an enumeration its elements, which are repeated as many times as is their multiplicity.

Example 1.2. The multiset $(\{a, b, c\}, \{(a, 3), (b, 1), (c, 5)\})$ may be written as $\{a, a, a, b, c, c, c, c, c\}$.

Remark 1.3. A set can be seen as a special case of a multiset where each element has a multiplicity of 1.

Definition 1.4. Let $\mathbb{A}$ be a set. A multiset $(\mathbb{B}, m)$ is called a **submultiset** of $\mathbb{A}$ if $\mathbb{B} \subset \mathbb{A}$. This is denoted as

$$(\mathbb{B}, m) \subset \mathbb{A}$$

The previous definition allows for a generalization of the concept of a power set, which is usually denoted as $\mathcal{P}(\mathbb{A})$ or $2^{\mathbb{A}}$.

Definition 1.5. Let $\mathbb{A}$ be a set. A **power multiset** of $\mathbb{A}$ is the set

$$\mathcal{P}^M(\mathbb{A}) = \{(\mathbb{B}, m) | (\mathbb{B}, m) \text{ is a multiset} \wedge (\mathbb{B}, m) \subset \mathbb{A}\}$$

Remark 1.6. A power set of $\mathbb{A}$ is sometimes expressed as the set of all function from $\mathbb{A}$ to $\{0, 1\}$, written as $\{0, 1\}^{\mathbb{A}}$. $\{0, 1\}$ coincides with the von Neumann ordinal 2, therefore the power set is often denoted $2^{\mathbb{A}}$. Similarly, the power multiset may be expressed as the set of all functions from $\mathbb{A}$ to $\mathbb{N}$ (where $0 \in \mathbb{N}$). It is therefore possible to use an analogous notation $\mathcal{P}^M(\mathbb{A}) = \mathbb{N}^{\mathbb{A}}$. This analogy extends to the size of the sets in question where

$$\left|2^{\mathbb{A}}\right| = 2^{|\mathbb{A}|} \qquad \text{and} \qquad \left|\mathbb{N}^{\mathbb{A}}\right| = +\infty$$

for all $\mathbb{A} \neq \emptyset$.

Definition 1.7. Let $\mathbb{B} = (\mathbb{A}, m)$ be a multiset where $\mathbb{A}$ is finite. The cardinality of the multiset $\mathbb{B}$ is

$$|\mathbb{B}| = \sum_{a \in \mathbb{A}} m(a)$$

The summation of two multisets can be easily defined as follows.

Definition 1.8. Let $\mathbb{B}_1 = (\mathbb{A}_1, m_1)$ and $\mathbb{B}_2 = (\mathbb{A}_2, m_2)$ be two multisets. Then the sum of $\mathbb{B}_1$ and $\mathbb{B}_2$ is

$$\mathbb{B}_1 \oplus \mathbb{B}_2 = (\mathbb{A}_1 \cup \mathbb{A}_2, m)$$

where

$$m(a) = \begin{cases} m_1(a) & \text{for} \quad a \in \mathbb{A}_1 \setminus \mathbb{A}_2 \\ m_2(a) & \text{for} \quad a \in \mathbb{A}_2 \setminus \mathbb{A}_1 \\ m_1(a) + m_2(a) & \text{for} \quad a \in \mathbb{A}_1 \cap \mathbb{A}_2 \end{cases}$$

1.2 An algebraic formalism for multi-instance learning

Multi-instance learning (see Dietterich et al., 1997) in its original form is an approach to representing and solving problems where there exist samples from a space $\mathcal{X}$ and their corresponding labels from a space $\mathcal{Y}$ (the labels are also often called classes) and the goal is to find a mapping of samples to labels which correctly predicts the targets on the data. Unlike classical supervised learning, there is no known solution consisting of pairs of samples and labels. Instead, the samples are grouped together into so-called bags, which are defined as follows:

Definition 1.9. Let $\mathcal{X}$ be a set of samples. Then the **bag space** is a multiset $\mathcal{B}$ with the properties

1. $\mathcal{B} \subset \mathcal{P}^M(\mathcal{X})$
2. $(\forall x \in \mathcal{X})(\exists B \in \mathcal{B})(x \in B)$

Elements of the bag space are called **bags**.

Multi-instance learning classification uses a dataset consisting of pairs of bags and labels to learn a mapping between them. Dietterich et al., 1997 provides the following example of a multi-instance problem:

Example 1.10. Suppose there is a keyed lock on the door to the supply room in an office. Each staff member has a key chain containing several keys. One key on each key chain can open the supply room door. For some staff members, their supply room key opens only the supply room door; while for other staff members, their supply room key may open one or more other doors (e.g., their office door, the mail room door, the conference room door).

Suppose you are a lock smith and you are attempting to infer the most general required shape that a key must have in order to open the supply room door. If you knew this required shape, you could predict, by examining any key, whether that key could unlock the door. What makes your lock smith job difficult is that the staff members are uncooperative. Instead of showing you which key on their key chains opens the supply room door, they just hand you their entire key chain and ask you to figure it out for yourself! Furthermore, you are not given access to the supply room door, so you can't try out the individual keys. Instead, you must examine the shapes of all the keys on the key rings and infer the answer.

In this example, the keys (or, more precisely their descriptions by feature vectors) are the samples from $\mathcal{X}$. The key chains are bags. Even though it makes sense to ask for each key whether it unlocks the supply room, this information is not provided. Instead, only a key-chain-level information is available. On the other hand, it may be stated that a key chain opens the supply room door if and only if it contains a key which opens the door. This naturally leads to the following definition:

Definition 1.11. Let $\mathcal{X}$ be an sample space and $\mathcal{Y}$ a label space such that

$$(\forall x \in \mathcal{X})\,(\exists_1 y_x \in \mathcal{Y})$$

Let $\mathcal{B}$ be a bag space. Let maximum be well-defined in $\mathcal{Y}$. Then the label of the bag $B \in \mathcal{B}$ is

$$y_B = \max_{x \in B} (y_x) \in \mathcal{Y}$$

As will later be shown in sections 1.4.2 and 1.4.3, this isn't the only interpretation of multi-instance learning. There are problems where the instance-level labels are not only no known, but even not well-defined as the bag isn't seen as a mere collection of objects but as a distinct object of its own.

1.3 A probabilistic formalism for multi-instance learning

An alternative formalism for multi-instance learning was first introduced in Pevný; Somol, 2017 and builds on the previous work Muandet et al., 2012.

Let for the space $\mathcal{X}$ exist a measurable space $(\mathcal{X}, \mathcal{A})$, where $\mathcal{A}$ is a σ-algebra of $\mathcal{X}$. Let $\mathcal{P}^{\mathcal{X}}$ denote the set of all probability measures on $(\mathcal{X}, \mathcal{A})$. A bag B is viewed as a set of realizations of a particular probability distribution $P \in \mathcal{P}^{\mathcal{X}}$, that is

$$B = \{x_i | x_i \sim P, i \in \{1, \ldots, m\}, m \in \mathbb{N}\}$$

A core assumption of this formalism is that $P = P(P_B, y_B)$ where P_B is the probability distribution of instances in the bag B and y_B is the label assigned to B. That is, the probability distribution of the bag is dependent on its assigned label.

1.4 Approaches to solving multi-instance problems

In this section, there are described the three major approaches to solving multi-instance classification problems. The section is sourced primarily from Pevný; Somol, 2017 and Pevný; Somol, 2016.

1.4.1 Instance-space paradigm

Instance-space paradigm is the original approach proposed by Dietterich et al., 1997. The existence of labels on the level of instances is presumed, even though such labels aren't known even in the training phase. The goal of this approach is to model the unknown instance-level labelling as $f : \mathcal{X} \to \mathcal{Y}$ and use it to compute the bag-level label analogically to definition 1.11, that is

$$y_B = \max_{x \in B} \left(f\left(x\right) \right)$$

There are many applications of instance-space paradigm, the most important of which are described next. (Also see Andrews et al., 2002 C. Zhang et al., 2006)

BP-MIP, proposed by Zhou; M.-L. Zhang, 2002, is an application of multi-instance learning to binary classification, i.e. $\mathcal{Y} = \{-1, +1\}$. A feedforward neural network is used, its output for the j-th instance of the i-th bag denoted o_{ij}. An instance-level error function is defined as

$$E_{ij} = \begin{cases} 0 & \text{for} \quad y_{B_i} = -1 \wedge o_{ij} < 0.5 \\ 0 & \text{for} \quad y_{B_i} = +1 \wedge o_{ij} \geq 0.5 \\ \frac{1}{2}\left(o_{ij} - 0.5\right)^2 & \text{otherwise} \end{cases}$$

This instance-level error function is then used to compute the bag-level error function as [1]

$$E_i = \begin{cases} \max_{1 \leq j \leq |B_i|} E_{ij} & \text{for} \quad y_{B_i} = -1 \\ \min_{1 \leq j \leq |B_i|} E_{ij} & \text{for} \quad y_{B_i} = +1 \end{cases}$$

And this is in turn used to compute the global loss function

$$E = \sum_{i=1}^{|\mathcal{B}|} E_i$$

With such a well-defined global loss function, the back-propagation algorithm can be used with a slight modification described in the original article.

EM-DD, proposed by Q. Zhang et al., 2002, combines the **expectation-maximization** algorithm (see Dempster et al., 1977) with the **diverse density** algorithm (see Maron et al., 1998). The EM-DD algorithm starts with a target hypothesis $h \in \mathcal{Y}$, which is then refined with the EM algorithm. In the E-step, the instance thought to be most responsible for the label of each bag is picked. In the M-step, the gradient ascent algorithm is used to find a new hypothesis h' which maximizes DD (h). DD (h) corresponds to the likelihood that h is the actual target. These steps are then repeated, refining the hypothesis.

[1] The original article erroneously states the upper bound as $|b_j|$

1.4.2 Bag-space paradigm

The bag-space paradigm is a formalism in which the notion of instance-level labels is abandoned (that is, the labels are presumed to not only be unknown, but even undefined) and only bag-level labels are assumed to exist. The definition 1.11 is no longer used, so some other way of working with bags must be devised. In order to do this, a bag distance function or a bag kernel function is defined, having the form

$$k : \mathcal{B} \times \mathcal{B} \to \mathbb{R}_0^+$$

Because the notion of an instance-level label is no longer used, the sought classification function is of the form $f : \mathcal{B} \to \mathcal{Y}$.

Citation-kNN, proposed by J. Wang et al., 2000, defines a modified Hausdorff distance for bags $B_1, B_2 \in \mathcal{B}$ as

$$H(B_1, B_2) = \min_{x \in B_1} \min_{y \in B_2} \|x - y\|$$

The k-nearest neighbour algorithm (see Dasarathy, 1991) is modified for this application by adding a system of **citations** and **references**. The r-nearest references are the nearest neighbours from the vanilla kNN algorithm and the c-nearest citers are defined as

$$\text{Citers}(x, c) = \{x_i | \text{Rank}(x_i, x) \leq c \wedge x_i \in B\}$$

For each bag, the r-nearest references and the c-nearest citers are found using the modified Hausdorff distance. If among those there are more positive bags than there are negative ones, the bag is labelled as positive. Otherwise, the bag is labelled as negative.

For more applications using the bag-space paradigm, see J. Wang et al., 2000, Kwok et al., 2007, Gärtner et al., 2002, Haussler, 1999, Zhou; Sun, et al., 2008 and Muandet et al., 2012.

1.4.3 Embedded-space paradigm

In the embedded space paradigm, labels are only defined on the level of bags, same as in the bag-space paradigm. In order for these bag labels to be learned, an embedding function of the form $\phi : \mathcal{B} \to \bar{\mathcal{X}}$ must be defined, where $\bar{\mathcal{X}}$ is a latent space, which may or may not be identical to $\mathcal{X}$. Using this function, each bag can be represented by an object $\phi(B) \in \bar{\mathcal{X}}$, which makes it possible to use any off-the-shelf supervised learning algorithm acting on $\bar{\mathcal{X}}$. Among the simplest embedding functions are e.g. element-wise minimum, maximum and mean. A more complicated embedding function may for example apply a neural network to each instance of the bag and subsequently pool the instances using one of the aforementioned functions.

MILES, proposed by Yixin Chen; Bi, et al., 2006, uses an instance dictionary $\mathcal{D}$ to define an embedding function representing the degree of similarity between the bag and the dictionary, that is

$$\phi : \mathcal{B} \to \mathbb{R}^{|\mathcal{D}|}$$

which is defined element-wise as

$$\phi_i(B) = \sum_{x \in B} k(x, d_i) \quad \text{where} \quad d_i \in \mathcal{D}$$

where

$$k(x, d) = \begin{cases} \exp\left(-\frac{1}{\sigma^2} \|x - d\|^2\right) & \text{if } d \text{ is the nearest neighbour of } x \text{ in } \mathcal{D} \\ 0 & \text{otherwise} \end{cases}$$

The dictionary is populated by a 1-class support vector machine (see Zhu et al., 2004), which, however, has a high computational complexity.

Pevný; Dědič, 2020 and Dědič, 2017 present an approach to learning to classify HTTP traffic by utilizing sets of URLs. Neural networks are used to transform both instance-level and bag-level representations. The model is as follows. A deep neural network $f_I : \mathcal{X}_1 \to \mathcal{X}_2$ is used to transform each instance $x \in B$. An aggregation function is then used, being of the form

$$g : \mathcal{P}^M (\mathcal{X}_2) \to \bar{\mathcal{X}}_1$$

Finally, a second deep neural network $f_B : \bar{\mathcal{X}}_1 \to \bar{\mathcal{X}}_2$ is used to transform the representation of each bag. Combining these functions gives the embedding function

$$\phi (B) = f_B (g (\{f_I (\boldsymbol{x}) | \boldsymbol{x} \in B\}))$$

The functions f_I and f_B are realized by a deep neural network using ReLU as its activation function. The aggregation g is realized as an element-wise mean or maximum of all the vectors.

The structure of the data is highly exploited using the embedded-space paradigm. The approach is innovative in that it uses multiple layers of MIL nested in each other – that is the instances of a bag do not necessarily need to be feature vectors, but can be bag in themselves. The HTTP traffic of a particular client is therefore represented as a bag of all second-level domains the client has exchanged datagrams with. Each second-level domain is then represented as a bag of individual URLs which the client has connected to. Individual URLs are then split into 3 part, domain, path and query, and each part is represented as a bag of tokens, which can then be broken down eve further. The model is shown in figure 1.1. In the end, the model consists of 5 nested MIL problems.

An added benefit of MIL is also in its relatively easy explainability and interpretability. Pevný; Dědič, 2020 present a way to extract indicators of compromise and explain the decision using the learned MIL model.

For more applications using the embedded-space paradigm, see Cheplygina et al., 2015, Yixin Chen; J. Z. Wang, 2004 Foulds, 2008 and M.-L. Zhang et al., 2009.

1.4.4 A comparison of the approaches

For this work, the embedded-space paradigm was chosen. The main reason for this is the development multi-instance learning has gone through since its inception. In the pioneering work of Dietterich et al., 1997, it is assumed that a bag is just a convenient collection of instances which could be used directly, but have no available per-instance labels. This notion of MIL has, however, since evolved beyond these simple cases and the instance-space paradigm is no longer suitable. The previous work Pevný; Dědič, 2020 presents a clear example where multi-instance learning is used as a general toolkit for learning representations from structured data and even defines how to use nested MIL problems to represent complex hierarchical structures. To this end, the embedded space paradigm is used in this work as it is the only one for which such a structured, multi-layered approach has been proposed.

The bag-space paradigm has clear prior art in the domain of clustering. For large datasets, however, this approach is not feasible due to its high computational complexity. This presents another argument in favor of using the embedded-space paradigm, as the target application of this work is in a domain with extremely large datasets.

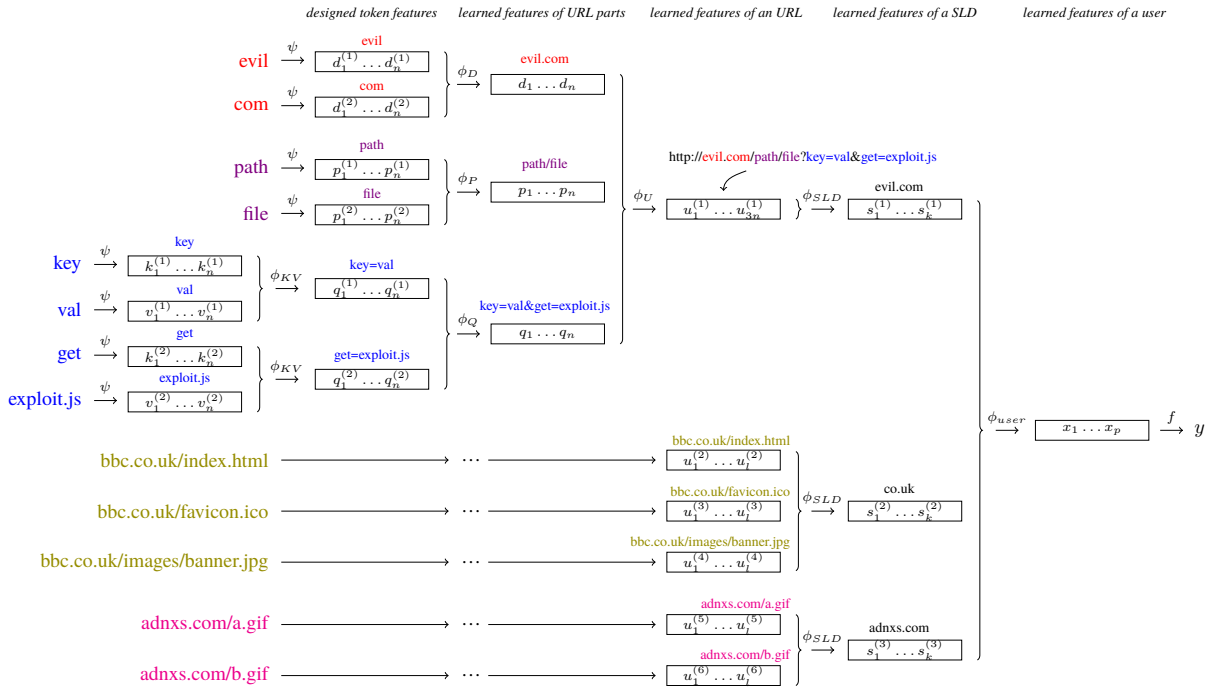


Figure 1.1: Hierarchical model of the network traffic of a single computer. The computer is modelled by a set of remote servers and each remote server is modelled by a set of messages (connections) from the given computer to it. Figure from Pevný; Dědič, 2020.

