NORTHWESTERN UNIVERSITY

Topics in Classification with Deep Learning

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Statistics

By

Yiming Xu

EVANSTON, ILLINOIS

December 2020

ABSTRACT

Topics in Classification with Deep Learning

Yiming Xu

The task of classification has been increasingly attracting attention from researchers in recent years. The objective is to assign labels given attributes of samples. The classification task is practical in real-world applications and is widely explored in fields such as computer vision, natural language processing and information retrieval. The recent advances of deep learning techniques provide many efficient solutions to the classification task. This dissertation includes four chapters: 1) k-Nearest Neighbors by Means of Sequence to Sequence Deep Neural Networks and Memory Networks, 2) Automatic Ontology Learning from Domain-Specific Short Unstructured Text Data, 3) Concept Drift and Covariate Shift Detection Ensemble with Lagged Labels and 4) Open Set Domain Adaptation by Extreme Value Theory.

In the first chapter, we mimic the k-Nearest Neighbors method by two families of deep networks. k-Nearest Neighbors is one of the most fundamental but effective classification models. We propose two families of models built on a sequence to sequence model and a memory network model to mimic the k-Nearest Neighbors model, which generate a

sequence of labels, a sequence of out-of-sample feature vectors and a final label for classification, and thus they could also function as oversamplers. We also propose 'out-of-core' versions of our models which assume that only a small portion of data can be loaded into memory.

In the second chapter, we provide an efficient and effective way to automatic ontology learning. Ontology learning is a critical task in industry, which deals with identifying and extracting concepts reported in text such that these concepts can be used in different tasks, e.g. information retrieval. The problem of ontology learning is non-trivial due to several reasons with a limited amount of prior research work that automatically learns a domain specific ontology from data. We propose a two-stage classification system to automatically learn an ontology from unstructured text. In the proposed model, the first-stage classifier classifies candidate concepts into relevant and irrelevant concepts and then the second-stage classifier assigns specific classes to the relevant concepts. The proposed system is deployed as a prototype in General Motors and its performance is validated by using complaint and repair verbatim data collected from different data sources.

In the third chapter, we propose a drift detection ensemble to detect concept drifts and covariate shifts and automatically select the retraining data. In model serving, having one fixed model during the entire often lift-long inference process is usually detrimental to model performance, as data distribution evolves over time, resulting in lack of reliability of the model trained on historical data. It is important to detect changes and retrain the model in time. The existing methods generally have three weaknesses: 1) using only classification error rate as signal, 2) assuming ground truth labels are immediately available after features from samples are received and 3) unable to decide what data to

use to retrain the model when change occurs. We address the first problem by utilizing six different signals to capture a wide range of characteristics of data, and we address the second problem by allowing lag of labels, where labels of corresponding features are received after a lag in time. For the third problem, our proposed method can automatically decide what data used to retrain to use on the signals.

In the fourth chapter, we solve the problem of open set domain adaptation by utilizing extreme value theory. Common domain adaptation techniques assume that the source domain and the target domain share an identical label space, which is not practical since when target samples are unlabeled we have no knowledge on whether two domains share the same label space. When the assumption is not satisfied, such methods fail to perform well because the additional unknown classes are also matched with the source domain during adaptation. In this chapter, we tackle the open set domain adaptation problem which assumes the source and the target label spaces only partially overlap, and the task becomes when the unknown classes exist, how to detect the target unknown classes and avoid aligning them with the source domain. We propose to 1) utilize an instance-level reweighting strategy for domain adaptation where the weights indicate the likelihood of a sample belonging to known classes and 2) model the tail of entropy distribution with Extreme Value Theory for unknown classes detection.

Acknowledgements

First of all, I would like to sincerely thank my primary advisor Professor Diego Klabjan. The completion of my dissertation would not have been possible without his nurturing. He is one of the smartest and most experienced person I know in the deep learning field, and he is also very humorous and meetings with him are never boring - I always enjoy his dry jokes. His immense knowledge, insight, encouragement and sense of humour have guided me through my PhD journey. I will forever be thankful for his support and guidance.

Apart from Professor Diego Klabjan, I would like to also express my gratitude to my other advisor Professor Wenxin Jiang. He has provided me with his invaluable knowledge and insight in the statistics field. The discussions and his advice are instrumental in my research.

I am also pleased to say thank you to Professor Hongmei Jiang, who is also my PhD thesis committee member. Her constructive comments mean a lot to me.

I would also like to especially thank Dr. Lin Chen, Prof. Jiebo Luo and Dr. Dnyanesh Rajpathak, who are outside Northwestern University but have helped and guided me immensely in the field of machine learning and deep learning. Their guidance is hard to forget.

Lastly, I am grateful to my parents and friends for their endless support and love. Their support and encouragement have been unconditional through these years. You are always there for me, supporting me spiritually throughout my life.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

k-Nearest Neighbors by Means of Sequence to Sequence Deep Neural Networks and Memory Networks

1.1. Introduction

Recently, neural networks have been attracting a lot of attention among researchers in both academia and industry, due to their astounding performance in fields such as natural language processing and image recognition. Interpretability of these models, however, has always been an issue since it is difficult to understand the performance of neural networks. The well-known manifold hypothesis states that real-world high dimensional data (such as images) form lower-dimensional manifolds embedded in the high-dimensional space [**1**], but these manifolds are tangled together and are difficult to separate. The classification process is then equivalent to stretching, squishing and separating the tangled manifolds apart. However, these operations pose a challenge: it is quite implausible that only affine transformations followed by pointwise nonlinear activations are sufficient to project or embed data into representative manifolds that are easily separable by class.

Therefore, instead of asking neural networks to separate the manifolds by a hyperplane or a surface, it is more reasonable to require points of the same manifold to be closer than points of other manifolds [**2**]. Namely, the distance between manifolds of different classes should be large and the distance between manifolds of the same class should be small. This distance property is behind the concept of k-Nearest Neighbor (kNN) [**3**].

Consequently, letting neural networks mimic kNN would combine the notion of manifolds with the desired distance property.

We explore kNN through two deep neural network models: sequence to sequence deep neural networks [**4**] and memory networks [**5**] [**6**]. A family of our models are based on a sequence to sequence network. The new sequence to sequence model has the input sequence of length 1 corresponding to a sample, and then it decodes it to predict two sequences of output, which are the classes of closest samples and neighboring samples not necessarily in the training data, where we call the latter as out-of-sample feature vectors. We also propose a family of models built on a memory network, which has a memory that can be read and written to and is composed of a subset of training samples, with the goal of using it for predicting both classes of close samples and out-of-sample feature vectors. With the help of attention over memory vectors, our new memory network model generates the predicted label sequence and out-of-sample feature vectors. Both families of models use loss functions that mimic kNN. Computational experiments show that the new sequence to sequence kNN model consistently outperforms benchmarks (kNN [**3**], random forest [**7**], XGBoost [**8**], lightGBM [**9**], a feed-forward neural network and a vanilla memory network) on structured datasets. The performance on some commonly used image and text datasets is comparable to many state-of-the-art deep models. We postulate that this is due to the fact that we are forcing the model to 'work harder' than necessary (producing out-of-sample feature vectors).

Different from general classification models, our models predict not only labels, but also out-of-sample feature vectors. Usually a classification model only predicts labels, but as in the case of kNN, it is desirable to learn or predict the feature vectors of neighbors

as well. Intuitively, if a deep neural network predicts both labels and feature vectors, it is forced to learn and capture representative information of input, and thus it should perform better in classification. Moreover, our models also function as synthetic oversamplers: we add the out-of-sample feature vectors and their labels (synthetic samples) to the training set. Experiments show that our sequence to sequence kNN model outperforms Synthetic Minority Over-sampling Technique (SMOTE) [**10**] and Adaptive Synthetic sampling (ADASYN) [**11**] most of the times on imbalanced datasets.

Usually we allow models to perform kNN searching on the entire dataset, which we call the full versions of models, but kNN is computationally expensive on large datasets. We design an algorithm to resolve this and we test our models under such an 'out-of-core' setting: only a batch of data can be loaded into memory, i.e. kNN searching in the entire dataset is not allowed. For each such random batch, we compute the K closest samples with respect to the given training sample. We repeat this R times and find the closest K samples among these KR samples. These closest K samples provide the approximate label sequence and feature vector sequence to the training sample based on the kNN algorithm. Computational experiments show that sequence to sequence kNN models and memory network kNN models significantly outperform the kNN benchmark under the out-of-core setting.

Our main contributions are as follows. First, we develop two types of deep neural network models which mimic the kNN structure. Second, our models are able to predict both labels of closest samples and out-of-sample feature vectors at the same time: they are both classification models and oversamplers. Third, we establish the out-of-core version of models in the situation where not all data can be read into computer memory or kNN

cannot be run on the entire dataset. The full version of the sequence to sequence kNN models and the out-of-core version of both sequence to sequence kNN models and memory network kNN models outperform the benchmarks, which we postulate is because learning neighboring samples enables the model to capture representative features.

1.2. Background and Literature Review

There are several works trying to mimic kNN or applying kNN within different models. [**12**] introduced the boundary forest algorithm which can be used for nearest neighbor retrieval. Based on the boundary forest model, in [**13**], a boundary deep learning tree model with differentiable loss function was presented to learn an efficient representation for kNN. The main differences between this work and our work are in the base models used (boundary tree vs standard kNN), in the main objectives (representation learning vs classification and oversampling) and in the loss functions (KL divergence vs KL divergence components reflecting the kNN strategy and L^2 norm). [**14**] introduced a text classification model which utilized nearest neighbors of input text as the external memory to predict the class of input text. Our memory network kNN models differ from this model in 1) the external memory: our memory network kNN models simply feed a random batch of samples into the external memory without the requirement of nearest neighbors and thus they save computational time and 2) they considered only a classification setting, while our models generate not only labels but also out-of-sample feature vectors. Most importantly, the loss functions are different: in [**14**] the authors used KL divergence as the loss function while we use a specially designed KL divergence and L^2 norm to force our models to mimic kNN.

The sequence to sequence model, one of our base models, has recently become the leading framework in natural language processing [**4**] [**15**]. In [**15**] an RNN encoder-decoder architecture was used to deal with statistical machine translation problems. In [**4**] the authors proposed a general end-to-end sequence to sequence framework. The major difference between our work and these studies is that the loss function in our work forces the model to learn from neighboring samples, and our models are more than just classifiers - they also create out-of-sample feature vectors that improve accuracy or can be used as oversamplers.

There are also a plethora of studies utilizing external memory in neural networks. [**5**] proposed the memory network model to predict the correct answer of a query by means of ranking the importance of sentences in the external memory. [**6**] introduced a continuous version of a memory network with a recurrent attention mechanism over an external memory, which outperformed the previous discrete memory network architecture in question answering.

In summary, the main differences between our work and previous studies are as follows. First, our models predict both labels of nearest samples and out-of-sample feature vectors rather than simply labels. Thus, they are more than classifiers: the predicted label sequences and feature vector sequences can be treated as synthetic oversamples to handle imbalanced class problems. Second, our work emphasizes on the out-of-core setting. All of the prior works related to kNN and deep learning assume that kNN can be run on the entire dataset and thus cannot be used on large datasets. Third, our loss functions are designed to mimic kNN, so that our models are forced to learn neighboring samples to capture the representative information.

1.2.1. Sequence to Sequence model

A family of our models are built on sequence to sequence models. A sequence to sequence (Seq2seq) model is an encoder-decoder model. The encoder encodes the input sequence to an internal representation called the 'context vector' which is used by the decoder to generate the output sequence. Usually, each cell in the Seq2seq model is a Long Short-Term Memory (LSTM) cell [**16**] or a Gated Recurrent Unit (GRU) cell [**15**].

Given input sequence $x_1, ..., x_T$, in order to predict output $Y_1^P, ..., Y_K^P$ (where the superscript P denotes 'predicted'), the Seq2seq model estimates conditional probability $P(Y_1^P, ..., Y_t^P | x_1, ..., x_T)$ for $1 \leq t \leq K$. At each time step t, the encoder updates the hidden state h_t^e, which can also include the cell state, by $h_t^e = f_h^e(x_t, h_{t-1}^e)$, where $1 \leq t \leq T$. The decoder updates the hidden state h_t^d by $h_t^d = f_h^d(Y_{t-1}^P, h_{t-1}^d, h_T^e)$, where $1 \leq t \leq K$. The decoder generates output y_t by

$$y_t = g(Y_{t-1}^P, h_t^d, h_T^e), \tag{1.1}$$

and $Y_t^P = q(y_t)$ with q usually being softmax function.

The model calculates the conditional probability of output $Y_1^P, ..., Y_K^P$ by

$\Pr(Y_1^P, ..., Y_K^P | x_1, ..., x_T) = \prod_{t=1}^K \Pr(Y_t^P | Y_1^P, ..., Y_{t-1}^P)$.

1.2.2. End to End Memory Networks

The other family of our models are built on an end-to-end memory network (MemN2N). This model takes $x_1, ..., x_n$ as the external memory, a 'query' x, a ground truth Y^{GT} and predicts an answer Y^P. It first embeds memory vectors $x_1, ..., x_n$ and query x into

continuous space. They are then processed through multiple layers to generate the output label Y^P.

MemN2N has K layers. In the t^{th} layer, where $1 \leq t \leq K$, the external memory is converted into embedded memory vectors $m_1^t, ..., m_n^t$ by an embedding matrix A^t. The query x is also embedded as u^t by an embedding matrix B^t. The attention scores between embedded query u^t and memory vectors $(m_i^t)_{i=1,2,...,n}$ are calculated by $p^t = softmax((u^t)^T m_1^t, (u^t)^T m_2^t, ..., (u^t)^T m_n^t)$.

Each x_i is also embedded to an output representation c_i^t by another embedding matrix C^t. The output vector from the external memory is defined as $o^t = \sum_{i=1}^{n} p_i^t c_i^t$. By a linear mapping H, the input to the next layer is calculated by $u^{t+1} = Hu^t + o^t$.

In the last layer, by another embedding matrix W, MemN2N generates a label for the query x by $Y^P = softmax(W(Hu^K + o^K))$.

1.3. kNN models

Our sequence to sequence kNN models are built on a Seq2seq model, and our memory network kNN models are built on a MemN2N model. Let K denote the number of neighbors of interest.

1.3.1. Vector to Label Sequence (V2LS) Model

Given an input feature vector x, a ground truth label Y^{GT} (a single class corresponding to x) and a sequence of labels $Y_1^T, Y_2^T, ..., Y_K^T$ corresponding to the labels of the $1^{st}, 2^{nd}, ..., K^{th}$ nearest sample to x in the entire training set, V2LS predicts a label Y^P and $Y_1^P, Y_2^P, ..., Y_K^P$,

the predicted labels of the $1^{st}, 2^{nd}, ..., K^{th}$ nearest samples. Since $Y_1^T, Y_2^T, ..., Y_K^T$ are obtained by using kNN upfront, the real input is only x and Y^{GT}. When kNN does not misclassify, Y^{GT} corresponds to majority voting of $Y_1^T, Y_2^T, ..., Y_K^T$.

The key concept of our model is to have x as the input sequence (of length 1) and the output sequence $Y_1^P, Y_2^P, ..., Y_K^P$ to correspond to $Y_1^T, Y_2^T, ..., Y_K^T$. The loss function also captures Y^{GT} and $Y_1^T, Y_2^T, ..., Y_K^T$.

In the V2LS model, by a softmax operation with temperature after a linear mapping (W_y, b_y), the label of the t^{th} nearest sample to x is predicted by $Y_t^P = softmax((W_y y_t + b_y)/\tau)$, where y^t is as in (1) for $t = 1, 2, ..., K$ and τ is the temperature of softmax.

By taking the average of predicted label distributions, the label of x is predicted by $Y^P = \sum_{t=1}^{K} Y_t^P/K$. Note that if Y_t^P corresponds to a Dirac distribution for each t, then Y^P matches majority voting. Temperature τ controls the "peakedness" of Y_t^P. Values of τ below 1 push Y_t^P towards a Dirac distribution, which is desired in order to mimic kNN [**17**] [**18**].

We design the loss function as $L_1 = \mathbb{E}\{\sum_{t=1}^{K} D_{KL}(Y_t^T||Y_t^P)/K + \alpha D_{KL}(Y^{GT}||Y^P)\}$, where the first term captures the label at the neighbor level, the second term for the actual ground truth and α is a hyperparameter to balance the two terms. The expectation is taken over all training samples, and D_{KL} denotes the Kullback-Leibler divergence. Due to the fact that the first term is the sum of KL divergence between predicted labels of nearest neighbors and target labels of nearest neighbors, it forces the model to learn information about the neighborhood. The second term considers the actual ground truth label: a classification model should minimize the KL divergence between the predicted label (average of K distributions) and the ground truth label. By combining the two terms,

the model is forced to not only learn the classes of the final label but also the labels of nearest neighbors. We let the t^{th} decoder cell predict the t^{th} nearest sample because the preceding decoder cells preserve closeness to the original input. In the subsequent cells, the information gets passed through more decoder cells and thus it is expected to deviate more from the input, which is why we let the t^{th} cell predict the t^{th} nearest neighbor.

In inference, given an input x, V2LS predicts Y^P and $Y_1^P, Y_2^P, ..., Y_K^P$, but only Y^P is the actual output; it is used to measure the classification performance. Note that it is possible that $argmax Y^P$ is different from the majority voted class among $argmax Y_1^P, argmax Y_2^P, ...,$ $argmax Y_K^P$ when kNN misclassifies.

1.3.2. Vector to Vector Sequence (V2VS) Model

We use the same structure as the V2LS model except that in this model, the inputs are a feature vector x and a sequence of feature vectors $X_1^T, X_2^T, ..., X_K^T$ corresponding to the $1^{st}, 2^{nd}, ..., K^{th}$ nearest sample to x among the entire training set (calculated upfront using kNN). V2VS predicts $X_1^P, X_2^P, ..., X_K^P$ which denote the predicted out-of-sample feature vectors of the $1^{st}, 2^{nd}, ..., K^{th}$ nearest sample. Since $X_1^T, X_2^T, ..., X_K^T$ are obtained using kNN, this is an unsupervised model.

The output of the t^{th} decoder cell y_t is processed by a linear layer (W_{x1}, b_{x1}), a $ReLU$ operation and another linear layer (W_{x2}, b_{x2}) to predict the out-of-sample feature vector $X_t^P = W_{x2} max\{W_{x1} y_t + b_{x1}, 0\} + b_{x2}, t = 1, 2, ..., K$. Numerical experiments show that $ReLU$ works best compared with $tanh$ and other activation functions.

The loss function is defined to be the sum of L^2 norms: $L_2 = \mathbb{E}\{\sum_{t=1}^{K} ||X_t^P - X_t^T||^2\}$. Since the predicted out-of-sample feature vectors should be close to the input vector,

learning nearest vectors forces the model to learn a sequence of approximations to something very close to the identity function. However, this is not trivial. First it does not learn an exact identity function, since the output is a sequence of nearest neighbors to input, i.e. it does not simply copy the input K times. Second, by limiting the number of hidden units of the neural network, the model is forced to capture the most representative and condensed information of input. A large amount of studies have shown this to be beneficial to classification problems [**19**] [**20**] [**21**].

In inference, we predict the label of x by finding the labels of out-of-sample feature vectors X_t^P and then perform majority voting among these K labels. The most voted label is regarded as the prediction of the current sample.

1.3.3. Vector to Vector Sequence and Label Sequence (V2VSLS) Model

In previous models, V2LS learns to predict labels of nearest neighbors and V2VS learns to predict feature vectors of nearest neighbors. Combining V2LS and V2VS together, this model predicts both X_t^P and Y_t^P. Given an input feature vector x, a ground truth label Y^{GT}, a sequence of nearest labels $Y_1^T, Y_2^T, ..., Y_K^T$ and a sequence of nearest feature vectors $X_1^T, X_2^T, ..., X_K^T$, V2VSLS predicts a label Y^P, a label sequence $Y_1^P, Y_2^P, ..., Y_K^P$ and an out-of-sample feature vector sequence $X_1^P, X_2^P, ..., X_K^P$. Since the two target sequences are obtained by kNN, the model still only needs x and Y^{GT} as input.

