CHAPTER 5
Open-World Learning

Classic supervised learning makes the *closed-world assumption*, meaning that all the test classes have been seen in training [Bendale and Boult, 2015, Fei and Liu, 2016, Fei et al., 2016]. Although this assumption holds in many applications, it is violated in many others, especially in dynamic and open environments, where instances of unexpected classes may appear in testing or applications. That is, the test/application data may contain instances from classes that have not appeared in training. To learn in such an environment, we need *open-world learning* (open-world classification or simply open classification), which has to detect instances of unseen classes during testing or model application, and incrementally learn the new classes to update the existing model without re-training the whole model from scratch. This form of learning is also called *cumulative learning* in Fei et al. [2016] and in the first edition of this book. In computer vision, open-world learning is called *open-world recognition* [Bendale and Boult, 2015, De Rosa et al., 2016].

In fact, open-world learning is a general problem, not limited to supervised learning. It can be broadly defined as learning a model that can perform its intended task and also identify new things that have not been learned before, and then incrementally learn the new things. Open-world learning can occur in different learning scenarios and paradigms. For example, in reading, the system may see a new word that it does not know, and then learns it by looking up the word in the dictionary. In human-machine conversation, the conversation agent may not understand something said by the human user and then asks the user to explain in order to learn it. In this chapter, we focus on open-world supervised learning. Learning during conversation will be discussed in Chapter 8.

Open-world learning basically performs a form of *self-motivated learning* because by recognizing that something new has appeared, the system has the opportunity to learn the new thing. Traditionally, self-motivated learning means that the learner has curiosity that motivates it to explore new territories and to learn new things. In the context of supervised learning, the key is for the system to recognize what it has not seen or learned before. If a learned model cannot recognize any new things, there is no way for the learner to learn new things or to explore by itself other than by being told or instructed by a human user or an external system, which is not ideal for a truly intelligent system. It also has great difficulty to function in a dynamic and open environment.

5.1 PROBLEM DEFINITION AND APPLICATIONS

Open-world Learning is defined as follows [Bendale and Boult, 2015, Fei et al., 2016].

1. At a particular time point, the learner has built a multi-class classification model F_N based on all past N classes of data $\mathcal{D}^p = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N\}$ with their corresponding class labels $\mathcal{Y}^N = \{l_1, l_2, \ldots, l_N\}$. F_N is able to classify each test instance to either one of the known classes $l_i \in \mathcal{Y}^N$ or reject it and put it in a rejected set R, which may include instances from one or more new or unseen classes in the test set.

2. The system or a human user identifies the hidden unseen classes C in R, and collects training data for the unseen classes.

3. Assume that there are k new classes in C that have enough training data. The learner incrementally learns the k classes based on their training data. The existing model F_N is updated to produce the new model F_{N+k}.

Open-world learning is a form of lifelong learning (LL) because it conforms to the definition of LL in Chapter 1. Specifically, the new learning task $\mathcal{T}_{N+1}$ is to build a multi-class open classifier based on all the past and the current classes. The knowledge base (KB) contains the past model F_N and possibly all the past training data.

We should note that the third task of learning new classes incrementally here is different from traditional *incremental class learning* (ICL) studied in different areas because traditional ICL still learns in the closed-world (i.e., it does not perform unseen class rejection) although it can add new classes incrementally to the classification system without re-training the whole model from scratch.

Let us see some example applications. For example, we want to build a greeting robot for a hotel. At any point in time, the robot has learned to recognize all existing hotel guests. When it sees an existing guest, it can call his/her name and chat. At the same time, it must also detect any new guests that it has not seen before. On seeing a new guest, it can say hello, ask for his/her name, take many pictures, and learn to recognize the guest. Next time when it sees the person again, it can call his/her name and chat like an old friend. The scenario in self-driving cars is very similar as it is very hard, if not impossible, to train a system to recognize every possible object that may appear on the road. The system has to recognize objects that it has not learned before and learn them during driving (possibly through interactions with the human passenger) so that when it sees the objects next time, it will have no problem recognizing them.

Fei et al. [2016] gave another example in text classification. The 2016 presidential election in the U.S. was a hot topic on social media, and many social science researchers relied on collected online user discussions to carry out their research. During the campaign, every new proposal made by a candidate was followed by a huge amount of discussions in the social media. A multi-class classifier is thus needed to organize the discussions. As the campaign went on, the initially built classifier inevitably encounters new topics (e.g., Donald Trump's plan for immigration reform or Hillary Clinton's proposal for tax increase) that had not been covered in previous training. In this case, the classifier should first recognize these new topics when they occur rather than classify them into some existing classes or topics. Second, after enough training examples of the new topics are collected, the existing classifier should incorporate the new classes or topics incrementally in a manner that does not require retraining the entire classification system from scratch.

Bendale and Boult [2015] made an attempt to solve the open-world learning problem (which was called open-world recognition in their paper) for image classification. Its method is called *Nearest Non-Outlier* (NNO), which is adapted from the traditional *Nearest Class Mean* (NCM) method for image classification using a metric learning technique proposed by Mensink et al. [2013]. In NCM, each image is represented as a feature vector and each class is represented by the class mean computed using the feature vectors of all the images in the class. In testing, each test image's feature vector is compared with each class mean and is assigned the class with the nearest class mean. However, this method cannot perform unseen class rejection. NNO enables rejection. For incremental learning, it simply adds the new class mean to the existing class mean set. The rejection capability of NNO was improved in Bendale and Boult [2016]. The new method, called OpenMax, is based on deep learning, which adapts the traditional SoftMax classification scheme to enable rejection by introducing a new model layer (also called OpenMax) to estimate the probability of an input being from an unseen class. However, its training needs examples from some unseen classes (not necessarily the test unseen classes) to tune the parameters. In the next two sections, we discuss two other methods. It was shown in Shu et al. [2017a] that its DOC method outperforms OpenMax for both open text and open image classifications without requiring any training unseen class examples.