Chapter 2

Clustering

Clustering is a prime example of a problem typically associated with unsupervised learning.[1] Xu et al., 2015 even call clustering the most important question of unsupervised learning. The task is one of grouping objects into **clusters** such that the objects in any particular cluster are in some sense more similar to each other that to the objects from other clusters. Alternatively, the problem can be solved by finding a similarity measure on the object space such that the objects deemed to be similar actually are under the measure. Oftentimes, such a similarity measure is based on some distance function such as the Euclidean, Manhattan, Cosine, Minkowski or Mahalanobis (see Mahalanobis, 1936) distance.

2.1 Definition of clustering

So far, the definition of terms such as ”cluster” or ”similar” has been left intuitive and vague. The reason for that is that there is no universal definition of what a cluster is. Estivill-Castro, 2002 argues that there even should not be a universal definition as the clusters are, in a large part, on the eye of the beholder. Notwithstanding that, there is at least an informal definition provided by Jain et al., 1988:

1. A cluster is a set of entities which are *alike* and entities from different clusters are not alike.

2. A cluster is an aggregation of points in the test space such that the *distance* between any two points in the cluster is less than the distance between any point in the cluster and any point not in it.

3. Cluster may be described as connected regions of a multi-dimensional space containing a relatively *high density* of points, separated from other such regions by a region containing a relatively low density of points.

Even with that definition, it is sometimes not clear what constitutes a cluster without some additional context. Jain et al., 1988 presents figure 2.1 as an example.

Xu et al., 2015 offers a comprehensive survey of different distance functions, similarity functions, performance metrics and clustering algorithms. The choice of the right clustering algorithm can play a crucial role in the efficiency of a clustering model. In this work, the proposed embedding is general enough that many clustering algorithms can be used. A comparison of clustering algorithms is, however, outside the scope of this work, so the k-nearest neighbour algorithm will be used for all the experiments, as it is the most explored one.

[1] In this work, both unsupervised and supervised approaches to clustering are explored.

2.2 Application to MIL

In the previous section it has been described what is a clustering. The link between clustering and multi-instance learning is, however, yet to be shown. In this section, the application of clustering to MIL is described.

The problem at hand is not one of clustering ordinary number vectors in a linear space. Instead, a clustering of objects represented by bags is explored, as that is the problem solved for datasets introduced later in chapters 4 and 5. While J. Wang et al., 2000 present a clustering of bags using a modified Hausdorff distance on bags and Kohout et al., 2018 present a clustering of network traffic fingerprints using maximum mean discrepancy, this approach was not chosen in this work. Among the main reasons for this choice is the prohibitively high computational complexity of bag-space paradigm approaches on large datasets and the opportunity to utilize the representations previously introduced for the particular datasets in Dědič, 2017 and Pevný; Dědič, 2020.

An approach based on the embedded-space paradigm (see section 1.4.3) for MIL was chosen. In order to utilize the structure of the data, a MIL model is used to represent each bag in the latent space $\bar{\mathcal{X}}$. This presents the issue of how to train the embedding function ϕ – MIL as presented in chapter 1 is a supervised algorithm, whereas clustering is a typically unsupervised problem.

The embedding function ϕ from section 1.4.3 is actually of the form

$$\phi = \phi\left(B, \boldsymbol{\theta}\right) \qquad \text{where} \quad B \in \mathcal{B}$$

where $\boldsymbol{\theta}$ are the parameters of the embedding, typically learned during the training phase. In the context of clustering, these parameters still need to be learned over the training phase somehow. This in and of itself presents another challenge though – off-the-shelf algorithms typically work in some kind of constant Hilbert space, whereas in this example, the latent space needs to change over the learning period. As is the case for MIL itself, end-to-end learning was chosen as an approach to solve both of these problems. A clustering-loss function L_C is chosen such that

$$L_C : \mathcal{P}^M\left(\bar{\mathcal{X}}\right) \to \mathbb{R}$$

Given such clustering-loss function, an actual loss for the embedding model ϕ and its parameters $\boldsymbol{\theta}$ can be computed as

$$L\left(\phi, \boldsymbol{\theta}\right) = L_C\left(\{\phi\left(B, \boldsymbol{\theta}\right) | B \in \mathcal{B}\}\right)$$

If the cluster-loss L_C is chosen correctly, minimizing L over the learning period will yield a latent space $\bar{\mathcal{X}}$ in which the bags are already naturally clustered according to the design of the cluster-loss function. Applying any off-the-shelf clustering algorithm on $\bar{\mathcal{X}}$ will then give good results. How to choose the cluster-loss function L_C is the focus of chapter 3.

2.3 Clustering evaluation metrics

Xu et al., 2015 present different clustering performance metrics. In this work, several metrics are used. The primary metrics are based on the Silhouette coefficient, the secondary ones on the kNN algorithm (see Dasarathy, 1991). The metrics are somewhat different to the ones traditionally used in evaluating clustering methods as all the datasets used in this work are originally classification datasets and therefore have classes available. Each class can then be viewed as a cluster target and the learned clustering evaluated against these targets.

The first two metrics measure the **homogeneity** (that is the property of instances of one cluster being close to one another) and **separation** (that is the property of instances of different clusters being far apart)

of clusters (see Everitt et al., 2001). To measure the homogeneity of a cluster, the average distance between items in a cluster is measured and averaged over all clusters, giving the following metric for items x_j in clusters C_i (defined by the classes in the data):

$$\text{homo}\left(C_1, \ldots, C_n\right) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{|C_i|} \sum_{\substack{x_j, x_k \in C_i \\ x_j \neq x_k}} \|x_j - x_k\|$$

To make this metric computationally less complex, it can be simplified by extracting the centre μ of the cluster:

$$\mu\left(C_i\right) = \frac{1}{|C_i|} \sum_{x_j \in C_i} x_j$$

$$\text{homo}\left(C_1, \ldots, C_n\right) = \frac{1}{n} \sum_{i=1}^{n} \sum_{x_j \in C_i} \|x_j - \mu\left(C_i\right)\|$$

To measure the separation of clusters, the average distance between cluster centres was taken:

$$\text{sep}\left(C_1, \ldots, C_n\right) = \frac{2}{n\left(n-1\right)} \sum_{\substack{i,j \in \hat{n} \\ i<j}} \|\mu\left(C_i\right) - \mu\left(C_j\right)\|$$

The final primary metric is akin to the Silhouette coefficient:

$$\text{silhouette}\left(C_1, \ldots, C_n\right) = \frac{\text{sep}\left(C_1, \ldots, C_n\right)}{\text{homo}\left(C_1, \ldots, C_n\right)}$$

For the secondary metrics, the kNN algorithm is used and its accuracy (i.e. the fraction of correctly classified samples) measured. The kNN algorithm was seeded with the training data and its accuracy assessed on the testing data. This was done over the learning period, giving the first metric. Secondly, on the final model, the kNN classifier has been seeded with different amount of data and its performance measured (typically, this data is called the training data of the kNN classifier, in this work, however, it is referred to as **seed data** to avoid confusion with the data used to train the embedding). This gives a view into the robustness of the clustering, as high-quality embeddings would need only a small amount of seed data points to reach relatively high accuracy. For this last measure, the kNN algorithm was run thrice and the results averaged to compensate for high dependence on the particular seed points chosen if there is only a few of them.

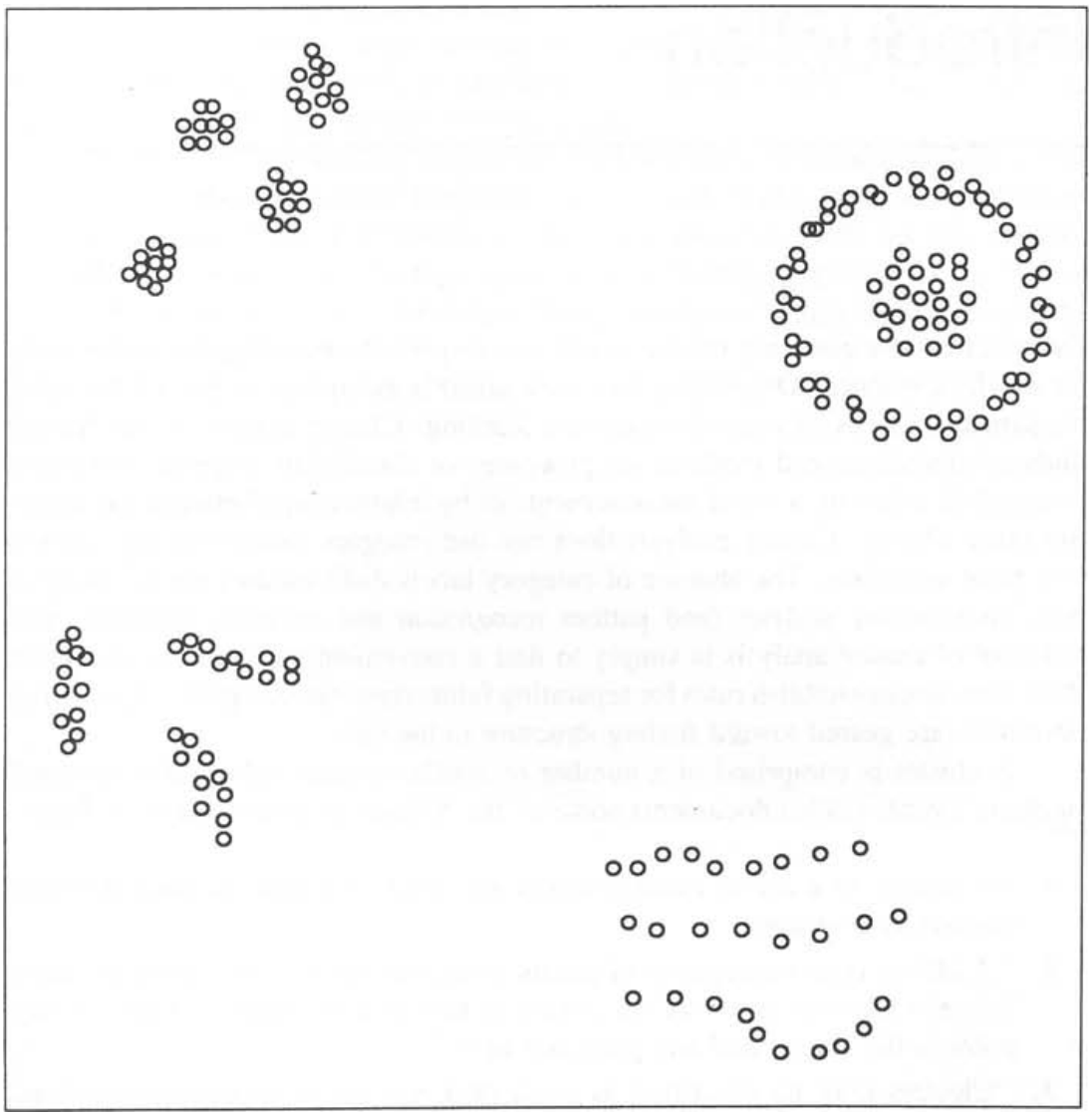


Figure 2.1: An example of the ambiguity of clusters from Jain et al., 1988. At the global level, there are four clusters in the scatter plot. At a local level, or a lower similarity threshold, twelve clusters can be recognised however.

Chapter 3

Learning a metric for clustering

In section 2.2, a method for learning a representation ϕ was shown. In order to learn the parameters of the representation, a clustering-loss function is required in the form

$$L_C : \mathcal{P}^M\left(\bar{\mathcal{X}}\right) \to \mathbb{R}$$

In this chapter, three ways of constructing such a clustering-loss function are explored. The chapter is structured as follows: First, an unsupervised method constructing a clustering loss-function is presented, followed by two supervised methods. Each of the methods builds on some prior art, which is introduced first, followed by an explanation of how the prior art was modified to solve the problem at hand.

3.1 Contrastive predictive coding

Contrastive predictive coding is a technique first introduced by Oord et al., 2019. In this section, the original article is first presented, with its application to MIL clustering introduced subsequently.

3.1.1 Original method

Contrastive predictive coding (also referred to as **CPC**) builds the model on the ideas from predictive coding (see Elias, 1955), a technique from information theory. In predictive coding, a sequence of messages is transmitted. Both the transmitter and the receiver try to predict future messages from all the past ones. Only the difference between the predicted and actual message is then actually transmitted.

CPC is an approach to representing time series by modelling future data from the past. In order to do that, the model learns high-level representations of the data and discards noise in the data. The further in future the model predicts, the less shared information is available and thus the global structure needs to be inferred better.

First, a non-linear encoder g_{enc} is used to map an input sequence of data to their latent representations:

$$g_{\text{enc}} : x_t \mapsto z_t$$

Next, an autoregressive model g_{ar} summarizes all latent representations up to a certain point t and produces a context latent representation:

$$c_t = g_{\text{ar}}\left(\{z_i | i \leq t\}\right)$$

Modelling $p_k\left(x_{t+k}|c_t\right)$ directly using a conditional generative model is computationally very expensive, which is the reason why this approach hasn't been chosen. Instead, the density ratio which preserves the

mutual information between x_{t+k} and c_t has been modelled as follows:

$$f_k\left(x_{t+k}, c_t\right) \propto \frac{p\left(x_{t+k}|c_t\right)}{p\left(x_{t+k}\right)} \tag{3.1}$$

A log-bilinear model was used:

$$f_k\left(x_{t+k}, c_t\right) = \exp\left(z_{t+k}^T W_k c_t\right)$$

Both g_{enc} and g_{ar} have been trained to jointly optimize a loss based on the Noise Contrastive Estimation (see Gutmann et al., 2010), which is called **InfoNCE** in Oord et al., 2019. Given a set $\mathbb{X} = \{x_1, \ldots, x_N\}$ containing one positive sample from $p\left(x_{t+k}|c_t\right)$ and $N-1$ negative samples from $p\left(x_{t+k}\right)$, the InfoNCE loss is as follows:

$$L_N = -\mathop{\mathbb{E}}_{\mathbb{X}}\left[\log f_k\left(x_{t+k}, c_t\right) - \log \sum_{x_j \in \mathbb{X}} f_k\left(x_j, c_t\right)\right]$$

Optimizing L_N will result in f_k estimating the density ratio from equation 3.1.

The model is illustrated in figure 3.1.

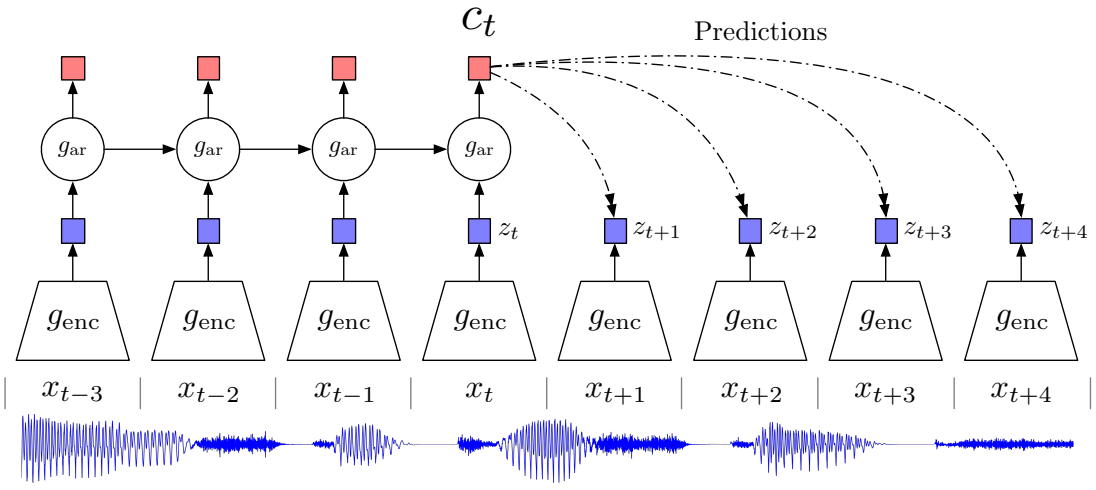


Figure 3.1: The Contrastive Predictive Coding model. Image from Oord et al., 2019

3.1.2 Application to MIL

The core idea taken from CPC to this work is that of the InfoNCE loss. The actual application is based on the following idea. If a bag is split into two parts, it is reasonable to expect that the representations of these two parts would be close to one another. On the other hand, if a random bag were to be drawn from the data (as a *simple random sample with replacement*), it is reasonable to expect it to be relatively far from any actual bag present in the data. These assumptions can be proven correct by using the probabilistic formalism for MIL (see section 1.3). Given that each bag B_k is viewed as a set of realizations of a probability distribution $P_k \in \mathcal{P}^{\mathcal{X}}$, it follows that the two parts of the bag, $B_k^{(1)}$ and $B_k^{(2)}$ are sets of realizations from the same distribution P_k and therefore should be statistically indistinguishable. On the other hand, a randomly sampled bag B_j' does not share the same probability distribution. Using these assumptions, the following clustering loss is constructed:

$$B_k^{(1)} \oplus B_k^{(2)} = B_k \in \mathcal{B}$$

$$\forall j \; B'_j \text{ is a bag of randomly sampled instances from } \mathcal{X}$$

$$L_{\text{CPC}} = \log \left\| \phi\left(B_k^{(1)}\right) - \phi\left(B_k^{(2)}\right) \right\|^2 - \log \sum_{j=1}^{K} \left\| \phi\left(B_k^{(1)}\right) - \phi\left(B'_j\right) \right\|^2$$

where $\oplus$ is the multiset sum (see definition 1.8). The first term of the loss function depicts the notion that the representations of the two parts of the bag should be close to one another and corresponds to the first term of InfoNCE, which maximizes the prediction of a matching future sample from the current context. The second term depicts the notion that a random bag should be far from all the bags[1] and corresponds to the second term of InfoNCE, which minimizes the prediction of a random sample from the current context. Choosing to only use the first part of the bag in the second term has no effect as the two parts are chosen randomly. The value $K \in \mathbb{N}$ is a hyper-parameter of this method.

This method can be further modified to make it less computationally complex. In order to not have to draw a lot of random bags, it can be reasonably expected that, on average, the representations of two parts of two mismatched bags $B_{k_1}^{(1)}$ and $B_{k_2}^{(2)}$ should be far apart. The matrix $\mathbf{D}$ is constructed as

$$\mathbf{D}_{ij} = \left\| \phi\left(B_i^{(1)}\right) - \phi\left(B_j^{(2)}\right) \right\|_2^2$$

The distances of the corresponding halves are found on the diagonal of $\mathbf{D}$, whereas the distances of mismatched halves are in the rest of the matrix. Under this assumption the final loss for the CPC method is

$$L_{\text{CPC}} = \frac{1}{n} \sum_{i=1}^{n} \left(\log\left(\mathbf{D}_{ii}\right) - \log \sum_{\substack{j=1 \\ j \neq i}}^{n} \mathbf{D}_{ij} \right)$$

where n is the number of bags.