The loss function is a weighted sum of the two loss functions in V2LS and V2VS: $L = L_1 + \lambda L_2$, where λ is a hyperparameter to account for the scale of the L^2 norm and the KL divergence.

The L^2 norm part enables the model to learn neighboring vectors. As discussed in the V2VS model, this is beneficial to classification since it drives the model to capture representative information of input and nearest neighbors. The KL part of the loss function focuses on predicting labels of nearest neighbors. As discussed in the V2LS model, the two terms in the KL loss force the model to learn both neighboring labels and the ground truth label. Combining the two parts, the V2VSLS model is able to predict nearest labels and out-of-sample feature vectors, as well as one final label for classification. The model is structured in this way because the K^{th} decoder cell output corresponds to the K^{th} nearest neighbor, so that the model is forced to better mimic kNN.

1.3.4. Memory Network - kNN (MNkNN) Model

The MNkNN model is built on the MemN2N model, which has K layers stacked together. After these layers, the MemN2N model generates a prediction. In order to mimic kNN, our MNkNN model has K layers as well but it generates one label after each layer, i.e. after the t^{th} layer, it predicts the label of the t^{th} nearest sample. Similar to the Seq2seq kNN models, the t^{th} layer predicts the t^{th} nearest sample because the preceding layers preserve closeness to the input. Therefore, we let the preceding layers predict the closest nearest neighbors to mimic kNN.

This model takes a feature vector x, its ground truth label Y^{GT}, a random subset $x_1, x_2, ..., x_n$ from the training set (to be stored in the external memory) and $Y_1^T, Y_2^T, ..., Y_K^T$ denoting the labels of the $1^{st}, 2^{nd}, ..., K^{th}$ nearest samples to x among the entire training set (calculated upfront using kNN). It predicts a label Y^P and a sequence of K labels of closest samples $Y_1^P, Y_2^P, ..., Y_K^P$.

After the t^{th} layer, by a softmax operation with temperature after a linear mapping (W_y, b_y), the model predicts the label of t^{th} nearest sample by $Y_t^P = softmax((W_y(Hu^t + o^t) + b_y)/\tau)$ where $t = 1, 2, ..., K$. The final label of x is calculated by $Y^P = \sum_{t=1}^{K} Y_t^P / K$.

The loss function of MNkNN is: $\overline{L_1} = \mathbb{E}\{\sum_{t=1}^{K} KL(Y_t^T || Y_t^P)/K + \alpha KL(Y^{GT} || Y^P)\}$ which is the same as in V2LS. The first term accounts for learning neighboring information, and the second term forces the model to provide the best single candidate class.

In inference, the model takes a query x and random samples $x_1, x_2, ..., x_n$ from the training set, and generates the predicted label Y^P as well as a sequence of nearest labels $Y_1^P, Y_2^P, ..., Y_K^P$.

1.3.5. Memory Network - kNN with Vector Sequence (MNkNN_VEC) Model

This model is built on MNkNN, but it predicts out-of-sample feature vectors X_t^P as well. MNkNN_VEC takes a query feature vector x, its corresponding ground truth label Y^{GT}, a random subset $x_1, x_2, ..., x_n$ from the training dataset (to be stored in the external memory), $Y_1^T, Y_2^T, ..., Y_K^T$ and $X_1^T, X_2^T, ..., X_K^T$ denoting labels and feature vectors of the $1^{st}, 2^{nd}, ..., K^{th}$ nearest samples to x among the entire training set (calculated both upfront using kNN). MNkNN_VEC predicts a label Y^P, a sequence of labels $Y_1^P, Y_2^P, ..., Y_K^P$ and a sequence of out-of-sample feature vectors $X_1^P, X_2^P, ..., X_K^P$.

By a linear mapping T, a $ReLU$ operation and another linear mapping (W_x, b_x), the feature vectors are then calculated by $X_t^P = W_x max\{T(Hu^t + o^t), 0\} + b_x$.

Same as the V2VSLS model, combining the L^2 norm and the KL divergence together, the loss function is defined as $\overline{L} = \overline{L_1} + \lambda\, \mathbb{E}\{\sum_{t=1}^{K} ||X_t^P - X_t^T||^2\}$.

1.3.6. Out-of-Core Models

In the models exhibited so far, we assume that kNN can be run on the entire dataset exactly to compute the K nearest feature vectors and corresponding labels to an input sample. However, there are two problems with this assumption. First, this can be very computationally expensive if the dataset size is large. Second, the training dataset might be too big to fit in memory. When either of these two challenges is present, an out-of-core model assuming it is infeasible to run a 'full' kNN on the entire dataset has to be invoked. The out-of-core models avoid running kNN on the entire dataset, and thus save computational time and resources.

Let B be the maximum number of samples that can be stored in memory, where $B > K$. For a training sample x, we sample a subset S from the training set (including x) where $|S| = B$, then we run kNN on S to obtain the K nearest feature vectors and corresponding labels to x, which are denoted as $Y^T(S) = \{Y_1^T, Y_2^T, ..., Y_K^T\}$ and $X^T(S) = \{X_1^T, X_2^T, ..., X_K^T\}$ for x in the training process. The previously introduced loss functions L and $\overline{L}$ depend on $x, Y^{GT}, X^T(S), Y^T(S)$ and the model parameters Θ, and thus our out-of-core models are to solve

$$\min_{\Theta} \mathbb{E}_x \, \mathbb{E}_S \{\widetilde{L}(x, Y^{GT}, X^T(S), Y^T(S), \Theta)\}$$

where $\widetilde{L}$ is either L or $\overline{L}$.

Sampling a set of size B and then finding the nearest K samples only once are insufficient on imbalanced datasets, due to the low selection probability for minor classes. To resolve this, we iteratively take R random batches: each time a random batch is taken, we

update the closest samples $X^T(S)$ by the K closest samples among the current batch and the K previous closest samples. These resulting nearest feature vectors and corresponding labels are used in the loss function. Note that we allow the previously selected samples to be selected in later sampling iterations. The entire algorithm is exhibited in Algorithm 1.

ALGORITHM 1: Out-of-core framework

```
for epoch = 1,...,T do
    for training sample x do
        Let X^T = ∅;
        for r = 1 to R do
            Randomly draw B samples from training set;
            U = nearest K samples to x among the B samples;
            Let X^T be the nearest K samples to x among U ∪ X^T;
        end
        Update parameters by a gradient iteration:
         Θ^R= Θ^R − α∇L̃(x, Y^GT, X^T, Y^T, Θ^R);
    end
end
```

1.4. Computational Experiments

In this section, we evaluate our models on 9 classification datasets: Network Intrusion (NI) [**22**], Forest Covertype (COV) [**23**], SensIT [**24**], Credit Card Default (CCD) [**25**], MNIST [**26**], CIFAR-10 [**27**], News20 [**28**], IMDb [**29**] and Reuters [**30**], which are all publicly available. Among the 9 datasets, the first 4 are structured and the remaining are unstructured.

For each dataset we experiment with 5 different seeds and all reported numbers are averages taken over 5 random seeds. We discuss the performance of the models in two aspects: classification and oversampling.

1.4.1. Classification

Experimental Setup As comparisons against memory network kNN models and sequence to sequence kNN models, we use kNN with Euclidean metric and several currently best classification models random forest (RF), extreme gradient boosting (XGB), lightGBM (LGBM), a 4-layer feed-forward neural network (FFN) trained using the Adam optimization algorithm (which has been calibrated) with dropout and batch normalization and MemN2N (since MNkNN and MNkNN_VEC are built on MemN2N) as benchmarks. Value $K = 5$ is used in all models because it yields the best performance with low standard deviation among $K = 1, 2, ..., 20$. Increasing K beyond $K = 5$ is somewhat detrimental to the F-1 scores while significantly increasing the training time.

In the sequence to sequence kNN models, LSTM cells are used. In the memory network kNN models, the size of the external memory is 64 since we observe that models with memory vectors of size 64 generally provide the best F-1 scores with acceptable running time. Both sequence to sequence kNN models and memory network kNN models are trained using the Adam optimization algorithm with initial learning rate set to be 0.01. We also find that $\tau = 0.85$, $\lambda = 0.12$ and $\alpha = 9.5$ provide overall the best F-1 scores.

We first experiment on structured datasets not requiring special embeddings, i.e. NI, COV, SensIT and CCD. We only consider 3 classes in NI and COV datasets due to significant class imbalance.

Overall Results of Full Model on Structured Data We first discuss the full models that can handle all of the training data, i.e. kNN can be run on the entire dataset. Table 1.1 shows that in the full model case, V2VSLS consistently outperforms the best classification models on all four datasets. t-tests show that it significantly outperforms

benchmarks at the 5% level on all four datasets. For our kNN models, V2LS significantly outperforms V2VS, because V2VS tries to reconstruct the feature level information, which does not utilize the label information. Moreover, it can also be seen that predicting not only labels but feature vectors as well is reasonable, since V2VSLS consistently outperforms V2LS and MNkNN_VEC consistently outperforms MNkNN. Models predicting feature vectors outperform models not predicting feature vectors on all datasets. These memory based models exhibit subpar performance, which is expected since they only consider 64 training samples at once (despite using exact labels).

Table 1.1. F-1 score comparison of full models.

	NI	**COV**	**SensIT**	**CCD**	**MNIST**	**CIFAR-10**	**News20**	**IMDb**	**Reuters**
kNN	90.54	91.15	82.56	63.81	98.91	93.12	62.14	86.25	72.34
RF	90.44	93.76	82.70	66.94	98.87	92.69	58.55	87.41	73.70
XGB	87.53	91.98	82.56	66.95	99.12	91.55	69.81	88.51	74.24
LGBM	90.23	89.85	83.29	65.68	99.57	92.18	70.59	88.60	74.98
SVM	89.28	90.59	83.15	66.01	98.86	92.95	71.84	87.75	73.97
FFN	88.53	91.83	83.67	65.37	99.51	94.41	**72.93**	88.33	74.83
MemN2N	79.36	77.98	75.17	61.83	96.20	90.13	54.20	81.01	69.87
LambdaRank	45.58	59.81	41.03	38.91	70.18	62.59	49.71	65.97	41.69
kNN-AN	64.18	69.64	54.29	52.18	89.59	81.72	55.01	67.80	59.81
V2LS	91.28	93.94	84.93	68.38	99.51	94.18	72.39	87.71	75.77
V2VS	86.18	90.39	74.84	64.23	98.17	92.98	70.11	86.27	72.10
V2VSLS	**92.07**	**94.97**	**86.24**	**69.87**	**99.70**	**94.86**	72.68	**89.83**	**76.11**
MNkNN	83.83	80.12	79.58	67.26	94.38	89.10	62.33	84.18	69.28
MNkNN_VEC	84.59	83.94	83.41	68.82	97.29	93.02	71.54	87.85	74.17
Set-based model	91.25	94.10	85.51	68.77	99.51	93.39	70.88	88.19	74.91
Swapped V2VSLS	91.79	94.56	85.99	69.42	99.43	94.01	72.17	89.51	75.70

Overall Results of Full Model on Unstructured Data To provide insights of how our model performs on unstructured data, we further evaluate on the image and text datasets: MNIST, CIFAR-10, News20, IMDb and Reuters. The embeddings are fed into classifiers.

We compare V2VSLS with some of the most popular classification models in Table 1.1. On News20, 7-layer FFN performs slightly better than V2VSLS, but V2VSLS consistently outperforms other classification models on all other datasets. There is a slight gap attributed to the single stage employed by pure deep learning models v.s. our experiment that has two stages (embedding construction, kNN). Nevertheless, the performance of V2VSLS on these unstructured datasets still outperforms many currently popular models.

Comparison with a Set-Based Model and Swapped Order V2VSLS We evaluate the necessity of modeling nearest neighbors as a sequence, instead of as a set. First, we compare the set-based model with the V2VSLS model. Note that compared to V2VSLS, the set-based neural network model also predicts K labels, K nearest neighbors and a final label for classification, but it does not model the K labels and nearest neighbors as a sequence. The only difference is that the set-based model's outputs are orderless. As shown in Table 1.1, the set-based model which removes the order of nearest neighbors suffers from a consistent performance drop across datasets. The set-based model still outperforms most of the existing popular classification methods, however, which again validates that predicting nearest neighbors is beneficial for classification.

Following the orderless nearest neighbors experiment, we arbitrarily swap the first and the third nearest neighbor of the order in the training data. Intuitively, if the performance drops after swapping the nearest neighbors in the training data, utilizing the order information of nearest neighbors is crucial. The results are shown in Table 1.1. V2VSLS with swapped order performs worse than V2VSLS with original order, but it still outperforms the set-based model consistently, which validates that keeping the order of the nearest neighbors is necessary.

Comparison with other Related Benchmark Models We first compare V2VSLS with the popular ranking-based model LambdaRank [**31**] with lightGBM as the classifier. LambdaRank ranks the feature vectors in the training set for a given query by similarity. In inference, for a provided feature vector query, LambdaRank ranks the feature vectors in the training set and obtains the nearest $K = 5$ samples. Finally, the label of the query is obtained by majority voting among the corresponding labels of those nearest samples in the training set. The results are provided in Table 1.1. The performance improvement of approximately 40% is observed in the experiment, which shows the significant superiority of V2VSLS over the conventional ranking-based method.

To compare our work with another similar work which utilizes the nearest neighbors to make predictions, we implemented the kNN-Augmented Networks from [**14**]. The comparison is shown in Table 1.1. In the implementation of kNN-Augmented Networks, we have fine-tuned the hyper-parameters: $K = 5$, $I = 8$, learning rate=0.001 and the Adam optimizer. There is a substantial gap between our models and the kNN-Augmented Networks, that we were unable to close despite a significant effort to fine tune the hyper-parameters.

Overall Results of Out-of-Core Model Next, we validate our models in the out-of-core scenario which avoids running kNN on the entire dataset for saving computational resources. In the out-of-core versions of our models, R is set to be 50, since we observe that increasing 50 only has a slight impact on F-1 scores. However, this substantially increases the running time.

Table 1.2. F-1 score of out-of-core model with R=50.

	NI	**COV**	**SensIT**	**CCD**
kNN	73.87	63.87	61.40	59.41
V2LS	90.63	90.29	82.47	67.51
V2VS	81.92	71.29	69.12	61.36
V2VSLS	**91.27**	**92.89**	**83.38**	**69.21**
MNkNN	81.89	78.58	78.80	66.16
MNkNN_VEC	83.19	81.72	82.32	68.15

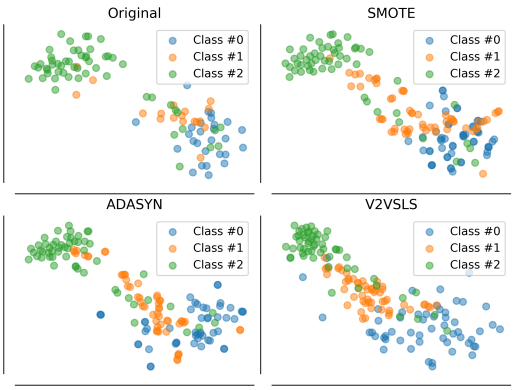

Figure 1.1. Visualization of oversampling methods.

Table 1.2 shows the results of our models under the out-of-core assumption when $R = 50$ and $B = 64$. Both V2VLSL and MNkNN_VEC significantly outperform the kNN benchmark based on t-tests at the 5% significance level. The kNN benchmark provides a low score since we restrict the batch size (or memory size) to be 64, and it turns out that kNN is substantially affected by the randomness of batches. Our models are robust under the out-of-core setting, because the weight of the ground truth label in the loss function is relatively high so that even if the input nearest sequences are noisy, they still can focus on learning the ground truth label and making reasonable predictions.

Table 1.3. Full model and out-of-core (OOC) model comparison on SensIT.

	kNN	**V2LS**	**V2VS**	**V2VSLS**	**MNkNN**	**MNkNN_VEC**
Full F-1	82.56	84.93	74.84	86.24	79.58	83.41
OOC F-1	61.40	82.47	69.12	83.38	78.80	82.32
Full time (s)	312	443+635	857+1358	1391+1802	443+692	1391+1081
OOC time (s)	193	287+619	488+1316	741+1846	287+703	741+1055

Full Model and Out-of-Core Model Comparison Table 1.3 shows a comparison between the full and out-of-core models with $R = 50, B = 64$ on the SensIT dataset. The running time of our models are broken down to two parts: the first part is the time to

obtain sequences of K nearest feature vectors and labels and the second part is the model training time. Under the out-of-core setting, overall the kNN sequence preprocessing time is saved by approximately 40% while the models perform only slightly worse.

1.4.2. Oversampling

Experimental Setup When class distributions are highly imbalanced, many classification models have low accuracy or F-1 score on the minority class. A simple but effective way to handle this problem is to oversample the minority class. Since V2VSLS and MNkNN_VEC are able to predict out-of-sample feature vectors, we also regard our models as oversamplers and we compare them with two widely used oversampling techniques: SMOTE and ADASYN. We only test V2VSLS since it scales better than MNkNN_VEC and the prediction performance is comparable. In our experiments, we evaluate our model on four imbalanced datasets. We first fully train the model, and then for each sample from the training set, V2VSLS predicts $K = 5$ out-of-sample feature vectors which are regarded as synthetic samples. We add them to the training set if they are in a minority class until the classes are balanced or there are no minority training data left for creating synthetic samples. In our oversampling experiments, we use $\lambda = 1.3$ and $\alpha = 3$.

Overall Results on Oversampling Table 1.4 shows the F-1 scores of FFN, extreme gradient boosting and random forest classification models, with different oversampling techniques, namely, original training set without oversampling, SMOTE, ADASYN and V2VSLS. V2VSLS performs the best among all combinations of classification models and oversampling techniques, as shown in Table 1.5. Although most of the time models on datasets with three oversampling techniques outperform models on datasets without

Table 1.4. Oversampling: F-1 score comparison.

	NI	**COV**	**SensIT**	**CCD**
FFN-original	89.64	91.83	83.67	65.37
FFN-SMOTE	89.99	91.18	83.43	66.32
FFN-ADASYN	90.38	90.67	83.72	66.51
FFN-V2VSLS	**90.89**	**92.05**	**83.94**	**66.82**
XGB-original	87.53	91.98	82.56	66.95
XGB-SMOTE	87.79	91.86	82.87	66.56
XGB-ADASYN	**88.39**	**92.56**	**83.42**	66.20
XGB-V2VSLS	87.62	92.43	82.46	**66.96**
RF-original	90.44	93.76	82.70	66.94
RF-SMOTE	89.97	93.88	83.01	66.13
RF-ADASYN	89.39	93.83	**83.34**	67.14
RF-V2VSLS	**90.79**	**94.36**	82.75	**68.08**

oversampling, the classification performance still largely depends on the classification model used and which dataset is considered.

Table 1.5. Oversampling techniques comparison.

	NI	**COV**	**SensIT**	**CCD**
Best model	FFN+V2VSLS	RF+V2VSLS	FFN+V2VSLS	RF+V2VSLS
Best F-1	90.89	94.36	83.92	68.08
Better than best SMOTE	0.51%	0.61%	2.28%	1%
Better than best ADASYN	0.6%	0.56%	0.26%	1.4%

Synthetic Samples Visualization Figure 1.1 shows a t-SNE [**32**] visualization of the original set and the oversampled set, using SensIT dataset, projected onto 2-D space. Although SMOTE and ADASYN overall perform well, their class boundaries are not as clean as those obtained by V2VSLS.

1.5. Conclusion

In summary, we find that it is beneficial to have models learn not only labels but also feature vectors. In our work, we develop two types of deep neural network models

mimicking kNN which are able to predict both the labels of closest samples and out-of-sample feature vectors at the same time. In experiments, our proposed models outperform the benchmark methods in both classification and oversampling tasks.

CHAPTER 2

Automatic Ontology Learning from Domain-Specific Short Unstructured Text Data

2.1. Introduction

Over 90% of organizational memory is captured in the form of unstructured as well as structured text. The unstructured text takes different forms in different industries, e.g. body of email messages, warranty repair verbatim, patient medical records, fault diagnosis reports, speech-to-text snippets, call center data, design and manufacturing data and social media data. Given the ubiquitous nature of unstructured text it provides a rich source of information to derive valuable business knowledge. For example, in an automotive (which is used as our running example) or aerospace industry in the event of fault or failure, repair documents (commonly referred to as verbatim) are captured [**33**]. These repair verbatims provide a valuable source of information to gain an insight into the nature of fault, symptoms observed along with fault, and corrective actions taken to fix the problem after systematic root cause investigation [**34**]. It is important to extract the knowledge from such verbatims to understand different ways by which the parts, components, modules and systems fail during their usage and under different operating conditions. Such knowledge can be used to improve the product quality and more importantly to ensure an avoidance of similar faults in the future. However, efficient and timely extraction, acquisition and formalization of knowledge from unstructured text

poses several challenges: 1. the overwhelming volume of unstructured text makes it difficult to manually extract relevant concepts embedded in the text, 2. the use of lean language and vocabulary results into an inconsistent semantics, e.g. 'vehicle' vs 'car,' or 'failing to work' vs. 'inoperative,' and finally, and 3. different types of noise are observed, e.g. misspellings, run-on words, additional white spaces and abbreviations.

An ontology [**35**] provides an explicit specification of concepts and resources associated with domain under consideration. A typical ontology (or a taxonomy) may consist of concepts and their attributes commonly observed in a domain, relations between the concepts, a hierarchical representation of concepts and concept instances representing ground-level objects. For example, the concept 'vehicle' can be used to formalize a locomotive object and vehicle instances, such as 'Chevrolet Equinox.' An ontological framework and the concept instances can be used to share the knowledge among different agents in a machine-readable format (e.g. RDF/S [1]) and in an unambiguous fashion. Hence, ontologies constitute a powerful way to formalize the domain knowledge to support different application, e.g. natural language processing [**36**] [**37**], information retrieval [**38**], information filtering [**39**], among others.

Given the overwhelming scale of data in the real world and an ever-changing competitive technology landscape, it is impractical to construct an ontology manually that scales at an industry level. To overcome this limitation, we propose an approach whereby an ontology is constructed by training a two-stage machine learning classification algorithm. The classifier to extract and classify key concepts from text consists of two stages: 1. in the first stage, a classifier is trained to classify the multi-gram concepts in a verbatim

[1]https://www.w3.org/TR/rdf-schema/

into relevant concepts and irrelevant concepts and 2. in the second stage, the relevant concepts are further classified into their specific classes. It is important to note that a concept can be a relevant in one verbatim, e.g. 'check engine light **is on**,' but irrelevant in another verbatim, e.g. 'vehicle **is on** the driveway.' Our input text corpus consists of short verbatims and the goal is to identify additional new concepts and classify them into their most appropriate classes. In the first-stage, our classifier takes as the input labelled training data consisting of n-grams generated from each verbatim and the label related to each n-gram, where n ranges from 1 to 4. The labelling process is performed manually and also by using an existing incomplete domain ontology. More specifically, if a concept is already covered in a domain ontology then its existing class is used as a label for a n-gram; otherwise a human reader provides a label. In our classification model, we use both linguistic features (e.g. part of speech (POS)), positional features (e.g. start and end index in verbatim, length of verbatim), and word embedding features (word2vec [**40**]). The problem of polysemy poses a significant challenge since they occur frequently in short text. In our approach, we introduce a new feature that handles the problem of polysemy as follows: Given a 1-gram, we cluster their embedding vectors with the number of clusters equal to the number of polysemy of a 1-gram based on WordNet [**41**] and then for an occurrence of a 1-gram, we use centroid of the closest cluster as a representative feature. For higher n-grams, e.g. 4-gram we observe limited positive samples in the training data and we perform two rounds of active learning to boost the number of positive samples. Our two-stage classification model is deployed as a proof-of-concept in General Motors and the experiments have shown it to be an effective approach to discover new concepts of high quality.

Through our work, we claim the following key contributions. 1. In real-world industry, data comes from disparate sources and therefore, the relevant concepts are heterogeneous both in terms of the lean language (e.g. 'unintended acceleration' and 'lurch forward') and distributions. We successfully identify collaborative, common set of features to train a machine learning classification model that classifies heterogeneous concepts with high accuracy. 2. The problem of polysemy, e.g. 'car **on** driveway' v.s. 'check engine light is **on**' is ubiquitous in our data. A new type of feature named polysemy centroid feature (discussed in section 2.4.3) is introduced, which handles the problem of polysemy in our data. 3. Abbreviations are common in real-world data and their disambiguation is important for the correct interpretation of data. We successfully disambiguate abbreviations and to the best of knowledge ours is the first proposal to disambiguate domain-specific abbreviations by combining a statistical and a machine learning model. 4. The proposed model is a practical system that is deployed as a tool in General Motors for an in-time augmentation of domain specific ontology. The system is scalable in nature and handles the industrial scale repair verbatim data.

The rest of the chapter is organized as follows. In the next section, we provide a review of the relevant literature. In Section 2.3, the problem description and an overview of our approach are discussed. In Section 2.4, we discuss data preprocessing algorithms that are used to clean the data and then discuss the process of feature engineering to identify key features that are used to train the classifiers. In Section 2.5, we discuss in detail the experiments and evaluation of our classification models. In Section 2.6, we conclude the chapter by reiterating the main contributions.

2.2. Background and Related Works

A plethora of works have been done in ontology learning [**42**]. There were three major approaches: statistical methods (e.g. weirdness, TF-IDF), machine learning methods (e.g. bagging, Naïve Bayes, HMM, SVM), and linguistic approaches (e.g. POS patterns, parsing, WordNet, discourage analysis).

[**43**] built an ontology learning system by collecting evidence from heterogeneous sources in a statistical approach. The candidate concepts were extracted and organized in the 'is-a' relations by using chi-squared co-occurrence significance score. In comparison with [**43**], we use a structured machine learning approach, which can be applied on unseen datasets. In [**43**], all evidence was integrated into a large semantic network and the spreading activation method was used to find most important candidate concepts. The candidate concepts are then manually evaluated before adding to an ontology. In comparison with this, in our approach the latent features, e.g. context features, polysemy features from the data are identified to train a machine learning classifier. Hence, it exploits richer data characteristics compared to [**43**]. Finally, ours is a probability based classifier and it can be applied to any new data to extract and classify important concepts effectively. The only manual intervention involved in our approach is to assign labels to the n-grams included in the training data. Finally, the model proposed by [**43**] is deterministic in nature and it does not consider the notion of context. Hence, it is very difficult to imagine how such model can be generalized to extract concepts specified in different context in the new data.

[**44**] makes use of the cosine similarity, TF-IDF, a C-value statistic, and POS to extract the candidate concepts to construct an ontology. This work was done in a statistical and

linguistic approach. The key difference between our work and the one proposed in [**44**], is ours is a principled machine learning model. It makes our system scalable to extract and classify multi-gram terms from industrial scale new data without manual intervention. The linguistic features, e.g. POS exploits syntactic information for better understanding text.

[**45**] constructs a hierarchical ontology by employing support vector machine (SVM). The SVM model heavily relies on the part-of-speech (POS) as the primary feature to determine classification hyperplane boundary. In comparison to [**45**], in our approach the POS is used as one of the features, but we also consider additional features, such as the context, polysemy, and word embedding to establish the context of a unigram or multi-gram concepts. Moreover, we also perform two rounds of active learning to further boost the classifier performance. As word embedding features are not considered by [**45**] it is difficult to envisage how the context associated with each concept was considered during their extraction.

[**46**] evaluates the effectiveness of word2vec features in ontology construction. The statistic based on 1-gram and 2-gram counts was used to extract the candidate concepts. However, the actual ontology was then constructed manually. In our work, we not only train a word2vec model to develop word embedding based context features, but other critical features, such as POS, polysemy features, etc. are also used to train a robust probabilistic machine learning model. The word embedding features included in our approach dominate statistical features and, therefore, other statistical features are not used in our approach.

[**47**] constructs an ontology by using the 'weirdness' statistic. The collocation analysis was performed along with domain expert verification process to construct a final ontology. There are two key differences between our approach and the one proposed by [**47**]. Firstly, in our approach the labelled training data along with different features as well as stop words are used to train a classification model, while in their approach the notions of 'weirdness' and 'peakedness' statistics are used to extract the candidate concepts. Secondly, in their work, there was a heavy reliance on domain experts to verify and curate newly constructed ontology. With our approach, no such manual intervention is needed during concept extraction stage or classification stage. Hence, our system can be deployed as a standalone tool to learn an ontology from an unseen data.

In our work, we also propose a new approach to disambiguate abbreviations. There are several related works. [**48**] extract features, such as concept unique identifiers and then built a classification model. [**49**] identify context based features to train a classifier, but they assumed an ambiguous phrase only with one correct expansion in the same article. [**50**] propose a word embedding based approach to select the expansion from all possible expansions with largest embedding similarity. There are two major differences between our approach and these works. First, we propose a new model that seamlessly combines the statistical approach (TF-IDF) with machine learning model (Naïve Bayes classifier). That is, we measure the importance of each concept in terms of TF-IDF and then estimate the posterior probability of each possible expansion. Alternate approaches either only apply machine learning model or simply calculate statistical similarity between abbreviation and possible expansions. Second, in these works a strong assumption is made that each abbreviation only has a single expansion in the same article and therefore the

features are conditionally independent. No such assumption is made in our approach and therefore it is more robust.

2.3. Problem Statement and Approach

In industry, data comes from several disparate sources and it can be useful in providing valuable information. However, given the overwhelming size of real-world data, manual ontology creation is impractical. Moreover, there are limited systems reported in literature that can be readily tuned to construct an ontology from the data related to different domains. In this work, we primarily focus on unstructured short verbatim text (commonly collected in automotive, aerospace, and other heavy equipment manufacturing industries).

This is a typical verbatim collected in automotive industry: *"Customer states the engine control light is illuminated on the dashboard. The dealer identified internal short to the fuel pump module relay and the fault code P0230 is read from the CAN bus. The fuel pump control module is replaced and reprogrammed. All the fault codes are cleared."* As shown in Figure 2.1, the domain model (classes and relations among them) of a specific domain, e.g. automotive, is designed by the domain experts, which have the common understanding of a domain. Our algorithm is trained by using a training dataset from a specific domain. In the first stage, the objective is to extract and classify all the relevant technical concepts reported in each verbatim, such as *'engine control light'*, *'fuel pump module relay'*, *'is illuminated'*, *'internal short'*, *'fuel pump control module'*, and *'replaced and reprogrammed'*. In the second stage, the relevant concepts are further classified into their specific classes. For instance, **part:** (engine control light, fuel pump module relay, fuel pump control module), **symptom:** (is illuminated, internal short), and **action:**

(replaced and reprogrammed). The classified technical concepts populates the domain model, which can be used within different applications, such as natural language processing, information retrieval, fault detection and root cause investigation, among others.

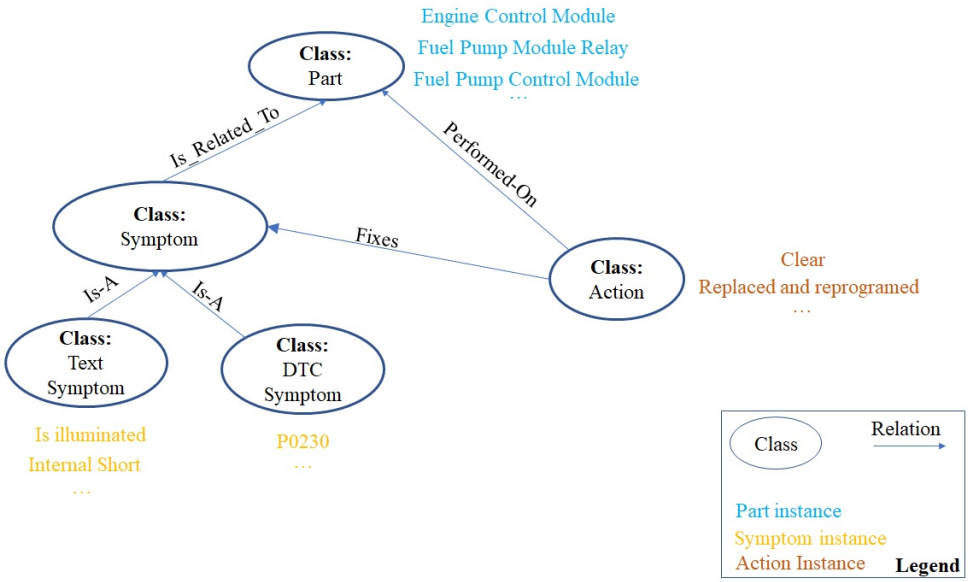

Figure 2.1. The domain model is designed by the domain experts. The classifier is trained to extract the technical concepts and they are classified into their specific classes to populate the domain model.

The classification process in our approach starts by constructing a corpus of millions of verbatims. Since a corpus usually contains different types of noise, these noises are cleaned by using a text cleaning pipeline. It consists of misspelling correction, run-on words correction, removal of additional white spaces, and abbreviation disambiguation. From each verbatim, all the stop words are deleted due to their non-descriptive nature and because they do not add any value to classifier training by using the domain specific vocabulary. Next, each verbatim is converted into n-grams ($n = 1, 2, 3, 4$) and these n-grams constitute the training data. For each n-gram, labels are assigned to indicate

whether it is a relevant technical or irrelevant non-technical concept and also a specific class (e.g. part, symptom, action in case of automotive domain) is assigned to each relevant technical concept. The labelling task is performed by using an existing seed ontology and also by using human reviewers. The process of generating the training dataset is discussed in further in Section 2.4.2.

As we discuss in Section 2.4.3, we identify several unique features related to each n-grams, such as POS, polysemy, word2vec, etc. The labeled n-grams and their corresponding features are used to train our classification model. The relevant concepts and their features are then fed to the second stage classification model, which is trained to assign specific classes to them. In our domain, the number of positive samples decrease as the size of n-gram grows. Hence, the training data consists of a limited number of 4-grams. To overcome this problem, two rounds of active learning are performed to boost the number of training samples of 4-grams in the training data. Active learning also helps to improve the overall performance of our model. In the inference stage (i.e. when applied on the new data), the model takes raw verbatims and preprocesses by using our data cleaning pipeline and then it extracts all candidate concepts without stop words and noise words. Finally, these candidate concepts are fed as input to our two-stage classification system. Figure 2.2 shows the overall process of our two-stage classification system.

2.4. Model Specifications

As discussed in the previous section, our data consists of different types of noise and it is important to clean the raw data before it can be used for feature engineering and then to train our classifiers. Below, we discuss each data cleaning steps in further details.

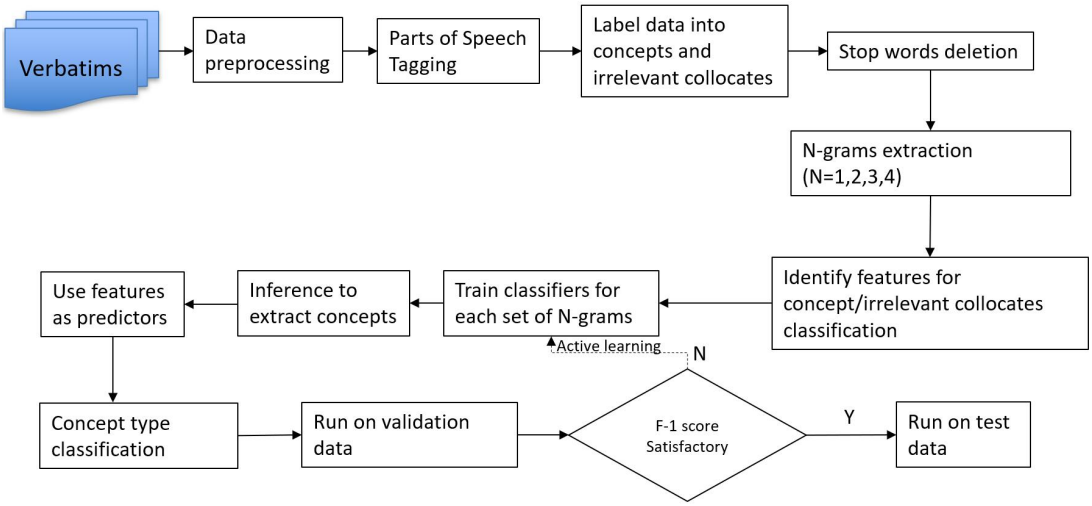

Figure 2.2. The overall methodology and flow of the two-stage classification model.

2.4.1. Data preprocessing

In particular, four different data cleaning algorithms are used to clean the data: misspelling correction, run-on words correction, removal of additional white spaces, and abbreviation disambiguation.

1. Misspellings correction. We consider all possible corrections of a misspelled 1-gram each with Levenshtein distance of 1. If there is only one correction, we replace the misspelled 1-gram by the correction. Otherwise, for each candidate correction we define its semantic similarity score to be the product of its logarithm of frequency and the word2vec similarity between the misspelled 1-gram and its correction. The misspelled 1-gram is replaced by the correction with the maximum similarity score.

2. Run-on words correction. We split run-on words into a 2-gram by inserting a white space between each pair of neighboring characters. For a specific split, if both the left 1-gram and the right 1-gram are correct we retain such split as the correct one.

If there are multiple possible splits with correct 1-grams, then for each correct split its semantic similarity score is defined to be the maximum of word2vec similarities between the run-on 1-gram and the two 1-grams. The split with maximum similarity score is replaced as the correct split.

3. Removal of additional white spaces. We also observe several cases in the data where there are additional white spaces inserted in a 1-gram, e.g. 'actu ator.' We try to remove the additional white spaces to see whether it turns the two incorrect 1-grams into a correct 1-gram and if it does, then we employ this correction.

4. Abbreviation disambiguation. The use of abbreviations, e.g. 'TPS is shorted' is ubiquitous in a corpus and it is critical to disambiguate their meaning to correctly populate our domain model. Typically, an abbreviation is a concept that can be mapped to more than one possible expansion (or full form), for example, 'TPS' could stand for 'Tank Pressure Sensor,' 'Tire Pressure Sensor' or 'Throttle Position Sensor.' The abbreviations mentioned in our data are identified by using the domain specific dictionary, which consists of commonly observed abbreviations and their possible full forms. For an identified abbreviation with a single full form, we replace that specific abbreviation with its full form. Otherwise we employ the following model.

Suppose an abbreviation *abbr* (e.g. TPS) has N possible full forms, namely, $\{ff_1, ff_2, ..., ff_N\}$, where $N > 1$. For 'TPS' we have three possible full forms: 'Tank Pressure Sensor,' 'Tire Pressure Sensor' or 'Throttle Position Sensor.' We first collect the 1-gram concepts, which co-occur with *abbr* from the entire corpus. The context concepts co-occurring with *abbr* are denoted as C_{abbr} and the set of all co-occurring concepts related to each possible expansion, say ff_n, $1 \leq n \leq N$ are denoted as C_n. To prevent meaningless expansions and

to compare the posterior probabilities of ff_i and ff_j, we only focus on the intersection of these sets: $V = \cap_{n=1}^{N} C_n \cap C_{abbr}$. Having identified the relevant intersecting context concepts, we measure the importance of each concept that is a member of intersection in terms of its TF-IDF score.

Let $v_u \in \mathbb{R}^{|V|}$ be the TF-IDF vector of collocate $u = abbr$ or full form $u = ff_i$. Given that ff_n is associated with $abbr$, the probability of co-occurring concepts given ff_n is then estimated as $P(abbr|ff_n) = \prod_{i=1}^{|V|} (\frac{v_{ff_n,i}}{\sum_{j=1}^{|V|} v_{ff_n,j}})^{v_{abbr,i}}$.