3.2 Triplet loss

Triplet loss is a much more direct approach based on Weinberger et al., 2006. In this section, the original article is first presented, with its application to MIL clustering introduced subsequently.

3.2.1 Original method

The performance of the k-nearest neighbour algorithm depends heavily on the distance metric used. Typically, the Euclidean distance is used. However, ideally, the metric would adapt to the problem at hand. Weinberger et al., 2006 present a way to learn a Mahalanobis distance metric for kNN such that it has the *homogeneity* and *separation* properties of clustering.

Let $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ be a training dataset with $\boldsymbol{x}_i \in \mathbb{R}^d$ and y_i discrete class labels. The goal is to learn a linear transformation

$$\boldsymbol{L} : \mathbb{R}^d \to \mathbb{R}^d$$

This linear transformation is then used to compute squared distances as

$$\mathcal{D}\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \left\| \boldsymbol{L}\left(\boldsymbol{x}_i - \boldsymbol{x}_j\right) \right\|_2^2$$

[1] On the off-chance a random bag would be close to $B_k^{(1)}$, this would be outweighed by the other terms.

A helper matrix $\mathbf{y}$ is used such that

$$\mathbf{y}_{ij} = \begin{cases} 0 & \text{for} \quad y_i \neq y_j \\ 1 & \text{for} \quad y_i = y_j \end{cases}$$

In addition to this, for each input $\boldsymbol{x}_i$, k target neighbours are defined that are supposed to be close to $\boldsymbol{x}_i$. Euclidean distance may be used to find the target neighbours. A matrix η is used to indicate target neighbours where $\eta_{ij} = 1$ when $\boldsymbol{x}_j$ is a target neighbour of $\boldsymbol{x}_i$ and 0 otherwise.

The cost function features two competing terms. The first term depicts the notion that an input should be close to its target neighbours. The second term depicts the notion that inputs of a different class should be far from one another. The loss is of the form

$$\varepsilon(\boldsymbol{L}) = \sum_{i,j=1}^{n} \eta_{ij} \left\| \boldsymbol{L}(\boldsymbol{x}_i - \boldsymbol{x}_j) \right\|_2^2 + c \sum_{i,j,l=1}^{n} \eta_{ij} (1 - \mathbf{y}_{il}) \left\{ \left\| \boldsymbol{L}(\boldsymbol{x}_i - \boldsymbol{x}_l) \right\|_2^2 - \left\| \boldsymbol{L}(\boldsymbol{x}_i - \boldsymbol{x}_j) \right\|_2^2 \right\}_+$$

where $c > 0$ is a hyper-parameter of this method and $\{\cdot\}_+$ is the hinge loss, i.e.

$$\{x\}_+ = \max(0, 1 - x)$$

The loss can be reformulated as an instance of semidefinite programming (see Vandenberghe et al., 1996). This requires the distance $\mathcal{D}$ to be reformulated as

$$\mathcal{D}(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i - \boldsymbol{x}_j)^\top \boldsymbol{M} (\boldsymbol{x}_i - \boldsymbol{x}_j)$$

$$\boldsymbol{M} = \boldsymbol{L}^\top \boldsymbol{L}$$

This makes it clear that $\mathcal{D}$ truly is a Mahalanobis distance measure. The hinge loss is replaced by introducing slack variables ξ_{ij} for all i, j such that $\mathbf{y}_{ij} = 0$. The resulting semidefinite program is:

Minimize $\sum_{i,j=1}^{n} \eta_{ij} (\boldsymbol{x}_i - \boldsymbol{x}_j)^\top \boldsymbol{M} (\boldsymbol{x}_i - \boldsymbol{x}_j) + c \sum_{i,j,l=1}^{n} \eta_{ij} (1 - \mathbf{y}_{il}) \xi_{ijl}$

Subject to:

(1) $(\boldsymbol{x}_i - \boldsymbol{x}_l)^\top \boldsymbol{M} (\boldsymbol{x}_i - \boldsymbol{x}_l) - (\boldsymbol{x}_i - \boldsymbol{x}_j)^\top \boldsymbol{M} (\boldsymbol{x}_i - \boldsymbol{x}_j) \geq 1 - \xi_{ijl}$

(2) $\xi_{ijl} \geq 0$

(3) $\boldsymbol{M}$ is positive semidefinite

While this program can be solved by standard general-purpose solvers, Weinberger et al., 2006 use a custom solver.

3.2.2 Application to MIL

The idea taken from Weinberger et al., 2006 is the triplet loss itself as the optimizations using semidefinite programming are not of interest for this work.

The matrix $\mathbf{y}$ is defined the same way as in the original article:

$$\mathbf{y}_{ij} = \begin{cases} 0 & \text{for} \quad y_i \neq y_j \\ 1 & \text{for} \quad y_i = y_j \end{cases}$$

This makes triplet loss a supervised method. The matrix η can be defined similarly:

$$\eta_{ij} = \begin{cases} 1 & \text{if bag } B_j \text{ is a target neighbour for the bag } B_i \\ 0 & \text{otherwise} \end{cases}$$

Next, the pair-wise distance matrix $\mathbf{D}$ is used, in the same form as in section 3.1.2:

$$\mathbf{D}_{ij} = \left\| \phi\left(B_i^{(1)}\right) - \phi\left(B_j^{(2)}\right) \right\|_2^2$$

The loss itself is then expressed as:

$$L_{\text{triplet}} = \sum_{ij} \eta_{ij} D_{ij} + c \sum_{ijl} \eta_{ij} \left(1 - y_{il}\right) \left\{D_{il} - D_{ij}\right\}_+$$

where $c > 0$ is a hyper-parameter of the method. Although Weinberger et al., 2006 suggest finding η as the k-nearest neighbours of a data point, in this work, the loss has been simplified by setting $\eta_{ij} = \mathbf{y}_{ij}$.

3.3 Magnet loss

Magnet loss is an enhancement of the previously described triplet loss, first introduced by Rippel et al., 2015. In this section, the original article is first presented, with its application to MIL clustering introduced subsequently.

3.3.1 Original method

The idea of magnet loss builds upon triplet loss. In the case of triplet, the loss function essentially goes over all triplets of a data point, its target neighbour and a point from a different class and optimizes their distances (hence the name "triplet loss"). Magnet loss instead maintains an explicit model of the distributions of different classes in the representation space. To capture the distributions, the model uses simple clustering models for each class. In the case of the triplet loss, the target neighbourhood is set *a priori* and in the original input space – but using distances in the original input space is exactly what triplet loss and magnet loss aim to circumvent. To this end, magnet loss uses similarity in the space of representations and updates it periodically. Magnet loss also pursues local separation instead of a global one – representations of different classes should be separated, but can be interleaved. The difference between triplet loss and magnet loss is visualized in figure 3.2.

Let $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ be a training dataset with inputs $\boldsymbol{x}_i \in \mathbb{R}^d$ and y_i being C discrete class labels. A general parameterized map $\boldsymbol{f}\left(\cdot; \Theta\right)$ embeds the inputs into a representation space:

$$\boldsymbol{r}_i = \boldsymbol{f}\left(\boldsymbol{x}_i; \Theta\right)$$

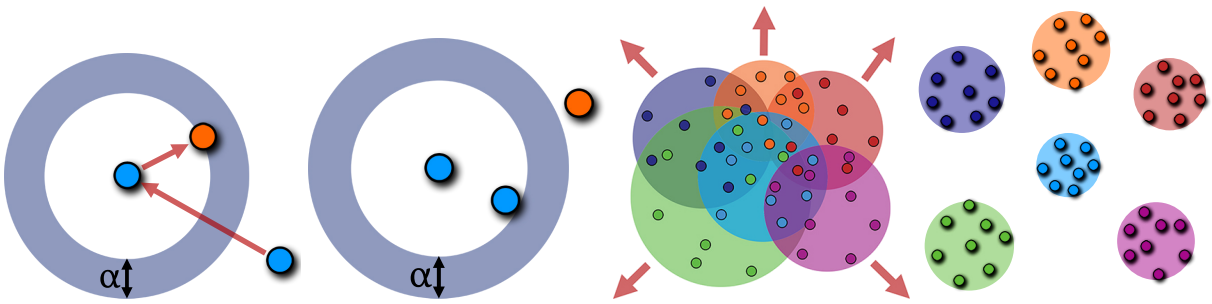
For each class c, K cluster assignments are obtained by using the K-means++ algorithm (see MacQueen, 1967 and Arthur et al., 2007) where $K \in \mathbb{N}$ is a hyper-parameter of the method:

$$\mathcal{I}_1^c, \ldots, \mathcal{I}_K^c = \underset{I_1^c, \ldots, I_K^c}{\arg\min} \sum_{k=1}^{K} \sum_{\boldsymbol{r} \in I_k^c} \left\| \boldsymbol{r} - \frac{1}{\left|I_k^c\right|} \sum_{\boldsymbol{r} \in I_k^c} \boldsymbol{r} \right\|_2^2$$

the centre of each cluster[2] is denoted $\boldsymbol{\mu}_k^c$:

$$\boldsymbol{\mu}_k^c = \frac{1}{\left|\mathcal{I}_k^c\right|} \sum_{\boldsymbol{r} \in \mathcal{I}_k^c} \boldsymbol{r}$$

[2] Rippel et al., 2015 incorrectly uses $\boldsymbol{\mu}_k^c$ as the centre of both $\mathcal{I}_k^c$ as well as the intermediary cluster I_k^c.



(a) Triplet loss: before (b) Triplet loss: after (c) Magnet loss: before (d) Magnet loss: after

Figure 3.2: Visualization of the difference between triplet loss and magnet loss. While triplet loss only considers one data point at a time, magnet loss operates on a neighbourhood of clusters. Image from Rippel et al., 2015

For each input, the centre of the cluster in which its representation falls is denoted as

$$\boldsymbol{\mu}\left(\boldsymbol{r}\right) = \arg\min_{\boldsymbol{\mu_k^c}} \|\boldsymbol{r} - \boldsymbol{\mu}_k^c\|$$

and the distance of its representation from the centre of the cluster is denoted as

$$N_i = \|\boldsymbol{r}_i - \mu\left(\boldsymbol{r}_i\right)\|_2^2$$

The variance of the distance of all the representations from their respective centres can then be calculated as

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n} N_i$$

For each input, the distance to all the inputs of a different class is calculated as

$$M_i = \sum_{c \neq y_i}\sum_{k=1}^{K} \exp\left(-\frac{1}{2\sigma^2}\|\boldsymbol{r}_i - \boldsymbol{\mu}_k^c\|_2^2\right)$$

The magnet loss is then

$$L\left(\Theta\right) = \frac{1}{n}\sum_{i=1}^{n}\left\{-\log\frac{\exp\left(-\frac{1}{2\sigma^2}N_i - \alpha\right)}{M_i}\right\}_+$$

where $\alpha \in \mathbb{R}$ is a hyper-parameter of the method, which has the meaning of the desired cluster separation gap with respect to the variance.

This loss function is made computationally less expensive by only sampling a certain amount of clusters for a class, then sampling a certain amount of inputs in each cluster. The rest of the data then isn't considered. This optimization yields the final training procedure:

1. Sample a seed cluster $I_1 \sim p_I\left(\cdot\right)$.
2. Find the $M-1$ nearest **impostor clusters** $I_2, \ldots I_M$ for I_1.
3. For each cluster I_m, sample D examples $\boldsymbol{x}_i^m, \ldots, \boldsymbol{x}_D^m \sim p_{I_m}\left(\cdot\right)$.

A stochastic approximation of the loss function is then constructed as

$$\boldsymbol{r}_d^m = \boldsymbol{f}\left(\boldsymbol{x}_d^m; \Theta\right)$$

$$\hat{\boldsymbol{\mu}}_m = \frac{1}{D}\sum_{d=1}^{D} \boldsymbol{r}_d^m$$

$$\hat{\mathbf{N}}_{md} = \left\|\boldsymbol{r}_d^m - \hat{\boldsymbol{\mu}}_m\right\|_2^2$$

$$\hat{\sigma}^2 = \frac{1}{MD-1}\sum_{m=1}^{M}\sum_{d=1}^{D}\hat{\mathbf{N}}_{md}$$

$$\hat{\mathbf{M}}_{md} = \sum_{\substack{\hat{\boldsymbol{\mu}} \\ C(\hat{\boldsymbol{\mu}})\neq C\left(\boldsymbol{r}_d^m\right)}} \exp\left(-\frac{1}{2\hat{\sigma}^2}\left\|\boldsymbol{r}_d^m - \hat{\boldsymbol{\mu}}\right\|_2^2\right)$$

$$L\left(\Theta\right) = \frac{1}{MD}\sum_{m=1}^{M}\sum_{d=1}^{D}\left\{-\log\frac{\exp\left(-\frac{1}{2\hat{\sigma}^2}\hat{\mathbf{N}}_{md} - \alpha\right)}{\hat{\mathbf{M}}_{md}}\right\}_+$$

where $C\left(\boldsymbol{r}\right)$ is the class of the representation $\boldsymbol{r}$.

The cluster index is refreshed periodically between optimization passes with the frequency of the updates being a hyper-parameter of the method, although Rippel et al., 2015 suggest updating it before each optimization pass.

3.3.2 Application to MIL

The magnet loss is taken almost verbatim from the original article. The simplification by sampling clusters and data points is not used. For posterity, the complete formulation of the method is as follows:

$$\boldsymbol{r}_i = \phi\left(B_i\right)$$

$$\mathcal{I}_1^c, \ldots, \mathcal{I}_K^c = \underset{I_1^c,\ldots,I_K^c}{\arg\min}\sum_{k=1}^{K}\sum_{\boldsymbol{r}\in I_k^c}\left\|\boldsymbol{r} - \frac{1}{\left|I_k^c\right|}\sum_{\boldsymbol{r}\in I_k^c}\boldsymbol{r}\right\|_2^2$$

$$\boldsymbol{\mu}_k^c = \frac{1}{\left|\mathcal{I}_k^c\right|}\sum_{\boldsymbol{r}\in\mathcal{I}_k^c}\boldsymbol{r}$$

$$\boldsymbol{\mu}\left(\boldsymbol{r}\right) = \underset{\boldsymbol{\mu_k^c}}{\arg\min}\left\|\boldsymbol{r} - \boldsymbol{\mu}_k^c\right\|$$

$$N_i = \left\|\boldsymbol{r}_i - \mu\left(\boldsymbol{r}_i\right)\right\|_2^2$$

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n} N_i$$

$$M_i = \sum_{c\neq y_i}\sum_{k=1}^{K}\exp\left(-\frac{1}{2\sigma^2}\left\|\boldsymbol{r}_i - \boldsymbol{\mu}_k^c\right\|_2^2\right)$$

$$L_{\text{magnet}} = \frac{1}{n}\sum_{i=1}^{n}\left\{-\log\frac{\exp\left(-\frac{1}{2\sigma^2}N_i - \alpha\right)}{M_i}\right\}_+$$

where y_i is the class label assigned to the bag B_i (making magnet loss a supervised method), n is the number of bags in the dataset and $K \in \mathbb{N}$ and $\alpha \in \mathbb{R}$ are hyper-parameters of the method.

The choice of the hyper-parameter K is non-obvious for most problems. Due to this, alternative methods for obtaining the cluster index which would not need hyper-parameter tuning were explored. One such method, **self-tuning spectral clustering** (see Zelnik-manor et al., 2005) was implemented as a replacement for the K-means algorithm used in the original method. Section 4.6 presents a comparison of the performance of these two methods.

3.4 A comparison of the methods

Each of the three methods presented in this chapter is a unique approach to solving the same problem. Out of the three, the approach based on contrastive predictive coding seems the most promising, as it is the only unsupervised method and could therefore be used in a broader class of applications. This, however, also has its drawbacks – unsupervised methods are generally harder to learn correctly. For that reason, the two supervised methods were selected as a safer option. Comparing them, since magnet loss is an enhancement of triplet loss, it is reasonable to assume it would perform the same or better than triplet loss. One drawback of magnet loss is the higher number of hyper-parameters which need to be tuned in order for the method to work correctly. This problem is partially solved by the introduction of self-tuning spectral clustering. Chapter 4 evaluates all of the methods on publicly available datasets and chapter 5 then applies all the methods to a corporate dataset in the domain of network security.

Chapter 4

Evaluation on publicly available datasets

The three loss functions presented in chapter 3 were experimentally evaluated on publicly available datasets. This chapter is structured as follows: The datasets and the models are presented, along with all the used configuration options so that the experiments would be independently replicable. Two baseline models are presented as a comparison to the proposed methods. Then, for triplet loss and magnet loss, the hyper-parameters are discussed (that is the parameters to be set by the architect of the model) and optimal values selected by performing a grid-search of the parameter space. Finally, all the methods are compared using their optimal configuration. When comparing the methods, it is important not only to look at the performance metrics at the end of the training phase, but to actually observe their development during the whole training process. It would make no sense to select a method which has a better initial performance, but isn't affected by the learning process. Also note that in all comparisons, both the value of silhouette $(\cdot)$ and the accuracy of a kNN classifier built on the embedding are taken into consideration as they may or may not follow each other.

4.1 Used datasets

The evaluation was done on a set of 20 datasets. These are, in alphabetical order:

1. The "BrownCreeper" and "WinterWren" datasets. See Briggs et al., 2012. A dataset of bird songs where each bag represents a recording of one or multiple birds. Originally a 13-class dataset, converted to binary classification datasets by selecting a target class.

2. The "CorelAfrican" and "CorelBeach" datasets. See Yixin Chen; Bi, et al., 2006. A dataset of object images where each bag is an image, consisting of segments described by 4×4 patch features. Originally a 20-class dataset, converted to binary classification datasets by selecting a target class.

3. The "Elephant", "Fox" and "Tiger" datasets. See Andrews et al., 2002. A dataset of animal images where each bag is an image, consisting of segments. Originally a multi-class dataset (also with animals other that the 3), converted to binary classification datasets by selecting a target class.

4. The "Musk1" and "Musk2" datasets. See Dietterich et al., 1997. A dataset of molecules where each bag is a set of the different shapes the molecule can fold into (so-called *conformers*). The goal is to predict whether a molecule has a musky smell or not. If at least one of the conformers of a molecule can cause it to smell musky, the molecule is positive.