This formula computes a probability and if $abbr$ and ff_n are truly interchangeable then they have the same underlying distribution probability of their co-occurring concepts. Furthermore, we estimate the prior probability $P(ff_n)$ of ff_n from its document frequency. Therefore, by the Bayes theorem, $P(ff_n|abbr) \propto P(abbr|ff_n) \cdot P(ff_n)$. We then replace abbreviation $abbr$ by the full form with the largest posterior probability.

2.4.2. Preparation of training set

The process of classifying the data starts by labeling the raw n-grams generated from the cleaned data to construct the training set. Given the scale of real-world data, it is impossible to manually label each raw sample. To overcome this problem, we make use of the seed (incomplete) ontology and tag all such n-grams that are already covered in the seed ontology with the label. For instance, if a specific n-gram, e.g. 'engine control light' is already covered in the existing seed ontology then it is assigned the label of relevant technical concept and then a specific class of a n-gram is borrowed from the seed ontology, e.g. the technical concept 'engine control light' is assigned a specific class 'part.' For the purpose of avoiding repetitions and keeping concepts as complete as possible, only the

longest concepts are marked as a true concept. That is, in a specific verbatim a concept 'engine control module' is marked as the relevant concept, then its sub-grams, such as 'engine,' 'control,' 'module,' 'engine control,' 'control module' are labelled as irrelevant concepts in that specific verbatim. Since the seed ontology is incomplete it is difficult to imagine that it covers all the n-grams included in the training dataset. The n-grams that are not covered in the seed ontology are labelled by domain experts. Please note that we impose a specific frequency threshold (tuned empirically) and the n-grams with their frequency above the threshold are used for manual labelling.

In inference, given a verbatim, we collect all possible n-grams without stop words and noise words in them and they are passed to the two-stage classification system.

2.4.3. Feature engineering

In our model, different types of features, such as discrete linguistic features, word2vec features, polysemy centroid features, and the context based features are identified.

1. Discrete linguistic features. From the data the following linguistic features are identified: 1) the POS tags related to each n-gram is used which is assigned by employing Stanford parts of speech tagger [**51**], 2) the POS tags of the three nearest left side 1-grams of the n-gram, 3) the POS tags of the three nearest right side 1-grams of the n-gram, 4) the POS tag of the nearest concept on the left side of the n-gram, 5) the POS tag of the nearest concept on the right side of the n-gram.

2. Word2vec features. We also consider the continuous word2vec vector associated with each n-gram as one of the features to improve the model performance. We train a Skip-Gram model with respect to frequent 1-grams. When the word2vec embedding is

not available, we consider it as a zero vector. For a n-gram, the associated feature vector is the average word2vec embedding of all its 1-grams.

3. Context features. We consider the 'context' word2vec feature of each n-gram. For a n-gram T, we take the 3 left 1-grams and 3 right 1-grams of T in the verbatim and then obtain the word2vec embeddings of 6 1-grams. The context feature is constructed by the concatenation of the average of the 3 embeddings on the left and the average of the 3 embeddings on the right. If a specific n-gram is towards the beginning or an end of a verbatim and then naturally less than 3 embeddings get constructed, but in such cases all the empty n-grams are not considered while constructing the average embedding. If none of the context terms are available (in our domain it is possible that only one n-gram makes a complete sentence), we set an average embedding to be the zero vector.

4. Polysemy centroid features. In our data, the n-grams appearing in different verbatim may have different semantic meanings as their context changes. Given the number of meanings of a specific n-gram extracted from WordNet, we cluster all the context features of such n-gram into a specific number of clusters. We take a viewpoint that the cluster centroid of each cluster essentially provides a representative feature (indicative of different meanings) of a n-gram. In this way, we distinguish between different semantic meanings of the same n-gram based on its context. Specifically, we consider the polysemy of a 1-grams and the following two steps as shown in Figure 2.3 are employed: 1. For each n-gram T, we randomly sample 1,000 verbatims in which T is mentioned and calculate the context feature vector $V(T)$ for T in each selected verbatim. Then, we use WordNet to obtain the number p polysemies of T. Further, the k-Means clustering algorithm is used to cluster these 1,000 $V(T)$ vectors, with the number of clusters set to p. 2. Having

generated polysemy centroids for a n-gram T', we find the context vector from its verbatim. The feature vector of T' corresponds to the closest centroid among those obtained in step 1 for T' with respect to the context features of T'.

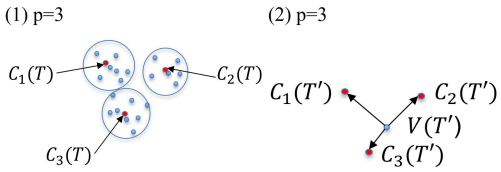

Figure 2.3. (1) Obtain all possible polysemy centroids of a collocate: for a collocate T, we cluster context vectors and save the cluster centroids $C_1(T), ..., C_p(T)$. (2) Create polysemy centroid feature of a collocate: for a new collocate T', let $m = argmin\{d(V(T'), C_1(T')), ..., d(V(T'), C_{p'}(T'))\}$ denote the index of the closest centroid, where d is the Euclidean distance. Vector $C_m(T')$ is our polysemy feature for T'.

5. Features based on the incomplete ontology. We also find that a seed ontology plays a significant role in classification. For a n-gram, we split it into 1-grams and add a feature vector of the same length as the n-gram, with each element being set to be 1 if such 1-gram exists in the seed ontology, otherwise 0.

2.4.4. Classification

In our work, we train a random forest model as our classification model, but we have also experimented with support vector machine, gradient boosted trees, and Naïve Bayes models. The model selection experiments showed that the random forest model outperformed all other models. As a part of model training process, we fine-tune the following important hyperparameters of random forest: the number of trees in the forest is 10, no maximum depth of a tree, the minimum number of samples required to split an internal node is 2.

To further boost model performance, we have also introduced two rounds of active learning. For this, eight different classifiers are trained by feeding randomly sampled data. All the samples with four positive and four negative votes are collected. We then pass all such samples that the classifiers fail to classify consistently into their correct classes to human reviewers for manual labelling. All the samples generated from the two rounds of active learning are added to the training data.

We also analyze feature importance by using the backward elimination process. Within the backward elimination process, our model initially starts with all features and then randomly drops one feature at a time, and we train a new model by using the remaining features. This is done for all features. Then we remove the feature that yields the largest improvement to the F1-score when removed. This process is repeated iteratively until removing any feature does not improve the F1-score. The final set of features kept are Word2vec, Polysemy, POS, Context and Existing Ontology, which are the most important features in our model. The features dropped are left POS, right POS, left three POSes, right three POSes.

2.5. Computational Study

In this section, we provide experimental results to validate our model. While our model can be applied to any domain, in our work, we validate our ontology learning system on a subset of an automotive repair (AR) verbatim corpus collected from an automotive original equipment manufacturer, the vehicle ownership questionnaire (VOQ) complaint verbatim collected from National Highway Traffic Safety Administration[2], and Survey data. The AR data contains more than 15 million verbatims, each of which on average

[2]https://www.nhtsa.gov/

contains 19 1-grams. Here is a typical AR verbatim: *c/s service airbag light on. pulled codes P0100 & ..solder 8 terminals on both front seats as per special policy 300b.clear codes test ok.* The classification models are trained on AR. To study the generality of our model, we also test on VOQ, which contains more than 300,000 verbatims. The VOQ verbatims are significantly different from AR primarily because the VOQ verbatim are reported directly by customers, and thus they are more verbose and less technical in nature in comparison with AR. A sample from VOQ reads: *heard a pop. all of the sudden the car started rolling forward...* Finally, the Survey data is generated from the telephone conversation between customer and service representative. It also consists of different faults as the ones reported in AR, but they are longer with more description. Our existing seed ontology consists of about 9,000 n-grams associated with three different classes, labeled as Part, Symptom, and Action herein.

The classification system is implemented in Python 2.7 and Apache Spark 1.6 and ran on a 32-core Hadoop cluster. The extracted ontology is added to the central database where it can be accessed by different business divisions, e.g. service, quality, engineering, manufacturing. Below we discuss the evaluation of the abbreviation disambiguation as well as both the stages of our classification model.

2.5.1. Evaluation of abbreviation disambiguation

To evaluate the performance of the abbreviation disambiguation algorithm, we generate three separate test datasets from the AR data source. On average, 5% of AR verbatims contain an abbreviation and each abbreviation has more than 2 expansions. Table 2.1

summarizes the results of the abbreviation disambiguation algorithm experiment. All the results are manually evaluated by domain experts.

Table 2.1. The result summary of abbreviation disambiguation algorithm. N_{raw} denotes the number of raw verbatims, N_c denotes the number of abbreviations corrected and $N_{correct}$ denotes the number of correct abbreviation corrections.

Data	N_{raw}	N_c	$N_{correct}$	**Accuracy**
AR 1	10,000	204	154	0.75
AR 2	30,000	374	278	0.74
AR 3	45,000	407	312	0.77

As it can be seen in Table 2.1, the performance of the algorithm is stable, i.e. accuracy does not vary across the three test datasets. On average, 75% of our corrections are correct, which shows our algorithm is able to capture correct expansions of abbreviations. Note that there might be abbreviations that are not captured by our algorithm if abbreviations are not in the abbreviation list.

2.5.2. Performance of classifiers

One of the bottlenecks of supervised machine learning approach is to assemble a large volume of manually labeled data. Recall that the training data is from the AR data. Since the entire AR data is large, our training set is sampled from AR in the following way: for each n-gram ($n = 1, 2, 3, 4$), we randomly sample 50,000 relevant and irrelevant concepts, which we regard as the training set for the n-gram model. Among 100,000 training samples, only 2,000 n-grams are manually labeled and 2,000 are generated by active learning. For evaluation, we generate three different test datasets.

The datasets are first preprocessed using the data preprocessing pipeline and the cleaned data are used in inference. The first test dataset consists of 3,000 randomly

selected repair verbatim from the AR data. The model classified n-grams into relevant and irrelevant concepts and then classified the relevant concepts into their specific classes of either Part, Symptom, or Action. We then randomly selected 1,500 classified n-grams for their evaluation by domain experts to calculate precision, recall and F1-score. The second test dataset consists of 23,000 VOQ verbatims and from the classified n-grams we randomly selected 1,500 n-grams for their evaluation by domain experts. The third test dataset consists of 46,000 verbatims and from the classified n-grams we randomly selected 1,000 n-grams for their evaluation by domain experts. The randomly drawn samples used in evaluation are reviewed in the context of actual verbatim in which they are reported. Moreover, in the AR test set among those n-grams classified as the relevant concepts, slightly less than 30% of concepts are newly discovered by our algorithm, which are not previously covered in the seed ontology. This is a useful finding because newly discovered concepts provide additional coverage to detect new faults/failures for improved decision making. Please also note that the proposed system is not evaluated against other algorithms that are presented in the related work, because the systems reported in the literature are end-to-end solutions and to make a fair comparison all the components used by these systems are necessary. The precision, recall, and F1-score for the test datasets based on the domain expert results are given in Table 2.2.

Table 2.2. The evaluation of relevant concepts and irrelevant concepts classification algorithm.

Dataset	**Precision**	**Recall**	**F1-score**
AR	0.81	0.90	0.85
VOQ	0.89	0.47	0.62
Survey	0.80	0.79	0.79

As we can observe in Table 2.2, the classification F1-score on the AR dataset of relevant and irrelevant concepts is relatively high since the test and training sets are from the similar distribution, in which case the ontology learning system performs very well. In VOQ, since the test data is not from a similar distribution, i.e. the VOQ verbatims are more verbose, the performance on VOQ data is much worse than that on AR. The Survey data is conditionally sampled from AR, and therefore is also from a similar distribution as training, which results in good classification performance. Moreover, on AR, the F1-score for each n-gram is 0.88, 0.81, 0.83, 0.86 for $n = 1, 2, 3, 4$, respectively. The F1-score for 1-gram is better primarily because we have a polysemy centroid feature to capture polysemy meanings of 1-grams, which very likely have different polysemies. For higher grams, the performance is also good, and we presume this is because longer concepts are more easily captured by the algorithm while shorter concepts can be easily confused with irrelevant n-grams.

Table 2.3. The evaluation of relevant concept type classification algorithm.

Dataset	**Precision**	**Recall**	**F1-score**
AR	0.82	0.82	0.82
VOQ	0.84	0.65	0.73
Survey	0.82	0.80	0.81

We follow the same approach to evaluate the performance of the second stage classifier which takes as the input the relevant concepts classified by the first stage classifier and assigns specific classes, i.e. Part, Symptom, or Action. The test set sizes are 800, 1,500, 900 for AR, VOQ and Survey respectively. Note that the 'concepts' passed to the second stage classifier could be incorrectly classified by the first stage classifier, i.e. some inputs could be irrelevant concepts. Each irrelevant concept input to the second stage classifier

is counted as falsely predicted regardless of the type predicted by the classifier. Despite of this, as it can be seen from Table 2.3, the second stage classification model shows good precision rate, but the recall rate is comparatively lower, due to the false negative rate, i.e. the classifier misses out on assigning types to long phrases. It is important to note that although the VOQ dataset is generated from a completely different data source, the second stage classifier shows a very good performance.

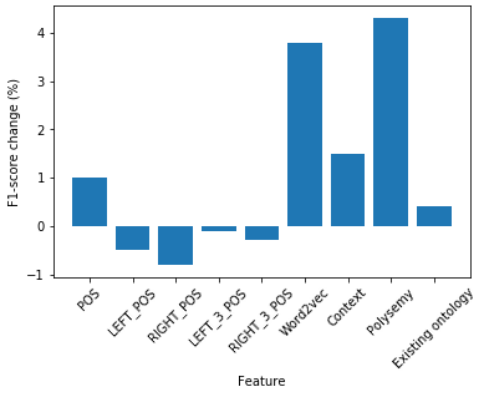

Figure 2.4. Change of F1-score when dropping each feature.

Next, we calculate feature importance by recording how much F1-score drops when we remove each feature. The higher the value, the more important the feature. As it can be seen in Figure 2.4, the features that contribute most to the F1-score are Word2Vec, Context, Polysemy and POS, which is consistent with our observation in backward elimination algorithm. The two most important features are Polysemy (4.3%) and Word2vec (3.8%), which shows the significance of applying word embeddings to the problem of ontology learning.

Table 2.4 shows typical examples of the correctly and incorrectly classified relevant and irrelevant concepts. There are some critical reasons that are identified which contribute to

Table 2.4. Examples of classification results, where 'None' denotes irrelevant concepts.

Collocate	**Predicted**	**True Type**
RECOVER	Action	Action
NO POWER PUSHED	None	None
HIGH MOUNT BRAKE BULB	Part	Part
PARK LAMP	None	Part
ROUGH IDLE RIGHT SIDE	Part	None
ENGINE CUTS OFF	None	Symptom

the misclassification. First, the POS tags associated with each concept considered during the training stage is one of the crucial features and it turns out that POS tags assigned by the Stanford's POS tagger are inconsistent in our data. For example, in 'PARK LAMP,' the POS tagger tags it as 'VBN NNP,' while it should be tagged as 'NNP NNP' since 'PARK' here is not a verb. Second, there is variance in stop and noise words in real-world data. While standard English stop words and noise words allow us to reduce the non-descriptive concepts in the data, we need a more comprehensive stop and noise word customized dictionary specific to automotive domain. Moreover, such a dictionary needs to be a living document that requires timely augmentation to ensure as complete coverage to such words as possible. For example, 'OFF,' which is usually regarded as a stop word in English should not be in our customized stop words list as it appears in concepts like 'ENGINE CUTS OFF.' Third, concepts that are combinations of two different class types contribute to misclassification. In our data, concepts such as 'ENGINE CUTS OFF' consist of two classes fused together, i.e. 'ENGINE' is class Part, while 'CUTS OFF' is class Symptom. To handle such cases, we need to have more representatives within the training dataset.

We also perform another experiment in order to assess the effectiveness of our model in re-discovering the relevant concepts that are already included in the existing seed ontology. For this experiment, we randomly removed the relevant concepts related to the three classes, referred to as class Part, class Symptom, and class Action in the existing seed ontology. Specifically, we removed 250 class Part concepts, 127 class Symptom concepts, and 23 class Action concepts. Since the concepts in the seed ontology are acquired from the AR data, 12,000 AR verbatim are used in this experiment. The test dataset of 12,000 verbatim is cleaned and the trained classification model is applied. The model extracted and classified the relevant and irrelevant concepts and then the relevant concepts are classified into Part, Symptom or Action. Table 2.5 shows the results of this experiment in terms of precision, recall, and F1-score.

Table 2.5. The reconstruction of existing seed ontology from the AR data.

Technical class	**Precision**	**Recall**	**F1-score**
Part	0.89	0.83	0.85
Symptom	0.86	0.79	0.82
Action	0.90	0.86	0.88

As it can be seen from Table 2.5, our classification model has shown promising F1-score and identified the key relevant concepts that were randomly removed from the seed ontology. The closer analysis of the results revealed that our model suffered particularly in classifying 4-grams concepts. There are two reasons behind this: 1. In some cases, the correction of 4-gram concepts by the data cleaning pipeline showed limited accuracy. For example, a 4-gram concept 'P S STEERING RACK' was converted into 'Power Steering Steering Rack' (as the first two 1-grams 'P' and 'S' are

converted into 'Power' and 'Steering'). Then classifier marked such concept as a member of class Part, but a domain expert considers it to be an irrelevant concept. 2. As discussed earlier, the Stanford POS tagger assigns inconsistent tags to the class Symptom concepts in our domain. Further investigation revealed that the Stanford POS tagger assigns a POS tag to a term by estimating a tag sequence probability, i.e. $p(t_1 \dots t_n | w_1 \dots w_n) = \prod_{i=1}^{n} p(t_i | t_1 \dots t_{i-1}, w_1 \dots w_n) \approx \prod_{i=1}^{n} p(t_i | h_i)$.

In our domain, this notion of maximum likelihood showed weaknesses primarily due to the sparse context words associated with higher n-grams. Since the context words around 4-grams change based on verbatim in which they appear the same concept gets different POS tags. For example, the concept 'air pressure compressor sensor' in one verbatim gets the POS tag of 'NNP NN NN NN,' while in another verbatim it is tagged as 'NN NN NN NN.' The POS is one of the important features in our classification model, which ends up impacting the classification accuracy. Therefore, it is important to note that our model shows good F1-score given the complex nature of real-world data, both in terms of identifying new concepts as well as in reconstructing existing concepts.

The ontology learning system discussed in this work is deployed in General Motors. The proposed model is run once every two months in order to extract and classify new concepts, which are reported in the AR data. The newly extracted concepts are added to the existing ontology to improve in-time coverage. This new ontology provides a semantic backbone to the 'fault detection tool,' which is used to build the fault signatures from different data sources to identify key areas of improvement.

2.6. Conclusion

We propose an effective and efficient two-stage classification system for automatically learning an ontology from unstructured text. The proposed framework initially cleans noisy data by correcting different types of noise observed in verbatims. The corrected text is used to train our two-stage classifier. In the first stage, the classification algorithm automatically classifies n-grams into relevant concepts and irrelevant concepts. Next, the relevant concepts are classified to their specific classes. In our approach, different types of features are used and we not only use surface features, e.g. POS, but also identify latent features, such as word embeddings and polysemy features associated with n-grams. In particular, the introduction of novel polysemy controid feature helps in correctly classifying n-grams. As shown in the evaluation, the combination of surface features together with latent features provides necessary discrimination to correctly classify collocates. The evaluation of our system using real-world test data shows its ability to extract and classify n-grams with high F1-score. The proposed model has been successfully deployed as a proof of concept in General Motors for an in-time augmentation of a domain ontology.

CHAPTER 3

Concept Drift and Covariate Shift Detection Ensemble with Lagged Labels

3.1. Introduction

In most challenging real-world supervised machine learning tasks in model serving such as sentiment analysis of Twitter users and weather forecasting, data evolve and change over time, causing the machine learning models built on historical data to become increasingly unreliable. One reason is usually that the classification or regression models assume stationarity, i.e. the training and test data should be independent and identically distributed [**52**, **53**]. This assumption is often violated, leading to limited generalization ability. The changes of the relationship between features and labels are referred to as a concept drift, while the changes of features only are referred to as covariate shift. When either one occurs, the model performance deteriorates. In order to handle the concept drift or the covariate shift, the easiest way is to retrain the model as soon as a batch of new labeled data comes. This is impractical as 1) continuous retraining is extremely computationally time-consuming and 2) it is a waste of effort when the data distribution does not change. A better strategy is to require the model serving system to continuously diagnose signals such as the classification error and feature reconstruction error, and to automatically adapt to changes in data over time. For example, when the classification

error rate increases significantly, the system is supposed to detect the signal and retrain the model.

We consider the model serving process where we continuously receive new batches of data (a batch can as well be a single sample corresponding to streaming) for inference. The data comes without labels which can either arrive immediately after the batch or at any time in the future. Before a new batch arrives, the system needs to make a decision to retrain the model or not and it needs to select a subset of samples to use in retraining if triggered.

There exist several works on drift detection [**54**–**59**] that mainly focus on concept drift with classification performance as the signal for detection. However, this may be problematic. On the one hand, having only one signal may greatly increase the likelihood of false detection or missed detection. On the other hand, drift detection relying on classification performance requires in-time online labeling and is impractical in real-world applications, as in-time online labeling is time-consuming, costly and requires a large amount of human intervention [**53**, **60**, **61**].

We address the first problem by including six different signals, capturing different characteristics of data changes such as a lagged classification error rate and model uncertainty. They function as an ensemble and use a tailored majority voting strategy for drift detection, and thus reduce the reliance on one specific signal and the model is less sensitive to anomaly data.

Although unavoidable, in order to reduce the reliance on in-time online labeling, we address the second problem by introducing the *lag of labels* setting. In this setting, the system receives the labels of input data after certain time periods to allow time for

labeling, instead of receiving labels immediately after a batch as in previous works. In order to detect drift effectively and not waiting till labels are available, our proposed method utilizes a lagged classification error rate for concept drift and other signals for covariate shift, which monitor feature distribution changes as an early indicator of drift. This *lag of labels* scenario is prevalent in real-world applications due to domain expert labeling efforts.

Moreover, most of the existing drift detection algorithms do not have mechanisms to determine what data to use for model retraining when drift is reported, which poses a challenge when applied to real-world applications, because not only do we care about effective and timely drift detection, but the model performance in serving is important as well. Our proposed method automatically determines the data that are used for retraining by collecting samples that are in the warning zone, which can be easily deployed into model serving without human efforts to determine retraining data.

In our work, we propose Concept Drift and Covariate Shift Detection Ensemble (CDCSDE), a drift detection ensemble algorithm in the *lag of labels* setting, where the system receives the labels of input features after a time period due to labeling costs. The ensemble system is composed of six drift detection modules, capturing different characterstics of streaming data such as misclassification rate and feature reconstruction error. The proposed system is also able to decide when to retrain the model and automatically select the data to be used for retraining. We evaluate CDCSDE on both structured and unstructured datasets, and on simulated and real-world datasets. The results show that the proposed method consistently outperforms all the benchmark drift detection models by a large margin.

Our contributions are summarized as follows.

- We propose a novel and effective method for drift detection in the *lag of labels* setting.
- The proposed method can detect both concept drift and covariate shift; it can determine when to retrain and what data to use to retrain automatically, by utilizing an ensemble of six different drift detectors.
- CDCSDE is suitable for both structured and unstructured data.
- We conduct extensive experiments on popular drift detection benchmark datasets. The results show the proposed method consistently outperforms all other methods by a large margin.

The rest of the chapter is organized as follows. In the next section, we provide a review of the relevant literature. In Section 3.3, we provide the problem description while in Section 3.4 we discuss the proposed approach in detail. In Section 3.5, we show the experimental study results. We conclude the chapter by reiterating the main contributions In Section 3.6.

3.2. Related Work

There are several works utilizing statistical control to monitor and detect drift in data streaming. A Cumulative Sum control chart [**56**] (CUSUM) is a sequential analysis technique, which is typically used for monitoring change detection. The system reports an alarm as soon as the cumulative sum of incoming data exceeds a user-specified threshold value. The Page Hinkley [**56**] (PH) test, a variant of CUSUM, is also a sequential analysis technique often used for change detection in the average of a Gaussian signal. Similar to

CUSUM, the PH test alarms a user a change in the distribution when the test statistic of incoming data is greater than a user-specified threshold. The exponentially weighted moving average [**58**] (EWMA) method monitors the mis-classification rate of a classifier for change detection. It calculates the recent error rate by down-weighting the previous data progressively and reports a drift when the EWMA estimator exceeds an adaptive threshold value. Slightly different from the previous approaches, Adaptive Windowing [**57**] (ADWIN) is an adaptive sliding window algorithm for drift detection, which keeps updated statistics from a window of a variable size. The algorithm takes the binary prediction results for incoming data as input, and decides the size of the window by cutting the statistics window at different points and it analyzes the average of statistics over these sub-windows. Whenever the absolute value of the difference between the two averages from two sub-windows exceeds a threshold, the algorithm concludes that the corresponding expected values are different and reports a drift. The Drift Detection Method [**54**] (DDM) is the most widely used concept drift detection method based on the assumption that the model error rate would decrease as the number of analyzed samples increases, provided that the data distribution is stationary. Same as ADWIN, DDM uses a binomial distribution to describe the model performance. DDM then calculates the sum of the overall classification error and its standard deviation. The most significant difference between DDM and the previous works is that when the sum of the two statistics exceeds a threshold, either drift is detected or the algorithm warns that drift may occur soon. The data that DDM flags as a warning can be potentially used for future retraining.

There are four major differences between our proposed algorithm and the existing works. First, most of the existing works utilize a user-specified threshold for drift detection, and thus the performance of the drift detector largely depends on the choice of the threshold with the model performance being extremely sensitive to it. Our proposed algorithm does not require such threshold. Second, most of the existing works focus on an ideal supervised setting, i.e. they assume the labels are always available as soon as the input features are received, which does not capture many real-world applications due to labeling costs. Our proposed algorithm assumes the *lag of labels* setting, which can report a drift as an early indicator even though the labels are not yet available. Third, these works utilize only a single statistic, e.g. the classification error rate, to monitor changes in data, which suffers from the existence of outliers, while our work utilizes an ensemble of six drift detectors that can capture different characteristics of incoming data. Fourth, the previous works only alarm the user about changes in data, omitting deciding the data to be used for retraining. In our work, we progressively select the retraining data based on the calculated warning zone.

3.3. Problem Formulation

A data stream is a data set where observations have time stamps, which induces either a total or a partial order between observations [**62**]. In our work, we assume classification as the only task, although our proposed method can easily generalize to the unsupervised setting where no labels are available.

Suppose the joint distribution $p(X, Y)$ generates random variables X and Y, where X denotes the features for classification and Y denotes the corresponding labels. We further

denote $p_t(X, Y)$ as the joint distribution at time t. Following the conventional definition, concept drift occurs when

$$P_n(X, Y) \neq P_m(X, Y)$$

for time n and m, i.e., the joint distribution changes from time n to time m ($n < m$), which often results in model performance degradation, as the model trained to fit one distribution no longer works for the other one.

Similarly, covariate shift occurs when

$$P_n(X) \neq P_m(X)$$

for times n and m, i.e. the feature distribution has changed. Detection of covariate shift is also of great importance, as on the one hand, it can be used as an early indicator of concept drift, especially when labels are not available or cannot be obtained in-time. On the other hand, most parametric models output probability distributions, which are useful information to learn the model confidence about such predictions. If the model is no longer confident about the predictions, it is desirable to alarm and retrain the model.

In real-world applications, a data stream is usually generated by different joint distributions, as characteristics of incoming data may change over time. If a concept drift or covariate shift occurs, the classification performance is often affected, thus there arises the need of drift detection and subsequent model retraining.

In a data stream in our work, we assume that labels Y_n can only be obtained after a time period l, i.e. at time n features X_n are available, while the corresponding labels Y_n

are available at time $n+l$. This is a more difficult but practical scenario as opposed to the setting where labels Y_n are available as soon as X_n arrives. In real-world scenarios, the labels may arrive at any time, and thus l may be a random variable following a statistical distribution.

Due to the aforementioned reasons, one cannot have a fixed model during the entire process of model serving, as incoming data distribution might have changed and causes performance degradation. The main tasks are: 1) how to monitor model performance, 2) how to decide if data distribution has changed and 3) what data to use to retrain if change is detected. The ultimate goal is to maximize the progressive accuracy of the provided classifier across all incoming data by addressing the three tasks.

3.4. Concept Drift Detection Ensemble and Model Retraining

In this section, we exhibit our approach to solve the drift detection and model retraining problems.

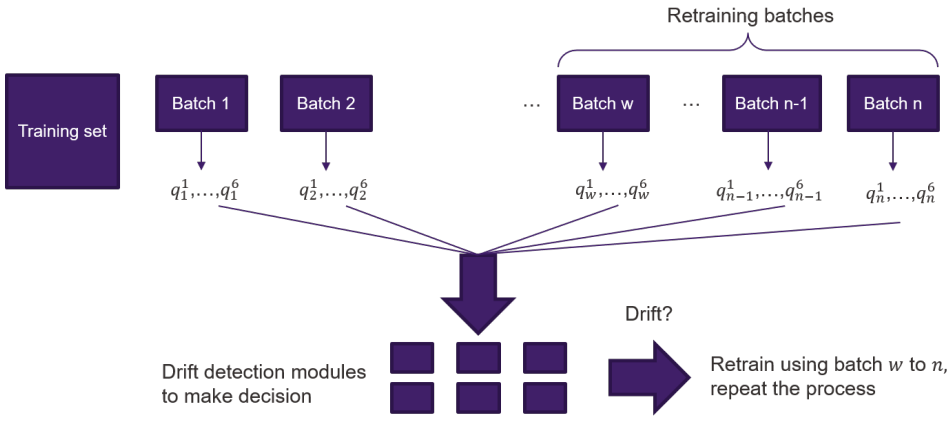

Figure 3.1. Overall approach. For each batch of incoming data, we calculate six descriptive statistics of time series, then utilize drift detection module for each of them to monitor drift and decide what data used to retrain.

We assume the label lag is l. (X_{tr}, Y_{tr}) are used to train the current classification model clf. We denote an incoming batch as $B_n = (X_n, Y_n)$, where Y_n are labels for feature set X_{n-l}. Sets $Y_1, Y_2, ..., Y_l$ are assumed to be empty.

Figure 3.1 shows the general approach of our system, which contains six different drift detection modules which capture different characteristics of data: EWMA of delayed kpi, model uncertainty, Hellinger distance, auto-encoder reconstruction error, SPN fitting loss and gradient changes, which are further explained in the following subsections. The combination of the six modules is able to detect both concept drifts and covariate shifts. Based on the tailored majority voting strategy among six modules, when the system decides to retrain the model, it utilizes proper batches selected by the system to retrain the model, after which the entire monitoring process is repeated.

3.4.1. Descriptive statistics calculation

We construct six time series which are fed to drift detection modules. At each time step six different scores are calculated, which are either based on only the current batch or based on the current batch as well as the previous batches. The six scores are then added to the six time series for drift detection. We provide details on how we calculate the scores in the following subsections.

EWMA of a delayed classification indicator: Let $kpi(Y, X)$ be the most important key performance indicator of clf, e.g. error rate or $1 - f1score$. We calculate the exponentially weighted moving average of delayed classification error rate as follows. Assume we trace back k batches to calculate the moving average, i.e. at time n, we use the kpi of batches $n - k + 1$, $n - k + 2$,..., n. As the labels are delayed by l based on our

assumption. If the weight decay for weighted moving average is w such that $\sum_{i=1}^{l} w^i = 1$, then the score is calculated as

$$q_n^1 = \sum_{i=n-k+1}^{n} kpi(Y_i, clf(X_{i-l})) \cdot w^{n-i+1}. \tag{3.1}$$

The main difference between (1) and the prior works is that the score defined in (1) avoids using the labels of current batch due to label delay, and it uses the delayed kpi for drift detection, while the prior works utilizing EWMA assume $l = 0$, i.e. they are not designed to handle *lag of labels* situation and they use a different statistical control method. Despite of this, it generalizes in a straightforward manner to the setting where features and labels arrive simultaneously ($l = 0$). Moreover, utilizing EWMA instead of focusing on a single batch is beneficial to the stability of the system, as it is more robust against potential outliers. Intuitively, when EWMA increases significantly, the model suffers from performance degradation and the drift has likely occurred.

Model uncertainty: It is also helpful to understand how confident the model is regarding predictions. In the second score, we first predict every sample in the current batch X_n to obtain the predicted probability distributions. For each of these distributions, we construct the histogram of the largest probability and the histogram of the second largest probability, and fit each histogram using a Gaussian distribution to obtain probability density functions N_1, N_2. The model uncertainty score is obtained by

$$q_n^2 = \int_{-\infty}^{\infty} min(N_1, N_2). \tag{3.2}$$

When the overlapping area increases, the mean and variance of the two Gaussians are close to each other, indicating that the largest probability and the second largest probability from predictions are very similar and thus the model is more uncertain regarding the predictions. A significant increase of q^2 is a signal for drift, since when the model predictions are uncertain, the likelihood of a change in distribution is high. On the other hand, when the model is uncertain regarding a distribution, even if the predictions are correct (i.e. q^1 does not vary much), it is still detrimental to the overall process of model serving, as a small fluctuation of the distribution may move the model decision boundary and makes the model vulnerable to future outliers.

Hellinger distance: In statistics and measure theory, the Hellinger distance is often used to quantify similarity between two distributions, and is also widely used in tasks such as anomaly detection and classification. We utilize Hellinger distance to construct the distance between two datasets. Having discretized the features, we define the Hellinger distance between the training set and the current batch as

$$q_n^3 = \frac{1}{|F|} \sum_{f \in F} \sqrt{\sum_{z \in f} (\sqrt{\frac{|X_{tr,f=z}|}{|X_{tr}|}} - \sqrt{\frac{|X_{n,f=z}|}{|X_n|}})^2}, \tag{3.3}$$

where F denotes the set of all features. By averaging over all features, we calculate the distance between two datasets or batches. If the distance between the training dataset

(i.e. the dataset that the model is fitted on) and the current batch is large, the existing model no longer fits the current distribution and needs to be retrained. Note that if the input data is unstructured such as image data, we calculate the Hellinger distance based on encoded feature vectors from an auto-encoder, instead of the raw features themselves.

Auto-encoder reconstruction error:

Auto-encoders are widely used in tasks such as dimension reduction and embedding learning. In our work, we employ an auto-encoder to measure how different the two datasets or batches are. We first train an auto-encoder using training features X_{tr} and obtain the training reconstruction MSE loss $L_{tr} = MSE(X_{tr}, AE(X_{tr}))$. For the current batch X_n, we calculate the test reconstruction loss $L_n = MSE(X_n, AE(X_n))$. The auto-encoder reconstruction score is defined as

$$q_n^4 = tanh(\frac{L_{te}}{L_{tr}}). \tag{3.4}$$

This score allows us to measure divergence by how large the test reconstruction error is compared to the training error. The increase of q^4 indicates that the auto-encoder is unable to fully reconstruct the incoming data and thus the covariate shift has occurred.

Sum-Product Networks:

The Sum-Product Network (SPN) is a deep probabilistic model widely used as black box density estimators by comparing the likelihoods on tasks such as image completion and image classification [**63**]. Similar as q^4, we use an SPN to monitor if the incoming distribution has changed or not, compared to the training data. After training an SPN using

training data X_{tr}, for batch X_n, we obtain the log of negative log-likelihood $log(-ll_n)$. The SPN module score is defined as

$$q_n^5 = log(-ll_n). \tag{3.5}$$

As vanilla SPNs not take unstructured data as input, in such cases, we feed the embedded features from the encoder of the auto-encoder to SPN. The higher the score is, the less likely X_n is generated by the same distribution as X_{tr}.

Gradient changes:

The changes of gradients can also be used to tackle the drift detection task, i.e. the larger the gradient changes, the more likely the data has changed. Instead of using the conventional gradients, we utilize natural gradients instead, which show promising performance in areas such as robotics and control. As natural gradients are rescaled by the Fisher information matrix, they are more stable and often used to solve issues such as catastrophic forgetting.

Let the optimal parameters from the training data (X_{tr}, Y_{tr}) be θ^*. We evaluate the natural gradients at time n using batch (X_{n-l}, Y_n) as $\nabla_N Loss = \nabla Loss(X_{n-l}, Y_n, \theta^*) \cdot F^{-1}$, where $Loss$ is the conventional loss function (cross-entropy in classification) and F is the Fisher information matrix, approximated by using Kronecker factorization [**64**]. The gradients change score is obtained by

$$q_n^6 = \frac{(\nabla_N Loss)^T (\nabla_N Loss)}{dim(\nabla_N Loss)}. \tag{3.6}$$

As gradient changes, the score should increase which is a signal for potential drifts.

3.4.2. Drift detection

Each descriptive statistic defined in the previous subsection generates a time series over time. The remaining problem is how to detect drift based on these time series. We describe next for the current batch X_n, the resulting algorithm to report drift, warning or safe signals, which utilizes an ensemble of six independent drift detection modules to monitor drift.

Similar to [**54**], to detect drift of a time series $\{q_n\}_{n=1,2,...}$, denote the progressive average by $p_i = \sum_{n=1}^{i} q_n/i$ and standard deviation by $s_i = std(p_1, p_2, ..., p_i)$. We also denote the minimum values at time i by p^i_{min} and s^i_{min} such that $p^i_{min} + s^i_{min} = \min_{1\leq n\leq i}(p_n + s_n)$. Note that different from [**54**], at each time step we receive a batch instead of a single observation, thus our time series is not generated by Bernoulli distributions. To monitor drift, we apply the following rules:

If $p_i + s_i > p^i_{min} + 2 \cdot s^i_{min}$, the system is in the warning zone;

If $p_i + s_i > p^i_{min} + 3 \cdot s^i_{min}$, we report drift and retrain the model using batches in the warning zone;

If $p_i + s_i < p^i_{min} + 2 \cdot s^i_{min}$, the system exits the warning zone and is safe from drift.

There are two assumptions for this statistical control module. First, the values in the time series decrease when the data distribution is stationary, and thus the system should either exit the warning zone or stay in safe zone. Second, a significant increase in the time series indicates the existence of a drift.

With a simple drift detection module, we would be able to detect drift and decide what data to use to retrain based on a single time series. Having six different time series, our method CDCSDE works as an ensemble. To detect a drift event, we employ a tailored majority voting rule as follows: if the drift detection module on q^1 reports drift, then the system reports drift and retrains all models (classifier, auto-encoder and SPN), as maximizing the classification accuracy is the ultimate goal; otherwise, if most of the remaining modules (i.e. not less than 3 modules) report a drift, CDCSDE will report drift and retrains all models.

If drift is reported, the data used to retrain is determined by CDCSDE as well. Suppose the latest warning zone (if a module exits the warning zone, then the previous warning zone is not included in future retraining) for each module is $W_i, i = 1, 2, ..., 6$. The union of these warning batches $\bigcup_{i=1}^{6} W_i$ is used to retrain all models.

3.5. Experimental Results

In this section, we report experiments on both simulated and real-world datasets to evaluate the proposed method. The simulated datasets have ground truth about drift. For simulated datasets, 5 different metrics are used: the mean accuracy value over all batches (MA), mean time between false alarms (MTFA), mean time to detection (MTD), missed detection rate (MDR), total number of drifts detected (TD). A good drift detection method should have high MA and MTFA (or no MTFA due to no false alarms) as well as low MTD and MDR. Low TD is also beneficial since it reduces retraining time and computational costs. Among these metrics, MA is the most important as in real-world model serving the overall model performance is what we care most about. In real-world

datasets only MA and TD are available as we do not have the knowledge on when drift 'indeed' occurs. We conduct experiments on both structured data and unstructured data to validate the effectiveness of CDCSDE.