5. The ”Mutagenesis1” and ”Mutagenesis2” datasets. See Srinivasan et al., 1995. A dataset consisting of a drug activity prediction problem. There is an easy (”Mutagenesis1”) and a hard (”Mutagenesis2”) version.

6. The ”Newsgroups1”, ”Newsgroups2” and ”Newsgroups3” datasets. See Zhou; Sun, et al., 2008. A text categorization dataset where each bag is a collection of posts from different newsgroups. A positive bag for a class is designed to contain 3% of posts from the target class and 97% randomly sampled from the other categories. Originally a 20-class dataset, converted to binary classification datasets by selecting a target class.

7. The ”Protein” dataset. See Ray et al., 2005a and Ray et al., 2005b. A dataset consisting of protein annotations making it a text categorization problem. The task is to decide whether a given pair should be annotated by a Gene Ontology (GO) code.

8. The ”UCSBBreastCancer” dataset. See Kandemir et al., 2014. A dataset consisting of 38 TMA image excerpts from breast cancer patients where each bag represents an image excerpt consisting of image patches. The goal is to predict whether the cancer is benign or malignant.

9. The ”Web1”, ”Web2”, ”Web3” and ”Web4” datasets. See Zhou; Jiang, et al., 2005. A dataset of webpages as ranked by 4 different users based on their being interesting. Each bag represents a webpage with instances being the links on the webpage.

All the datasets as used were made public in Dědič, 2019.

4.2 Used models

The models for evaluation were implemented in the Julia programming language (see Bezanson et al., 2017) using the Flux.jl framework for machine learning (see Innes, 2018) and the Mill.jl framework for multi-instance learning (see Pevný; Mandlík, 2019).

The embedding ϕ was realised by a MIL neural network consisting of 2 per-instance layers of 30 neurons, followed by aggregation formed by concatenating element-wise mean and element-wise maximum of all instances in a bag, followed by 2 per-bag layers of 30 neurons. All the neurons used the ReLU activation function (see Hahnloser et al., 2000). Layer weights were initialized using Glorot initialization (see Glorot et al., 2010), bias vectors were initialized to zeros. ADAM (see Kingma et al., 2014) was used as the optimization method. This particular architecture of the models was chosen based upon previous experience with using neural networks in multi-instance setting, particularly the works Dědič, 2017 and Pevný; Dědič, 2020. Several architectures were tested and the best one selected for the experiments.

For each of the datasets, 80% of bags was randomly chosen as the training data, with the rest being testing data. The models were trained using 100 mini-batches of size 50.

4.3 Mean model and classification model

In order to provide some baseline against which the models could be compared (as there is no prior art for this problem), two other models were introduced.

A non-machine-learning model was introduced as a baseline, which all models should surpass. This model implements the embedding ϕ as an element-wise mean of all instances of a bag. This represents a natural embedding of a bag as its expected value. This model is reffered to as the **mean model** in the rest of this work.

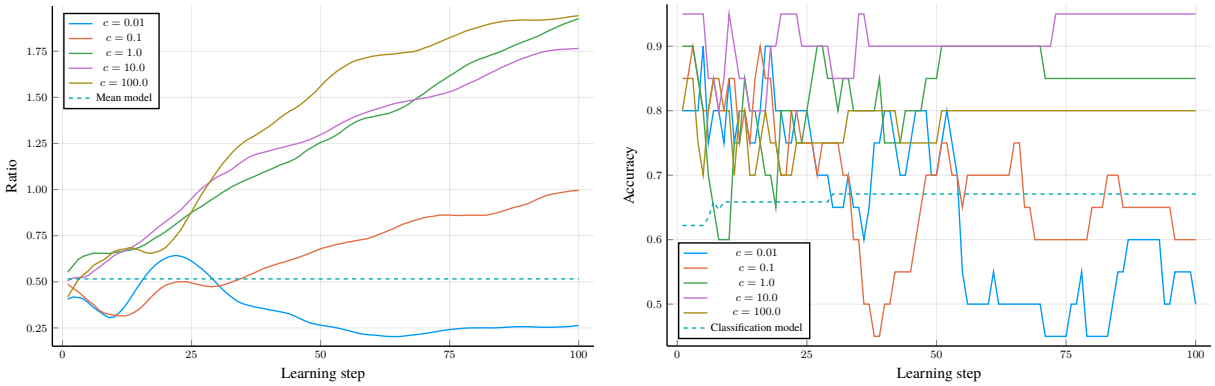
A classification model has been introduced as a target to beat. This model was realised by a MIL neural network consisting of 2 per-instance layers of 30 neurons, followed by aggregation formed by concatenating element-wise mean and element-wise maximum of all instances in a bag, followed by 2 per-bag layers of 30 neurons and a final layer of 2 output neurons. All the neurons used the ReLU activation function, except for the output neurons which used identity as their activation function. Layer weights were initialized using Glorot initialization, bias vectors were initialized to zeros. ADAM was used as the optimization method optimizing the cross-entropy loss. The accuracy of the model has been evaluated by selecting the optimal threshold on its output. This model mirrors the model used in the proposed methods, but replaces the clustering with simple classification. It is therefore not designed to be the optimal classification model. This model is reffered to as the **classification model** in the rest of this work.

4.4 Method hyper-parameter selection

Some of the three proposed clustering-losses have some hyper-parameters which need to be tuned, that is values that need to be chosen by the model architect as part of the design of the method. L_{CPC} has no hyper-parameters, L_{triplet} has a single hyper-parameter c which controls the weght on the term separating clusters, and L_{magnet} has three hyper-parameters, K, which is the number of clusters fora each class, α, which is the desired separation between clusters, and the cluster index update frequency. For L_{triplet} and L_{magnet}, a range of values was tried for each hyper-parameter in order to select the best configuration for each. The testing was done on the "Musk2" dataset, as it is sufficiently hard and simultaneously the best-known industry standard dataset for MIL.

4.4.1 Triplet loss

For L_{triplet}, the values $c \in \{0.01, 0.1, 1, 10, 100\}$ have been tested. Figure 4.1 shows the values of silhouette $(\cdot)$ (i.e. the ratio between the average inter-cluster distance and the average intra-cluster distance, see section 2.3 for the exact definition) and the accuracy of a kNN classifier built on the embedding for the different hyper-parameter values.



(a) The value of silhouette $(\cdot)$.

(b) The accuracy of a kNN classifier built on the embedding.

Figure 4.1: The values of the performance metrics over the learning period for triplet loss.

As can be seen from the figures, for low values of c, the performance is not good and, more importantly, isn't increasing over the learning period. In the end, the value $c = 1$ has been selected as it offers

good performance and the higher values show some instability in the learning process.

4.4.2 Magnet loss

For L_{magnet}, the values $K \in \{2, 3, 5, 8, 16, 20\}$, $\alpha \in \{0, 0.1, 0.5, 1, 10\}$ and the cluster index update frequency in $\{5, 10, 25, 70, 130, 200, 1000\}$ have been tested. Figure 4.2 shows the values of silhouette $(\cdot)$ for the different hyper-parameter values. The accuracy of the kNN classsifier built on top of the embedding was also calculated and is included in the appendix.

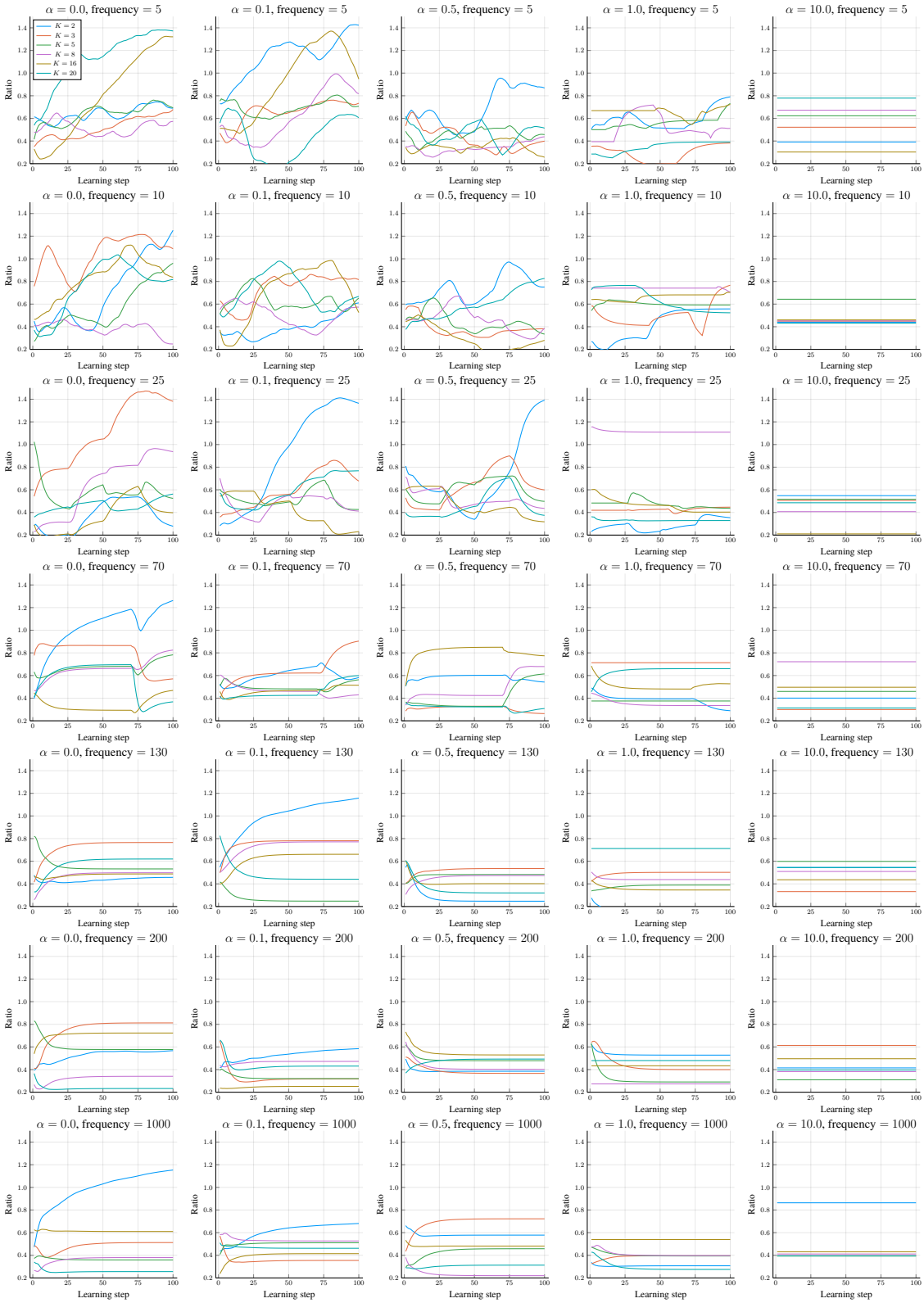
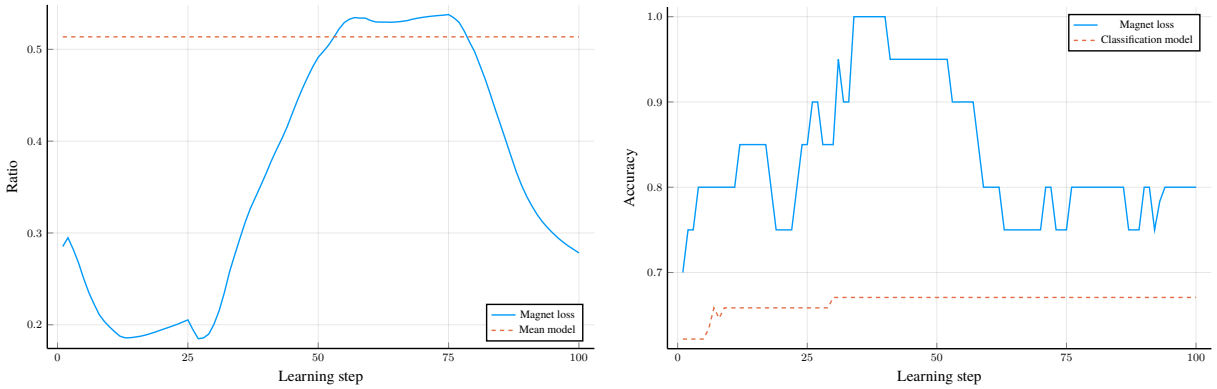


Figure 4.2: The value of silhouette $(\cdot)$ over the learning period for different values of K, α and the cluster index update frequency. Each subplot corresponds to a particular α and the cluster index update frequency. The series in each subplot correspond to a value of K.

As can be seen from the figures, the value of K doesn't affect the performance, therefore $K = 2$ was chosen due to its lowest computational complexity. For the values of α, the lower values offer better performance, with the values 0 and 0.1 being practically indistinguishable. Therefore $\alpha = 0$ was chosen. The cluster index update frequency offers no practical change in performance for values up to 25, with higher values having a negative impact. Therefore, due to computational constraints, the value of 25 was chosen. The selected hyper-parameter configuration is compared to the mean model and the classification model on the "Musk2" dataset in figure 4.3.



(a) The value of silhouette $(\cdot)$ over the learning period.

(b) The accuracy of a kNN classifier built on the embedding over the learning period.

Figure 4.3: Comparison of the selected hyper-parameter configuration for magnet loss to the mean model and the classification model.

4.5 Method comparison

The three methods were compared using their best hyper-parameter configuration. All of the methods were evaluated on all of the datasets, however, for better legibility, only the datasets "BrownCreeper", "Musk2", "Mutagenesis2" and "Web3" have been selected for the comparison as they contain a mixture of the most well-known, easy, hard and different domains. The rest of the figures, detailing the values of silhouette $(\cdot)$ and the accuracy of a kNN classifier based on the embedding as both a function of the learning step and as a function of the size of the seed dataset are presented in the appendix.

Figure 4.4 shows the value of silhouette $(\cdot)$ for the three methods, as evaluated on the 4 selected datasets. Table 4.1 shows the accuracy of the final model for each method and each dataset.

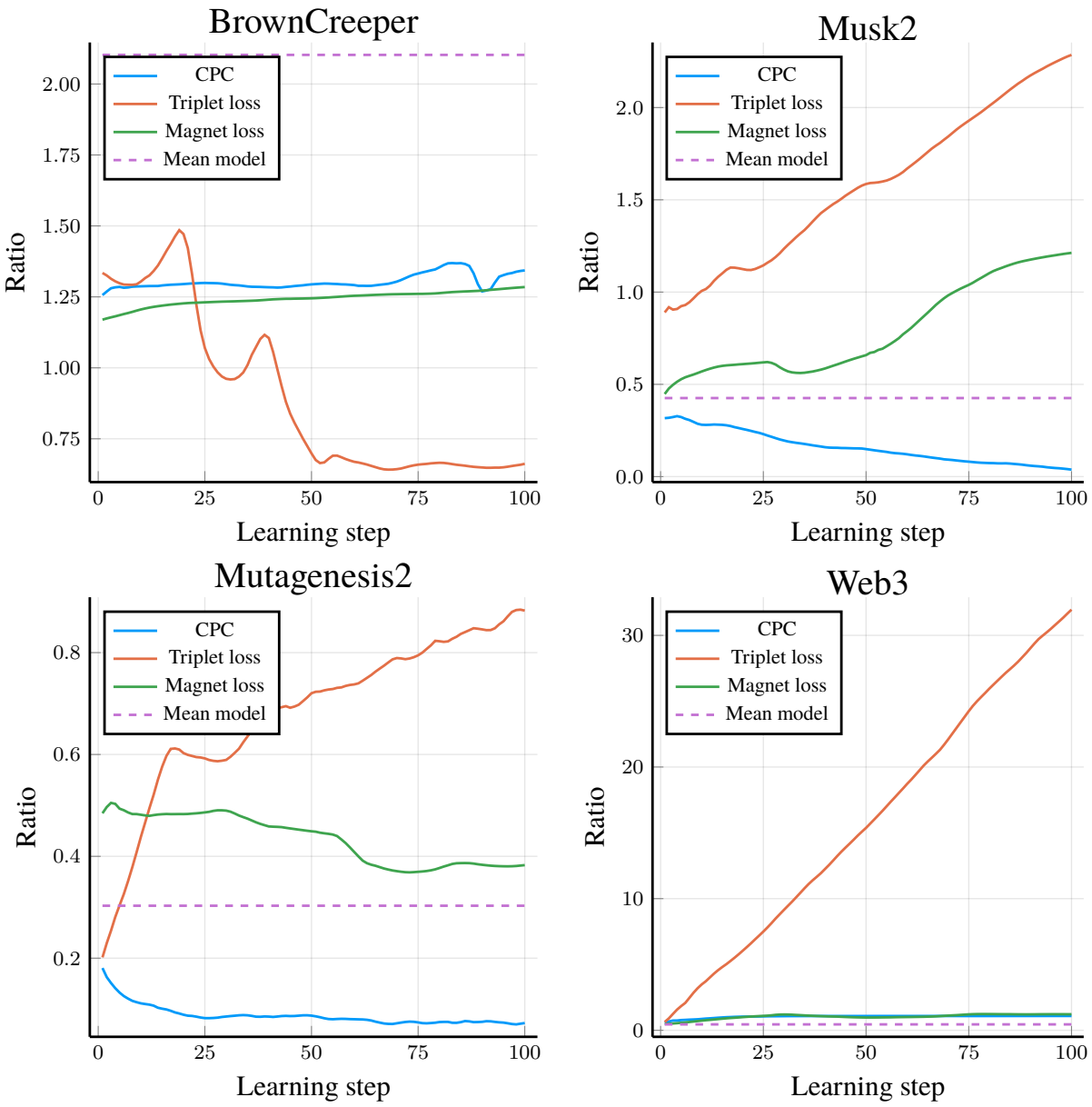


Figure 4.4: The value of silhouette $(\cdot)$ for the three methods over the learning period.

As can be seen from the figure and the table, triplet loss is the best performing among these 3 approaches as it is the only one which improves the metrics over the learning period (with the exception of the "BrownCreeper" dataset). The other two methods show no clear improvement. With that said, however, the performance varies wildly between datasets. On the "Web" datasets, the value of silhouette $(\cdot)$ quickly increases to vera high values, but the accuracy of the kNN classifier isn't accordingly high. This suggests that the embedding is quickly separating the different classes, but it does so in such a way that the kNN algorithm is not able to utilize this separation. This suggests that kNN might not be the right classification algorithm for this application.