In the following experiments, the *lag of labels* is assumed to follow the exponential distribution and thus the time between events is a Poisson process, i.e., a process where events occur continuously and independently at a constant rate. We employ scale = 4 and the sampled value is rounded down to a integer. We also study the sensitivity with respect to the *lag of labels* and we construct an ablation study.

3.5.1. Structured data stream

We first perform experiments on structured data which consist of the following widely used datasets.

1. Sea [**65**]: simulated data with abrupt drift, 50,000 samples, 3 features and 2 classes. There are 3 drifts in total, each occuring after 12,500 samples. **2. Sine2**: simulated data with abrupt drift, 100,000 samples, 2 features and 2 classes. There are 9 drifts in total, each occurs after 10,000 samples. **3. Elec** [**66**]: real-world data, 45,312 samples, 8 features and 2 classes, which are recorded every half an hour for two years from the Australian New South Wales Electricity Market. **4. Weather** [**67**]: real-world data, 18,159 samples, 8 features and 2 classes, which record weather measurements from over 7,000 weather stations worldwide to provide a wide scope of weather trends. **5. Temp** [**68**]: real-world data, 4,137 samples, 24 features and 2 classes, which are collected from a monitor system mounted in houses.

Across all datasets, we utilize a 2-layer auto-encoder with shape 'Input-6 neurons-Input' and one-layer feed-forward network with shape '6-Output' as the classifier which takes the embedded features from the auto-encoder as input. Also taking the embedded features as input, the SPN consists of 4 layers, i.e. normal, product, sum, product. We optimize our models with the Adam optimizer with the 0.001 initial learning rate. The batch size is set as 64.

Table 3.1. Results on simulated structured datasets.

	Sea					**Sine2**					
Method	**MA**	**MTFA**	**MTD**	**MDR**	**TD**	**MA**	**MTFA**	**MTD**	**MDR**	**TD**	**AVG**
PH	69.50	-	43.5	33.3	3	69.78	38.0	22.3	33.3	8	69.64
ADWIN	77.59	9.5	18.0	0	14	78.39	15.9	40.0	33.3	16	77.99
EWMA	75.73	15.5	22.0	0	7	47.92	31.5	25.0	44.4	11	61.83
DDM	85.32	-	42.5	33.3	2	77.38	-	**9.3**	55.6	5	81.35
CDCSDE	**88.08**	-	**17.7**	**0**	3	**83.69**	-	10.4	**22.2**	8	**85.89**

Table 3.2. Results on real-world structured datasets.

	Elec		**Weather**		**Temp**		
Method	**MA**	**TD**	**MA**	**TD**	**MA**	**TD**	**AVG**
PH	57.51	6	56.18	2	67.00	6	60.23
ADWIN	58.38	2	67.70	1	78.45	2	68.18
EWMA	59.61	8	32.29	1	74.39	3	55.46
DDM	58.63	1	67.70	1	66.25	1	64.19
CDCSDE	**67.71**	5	**74.97**	2	**82.80**	2	**75.16**

From Table 3.1 where '-' denotes not available, i.e. no false alarms or only one false alarm, 'AVG' denotes the average MA across all datasets, and the numbers in bold denote the best across all methods if applicable, we observe that among all evaluation metrics, CDCSDE consistently outperforms the benchmarks, especially in the most important metric MA, as it effectively detects drift and decides what data to use to retrain. Our relative improvements on average accuracy are 23.33%, 10.13%, 38.91%, 5.58% for PH,

ADWIN, EWMA and DDM, respectively. Regarding MTFA, CDCSDE predicts no false detection in Sea and only one false detection in Sine2, while other benchmarks do not exhibit a stable performance, resulting in more false alarms and high MTFA. Our method also achieves the best performance in MTD due to the timely alarm of drift, with 17.7 on Sea and 10.4 on Sine2, while other benchmarks generally being less sensitive to the drifts. Lastly, among all methods, CDCSDE achieves the smallest MDR and reasonable TD.

We also conduct experiments on Elec, Weather and Temp datasets to provide better insights on how our method performs on real datasets. The results are provided in Table 3.2 where '-' denotes not available, i.e. no false alarms or only one false alarm, 'AVG' denotes the average MA across all datasets, and the numbers in bold denote the best across all methods if applicable. Note that we do not have measures such as MTFA, MTD, etc. as they are real-world datasets without the ground truth information regarding drifts. From Table 3.2, CDCSDE consistently achieves the best performance in MA for all benchmarks, which again validates the effectiveness of CDCSDE. Our relative improvements on average accuracy are 24.79%, 10.24%, 35.52%, 17.09% for PH, ADWIN, EWMA and DDM, respectively. The TD of our method is also reasonable, while some other benchmarks either report too many drifts or do not report drift at all.

3.5.2. Unstructured data stream

In order to evaluate our method on unstructured data which is more ubiquitous in real-world applications, we conduct experiments utilizing conventional image classification datasets MNIST and USPS [**69**]. They contain the same 10 digits as labels (i.e. 0-9), but their distributions differ. The two datasets are widely used in domain adaptation tasks

due to a moderate domain gap. In our work, we employ them to create 5 different data streams to validate CDCSDE.

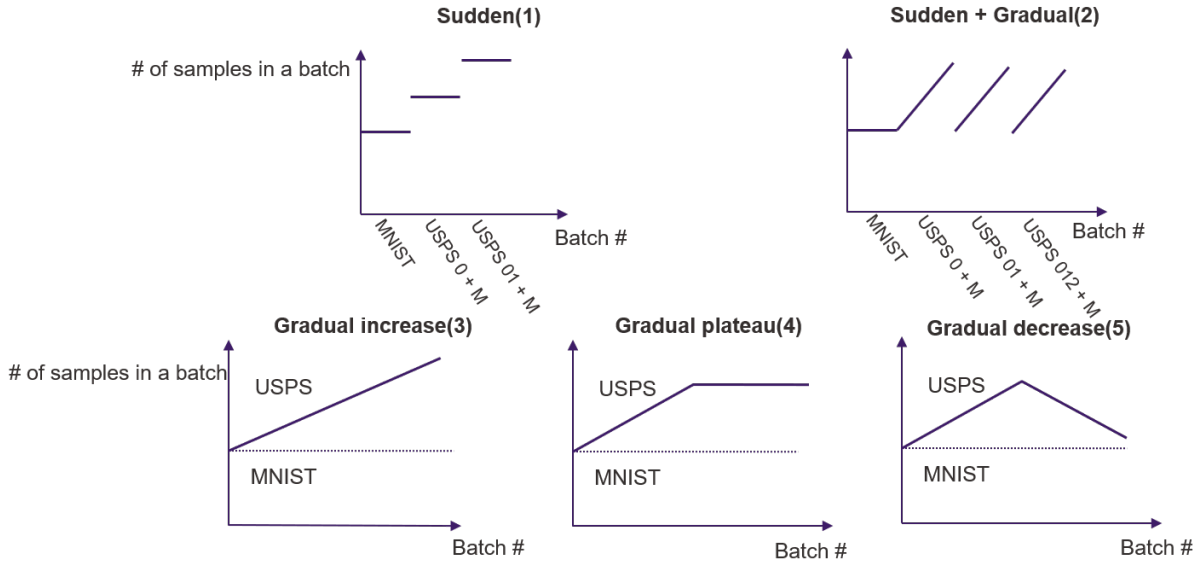

Figure 3.2. Unstructured datasets with various types of drifts.

Across all scenarios, we use MNIST as the 'base' dataset, i.e. any incoming batches contain MNIST 0-9. USPS is used as the 'drift' dataset, i.e. samples from USPS are added to the data stream in a specific way to introduce drifts. The batch size is set as 64 and the total number of samples is 60,000 (the size of the MNIST training set), and thus the number of batches is $60,000/64 = 938$. The details of each dataset are provided as follows, see also Figure 3.2.

1. Sudden drift: A fixed number of samples of one digit from USPS are introduced after each 100 batches. Therefore, there are 9 sudden drifts in total. **2. Sudden and gradual drift:** Samples of one digit from USPS are gradually introduced after each 100 batches. There are 9 sudden drifts in total. **3. Gradual drift - increase:** USPS 0-9 digits are gradually and increasingly introduced. **4. Gradual drift - plateau:** USPS

0-9 digits are gradually introduced. After the 500^{th} batch , the rate of adding USPS 0-9 digits is kept unchanged. **5. Gradual drift - decrease:** USPS 0-9 digits are gradually introduced. After the 500^{th} batch , USPS 0-9 digits are gradually removed at the same rate of introducing.

Table 3.3. Results on unstructured datasets with sudden drifts.

	Sudden					**Sudden and gradual**					
Method	**MA**	**MTFA**	**MTD**	**MDR**	**TD**	**MA**	**MTFA**	**MTD**	**MDR**	**TD**	**AVG**
PH	71.59	-	52.0	77.8	2	73.48	-	43.0	88.9	2	72.53
ADWIN	85.49	29.3	**23.4**	22.2	19	84.17	29.4	31.9	44.4	24	84.83
EWMA	90.92	53.9	29.4	33.3	11	92.01	49.2	26.1	33.3	19	90.47
DDM	87.00	-	25.5	77.8	2	85.51	-	35.5	77.8	2	86.26
CDCSDE	**97.41**	252.5	26.3	**11.1**	10	**97.89**	178.3	**18.3**	**11.1**	11	**97.65**

Across all datasets, we utilize a 2-layer CNN with output shape 500 as the encoder and an one-layer feed-forward network with shape '500-10' as classifier which takes the embedded features from the encoder as input. Also taking the embedded features as input, the SPN consists of 4 layers, i.e. normal, product, sum, product. We optimize our models by using the Adam optimizer with 0.001 initial learning rate. The batch size is 64.

We first experiment on the sudden drift, and sudden and gradual drift scenarios. From Table 3.3, we observe that when sudden drift occurs, the conventional benchmarks generally suffer from high MDR (MDR of PH and DDM are as high as 88.9% and 77.8%) due to the lack of ability for detecting changes in feature distributions, as they only take classification error rate as input. CDCSDE achieves 11.1% MDR, i.e. only 1 drift missed, which shows the ability of the proposed method for detecting both covariate shifts and concept drifts. CDCSDE's TD are 10 and 11 for two scenarios which is reasonable for a total of 9 drifts, while other benchmarks either report too many drifts (24 for ADWIN)

or too few drifts (2 for PH and DDM). Similarly, the MTFA metric of CDCSDE is much higher than the benchmarks when their TD is reasonable (as otherwise there will be no false alarms at all), indicating that the proposed method is much less likely to make false alarms. As a comparison, ADWIN reports a false alarm every 29.3 time steps on average. The MTD metric of the proposed method is 26.3 for the sudden dataset and 18.3 for the sudden and gradual dataset which is the lowest among all benchmarks, which again shows the ability of detecting drifts as low MTD indicates the method can respond to the changes in data distributions in time. Most importantly, the MA of CDCSDE consistently outperforms other benchmarks by a large margin due to the accurate and timely detection of drifts. Our relative improvements on average accuracy are 34.63%, 15.11%, 7.94%, 13.20% for PH, ADWIN, EWMA and DDM, respectively.

Table 3.4. Results on unstructured datasets with gradual drifts.

	Gradual increase		**Gradual plateau**		**Gradual decrease**		
Method	**MA**	**TD**	**MA**	**TD**	**MA**	**TD**	**AVG**
PH	85.63	3	81.28	2	81.12	4	82.68
ADWIN	89.45	13	91.65	9	87.17	7	89.42
EWMA	91.11	11	88.32	9	92.68	10	90.70
DDM	92.31	2	94.15	3	93.68	3	93.38
CDCSDE	**97.99**	3	**97.76**	4	**97.50**	3	**97.75**

We then conduct experiments only with gradual drifts to observe model performance if there are no sudden drifts in the data stream, the results shown in Table 3.4. From Table 3.4 we observe a relatively low TD for CDCSDE and the MA greatly exceeds the benchmarks across all three datasets, which shows the robustness of the proposed method for gradual drift as well. Our relative improvements on average accuracy are 18.23%,9.31%, 7.72%, 4.68% for PH, ADWIN, EWMA and DDM, respectively.

Ablation study In order to establish how important each component of CDCSDE is, we conduct an ablation study on three unstructured datasets: sudden, sudden and gradual, gradual decrease. The mean accuracies are shown in Table 3.5 and Table 3.6.

Table 3.5. Ablation study - increasingly adding components. 'AVG' denotes the average MA across all datasets.

	Sudden	**Sudden and gradual**	**Gradual decrease**	**AVG**
EWMA error rate	94.38	95.68	92.18	94.08
+ uncertainty	95.19	95.88	93.91	94.79
+ Hellinger	95.59	95.70	94.24	95.18
+ AE error	96.10	96.30	95.31	95.90
+ SPN likelihood	97.25	97.10	96.24	96.86
+ gradient norm	97.41	97.89	97.50	97.60

We first add each component at one time to observe performance changes reported in Table 3.5. The largest gains are from the SPN and gradient norm modules, which empirically shows that monitoring changes in feature distribution is beneficial to model performance.

Table 3.6. Ablation study - MA drop without each component.

	Sudden	**Sudden and gradual**	**Gradual decrease**	**AVG**
w/o EWMA error rate	**2.61**	**3.59**	**2.33**	**2.84**
w/o uncertainty	0.11	0.37	0.30	0.26
w/o Hellinger	0.21	0.28	0.17	0.22
w/o AE error	0.60	0.88	0.91	0.80
w/o SPN likelihood	1.38	1.60	1.10	1.36
w/o gradient norm	1.01	1.25	1.07	1.11

We then experiment removing each component to observe the accuracy drop in Table 3.6, which shows similar patterns as Table 3.5. The model without SPN suffers from the second largest performance drop, which again validates that modeling feature distribution as an early indicator of drift is important. It is not surprising that CDCSDE without

EWMA's error rate suffers from the largest performance degradation, as it utilizes both labels and features to calculate accuracy of predictions and a good accuracy in model serving is what we are ultimately interested in.

Varying the level of *lag of labels* We also experiment how the model performs on different levels of *lag of labels*. In addition to $l \sim exp(scale = 4)$ as in previous experiments, we conduct experiments on small lag $l \sim exp(2)$ and large lag $l \sim exp(10)$. The results are shown in Table 3.7.

Table 3.7. Results on unstructured datasets with different levels of *lag of labels*.

	Sudden					**Sudden and gradual**					
Method	**MA**	**MTFA**	**MTD**	**MDR**	**TD**	**MA**	**MTFA**	**MTD**	**MDR**	**TD**	**AVG**
	$l \sim exp(2)$										
PH	73.94	-	45.5	77.8	3	73.01	-	39.0	77.8	3	73.48
ADWIN	87.74	25.5	22.0	33.3	17	90.01	29.2	**21.2**	33.3	16	88.38
EWMA	92.01	59.0	**19.1**	33.3	12	90.19	29.9	26.8	44.4	14	90.60
DDM	87.64	39.0	32.0	88.9	3	87.10	-	30.0	77.8	2	86.26
CDCSDE	**97.57**	287.0	24.0	**11.1**	10	**97.71**	151.0	21.5	**11.1**	10	**97.65**
	$l \sim exp(4)$										
PH	71.59	-	52.0	77.8	2	73.48	-	43.0	88.9	2	72.53
ADWIN	88.10	31.6	**25.1**	33.3	19	84.41	28.4	29.5	55.6	14	86.26
EWMA	93.11	65.5	28.5	33.3	8	90.82	45.0	20.2	33.3	11	91.97
DDM	87.00	-	25.5	77.8	2	85.51	-	35.0	77.8	2	86.26
CDCSDE	**97.21**	252.5	26.3	**11.1**	10	**97.49**	178.3	**18.3**	**11.1**	11	**97.35**
	$l \sim exp(10)$										
PH	69.70	34.0	41.0	88.9	3	71.40	56.0	51.0	77.8	4	70.55
ADWIN	87.45	29.8	32.0	33.3	22	86.30	27.0	33.5	44.4	17	86.88
EWMA	86.98	55.3	39.5	44.4	10	89.33	58.2	28.3	44.4	13	88.16
DDM	84.57	33.5	45.5	77.8	4	82.05	40.0	36.0	77.8	4	83.31
CDCSDE	**96.01**	**191.0**	**28.6**	**22.2**	11	**95.41**	**117.0**	**24.9**	**11.1**	11	**95.71**

From Table 3.7, we observe that in general models perform better for smaller *lag of labels*, which is as expected because when the labels arrive sooner, such up-to-date information can be immediately taken into account. When the labeling costs are high, i.e. it takes more time for labels to arrive, the system can only use delayed labels to detect

drifts. Comparing among all methods, CDCSDE still shows stable performance across all metrics, consistently outperforming benchmarks by a considerable margin for all levels of *lag of labels*.

Fixed *lag of labels* Instead of using a stochastic *lag of labels* as in the previous experiments, we further experiment on employing a fixed *lag of labels* of $l = 2, 4$ and 10. The results are shown in Table 3.8.

Table 3.8. Results on unstructured datasets with fixed *lag of labels*.

	Sudden					**Sudden and gradual**					
Method	**MA**	**MTFA**	**MTD**	**MDR**	**TD**	**MA**	**MTFA**	**MTD**	**MDR**	**TD**	**AVG**
	$l = 2$										
PH	72.40	18.5	41.5	77.8	4	75.41	-	31.0	77.8	3	73.91
ADWIN	91.21	53.5	**23.5**	33.3	17	88.10	28.9	30.1	33.3	14	89.66
EWMA	90.01	41.0	32.3	44.4	12	90.83	35.2	23.6	44.4	10	90.42
DDM	89.02	54.5	28.5	77.8	4	87.45	38.5	29.0	88.9	3	88.24
CDCSDE	**97.73**	-	25.1	**22.2**	8	**97.49**	-	**20.9**	**11.1**	9	**97.61**
	$l = 4$										
PH	71.04	-	55.5	77.8	3	73.01	-	50.0	77.8	3	72.03
ADWIN	87.39	36.0	33.8	33.3	14	88.50	31.5	44.6	44.4	13	87.95
EWMA	89.59	35.5	39.3	44.4	12	88.19	40.0	27.5	44.4	13	88.89
DDM	86.24	39.0	32.0	88.9	3	84.20	-	30.0	77.8	2	86.26
CDCSDE	**97.27**	225.0	**30.1**	**11.1**	10	**97.51**	151.0	**22.5**	**11.1**	10	**97.39**
	$l = 10$										
PH	70.01	21.5	51.0	77.8	4	71.21	41.0	71.5	77.8	5	70.61
ADWIN	84.10	25.8	**33.0**	55.6	18	83.51	35.8	55.5	44.4	19	83.81
EWMA	85.19	49.0	41.0	55.6	8	83.91	48.8	**28.3**	44.4	23	84.55
DDM	83.87	41.0	50.3	66.7	5	81.29	51.3	36.5	77.8	5	82.58
CDCSDE	**96.51**	**170.0**	34.1	**22.2**	9	**95.80**	**108.5**	29.7	**33.3**	8	**96.16**

Compared with Table 3.3, we observe similar patterns in Table 3.8. The benchmarks generally are either too sensitive to drifts or unable to detect drifts. CDCSDE still shows robust performance across all metrics, consistently outperforming benchmarks by a considerable margin even in the fixed *lag of labels* scenario.

3.6. Conclusions

In this chapter, we present a novel and effective drift detection method in the practical *lag of labels* setting, which is able to detect both concept drift and covariate shift and automatically decide what data to use to retrain, with the help of the ensemble of different drift detectors. Extensive experiments on structured and unstructured data for different type of drifts have shown that our method consistently outperforms the state-of-the-art drift detection methods by a large margin.

CHAPTER 4

Open Set Domain Adaptation by Extreme Value Theory

4.1. Introduction

Recently, deep learning techniques have drawn a large amount of attention from people in both academia and industry, due to their astounding performance in fields such as computer vision and natural language processing [**15**, **70**–**78**]. However, one major disadvantage of deep learning is that neural networks generally require a large amount of training data to converge. When the training data are insufficient, the model performance is usually adversely affected. Sometimes even if the training data are sufficient, the domain gap, i.e. the difference between data distributions, between the source domain (the data we train model on) and the target domain (the desired target task) may still contribute to the low generalizability. This is because in conventional machine learning tasks, we usually assume training data distribution is the same as the testing data distribution. However, in real-world situation, testing data are uncontrollable, and thus the difference between the source domain and the target domain could be huge, which results in the overfitting problem, i.e. the model does not generalize well to the testing set.

In order to reduce the domain gap and better utilize the source domain knowledge, domain adaptation techniques have been proposed to resolve the issue. Domain adaptation assumes that the source domain has sufficient amount of labeled data to train a good model, while the desired target domain has insufficient amount of data to train the model.

Note that in most domain adaptation tasks, the target domain does not have labeled data at all, which is also known as unsupervised domain adaptation. Domain adaptation methods leverage the knowledge from the source domain with sufficient labeled data to learn a model that works well in the target domain with insufficient or no labeled data. Typically, domain adaptation methods reduce the domain gap by diminishing the divergence between the label-rich source and the label-scarce target domain [**79**–**90**]. A significant drawback of most of the existing works is that they assume the source label space and the target label space are identical, i.e. the target classes are assumed to have appeared during training process and training classes do not contain classes that are not in the target domain. This is also known as closed set domain adaptation (CSDA), which is impractical in real-world scenarios since it is never guaranteed that the target domain classes are the same as the source domain classes. Brutally aligning the source and the target domain when their label spaces are different is extremely detrimental to model's generalizability, which is also known as the negative transfer phenomenon, because the CSDA methods will try to align the additional unknown classes as well during adaptation.

To handle the domain adaptation tasks without assuming identical label spaces, open set domain adaptation (OSDA) methods were proposed to first detect the irrelevant or unknown samples and then avoid adaptation on the unknown samples and perform domain adaptation only on the known classes [**91**–**94**], which can be achieved by forcing the model to learn a clear boundary between known classes and unknown classes and a clear boundary within the known classes. After adaptation, the model is applied to the target samples: either a target sample is classified as one class in the known classes, or rejected as the unknown classes. However, all of the conventional OSDA methods employ a rejection

threshold hyperparameter, where if a score or statistic for a sample is higher than, for example, 0.3, then the sample will be rejected as an unknown class and discarded during adaptation, and thus the model's sensitivity is largely affected by this rejection threshold.

In our work, we handle the existing issues in OSDA by 1) utilizing an entropy-based instance-level reweighting strategy and 2) extreme value theory (EVT) which has proven to be useful in many classification tasks due to its ability to model extreme values [**95**–**99**]. We propose to use entropy of probability distribution of a sample to measure the likelihood it belongs to unknown classes. That is, the higher the entropy is, the more likely the sample belongs to the unknown classes, because the model is more uncertain regarding the prediction. We utilize the entropy values to construct an instance-level weight for domain adaptation, instead of setting a hard threshold. In this way, every sample is taken into account during adaptation process, and the sample with higher entropy will be focused less to avoid the negative transfer problem. In inference, we model the tail probability of entropy distribution by fitting a generalized extreme value (GEV) distribution, and we use the cumulative distribution function (CDF) score to indicate if a sample belongs to unknown or known classes. Experimental results on three conventional domain adaptation datasets show outperformance over both CSDA and OSDA benchmarks.

Through our work, we claim the following key contributions. First, we model the tail of entropy distributions with EVT to detect and reject unknown classes. Second, the entropy-based instance-level weighting strategy avoids setting a hard threshold manually and thus is more robust and stable. Third, we have done extensive experiments on three conventional datasets in domain adaptation, which show that our model outperforms benchmarks by a significant margin.

The rest of the chapter is organized as follows. In Section 4.2, we discuss the related literature and methods. In Section 4.3, we explain our method in details on how to solve the OSDA problem. In Section 4.4, we conduct experiments to validate the effectiveness of the proposed method. In Section 4.5, we conclude the chapter by reiterating the main contributions.

4.2. Related Work

Existing works on CSDA typically try to diminish the domain gap by minimizing a statistical divergence between two domains for adaptation or by an adversarial approach. MCD [**90**] utilized two task-specific classifiers to detect the target domain samples which are far from the source domain by maximizing the two classifiers' inconsistency regarding predictions. Based on the mean teacher model [**100**] that was originally proposed for semi-supervised learning tasks, the self-ensemble network [**89**] was proposed to calculate the exponentially moving average of the student model weights and it assigned the weights to the teacher model to reduce the domain gap. There are also a plethora of works on adversarial learning to reduce domain gap. DANN [**88**] reduced the domain gap by introducing a domain discriminator during training process to discriminate between domains, and the domain discriminator is optimized by a specially designed gradient reversal layer. ADDA [**85**] incorporated adversarial training to reduce domain gap by a discriminator to distinguish across domains. The soft labeling method [**79**] utilized a soft cross-entropy loss function to optimize for the domain invariance and thus aligning the source and target domain. WDGRL [**86**] was proposed to minimize the Wasserstein distance across domains adversarially for learning domain invariance. CADA [**101**] aligned

the joint distribution for labels and features across two domain by exploring classifier predictions for adversarial domain adaptation. The proposed method differs from the prior works in that the proposed method focuses on the practical OSDA setting while the prior works assume an identical label space across domains, and thus they cannot recognize the unknown classes samples and the unknown classes will be included during domain adaptation. While our method is able to detect the unknown classes by an instance-level weight based on entropy to avoid negative transfer.

There were also several existing works on OSDA to address the issue of unknown classes. STA [**102**] was proposed to develop a coarse-to-fine progressive separation method for unknown and known classes. OSBP [**92**] forced the generator to either match target samples with source known classes or ignore them in adaptation as unknown samples by adversarial training. Based on the work of SE [**89**] in CSDA, KASE [**93**] modified the adaptation loss and then aligned the target domain with the source domain to address the unknown classes. By factorization and joint separation, D-FRODA [**94**] represented the source and target classes with a shared embedding space for domain adaptation. ATI [**91**] was also proposed to detect the target samples that potentially belong to the known classes by learning a mapping from the source domain to the target domain. We see the difference between the proposed work and the prior works as the existing works usually utilized a hard threshold to recognize and reject the unknown samples, and thus the model accuracy largely depends on how the threshold is set. While the proposed method utilizes entropy to indicate the likelihood of unknown classes and construct a soft instance-level weight, and we further fit an EVT model on tail of entropy distribution to detect unknown classes. Therefore, by avoiding manual thresholding and introducing

EVT to detect unknown classes, the proposed method is more robust and suitable in the OSDA setting, which is also validated in experimental results.

4.3. Methodology

In this section, K denotes the number of known classes, x^s denotes a source sample, x^t denotes a target sample, y^s denotes a source label, X^s denotes the source samples distribution, X^t denotes the target samples distribution, D^s denotes the source domain, D^t denotes target domain, g denotes the feature extractor, clf denotes the K-way classifier to classify samples into one of the K known classes and clf^d denotes the binary domain classifier to classify samples into source or target domain.

4.3.1. Entropy Weighted Domain Adversarial Training

In our work, we train the model in an adversarial manner by a domain classifier clf^d to distinguish samples from source domain and target domain, where clf^d outputs a probability indicating the likelihood of belonging to the target domain. The domain classifier clf^d is optimized based on a binary cross-entropy loss. We train the feature extractor g to maximize the domain classification loss and the domain classifier to minimize the domain classification loss.

Domain classification loss is calculated by

$$L_d = E_{x^s \sim X^s} CE(clf^d(g(x^s)), \mathbf{0}) + E_{x^t \sim X^t} w(x^t) \cdot CE(clf^d(g(x^t)), \mathbf{1})$$

where

$$w(x^t) = \frac{1}{Z} exp\left(- \sum_{c=1}^{K} p_c(x^t) \log p_c(x^t) \right).$$

Please note that $p_c(x^t)$ denotes predicted probability for a target sample x^t for the c-th class, Z is a normalizing constant such that $\sum w\left(x^t\right) = 1$ and $\mathbf{0}$ and $\mathbf{1}$ are vectors containing all 0's and 1's for domain labels.

The feature extractor g is trained to maximize L^d so that the embedding from both domains are similar and the domain classifier clf^d is trained to minimize L^d for a good classification performance. This is to make the feature extractor fool the domain classifier and the domain classifier learns how to perform better from the feature extractor in an adversarial manner.

4.3.2. Entropy Maximization for Source Unknown Classes

In this chapter, we assume that the entropy of classifier predictions is high if the sample is from unknown classes and vice versa, because the classifier clf is optimized to minimize the cross-entropy loss on source known classes. Therefore, the known classes predictions are close to one-hot vectors with low entropy. Based on this assumption, the unknown classes detection task can be tackled if we are able to separate the known classes and unknown classes by their entropy values. In order to force the classifier to predict unknown classes with high entropy, we penalize the model if the prediction is different from the uniform distribution for a source unknown sample, because we want to force the unknown classes predictions as uncertain as possible. The loss function is calculated as

$$L_e = -E_{x^s \sim X_U^s} Ent(p(x^s)).$$

where X_U^s denotes the source known samples distribution. By maximizing the source unknown entropy and minimizing the source known entropy, we can further separate the known and unknown classes with a clear boundary based on their entropy values.

4.3.3. Domain Adversarial Optimization

Denote θ^g as the parameters of the feature extractor g, θ^c as the parameters of the classifier clf and θ^d as the parameters of the domain classifier clf^d. The objective function to minimize is calculated as

$$L = -\lambda_d L_d + \lambda_e L_e + \lambda_c L_c$$

where $L_c = E_{(x^s,y^s)\sim D^s} CE(clf(x^s), y^s)$ is the conventional cross-entropy classification loss on source known classes and $\lambda_d, \lambda_e, \lambda_c$ are weights for loss functions.

To force the domain classifier to minimize the adversarial loss and the feature extractor to maximize the adversarial loss, we seek the saddle point $\hat{\theta}^g, \hat{\theta}^c, \hat{\theta}^d$ of L satisfying the conditions as follows:

$$\begin{aligned} \hat{\theta}^g, \hat{\theta}^c &= \arg\min_{\theta^g,\theta^c} L(\theta^g, \theta^c, \hat{\theta}^d) \\ \hat{\theta}^d &= \arg\max_{\theta^d} L(\hat{\theta}^g, \hat{\theta}^c, \theta^d). \end{aligned} \tag{4.1}$$

On this saddle point, θ^d minimize the domain classification loss L_d, θ^c minimize the conventional classification loss L_c, θ^g maximize the domain classification loss (thus the feature divergence is minimized across two domains) and θ^g minimize the classification loss L_c (thus the features are discriminative).

4.3.4. Target Samples Classification

We propose to fit an GEV using the tail of entropy distribution of source samples. The probability density function for GEV is calculated as

$$PDF(x; loc, scale, conc) = t(x; loc, scale, conc)^{1+conc} \cdot exp(-t(x; loc, scale, conc))/scale$$

where

$$t(x; loc, scale, conc) = (1 + conc \cdot (x - loc)/scale))^{-1/conc}$$

and the CDF is calculated as

$$GEV(x; loc, scale, conc) = exp(-t(x; loc, scale, conc)),$$

where $loc, scale, conc$ are parameters associated with the distribution.

After fitting GEV using source samples' entropy values, for target samples, we estimate the CDF of GEV on entropy distribution, and assign each target sample a score in inference. If the CDF (i.e. probability) is higher than 0.5, then the sample is classified as a unknown sample; otherwise it is fed into the known-class classifier clf to predict into one of the K known classes.

4.4. Experimental Results

We evaluate our method by three conventional benchmark datasets in domain adaptation: Digits, Office-31 [**103**], VisDA [**104**]. In the experiments, the source domain samples are labeled and the target domain samples are not labeled, and the task is to either classify target samples into one of the known classes or reject as unknown. We use

the conventional OSDA metrics OS and OS* for performance measure, which denote the mean accuracy for all $K+1$ classes (all known classes and the additional unknown class) and the mean accuracy for the K known classes, respectively. The hyperparameters are set as $\lambda_d = 0.5, \lambda_e = 1, \lambda_c = 1$.

4.4.1. Digits Experiments

In the Digits experiments, we follow the conventions in OSDA to use USPS [**69**], SVHN [**105**] and MNIST to conduct evaluations, and we perform domain adaptation from SVHN to MNIST, USPS to MNIST and MNIST to USPS. Note that the USPS to MNIST and the MNIST to USPS tasks are easier than the SVHN to MNIST task, since MNIST and USPS both contain 2-dimensional black-and-white images and are hand-written digits, while SVHN contains 3-dimensional RGB images for real-world house numbers collected in Google Street View images. Therefore, the domain gap between SVHN to MNIST is significantly larger than the other two experiments. For consistency, we use digits 0, 1, 2, 3 as known classes, 4, 5, 6 as source unknown classes and 7, 8, 9 as target unknown classes, following the conventional protocols. The same CNN model is used across all methods for fair comparison.

Table 4.1 shows the overall performance comparison, where the model trained on source only performs the worst, which is as expected because it does not align the two domains at all. The CSDA benchmarks DAN and DANN perform slightly better than training only on source data, because they reduce domain misalignment, but they do not reject unknown classes in domain adaptation. The OSDA methods OSBP, STA and KASE outperform the CSDA methods but only improve by a small margin (1.6%,

Table 4.1. Accuracy (in %) on Digits dataset (best in bold). Note that 'AVG' denotes the average across all datasets.

Method	**S-M**		**U-M**		**M-U**		**AVG**	
	OS	OS*	OS	OS*	OS	OS*	OS	OS*
Source only	58.5	63.5	82.3	83.9	82.0	84.1	74.2	77.2
DAN [**87**]	66.2	67.0	86.9	88.0	89.1	90.5	80.7	81.8
DANN [**88**]	66.8	67.4	89.2	88.9	88.9	89.9	81.6	82.0
OSBP [**92**]	62.2	63.9	94.8	94.3	92.7	93.2	83.2	83.8
STA [**102**]	65.2	65.9	94.7	94.5	93.3	94.1	84.4	84.8
KASE [**93**]	67.2	68.1	94.7	95.1	93.6	94.5	85.2	85.9
Ours	**87.9**	**89.2**	**95.4**	**96.3**	**95.2**	**96.5**	**92.8**	**94.0**

2.8%, 3.6%, respectively), from which we can observe the difficulty in the OSDA scenario. While our method achieves 92.8% OS accuracy, outperforming the previous state-of-the-art OSDA method by 9.6%, 8.4% and 7.6% respectively. Our method also outperforms the CSDA benchmarks by 12.1% and 11.2%. In the most difficult task S-M, our method is able to achieve a more than 20% improvement on both OS and OS* accuracy, which again validates the effectiveness of the proposed method in the OSDA scenario. We also experiment on 5 different random seeds and the standard deviations for average OS and OS* are 0.49 and 0.37, indicating the performance is robust and stable. We can also observe that comparing the OS and OS* metrics, OS* is always higher than OS for the same method, meaning the unknown class accuracy is lower than the overall accuracy on known classes only. This points to the future research direction in the OSDA area that the unknown classes should be addressed more on their detection and separation. Our relative OS improvements regarding benchmarks are 25.1%, 15.0%, 13.7%, 11.5%, 10.0%, 8.92%.

Ablation study

In order to have a better understanding on what component in our model contributes most to the performance, we conduct ablation studies by removing components from our model. The ablation study is performed in Table 4.2. Note that in the second experiment (i.e. w/o EVT) we replace EVT by a binary classifier to classify target samples into known/unknown classes and this classifier is trained using source known and unknown samples.

Table 4.2. OS (in %) on Digits dataset without specific parts of our model. Note that 'AVG' denotes the average across all datasets.

Method	**S-M**		**U-M**		**M-U**		**AVG**	
	OS	OS*	OS	OS*	OS	OS*	OS	OS*
Ours w/o reweighting	71.5	73.9	89.4	92.0	87.4	88.7	82.8	84.9
Ours w/o EVT	83.2	84.7	84.9	86.8	88.5	90.1	85.5	87.2
Ours	**87.9**	**89.2**	**95.4**	**96.3**	**95.2**	**96.5**	**92.8**	**94.0**

From Table 4.2 we observe that removing soft reweighting strategy contributes most to the accuracy drop, i.e. the average OS drops by 10.0% without entropy-based reweighting. Removing EVT component is also detrimental to the performance, where the average OS drops by 7.3% without EVT. Including both reweighting and EVT in our model achieves the new state-of-the-art performance in OSDA.

Experiments with less source data

In domain adaptation tasks, usually we assume the source domain contains sufficient amount of labeled data. In order to see how our method performs with less source data, we experiment on limiting the source data to 10%, 25% and 50%. From Table 4.3, both the OS and OS* accuracy increase when we have more source data. The model trained with 10% data achieves comparable performance with the benchmarks, and the model trained with 25% source data already outperforms the previous state-of-the-art OSDA

Table 4.3. Performance of our method on Digits dataset with different percentage of source data.

Method	**S-M**		**U-M**		**M-U**		**AVG**	
	OS	OS*	OS	OS*	OS	OS*	OS	OS*
Source only	58.5	63.5	82.3	83.9	82.0	84.1	74.2	77.2
STA [**102**]	65.2	65.9	94.7	94.5	93.3	94.1	84.4	84.8
KASE [**93**]	67.2	68.1	94.7	95.1	93.6	94.5	85.2	85.9
10% data	73.2	74.6	90.1	91.3	87.5	89.8	83.6	85.2
25% data	79.1	80.9	92.3	93.1	92.0	92.5	87.8	88.8
50% data	85.1	86.3	93.9	95.8	94.6	95.8	91.2	92.6
All data	**87.9**	**89.2**	**95.4**	**96.3**	**95.2**	**96.5**	**92.8**	**94.0**

method. This validates that our method shows robust performance even when less source data are presented.

Sensitivity on hyperparameters

We also experiment on varying the loss function weights hyperparameters to observe their sensitivity. The results are shown in Table 4.4, where in general, the hyperparameter for source classification loss λ_c is less sensitive in that varying its value results in the least accuracy changes, which we postulate is because the source knowledge can already be learned well for a small weight. The hyperparameters for domain discriminator loss λ_d and unknown entropy maximization loss λ_e are more sensitive to the changes where either increasing or decreasing the values will result in an accuracy drop. The combination $\lambda_d = 0.5, \lambda_e = 1.0, \lambda_c = 1.0$ which provides the best accuracy is used as the final hyperparameters across all datasets.

Table 4.4. Model performance on varying loss function weights on Digits dataset.