Looking at the dependency of the accuracy on the size of the kNN seed data, some datasets clearly stand out as particularly hard for all of the methods (e.g. "Mutagenesis2"), while other have good results for all methods (e.g. "BrownCreeper"). Again, triplet loss is the clear winner as on most datasets it

Dataset	CPC	Triplet loss	Magnet loss
BrownCreeper	**0.900**	0.836	0.818
CorelAfrican	**0.950**	0.945	0.945
CorelBeach	0.965	**0.975**	0.970
Elephant	0.575	**0.875**	0.825
Fox	0.400	**0.625**	0.550
Musk1	0.667	**0.722**	0.667
Musk2	0.750	**1.0**	0.650
Mutagenesis1	0.658	**0.816**	0.789
Mutagenesis2	0.708	**0.750**	0.500
Newsgroups1	0.533	**0.900**	**0.900**
Newsgroups2	0.600	0.750	**0.850**
Newsgroups3	0.683	0.650	**0.700**
Protein	0.820	**0.846**	0.821
Tiger	0.642	**0.900**	0.575
UCSBBreastCancer	0.333	**0.750**	0.583
Web1	0.667	**0.733**	0.667
Web2	0.689	**0.733**	0.600
Web3	0.733	**0.867**	0.533
Web4	0.622	0.600	**0.667**
WinterWren	**0.936**	**0.936**	0.927

Table 4.1: The accuracy of the final models for the publicly available datasets.

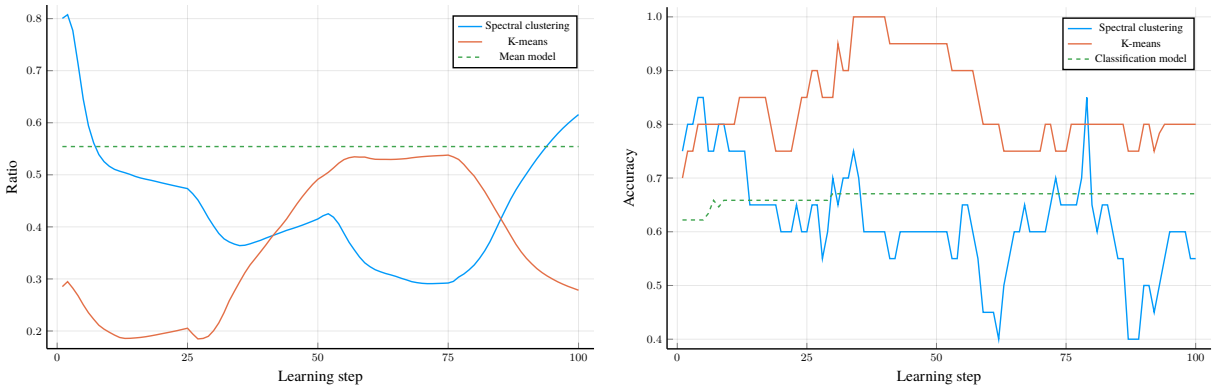
reaches its peak performance only with small amount of seed data.

Some of these findings go directly against the analysis presented in section 3.4, namely the prediction that magnet loss would be as good, if not better than triplet loss. The reason why it is not so may lie in the relative instability of the method, causing it to rapidly increase and the decrease all of the performance metrics.

4.6 Magnet loss with self-tuning spectral clustering

Figure 4.5 presents the difference between using K-means and self-tuning spectral clustering as the inner clustering algorithm in magnet loss.

As can be seen from the figures, the use of self-tuning spectral clustering brings no clear benefit to the performance of this method. The accuracy is lower then when the K-means algorithm was used, this can, however, be attributed to the instability of the method as a whole and may differ between runs.



(a) The value of silhouette $(\cdot)$ over the learning period.

(b) The accuracy of a kNN classifier built on the embedding over the learning period.

Figure 4.5: Comparison of the inner clustering algorithms for magnet loss.

Chapter 5

Evaluation on a corporate dataset

The tree methods were finally evaluated on a corporate dataset in the domain of computer security. It is reasonable to expect the methods to perform similarly to their behaviour on the publicly available datasets. However, this task is significantly harder, which influences each method differently.

5.1 The used dataset

The models were evaluated on a proprietary dataset provided by Cisco Cognitive Intelligence, consisting of records of network connections from clients (e.g. user computers or mobile devices) to some on-line services. The dataset represents HTTP traffic of more than 100 companies. Two datasets were collected, each spanning 1 day of traffic. The training data was traffic from 2019-11-18, while the data used for testing was collected the following day, 2019-11-19. For each connection, a proprietary classification system based on Jusko, 2017 provided labels, classifying the connections either as legitimate or malicious (connected to malware activity). The data was sampled to include 90% of negative bags and 10% of positive bags. Table 5.1 lists all the features provided in the dataset for each connection.

In addition to that, each connection contains the URL the client is connecting to. A hierarchical representation of the URL using MIL was added to the features – the MIL framework is powerful enough to enable a mix of traditional and MIL-based features. The URL model is based on the previous works Dědič, 2017 and Pevný; Dědič, 2020. The hierarchical model is visualized in figure 5.1.

The duration of the connection
The client port number
The IANA assigned internet protocol number
The server port number
The number of bytes sent from the client to the server
The number of bytes sent from the server to the client
The number of packets sent from the client to the server
The number of packets sent from the server to the client
The number of individual TCP connections
The number of SYN packets sent from the client to the server
The number of SYN-ACK packets sent from the server to the client
The number of RST packets sent from the client to the server
The number of RST packets sent from the server to the client
The number of FIN packets sent from the client to the server
The number of FIN packets sent from the server to the client

Table 5.1: The information provided for each connection in the corporate dataset.

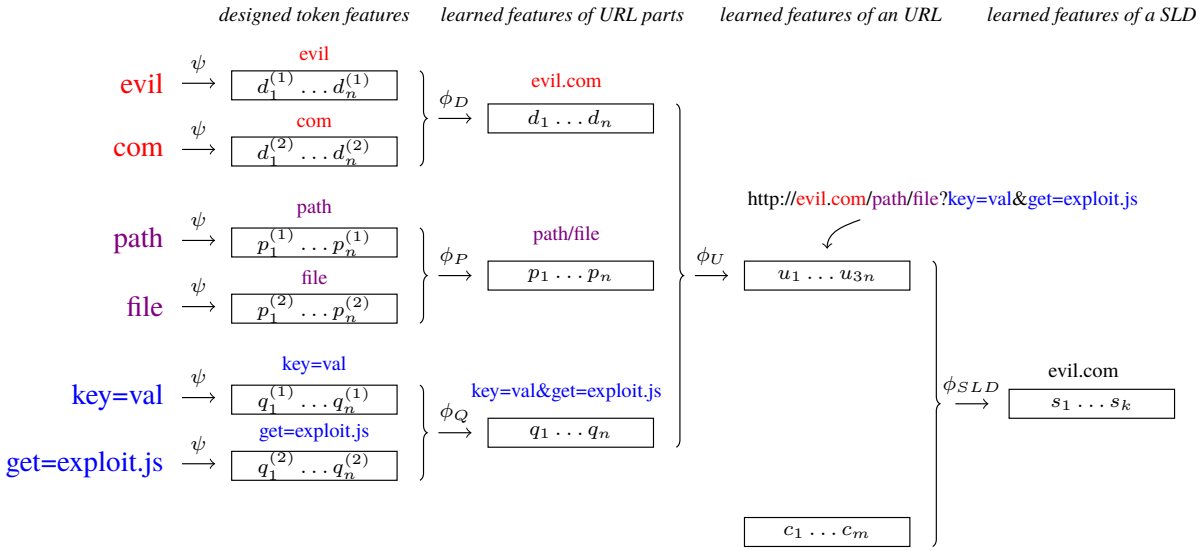


Figure 5.1: Hierarchical model of a URL. The vector $c_1, \ldots, c_m$ represents the connection features.

5.2 Used models

The models for evaluation were implemented in the Julia programming language (see Bezanson et al., 2017) using the Flux.jl framework for machine learning (see Innes, 2018) and the Mill.jl framework for multi-instance learning (see Pevný; Mandlík, 2019).

The embedding ϕ was realised by a MIL neural network. Using the notation from figure 5.1, ϕ_D, ϕ_P and ϕ_Q consisted of a layer of 20 neurons, followed by concatenating element-wise mean and element-wise maximum of all instances in a bag. After joining the URL parts by concatenating their representations, ϕ_U consisted of a layer of 120 neurons, followed by concatenation with the 15 connection features. Finally, ϕ_{SLD} consist of a layer of 135 neurons, followed by aggregation by concatenating element-wise

Legitimate traffic
Ad injector
Anonymization software
Advanced Persistent Threat (APT)
Banking trojan
Click fraud
Cryptocurrency miner
Data exfiltration
Exploit kit
Information stealer
Malicious advertising
Malicious content distribution
Malware distribution
Money scam
Potentially unwanted application (PUA)
Ransomware
Scareware
Spam botnet
Spam tracking
Trojan

Table 5.2: The malware classes distinguished in the multi-class dataset.

mean and element-wise maximum of all instances in a bag, followed by a layer with 270 neurons. All the neurons used the ReLU activation function (see Hahnloser et al., 2000). Layer weights were initialized using Glorot initialization (see Glorot et al., 2010), bias vectors were initialized to zeros. ADAM (see Kingma et al., 2014) was used as the optimization method. The models were trained using 20 mini-batches of 50 bags each.

A mean model and a classification model have been constructed similarily to the ones in section 4.3 The classification model is identical, with a final output layer of 2 neurons added. ADAM was used as the optimization method optimizing the cross-entropy loss. The accuracy of the model has been evaluated by selecting the optimal threshold on its output.

5.3 Multi-class classification

Apart from the task of learning a clustering based on a two-class labeling of bags as either legitimate or malicious, a multi-class classification system was also trained and evaluated. In this case, the data points were labeled as either legitimate or belonging to one of 19 classes of malware, described in table 5.2.

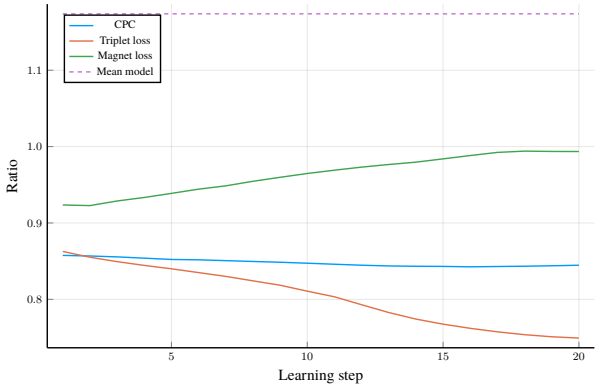
The data was sampled to include 81% of legitimate bags and 1% of every other class.

5.4 Method comparison

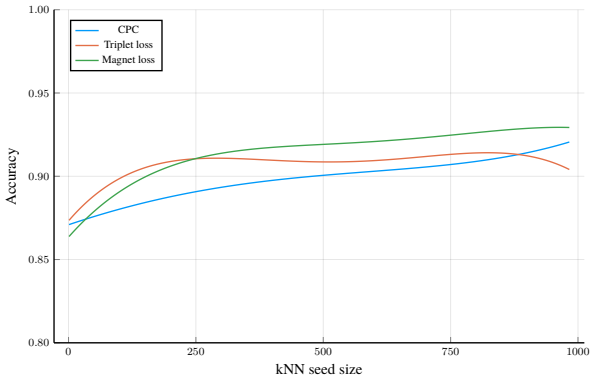
Figure 5.2a shows the value of silhouette $(\cdot)$ for all the methods over the learning period. Table 5.3 shows the accuracy of the final model for each method. Figure 5.2b shows the relation between the number of data points used to seed the kNN classifier and its accuracy. Due to the high noise (even when averaging

over three runs), in this figure the values are fitted by a polynomial of order 4 so that they could be compared. The original values are plotted in the appendix.

Figure 5.3a shows the value of silhouette $(\cdot)$ for all the methods for the 20-class classification problem over the learning period. Table 5.3 shows the accuracy of the final model for each method. Figure 5.3b shows the relation between the number of data points used to seed the kNN classifier and its accuracy. Due to the high noise (even when averaging over three runs), in this figure the values are fitted by a polynomial of order 4 so that they could be compared. The original values are plotted in the appendix.

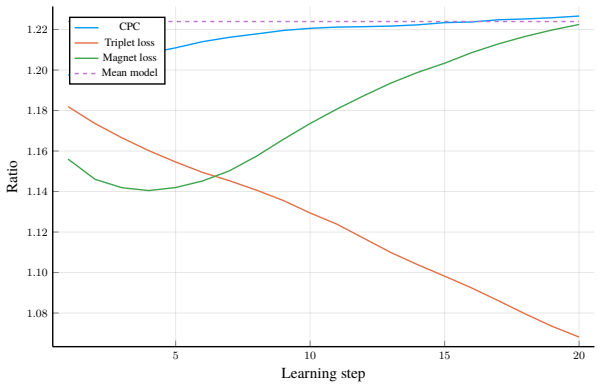


(a) The value of silhouette $(\cdot)$ over the learning period.

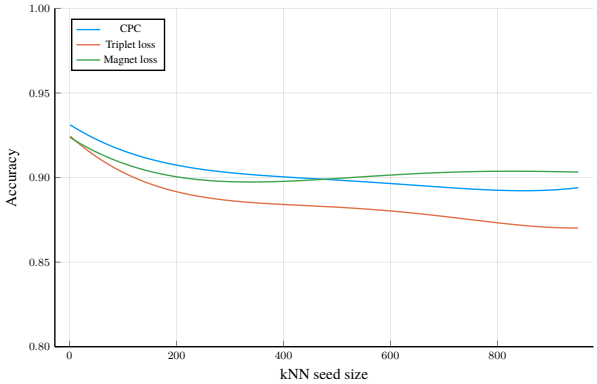


(b) The accuracy of a kNN classifier built on the final embedding as a function of the number of samples used to seed it. Average of three runs, fitted by a polynomial curve.

Figure 5.2: The performance metrics for two class classification on the corporate dataset.



(a) The value of silhouette $(\cdot)$ over the learning period.



(b) The accuracy of a kNN classifier built on the final embedding as a function of the number of samples used to seed it. Average of three runs, fitted by a polynomial curve.

Figure 5.3: The performance metrics for twenty class classification on the corporate dataset.

As can be seen from the figures and the table, for this problem, magnet loss performed the best. The model based on contrastive predictive coding is the second best and for the model based on triplet, the value of silhouette $(\cdot)$ actually goes down over the learning period. Figure 5.2b shows that triplet loss and magnet loss reach their peak performance for a small number of seed data, but CPC improves its

Classification problem	CPC	Triplet loss	Magnet loss
2 classes	0.920	0.910	**0.930**
20 classes	0.893	0.868	**0.904**

Table 5.3: The accuracy of the final models for the corporate datasets.

performance with every new seed point. For the multi-class version of the experiment, magnet loss still leads in terms of accuracy, but is somewhat surprisingly surpassed by CPC in terms of silhouette $(\cdot)$. Figure 5.3b shows an inverse relationship than expected, but this can be explained by the high number of classes which are only sampled for a sufficient number of seed data points.

Conclusion

In this work, an overview of multi-instance learning was provided with a mathematical formalism for describing it and a survey of the paradigms and methods. Clustering was introduced as a key task in unsupervised learning. Then, three methods for clustering based on multi-instance representations were introduced, one unsupervised and two supervised. For each of the methods, the prior art it builds on was presented, along with its modification for the purposes of multi-instance clustering. All three of the methods were theoretically and experimentally evaluated and compared. The experiments were conducted first on publicly available datasets in a reproducible fashion. Following that, a corporate dataset of network security data was used as it is the primary application in mind for this work.

Comparing the methods on publicly available datasets shows the method base on triplet loss to be the best performing one. This method outperformed the other two in the sense that it was the only one to improve its performance over the learning period. This is a surprising result, as magnet loss, which builds on triplet loss and enhances it, performed worse. When evaluating on the corporate dataset however, the method based on magnet loss outperformed both of the other methods, albeit none of the results were as good as anticipated. As this problem was a much harder one, this might hint at triplet loss not being able to handle such complex tasks, while the performance of magnet loss isn't impacted as much, making it surpass triplet loss. The method based on contrastive predictive coding performed poorly on both the publicly available datasets and the corporate one. However, the comaprison might not be fair as the CPC method is unsupervised, whereas the other two can utilize labels on the training data.

Clearly, more research is needed. The most promising method, CPC, performed poorly and should be investigated more. A thorough theoretical investigation of the method including finding its local extrema might explain its low performance, however, it was judged to be outside the scope of this work. Combining the representative power of multi-instance learning with the versatility of the clustering algorithms remains an open problem.

List of Tables

List of Figures

Bibliography

AMORES, Jaume, 2013. Multiple instance classification: Review, taxonomy and comparative study. *Artificial Intelligence* [online]. Vol. 201, pp. 81–105 [visited on 2017-05-31]. ISSN 0004-3702. Available from DOI: 10.1016/j.artint.2013.06.003.

ANDREWS, Stuart; TSOCHANTARIDIS, Ioannis; HOFMANN, Thomas, 2002. Support Vector Machines for Multiple-Instance Learning. *Advances in Neural Information Processing Systems*. Vol. 15, pp. 561–568.

ARTHUR, David; VASSILVITSKII, Sergei, 2007. K-Means++: The Advantages of Careful Seeding. In: vol. 8, pp. 1027–1035. Available from DOI: 10.1145/1283383.1283494.

BEZANSON, J.; EDELMAN, A.; KARPINSKI, S.; SHAH, V., 2017. Julia: A Fresh Approach to Numerical Computing. *SIAM Review* [online]. Vol. 59, no. 1, pp. 65–98 [visited on 2018-09-01]. ISSN 0036-1445. Available from DOI: 10.1137/141000671.

BRIGGS, Forrest; FERN, Xiaoli; RAICH, Raviv, 2012. Rank-loss support instance machines for MIML instance annotation. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Available from DOI: 10.1145/2339530.2339616.

CHEN, Ying; WU, Ou, 2012. Contextual Hausdorff dissimilarity for multi-instance clustering. In: *Fuzzy Systems and Knowledge Discovery (FSKD)* [online]. Sichuan, China: IEEE Computer Society Press, pp. 870–873 [visited on 2018-05-14]. ISBN 978-1-4673-0024-7. Available from DOI: 10.1109/FSKD.2012.6233889.

CHEN, Yixin; BI, Jinbo; WANG, J. Z., 2006. MILES: Multiple-Instance Learning via Embedded Instance Selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 28, no. 12, pp. 1931–1947. ISSN 0162-8828. Available from DOI: 10.1109/TPAMI.2006.248.

CHEN, Yixin; WANG, James Z., 2004. Image Categorization by Learning and Reasoning with Regions. *Journal of Machine Learning Research* [online]. Vol. 5, pp. 913–939 [visited on 2017-07-06]. ISSN ISSN 1533-7928. Available from: http://www.jmlr.org/papers/v5/chen04a.html.

CHEPLYGINA, Veronika; TAX, David M. J.; LOOG, Marco, 2015. Multiple instance learning with bag dissimilarities. *Pattern Recognition* [online]. Vol. 48, no. 1, pp. 264–275 [visited on 2017-07-04]. ISSN 0031-3203. Available from DOI: 10.1016/j.patcog.2014.07.022.

DASARATHY, Belur V., 1991. *Nearest neighbor (NN) norms: nn pattern classification techniques*. IEEE Computer Society Press. ISBN 978-0-8186-5930-0. Google-Books-ID: k2dQAAAAMAAJ.

DĚDIČ, Marek, 2017. *Hierarchické modely síťového provozu*. Prague. Bachelor thesis. Czech Technical University in Prague.

DĚDIČ, Marek, 2019. *MilDatasets.jl* [online]. Version v1.0.0 [visited on 2020-01-01]. Available from: https://github.com/marekdedic/MilDatasets.jl.

DEMPSTER, A. P.; LAIRD, N. M.; RUBIN, D. B., 1977. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)* [online]. Vol. 39, no. 1, pp. 1–38 [visited on 2017-07-01]. ISSN 0035-9246. Available from: http://www.jstor.org/stable/2984875.

DIETTERICH, Thomas G.; LATHROP, Richard H.; LOZANO-PÉREZ, Tomás, 1997. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence* [online]. Vol. 89, no. 1, pp. 31–71 [visited on 2017-05-31]. ISSN 0004-3702. Available from DOI: 10.1016/S0004-3702(96)00034-3.

ELIAS, P., 1955. Predictive coding–I. *IRE Transactions on Information Theory*. Vol. 1, no. 1, pp. 16–24. ISSN 2168-2712. Available from DOI: 10.1109/TIT.1955.1055126.

ESTIVILL-CASTRO, Vladimir, 2002. Why so many clustering algorithms: a position paper. *ACM SIGKDD Explorations Newsletter* [online]. Vol. 4, no. 1, pp. 65–75 [visited on 2019-12-27]. ISSN 1931-0145. Available from DOI: 10.1145/568574.568575.

EVERITT, Brian S.; LANDAU, Sabine; LEESE, Morven, 2001. *Cluster Analysis*. 4th ed. Taylor & Francis. ISBN 978-0-340-76119-9. Google-Books-ID: htZzDGlCnQYC.

FOULDS, James Richard, 2008. *Learning Instance Weights in Multi-Instance Learning* [online] [visited on 2017-07-06]. Available from: http://researchcommons.waikato.ac.nz/handle/10289/2460. Thesis. The University of Waikato.

GÄRTNER, Thomas; FLACH, Peter A.; KOWALCZYK, Adam; SMOLA, Alexander J., 2002. Multi-instance kernels. In: *ICML* [online]. Vol. 2, pp. 179–186 [visited on 2017-07-04]. Available from: http://sci2s.ugr.es/keel/pdf/algorithm/congreso/2002-Gartner-ICML.pdf.

GLOROT, Xavier; BENGIO, Yoshua, 2010. Understanding the difficulty of training deep feedforward neural networks. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* [online], pp. 249–256 [visited on 2018-09-01]. Available from: http://proceedings.mlr.press/v9/glorot10a.html.

GUTMANN, Michael; HYVÄRINEN, Aapo, 2010. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* [online], pp. 297–304 [visited on 2019-12-29]. Available from: http://proceedings.mlr.press/v9/gutmann10a.html.

HAHNLOSER, Richard H. R.; SARPESHKAR, Rahul; MAHOWALD, Misha A.; DOUGLAS, Rodney J.; SEUNG, H. Sebastian, 2000. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature* [online]. Vol. 405, no. 6789, pp. 947–951 [visited on 2018-08-27]. ISSN 1476-4687. Available from DOI: 10.1038/35016072.

HAUSSLER, David, 1999. *Convolution kernels on discrete structures* [online] [visited on 2017-07-04]. Available from: https://www.soe.ucsc.edu/sites/default/files/technical-reports/UCSC-CRL-99-10.pdf. Technical report, Department of Computer Science, University of California at Santa Cruz.

INNES, Mike, 2018. Flux: Elegant machine learning with Julia. *Journal of Open Source Software*. Available from DOI: 10.21105/joss.00602.

JAIN, Anil K.; DUBES, Richard C., 1988. *Algorithms for Clustering Data*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc. ISBN 978-0-13-022278-7.

JUSKO, Ján, 2017. *Graph-based Detection of Malicious Network Communities* [online] [visited on 2019-04-02]. Available from: https://dspace.cvut.cz/handle/10467/73702. PhD thesis.

KANDEMIR, Melih; ZHANG, Chong; HAMPRECHT, Fred A., 2014. Empowering Multiple Instance Histopathology Cancer Diagnosis by Cell Graphs. In: GOLLAND, Polina; HATA, Nobuhiko; BARILLOT, Christian; HORNEGGER, Joachim; HOWE, Robert (eds.). *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2014*. Cham: Springer International Publishing, pp. 228–235. Lecture Notes in Computer Science. ISBN 978-3-319-10470-6. Available from DOI: 10.1007/978-3-319-10470-6_29.

KINGMA, Diederik P.; BA, Jimmy, 2014. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]* [online] [visited on 2017-04-19]. Available from arXiv: 1412.6980.

KNUTH, Donald E., 1968. *The Art of Computer Programming*. Addison-Wesley. ISBN 0-201-03801-3.

KOHOUT, J.; PEVNÝ, T., 2018. Network Traffic Fingerprinting Based on Approximated Kernel Two-Sample Test. *IEEE Transactions on Information Forensics and Security*. Vol. 13, no. 3, pp. 788–801. ISSN 1556-6013. Available from DOI: 10.1109/TIFS.2017.2768018.

KWOK, James T.; CHEUNG, Pak-Ming, 2007. Marginalized Multi-Instance Kernels. In: *IJCAI* [online]. Vol. 7, pp. 901–906 [visited on 2017-07-04]. Available from: http://www.aaai.org/Papers/IJCAI/2007/IJCAI07-145.pdf.

MACQUEEN, J., 1967. Some methods for classification and analysis of multivariate observations. In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*. Berkeley, California: University of California Press, vol. 1, pp. 281–297.

MAHALANOBIS, P. C., 1936. On the Generalised Distance in Statistics. *Proceedings of the National Institute of Science, India*. Vol. 2, no. 1, pp. 49–55.

MARON, Oded; LOZANO-PÉREZ, Tomás, 1998. A framework for multiple-instance learning. In: *Advances in neural information processing systems* [online], pp. 570–576 [visited on 2017-07-01]. Available from: http://papers.nips.cc/paper/1346-a-framework-for-multiple-instance-learning.pdf.

MUANDET, Krikamol; FUKUMIZU, Kenji; DINUZZO, Francesco; SCHÖLKOPF, Bernhard, 2012. Learning from Distributions via Support Measure Machines. In: *Advances in neural information processing systems* [online], pp. 10–18 [visited on 2017-06-29]. Available from: http://papers.nips.cc/paper/4825-learning-from-distributions-via-support-measure-machines.

OORD, Aaron van den; LI, Yazhe; VINYALS, Oriol, 2019. Representation Learning with Contrastive Predictive Coding. *arXiv:1807.03748 [cs, stat]* [online] [visited on 2019-12-29]. Available from arXiv: 1807.03748.

PEVNÝ, Tomáš; DĚDIČ, Marek, 2020. *Nested Multiple Instance Learning in Modelling of HTTP network traffic* [Journal article submitted for publication].

PEVNÝ, Tomáš; MANDLÍK, Šimon, 2019. *Mill.jl* [online]. Before_zygote [visited on 2020-01-01]. Available from: https://github.com/pevnak/Mill.jl.

PEVNÝ, Tomáš; SOMOL, Petr, 2016. Discriminative Models for Multi-instance Problems with Tree Structure. In: *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security* [online]. New York, NY, USA: ACM, pp. 83–91 [visited on 2017-03-01]. AISec '16. ISBN 978-1-4503-4573-6. Available from DOI: 10.1145/2996758.2996761.

PEVNÝ, Tomáš; SOMOL, Petr, 2017. Using Neural Network Formalism to Solve Multiple-Instance Problems. In: *Advances in Neural Networks - ISNN 2017* [online]. Springer, Cham, pp. 135–142 [visited on 2018-05-23]. Lecture Notes in Computer Science. ISBN 978-3-319-59072-1. Available from DOI: 10.1007/978-3-319-59072-1_17.

RAY, Soumya; CRAVEN, Mark, 2005a. Learning Statistical Models for Annotating Proteins with Function Information using Biomedical Text. *BMC Bioinformatics* [online]. Vol. 6, no. 1, pp. S18 [visited on 2020-01-01]. ISSN 1471-2105. Available from DOI: 10.1186/1471-2105-6-S1-S18.

RAY, Soumya; CRAVEN, Mark, 2005b. Supervised versus multiple instance learning: An empirical comparison. In: pp. 697–704. Available from DOI: 10.1145/1102351.1102439.

RIPPEL, Oren; PALURI, Manohar; DOLLAR, Piotr; BOURDEV, Lubomir, 2015. Metric Learning with Adaptive Density Discrimination.

SRINIVASAN, A.; MUGGLETON, Stephen; KING, Robert, 1995. Comparing the use of background knowledge by inductive logic programming systems. In:

VANDENBERGHE, Lieven.; BOYD, Stephen., 1996. Semidefinite Programming. *SIAM Review* [online]. Vol. 38, no. 1, pp. 49–95 [visited on 2019-12-30]. ISSN 0036-1445. Available from DOI: 10.1137/1038003.

WANG, Jun; ZUCKER, Jean-Daniel, 2000. Solving Multiple-Instance Problem: A Lazy Learning Approach. In: LANGLEY, Pat (ed.). *Proceedings of the Seventeenth International Conference on Machine Learning* [online]. Stanford University, Stanford, CA, USA: Morgan Kaufmann, pp. 1119–1125 [visited on 2017-07-01]. Available from: http://cogprints.org/2124/.

WEINBERGER, Kilian Q; BLITZER, John; SAUL, Lawrence K., 2006. Distance Metric Learning for Large Margin Nearest Neighbor Classification. In: WEISS, Y.; SCHÖLKOPF, B.; PLATT, J. C. (eds.). *Advances in Neural Information Processing Systems 18* [online]. MIT Press, pp. 1473–1480 [visited on 2019-06-27]. Available from: http://papers.nips.cc/paper/2795-distance-metric-learning-for-large-margin-nearest-neighbor-classification.pdf.

XU, Dongkuan; TIAN, Yingjie, 2015. A Comprehensive Survey of Clustering Algorithms. *Annals of Data Science* [online]. Vol. 2, no. 2, pp. 165–193 [visited on 2019-12-26]. ISSN 2198-5812. Available from DOI: 10.1007/s40745-015-0040-1.

ZELNIK-MANOR, Lihi; PERONA, Pietro, 2005. Self-Tuning Spectral Clustering. In: SAUL, L. K.; WEISS, Y.; BOTTOU, L. (eds.). *Advances in Neural Information Processing Systems 17* [online]. MIT Press, pp. 1601–1608 [visited on 2020-01-06]. Available from: http://papers.nips.cc/paper/2619-self-tuning-spectral-clustering.pdf.

ZHANG, Cha; PLATT, John C.; VIOLA, Paul A., 2006. Multiple instance boosting for object detection. In: *Advances in neural information processing systems* [online], pp. 1417–1424 [visited on 2017-07-04]. Available from: http://papers.nips.cc/paper/2926-multiple-instance-boosting-for-object-detection.pdf.

ZHANG, Min-Ling; ZHOU, Zhi-Hua, 2009. Multi-instance clustering with applications to multi-instance prediction. *Applied Intelligence* [online]. Vol. 31, no. 1, pp. 47–68 [visited on 2017-07-06]. ISSN 0924-669X, 1573-7497. ISSN 0924-669X, 1573-7497. Available from DOI: 10.1007/s10489-007-0111-x.

ZHANG, Qi; GOLDMAN, Sally A., 2002. EM-DD: An Improved Multiple-Instance Learning Technique. In: *Advances in neural information processing systems* [online], pp. 1073–1080 [visited on 2017-07-01]. Available from: http://papers.nips.cc/paper/1959-em-dd-an-improved-multiple-instance-learning-technique.pdf.

ZHOU, Zhi-Hua; JIANG, Kai; LI, Ming, 2005. Multi-Instance Learning Based Web Mining. *Applied Intelligence* [online]. Vol. 22, no. 2, pp. 135–147 [visited on 2020-01-01]. ISSN 1573-7497. Available from DOI: 10.1007/s10489-005-5602-z.

ZHOU, Zhi-Hua; SUN, Yu-Yin; LI, Yu-Feng, 2008. Multi-Instance Learning by Treating Instances As Non-I.I.D. Samples. *Proceedings of the 26th International Conference On Machine Learning, ICML 2009*. Available from DOI: 10.1145/1553374.1553534.

ZHOU, Zhi-Hua; ZHANG, Min-Ling, 2002. Neural Networks for Multi-Instance Learning. In: *Proceedings of the International Conference on Intelligent Information Technology, Beijing, China* [online], pp. 455–459 [visited on 2017-06-26]. Available from: http://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/techrep02.pdf.

ZHU, Ji; ROSSET, Saharon; TIBSHIRANI, Robert; HASTIE, Trevor J., 2004. 1-norm support vector machines. In: *Advances in neural information processing systems* [online], pp. 49–56 [visited on 2017-07-01]. Available from: http://papers.nips.cc/paper/2450-1-norm-support-vector-machines.pdf.

Appendix A

Plots

This appendix contains all of the plots that weren't significant enough to include in chapters 4 and 5.

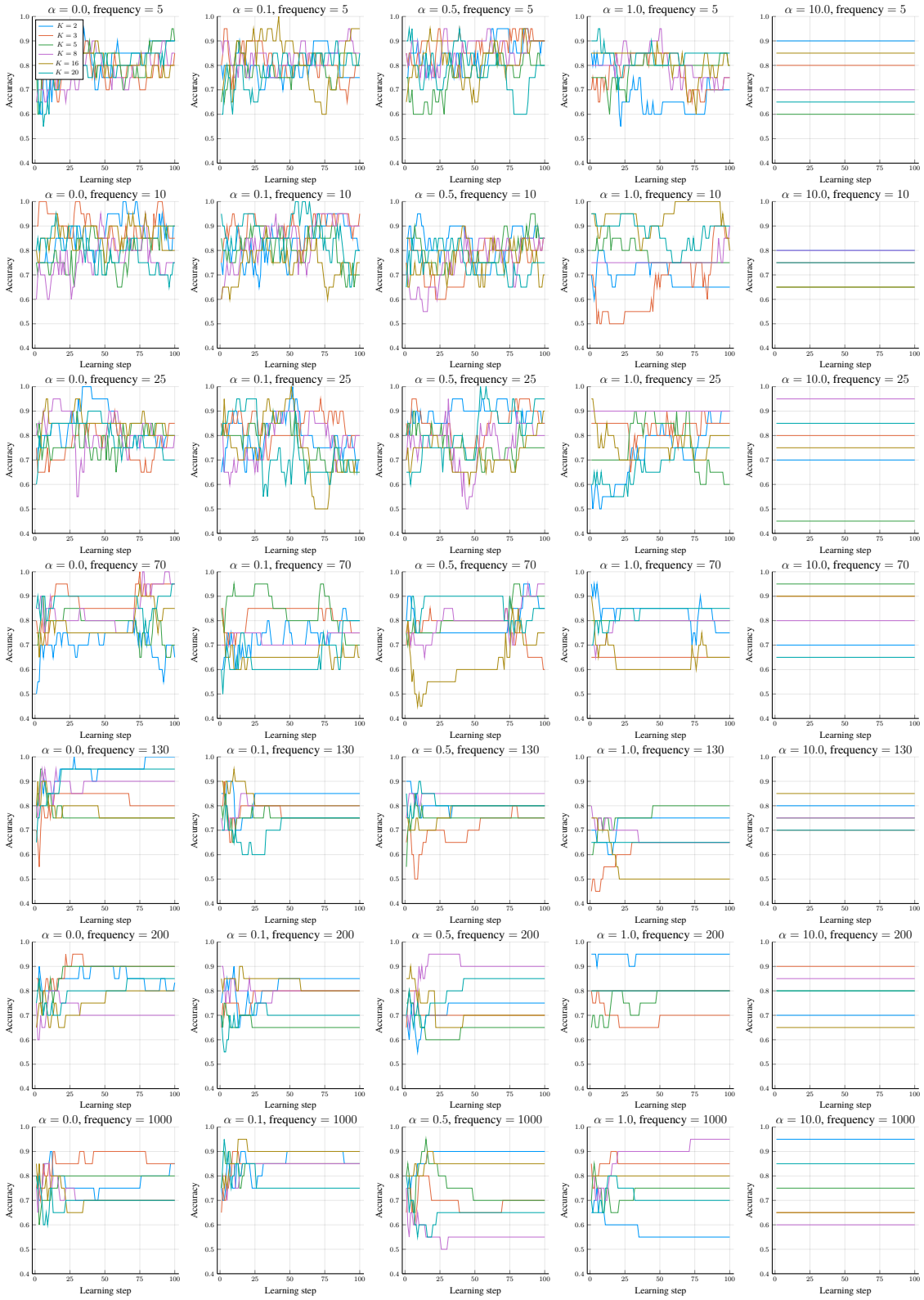


Figure A.1: The accuracy of a kNN classifier built on the embedding over the learning period for different values of K, α and the cluster index update frequency for magnet loss. Each subplot corresponds to a particular α and the cluster index update frequency. The series in each subplot correspond to a value of K.