Hyperparameters	**S-M**		**U-M**		**M-U**		**AVG**	
	OS	OS*	OS	OS*	OS	OS*	OS	OS*
$\lambda_d = 0.25, \lambda_e = 1.0, \lambda_c = 1.0$	86.1	87.3	95.3	95.9	93.7	94.2	91.7	92.5
$\lambda_d = 1.0, \lambda_e = 1.0, \lambda_c = 1.0$	84.4	87.0	94.1	94.8	90.5	92.2	89.7	91.3
$\lambda_d = 0.5, \lambda_e = 0.5, \lambda_c = 1.0$	81.4	83.0	91.1	92.9	91.9	93.1	88.1	89.7
$\lambda_d = 0.5, \lambda_e = 2.0, \lambda_c = 1.0$	83.8	85.7	92.6	94.6	93.7	95.1	90.0	91.8
$\lambda_d = 0.5, \lambda_e = 1.0, \lambda_c = 0.5$	86.8	88.3	94.8	95.9	94.7	95.6	92.1	93.3
$\lambda_d = 0.5, \lambda_e = 1.0, \lambda_c = 2.0$	85.7	87.4	93.2	95.0	94.5	95.5	91.1	92.6
$\lambda_d = 0.5, \lambda_e = 1.0, \lambda_c = 1.0$	87.9	89.2	95.4	96.3	95.2	96.5	92.8	94.0

4.4.2. Office-31 Experiments

We also experiment on the Office-31 dataset which is another frequently used domain adaptation dataset containing three domains: Amazon, DSLR and Webcam. The following 6 domain adaptation tasks are created: Amazon to DSLR, DSLR to Amazon, Amazon to Webcam, Webcam to Amazon, DSLR to Webcam and Webcam to DSLR. Each domain contains 31 classes, and the images are either collected directly from www.amazon.com or they are office environment images taken with different lighting etc. using a webcam or a digital single-lens reflex (DSLR) camera. Therefore, the domain gap between DSLR and Webcam is slightly smaller. Following the same experimental setup as in previous works, we set the 10 common classes with the Office-Caltech dataset [**81**] as the known classes, and the first 10 remaining classes in alphabetical order are set as the unknown classes for source domain and the last 11 classes are set as the unknown classes for target domain. We use the same CNN model AlexNet [**70**] as in previous works across all methods for fair comparison. The experimental results are presented in Table 4.5.

Table 4.5. Accuracy (in %) on Office-31 dataset (best in bold). Note that 'AVG' denotes the average across all datasets. '/' denotes the number is unavailable because the cited paper does not include such experiment and there are no codes publicly available.

Method	**A-D**		**A-W**		**D-A**		**D-W**		**W-A**		**W-D**		**AVG**	
	OS	OS*	OS	OS*	OS	OS*	OS	OS*	OS	OS*	OS	OS*	OS	OS*
Source only	68.5	71.8	59.6	63.2	54.8	58.1	88.2	90.5	49.8	53.2	92.6	92.9	68.9	71.6
DAN [**87**]	78.7	79.5	73.4	74.2	59.6	62.1	87.8	88.6	62.2	64.5	96.5	96.8	76.3	77.6
DANN [**88**]	79.7	80.5	77.2	79.3	55.2	56.1	90.7	91.3	65.7	67.0	97.9	98.3	78.0	78.8
ATI [**91**]	79.8	79.2	77.6	76.5	71.3	70.0	93.5	93.2	76.7	76.5	98.3	99.2	82.9	82.4
OSBP [**92**]	74.7	76.4	72.6	73.1	61.7	63.3	93.3	94.8	79.8	82.5	95.7	96.5	79.6	81.1
D-FRODA [**94**]	87.4	/	78.1	/	73.6	/	94.4	/	77.1	/	98.5	/	84.9	/
KASE [**93**]	86.6	88.1	80.1	80.5	78.6	79.8	95.4	95.9	82.2	83.4	98.9	98.8	87.0	87.8
Ours	**86.7**	**89.9**	**81.3**	**85.4**	**81.3**	**83.7**	**96.4**	**96.8**	**82.9**	**84.8**	**99.1**	**99.5**	**88.4**	**89.8**

From Table 4.5, the CSDA methods generally have accuracy below 80% since they are unable to address the additional unknown classes during adaptation. The OSDA methods outperform the CSDA methods in this experiment with a large margin, where the previous state-of-the-art method KASE achieves 87.0% OS accuracy and 87.8% OS* accuracy, significantly outperforming other methods. While our method shows 88.4% OS accuracy and 89.8% OS* accuracy, consistently beating the previous state-of-the-art method in all domain adaptation experiments for both known and unknown classes. Our relative OS improvements regarding benchmarks are 28.3%, 15.9%, 13.3%, 6.63%, 11.1%, 4.1% and 1.6%.

4.4.3. VisDA Experiments

The VisDA dataset [**104**] is a conventional but more difficult task in domain adaptation. The source domain contains synthetic images and the target domain contains real-world images. For fair comparison, we follow the conventions to set "bicycle," "bus," "car," "motorcycle," "train," "truck" as 6 known categories. In alphabetical order, the first 3

remaining categories "aeroplane," "horse," "knife" are set as source unknown categories and the last 3 remaining categories "person," "plant," "skateboard" are set as target unknown categories.

Table 4.6. Accuracy (in %) on VisDA-2017 (best in bold). 'UNK' denotes the additional unknown class.

Method	**bicycle**	**bus**	**car**	**motorcycle**	**train**	**truck**	**UNK**	**OS**	**OS***
Source only	42.9	65.9	58.9	80.5	80.2	8.7	10.1	49.6	56.2
DANN [**88**]	46.1	69.0	56.2	84.4	82.8	18.2	52.0	58.4	59.5
OSBP [**92**]	51.3	70.7	37.3	87.8	77.3	23.8	88.1	62.3	58.0
Ours	**56.0**	**71.2**	**59.1**	**88.4**	**83.0**	**25.0**	**88.3**	**67.3**	**63.8**

From Table 4.6, the previous state-of-the-art methods DANN and OSBP consistently outperform the model trained only on source data, while our method is able to achieve OS accuracy gains of 17.7%, 8.9%, 5.0% and OS* accuracy gains of 7.6%, 4.3%, 5.8% comparing with the benchmarks, from which the effectiveness of the proposed method is validated again even in this difficult task. Our relative OS improvements regarding benchmarks are 35.7%, 15.2% and 8.0%.

4.5. Conclusion

In our work, we propose an open set domain adaptation method which models the tail of entropy distributions using EVT and utilizes an instance-level reweighting strategy to detect and reject unknown samples. Experiments show that our method achieves the new state-of-the-art performance by beating the existing benchmarks by a large margin for three conventional domain adaptation datasets.

References

[1] G. Carlsson, T. Ishkhanov, V. de Silva, and A. Zomorodian, "On the local behavior of spaces of natural images," *International Journal of Computer Vision*, 2008.

[2] C. Olah, "Neural networks, manifolds, and topology," 2014. [Online]. Available: http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

[3] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, 1967.

[4] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Annual Conference on Neural Information Processing Systems*, 2014.

[5] J. Weston, S. Chopra, and A. Bordes, "Memory networks," *International Conference on Learning Representations*, 2015.

[6] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, "Weakly supervised memory networks," *Annual Conference on Neural Information Processing Systems*, 2015.

[7] L. Breiman, "Random forests," *Machine Learning*, 2001.

[8] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[9] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. M. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Conference on Neural Information Processing Systems*, 2017.

[10] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, 2002.

[11] H. He, Y. Bai, E. A. Garcia, and S. Li, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," *International Joint Conference on Neural Networks*, 2008.

[12] C. Mathy, N. Derbinsky, J. Bento, J. Rosenthal, and J. S. Yedidia, "The boundary forest algorithm for online supervised and unsupervised learning," *Association for the Advancement of Artificial Intelligence Conference*, 2015.

[13] D. Zoran, B. Lakshminarayanan, and C. Blundell, "Learning deep nearest neighbor representations using differentiable boundary trees," *arXiv Repository*, 2017. [Online]. Available: http://arxiv.org/abs/1702.08833

[14] Z. Wang, W. Hamza, and L. Song, "k-nearest neighbor augmented neural networks for text classification," *arXiv Repository*, 2017. [Online]. Available: http://arxiv.org/abs/1708.07863

[15] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine

translation," *Conference on Empirical Methods in Natural Language Processing*, 2014.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, 1997.

[17] A. Karpathy, "The unreasonable effectiveness of recurrent neural networks," 2015. [Online]. Available: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[18] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *Annual Conference on Neural Information Processing Systems*, 2015.

[19] D. Erhan, Y. Bengio, A. C. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *Journal of Machine Learning Research*, 2010.

[20] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. antoine Manzagol, "Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, 2010.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Conference on Computer Vision and Pattern Recognition*, 2016.

[22] S. Hettich and S. D. Bay, "Network intrusion," *The UCI KDD Archive*, 1999. [Online]. Available: http://kdd.ics.uci.edu

[23] J. A. Blackard and D. J. Dean, "Forest covertype," *UCI Machine Learning Repository*, 1998. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/covertype

[24] M. F. Duarte and Y. H. Hu, "Vehicle classification in distributed sensor networks," *Journal of Parallel and Distributed Computing*, 2004.

[25] I.-C. Yeh and C. hui Lien, "The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients." *Expert Systems with Applications*, 2009.

[26] Y. LeCun and C. Cortes, "MNIST handwritten digit database." [Online]. Available: http://yann.lecun.com/exdb/mnist/

[27] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)." [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[28] K. Lang, "Newsweeder: Learning to filter netnews," *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

[29] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 2011.

[30] F. Chollet, "Reuters newswire topics classification," 2015. [Online]. Available: https://keras.io/datasets

[31] C. J. C. Burges, R. Ragno, and Q. V. Le, "Learning to rank with nonsmooth cost functions," *Conference on Neural Information Processing Systems*, 2006.

[32] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, 2008.

[33] D. G. Rajpathak, R. Chougule, and P. Bandyopadhyay, "A domain-specific decision support system for knowledge discovery using association and text mining," *Knowledge and Information Systems*, 2011.

[34] D. G. Rajpathak, "An ontology based text mining system for knowledge discovery from the diagnosis data in the automotive domain," *Computers in Industry*, 2013.

[35] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, 1993.

[36] P. Cimiano, A. Hotho, and S. Staab, "Learning concept hierarchies from text corpora using formal concept analysis," *Journal of Artificial Intelligence Research*, 2005.

[37] M. Girardi and B. Ibrahim, "Using English to retrieve software," *Journal of Systems and Software*, 1995.

[38] S. E. Middleton, N. R. Shadbolt, and D. C. De Roure, "Ontological user profiling in recommender systems," *ACM Transactions on Information Systems*, 2004.

[39] P. Shoval, V. Maidel, and B. Shapira, "An ontology-content-based filtering method," *International Journal of Theories and Applications*, 2008.

[40] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv Repository*, 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[41] G. A. Miller, "Wordnet: A lexical database for English," *Communications of the ACM*, 1995.

[42] J. Lehmann and J. Völker, "Perspectives on ontology learning," *Studies on the Semantic Web*, 2014.

[43] G. Wohlgenannt, "Leveraging and balancing heterogeneous sources of evidence in ontology learning," *The Semantic Web. Latest Advances and New Domains*, 2015.

[44] K. Doing-Harris, Y. Livnat, and S. Meystre, "Automated concept and relationship extraction for the semi-automated ontology management (seam) system," *Journal of Biomedical Semantics*, 2015.

[45] M. A. Yosef, S. Bauer, J. Hoffart, M. Spaniol, and G. Weikum, "Hyena: Hierarchical type classification for entity names," *24th International Conference on Computational Linguistics*, 2012.

[46] I. Pembeci, "Using word embeddings for ontology enrichment," *International Journal of Intelligent Systems and Applications in Engineering*, 2016.

[47] K. Ahmad and L. Gillam, "Automatic ontology extraction from unstructured texts," *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, 2005.

[48] M. Stevenson, Y. Guo, A. Al Amri, and R. Gaizauskas, "Disambiguation of biomedical abbreviations," *Proceedings of the Workshop on Current Trends in Biomedical Natural Language Processing*, 2009.

[49] Y. HaCohen-Kerner, A. Kass, and A. Peretz, "Combined one sense disambiguation of abbreviations," *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, 2008.

[50] C. Li, L. Ji, and J. Yan, "Acronym disambiguation using word embedding," *Association for the Advancement of Artificial Intelligence Conference*, 2015.

[51] A. Ratnaparkhi, "A maximum entropy model for part-of-speech tagging," *Empirical Methods in Natural Language Processing*, 1996.

[52] I. Žliobaité, "Change with delayed labeling: When is it detectable?" *IEEE International Conference on Data Mining Workshops*, 2010.

[53] T. Sethi and M. Kantardzic, "On the reliable detection of concept drift from streaming unlabeled data," *Expert Systems with Applications*, 2017.

[54] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," *Advances in Artificial Intelligence – SBIA*, 2004.

[55] M. B.-G. Jose, J. D. Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavaldà, and R. Morales-bueno, "Early drift detection method," *Fourth international workshop on knowledge discovery from data streams*, 2006.

[56] E. S. Page, "Continuous Inspection Schemes," *Biometrika*, 1954.

[57] A. Bifet and R. Gavaldà, "Learning from time-changing data with adaptive windowing," *Proceedings of the 7th SIAM International Conference on Data Mining*, 2007.

[58] G. J. Ross, N. M. Adams, D. K. Tasoulis, and D. J. Hand, "Exponentially weighted moving average charts for detecting concept drift," *Pattern Recognition Letters*, 2012.

[59] P. Gonçalves Jr, S. Santos, R. Barros, and D. Vieira, "A comparative study on concept drift detectors," *Expert Systems with Applications*, 2014.

[60] E. Lughofer, E. Weigl, W. Heidl, C. Eitzinger, and T. Radauer, "Recognizing input space and target concept drifts in data streams with scarcely labeled and unlabelled instances," *Information Sciences*, 2016.

[61] G. Krempl, I. Žliobaite, D. Brzeziundefinedski, E. Hüllermeier, M. Last, V. Lemaire, T. Noack, A. Shaker, S. Sievi, M. Spiliopoulou, and J. Stefanowski, "Open challenges for data stream mining research," *SIGKDD Explorations Newsletter*, 2014.

[62] G. I. Webb, L. K. Lee, B. Goethals, and F. Petitjean, "Analyzing concept drift and shift from sample data," *Data Mining and Knowledge Discovery*, 2018.

[63] A. Vergari, N. D. Mauro, and F. Esposito, "Visualizing and understanding sum-product networks," *Machine Learning*, 2018.

[64] J. Martens and R. Grosse, "Optimizing neural networks with kronecker-factored approximate curvature," *International Conference on Machine Learning*, 2015.

[65] N. Street and Y. Kim, "A streaming ensemble algorithm (sea) for large-scale classification," *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2001.

[66] M. Harries and N. S. Wales, "Splice-2 comparative evaluation: Electricity pricing," *University of New South Wales, School of Computer Science and Engineering technical report*, 1999.

[67] G. Ditzler and R. Polikar, "Incremental learning of concept drift from streaming imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, 2013.

[68] F. Zamora-Martínez, P. Romeu Guallart, P. Botella-Rocamora, and J. Pardo Albiach, "On-line learning of indoor temperature forecasting models towards energy efficiency," *Energy and Buildings*, 2014.

[69] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.

[70] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Conference on Neural Information Processing Systems*, 2012.

[71] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv Repository*, 2014. [Online]. Available: https://arxiv.org/abs/1409.1556

[72] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[73] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," *Conference on Computer Vision and Pattern Recognition*, 2017.

[74] Y. Zhang, J. Hare, and A. Prügel-Bennett, "Learning to count objects in natural images for visual question answering," *International Conference on Machine Learning*, 2018.

[75] A. Singh, V. Natarajan, M. Shah, Y. Jiang, X. Chen, D. Batra, D. Parikh, and M. Rohrbach, "Towards VQA models that can read," *Conference on Computer Vision and Pattern Recognition*, 2019.

[76] P. Anderson, X. He, C. Buehler, D. Teney, M. Johnson, S. Gould, and L. Zhang, "Bottom-up and top-down attention for image captioning and visual question answering," *Conference on Computer Vision and Pattern Recognition*, 2017.

[77] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2015.

[78] R. Girshick, I. Radosavovic, G. Gkioxari, P. Dollár, and K. He, "Detectron," 2018. [Online]. Available: https://github.com/facebookresearch/detectron

[79] J. Hoffman, E. Tzeng, T. Darrell, and K. Saenko, "Simultaneous deep transfer across domains and tasks," *International Conference on Computer Vision*, 2015.

[80] P. Koniusz, Y. Tas, and F. Porikli, "Domain adaptation by mixture of alignments of second- or higher-order scatter tensors," *Conference on Computer Vision and Pattern Recognition*, 2017.

[81] B. Gong, Y. Shi, F. Sha, and K. Grauman, "Geodesic flow kernel for unsupervised domain adaptation," *Conference on Computer Vision and Pattern Recognition*, 2012.

[82] Y. Guo and M. Xiao, "Cross language text classification via subspace co-regularized multi-view learning," *International Conference on Machine Learning*, 2012.

[83] T. Yao, Y. Pan, C.-W. Ngo, H. Li, and T. Mei, "Semi-supervised domain adaptation with subspace learning for visual recognition," *Conference on Computer Vision and Pattern Recognition*, 2015.

[84] F. Qi, X. Yang, and C. Xu, "A unified framework for multimodal domain adaptation," *ACM Multimedia*, 2018.

[85] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," *Conference on Computer Vision and Pattern Recognition*, 2017.

[86] J. Shen, Y. Qu, W. Zhang, and Y. Yu, "Wasserstein distance guided representation learning for domain adaptation," *Association for the Advancement of Artificial Intelligence Conference*, 2017.

[87] M. Long, H. Zhu, J. Wang, and M. I. Jordan, "Unsupervised domain adaptation with residual transfer networks," *Conference on Neural Information Processing Systems*, 2016.

[88] Y. Ganin and V. S. Lempitsky, "Unsupervised domain adaptation by backpropagation," *International Conference on Machine Learning*, 2015.

[89] G. French, M. Mackiewicz, and M. H. Fisher, "Self-ensembling for visual domain adaptation," *International Conference on Learning Representations*, 2017.

[90] K. Saito, K. Watanabe, Y. Ushiku, and T. Harada, "Maximum classifier discrepancy for unsupervised domain adaptation," *Conference on Computer Vision and Pattern Recognition*, 2018.

[91] P. P. Busto and J. Gall, "Open set domain adaptation," *International Conference on Computer Vision*, 2017.

[92] K. Saito, S. Yamamoto, Y. Ushiku, and T. Harada, "Open set domain adaptation by backpropagation," *European Conference on Computer Vision*, 2018.

[93] Q. Lian, W. Li, L. Chen, and L. Duan, "Known-class aware self-ensemble for open set domain adaptation," *arXiv Repository*, 2019. [Online]. Available: https://arxiv.org/abs/1905.01068

[94] M. Baktash, M. Faraki, T. Drummond, and M. Salzmann, "Learning factorized representations for open-set domain adaptation," *International Conference on Learning Representations*, 2019.

[95] C. Geng, S.-J. Huang, and S. Chen, "Recent advances in open set recognition: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[96] P. Oza and V. Patel, "C2ae: Class conditioned auto-encoder for open-set recognition," *Conference on Computer Vision and Pattern Recognition*, 2019.

[97] S. Dang, Z. Cao, Z. Cui, Y. Pi, and N. Liu, "Open set incremental learning for automatic target recognition," *IEEE Transactions on Geoscience and Remote Sensing*, 2019.

[98] H. Zhang and V. M. Patel, "Sparse representation-based open set recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[99] A. Bendale and T. Boult, "Towards open set deep networks," *Conference on Computer Vision and Pattern Recognition*, 2016.

[100] A. Tarvainen and H. Valpola, "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results," *International Conference on Learning Representations*, 2017.

[101] M. Long, Z. Cao, J. Wang, and M. I. Jordan, "Conditional adversarial domain adaptation," *Conference on Neural Information Processing Systems*, 2018.

[102] H. Liu, Z. Cao, M. Long, J. Wang, and Q. Yang, "Separate to adapt: Open set domain adaptation via progressive separation," *Conference on Computer Vision and Pattern Recognition*, 2019.

[103] K. Saenko, B. Kulis, M. Fritz, and T. Darrell, "Adapting visual category models to new domains," *European Conference on Computer Vision*, 2010.

[104] X. Peng, B. Usman, N. Kaushik, J. Hoffman, D. Wang, and K. Saenko, "Visda: The visual domain adaptation challenge," *arXiv Repository*, 2017. [Online]. Available: https://arxiv.org/abs/1710.06924

[105] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Ng, "Reading digits in natural images with unsupervised feature learning," *Conference on Neural Information Processing Systems*, 2011.