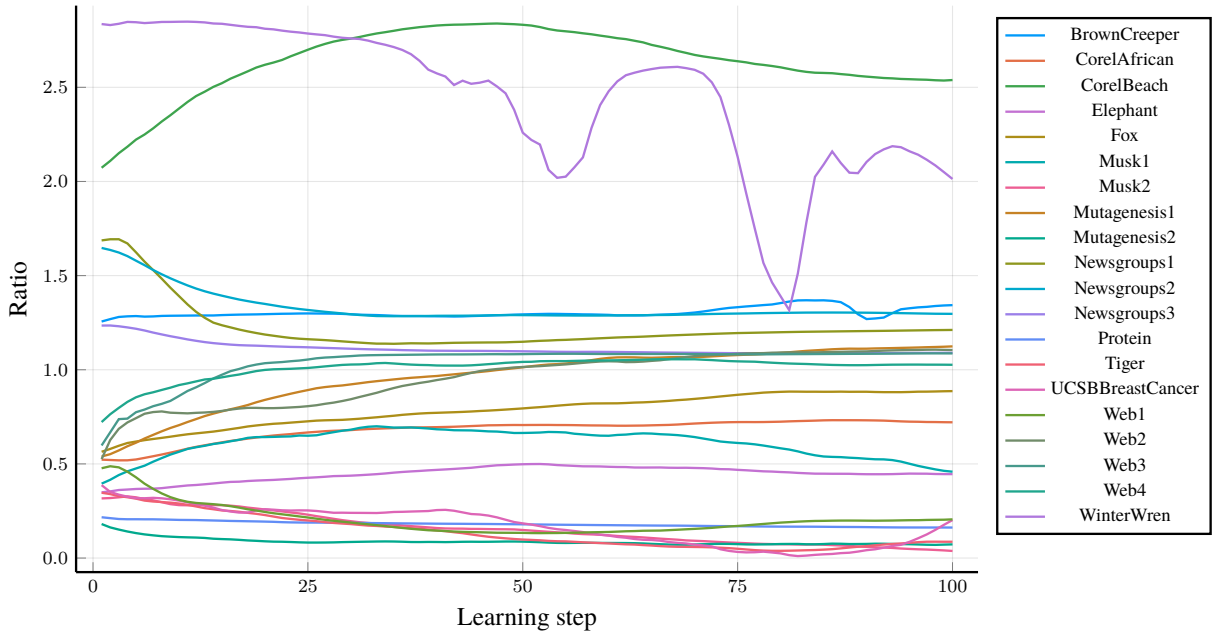


Figure A.2: The value of silhouette $(\cdot)$ for L_{CPC} over the learning period.

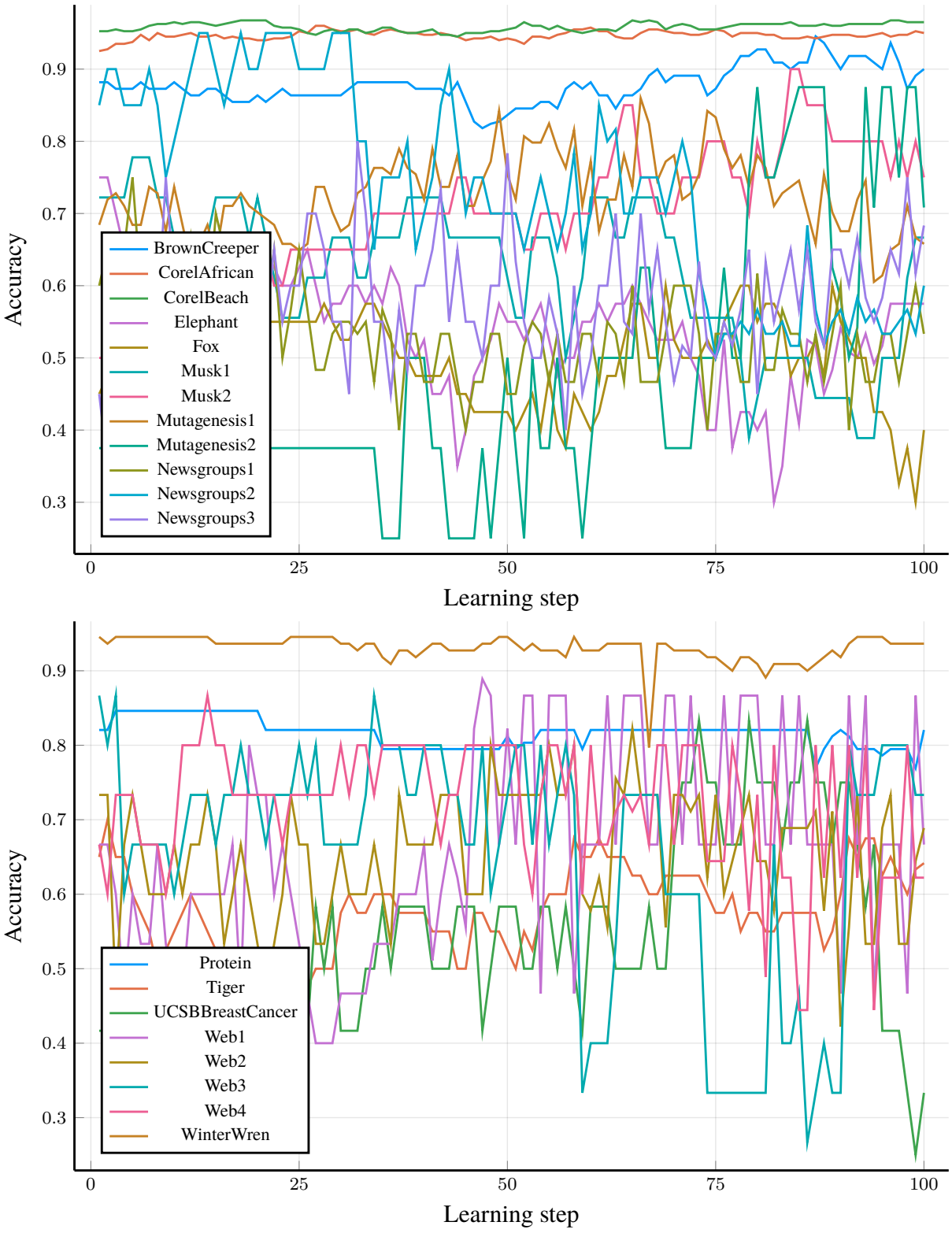


Figure A.3: The accuracy of a kNN classifier built on the embedding for L_{CPC} over the learning period.

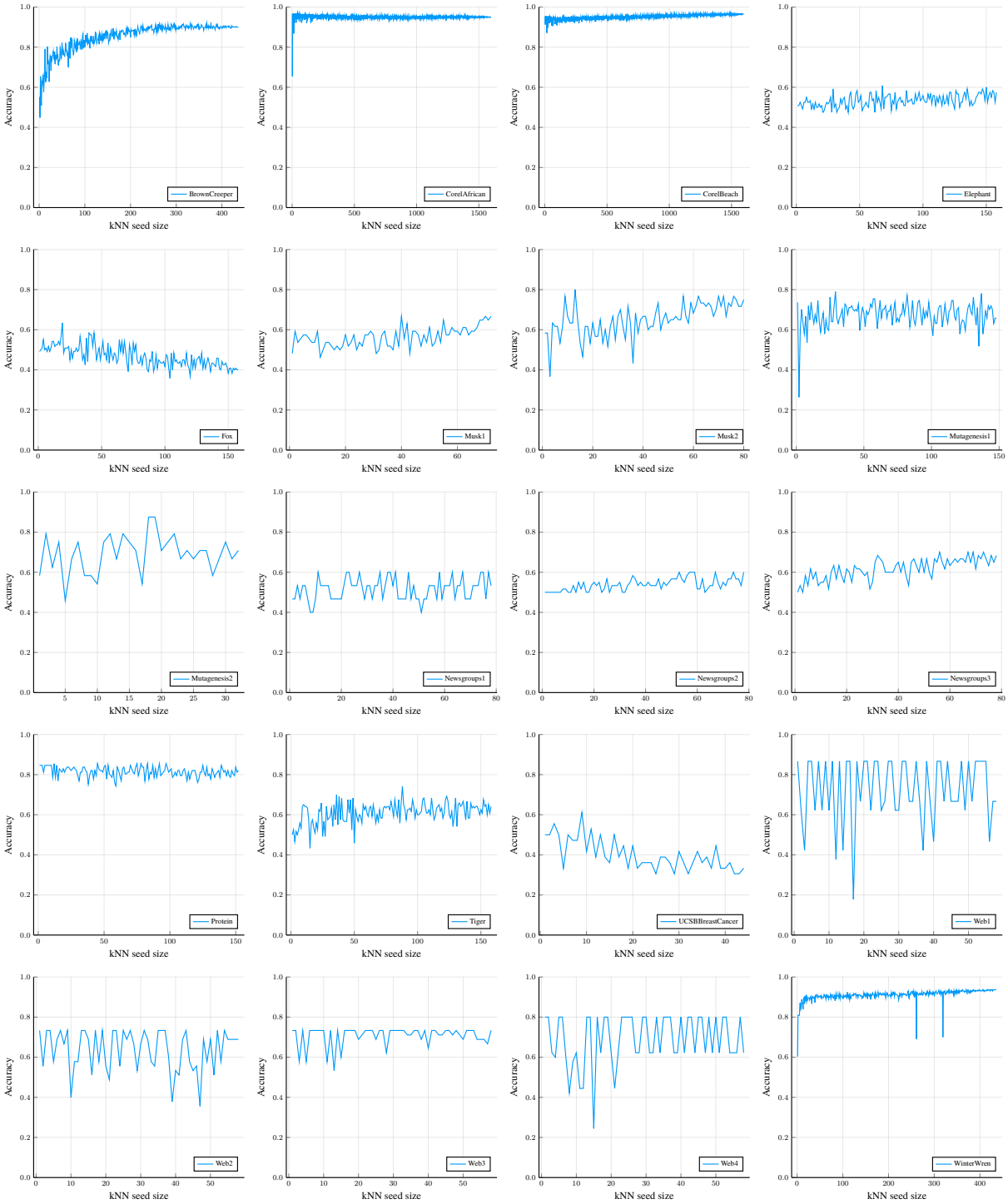


Figure A.4: The accuracy of a kNN classifier built on the final embedding for L_{CPC} as a function of the number of samples used to seed it. Average of three runs.

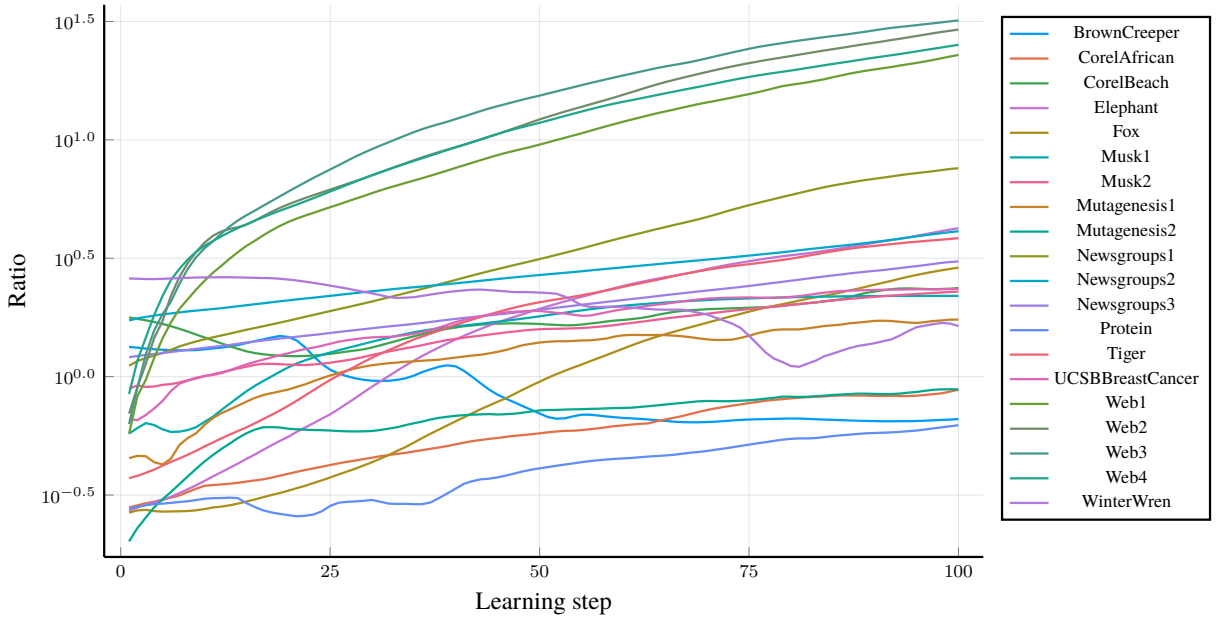


Figure A.5: The value of silhouette $(\cdot)$ for triplet loss over the learning period. Note the logarithmic scale on the y axis.

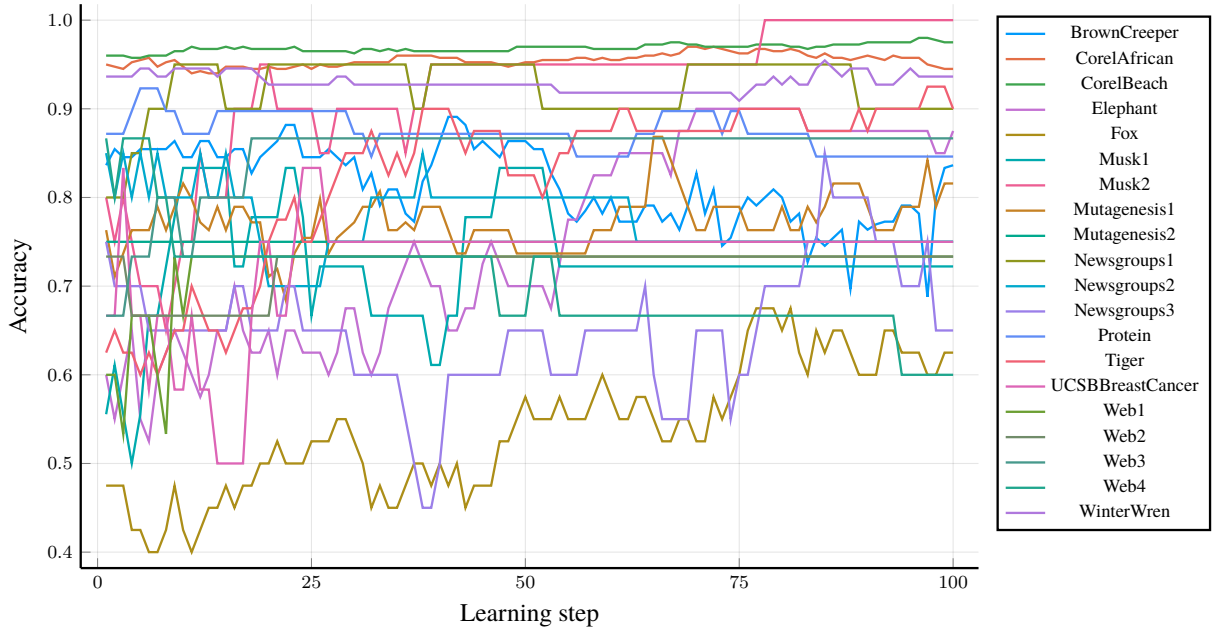


Figure A.6: The accuracy of a kNN classifier built on the embedding for triplet loss over the learning period.

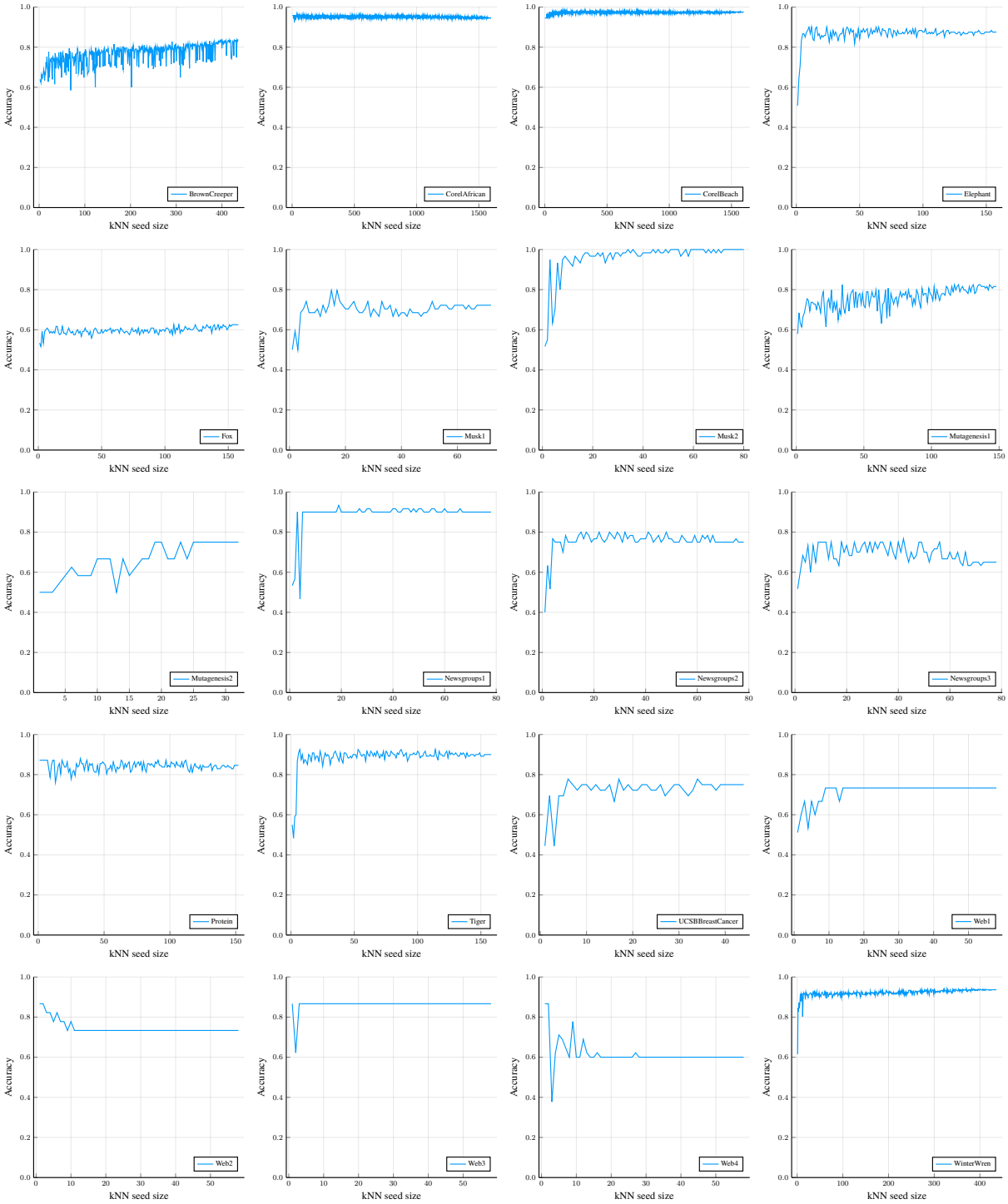


Figure A.7: The accuracy of a kNN classifier built on the final embedding for triplet loss as a function of the number of samples used to seed it. Average of three runs.

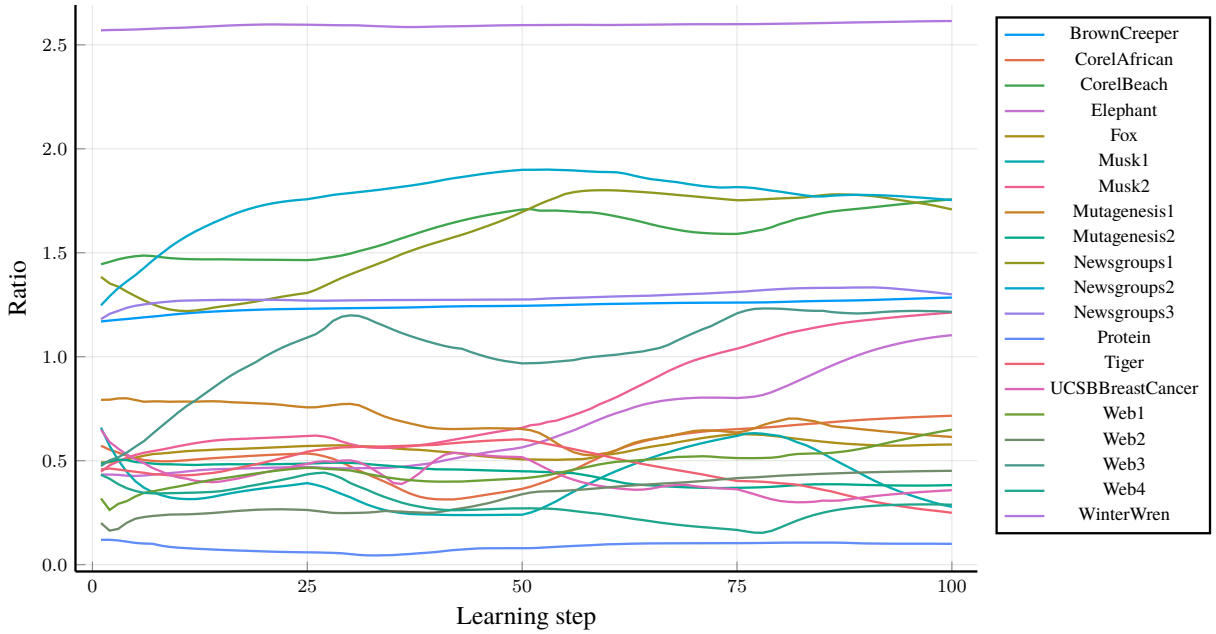


Figure A.8: The value of silhouette $(\cdot)$ for magnet loss over the learning period. Note the logarithmic scale on the y axis.

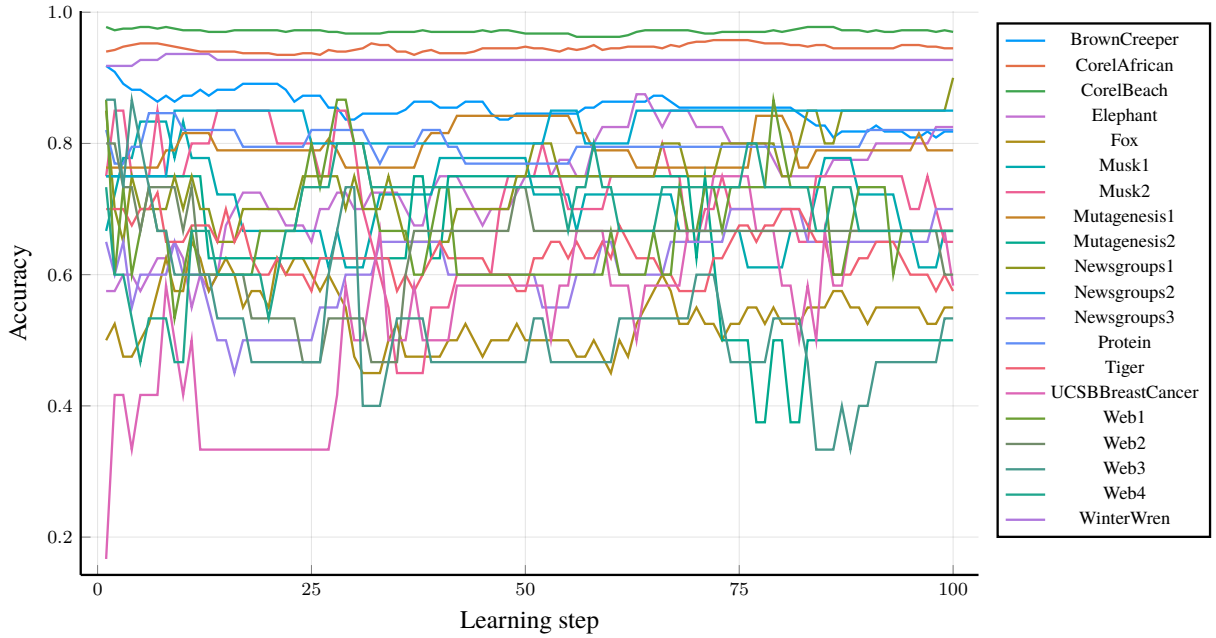


Figure A.9: The accuracy of a kNN classifier built on the embedding for magnet loss over the learning period.

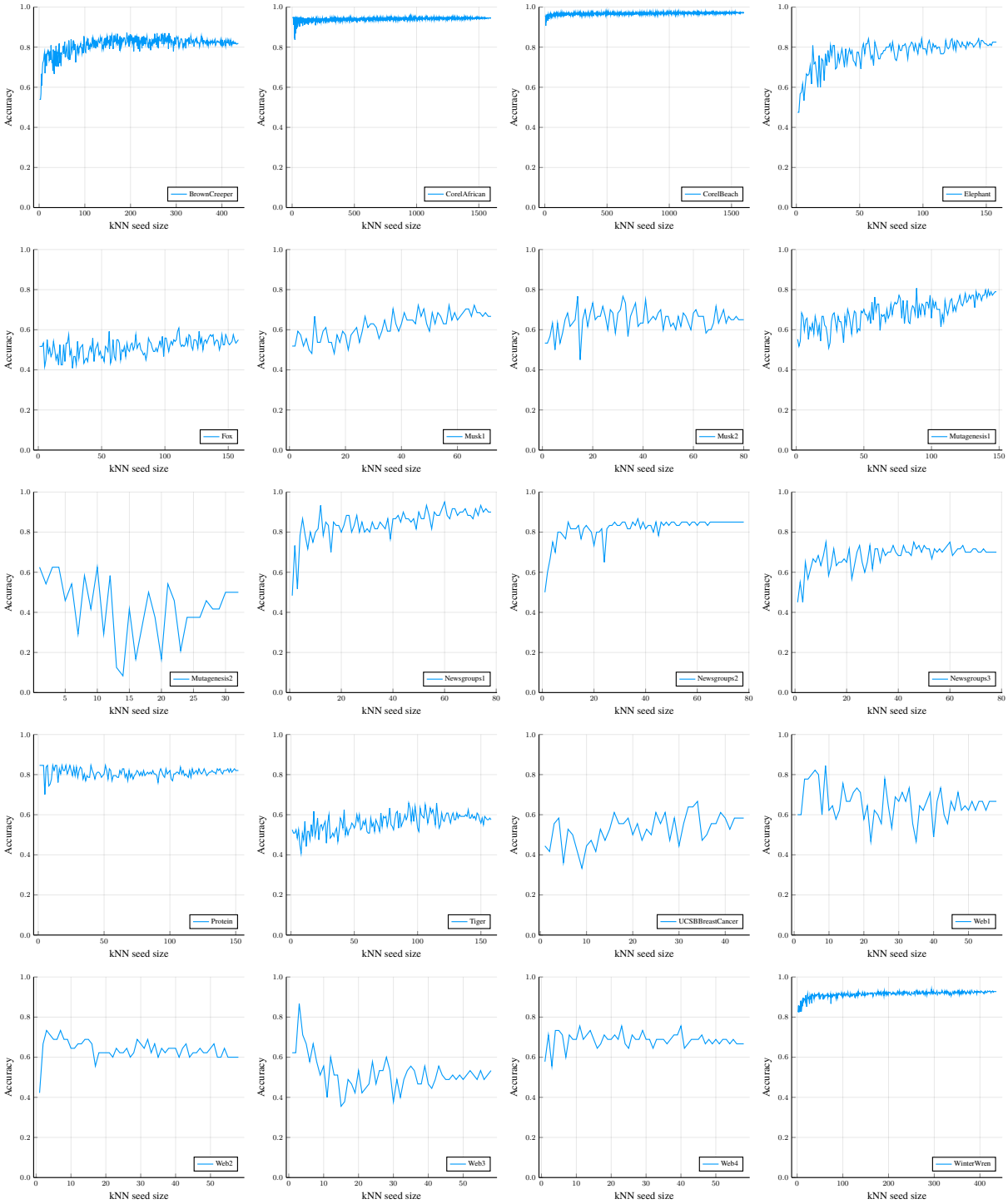


Figure A.10: The accuracy of a kNN classifier built on the final embedding for magnet loss as a function of the number of samples used to seed it. Average of three runs.

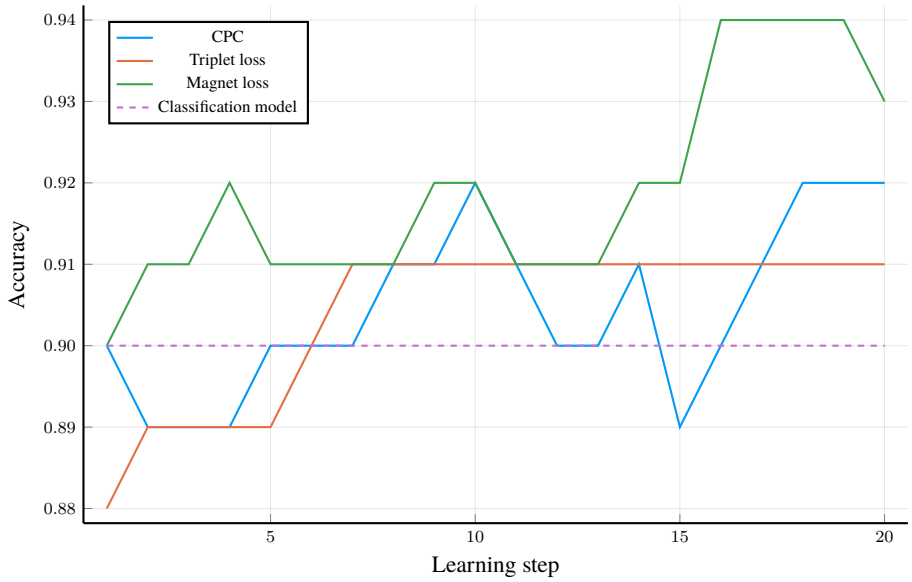


Figure A.11: The accuracy of a kNN classifier built on the final embedding over the learning period. Average of three runs on the corporate dataset.

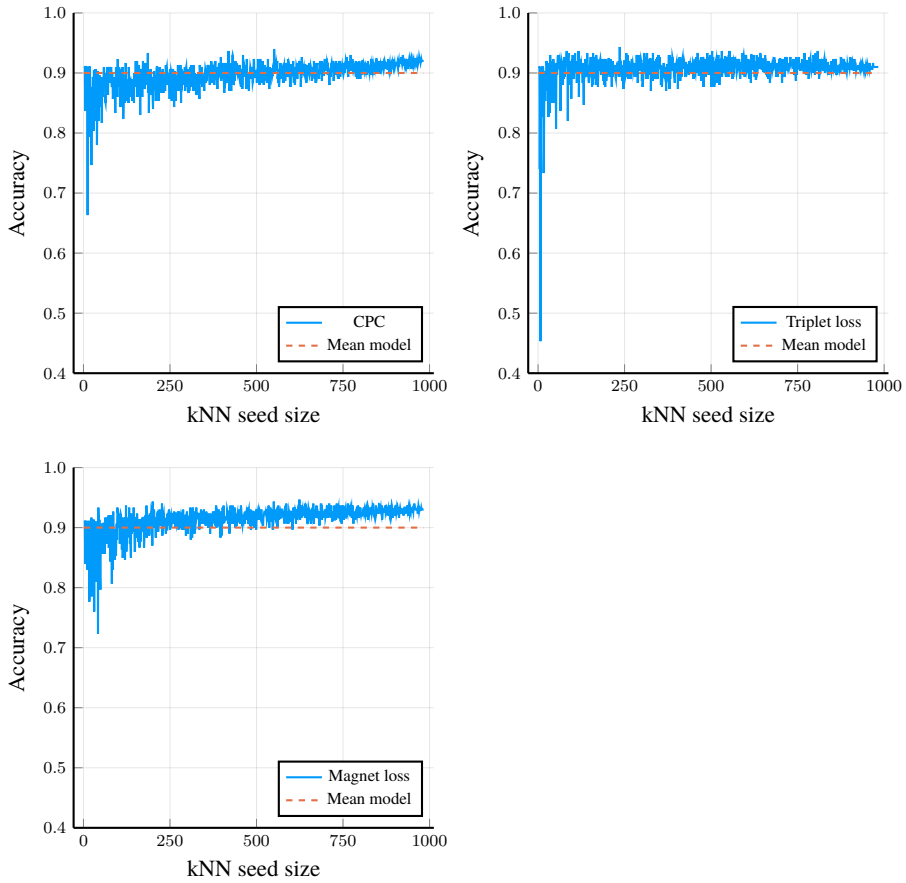


Figure A.12: The accuracy of a kNN classifier built on the final embedding as a function of the number of samples used to seed it. Average of three runs on the corporate dataset.

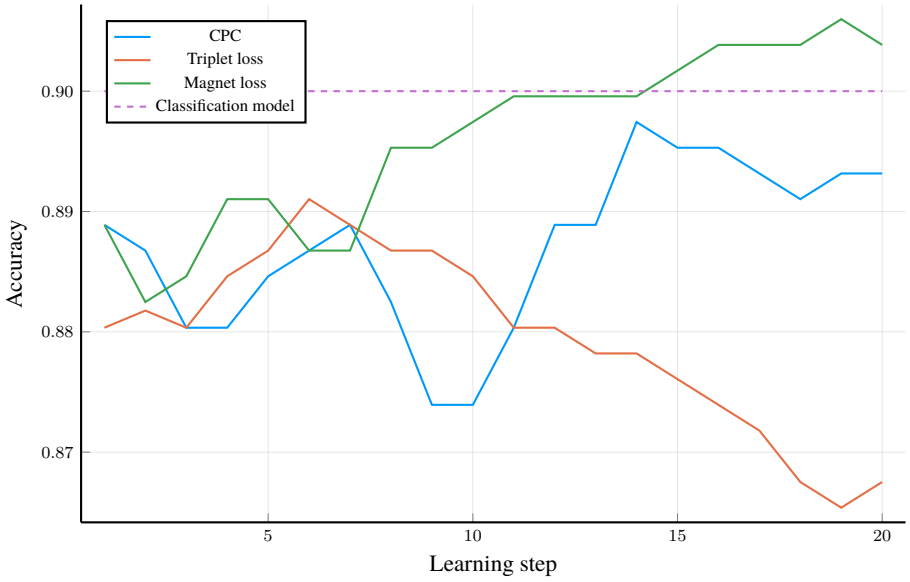


Figure A.13: The accuracy of a kNN classifier built on the final embedding over the learning period. Average of three runs on the corporate dataset with 20 classes.

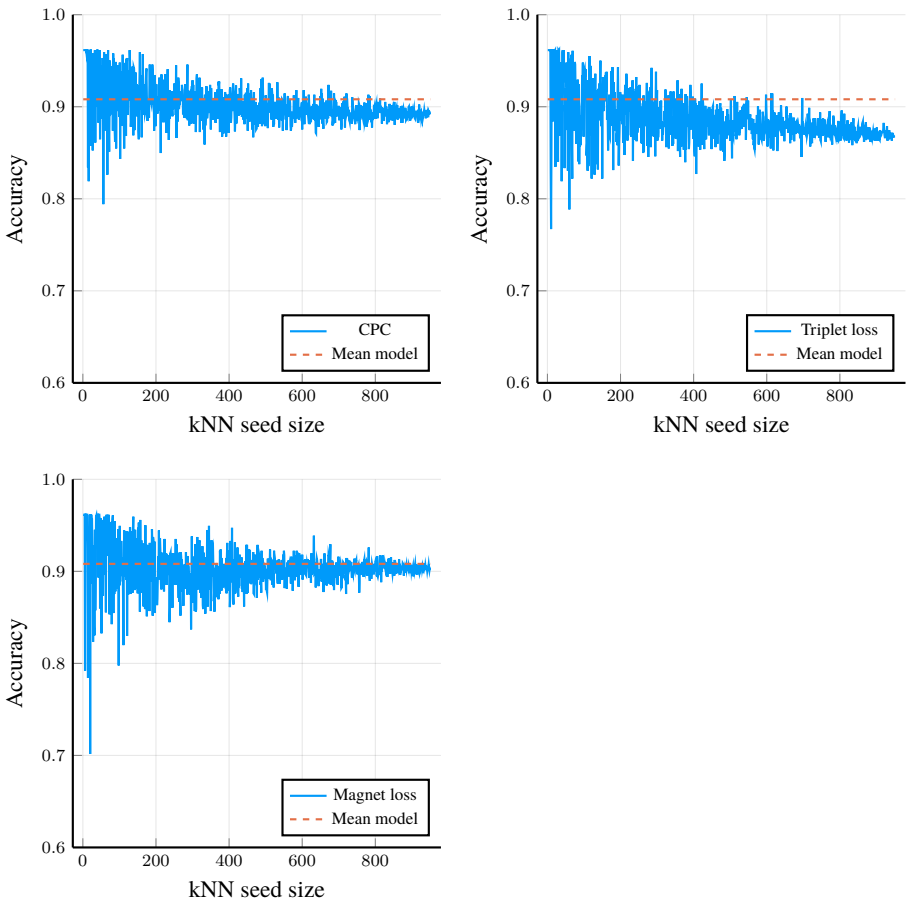


Figure A.14: The accuracy of a kNN classifier built on the final embedding as a function of the number of samples used to seed it. Average of three runs on the corporate dataset with 20 classes.