5.2 CENTER-BASED SIMILARITY SPACE LEARNING

Fei et al. [2016] proposed a technique to perform open-world classification based on a center-based similarity space learning method (called *CBS learning*), which we discuss below. We first discuss its training process for learning a new class incrementally and then its testing process, which is able to classify test instances to known/seen classes and also detect unseen class instances.

5.2.1 INCREMENTALLY UPDATING A CBS LEARNING MODEL

This sub-section describes incremental training in CBS learning, which was inspired by human concept learning. Humans are exposed to new concepts all the time. One way we learn a new concept is perhaps by searching from the already known concepts for ones that are similar to the new concept, and then trying to find the difference between these known concepts and the new one without using all the known concepts. For example, assume we have already learned the concepts like "movie," "furniture," and "soccer." Now we are presented with the concept of "basketball" and its set of documents. We find that "basketball" is similar to "soccer," but very different from "movie" and "furniture." Then we just need to *accommodate* the new concept "basketball" into our old knowledge base by focusing on distinguishing the "basketball" and "soccer" concepts, and do not need to worry about the difference between "basketball" and "movie" or "furniture," because the concepts of "movie" and "furniture" can easily tell that documents from "basketball" do not belong to either of them.

Fei et al. [2016] adopted this idea and used the 1-vs.-rest strategy of SVM for incremental learning of multiple classes (or concepts). Before the new class l_{N+1} arrives, the learning system has built a classification model F_N, which consists of a set of N 1-vs.-rest binary classifiers $F_N = \{f_1, f_2, \ldots, f_N\}$, for the past N classes using their training sets $\mathcal{D}^p = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N\}$ and corresponding class labels $\mathcal{Y}^N = \{l_1, l_2, \ldots, l_N\}$. Each f_i is a binary classifier built using the CBS learning method (see Section 5.2.3) for identifying instances of class l_i. When a new dataset $\mathcal{D}_{N+1}$ of class l_{N+1} arrives, the system goes through the following two steps to update the classification model F_N to build a new model F_{N+1} in order to be able to classify test data or instances of all existing classes in $\mathcal{Y}^{N+1} = \{l_1, l_2, \ldots, l_N, l_{N+1}\}$ and recognize any unseen class C_0 of documents.

1. Searching for a set of classes SC that are similar to the new class l_{N+1}.

2. Learning to separate the new class l_{N+1} and the previous classes in SC.

For step 1, the similarity between the new class l_{N+1} and the previous ones $\{l_1, l_2, \ldots, l_N\}$ is computed by running each of the 1-vs.-rest past binary classifiers f_i in $F_N = \{f_1, f_2, \ldots, f_N\}$ to classify instances in $\mathcal{D}_{N+1}$. The classes of those past binary classifiers that accept (classify as positive) a certain number/percentage λ_{sim} of instances from $\mathcal{D}_{N+1}$ are regarded as similar classes and denoted by SC.

Step 2 of separating the new class l_{N+1} and classes in SC involves two sub-steps: (1) building a new binary classifier f_{N+1} for the new class l_{N+1} and (2) updating the existing classifiers for the classes in SC. It is intuitive to build f_{N+1} using $\mathcal{D}_{N+1}$ as the positive training data and the data of the classes in SC as the negative training data. The reason for updating classifiers in SC is that the joining of class l_{N+1} confuses those classifiers in SC. To re-build each classifier, the system needs to use the original negative data employed to build the existing classifier f_i and the new data $\mathcal{D}_{N+1}$ as the new negative training data. The reason that the old negative training data is still used is because the new classifier still needs to separate class l_i from those old classes.

The detailed algorithm is given in Algorithm 5.6, which incrementally learns a new class with its data $\mathcal{D}_{N+1}$. Line 1 initializes SC to the empty set. Line 3 initializes the variable CT (count) to record the number of instances in $\mathcal{D}_{N+1}$ that will be classified as positive by classifier f_i. Lines 4–9 use f_i to classify each instance in $\mathcal{D}_{N+1}$ and record the number of instances that are classified (or accepted) as positive by f_i. Lines 10–12 check whether there are too many instances in $\mathcal{D}_{N+1}$ that have been classified as positive by f_i to render class l_i as similar to class l_{N+1}. λ_{sim} is a threshold controlling how many percents of instances in $\mathcal{D}_{N+1}$ should be classified to class l_i before considering l_i as similar/close to class l_{N+1}. Lines 14–17 build a new classifier f_{N+1} and update all the classifiers for classes in *SC*.

Algorithm 5.6 Incremental Class Learning

Input: classification model $F_N = \{f_1, f_2, \ldots, f_N\}$, past datasets $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N\}$, new dataset $\mathcal{D}_{N+1}$, similarity threshold λ_{sim}.
Output: classification model $F_{N+1} = \{f_1, \ldots, f_N, f_{N+1}\}$

```
1:  $SC = \emptyset$
2:  for each classifier $f_i \in F_N$ do
3:     $CT = 0$
4:     for each test instance $x_j \in \mathcal{D}_{N+1}$ do
5:        $class = f_i(x_j)$ // classify document $x_j$ using $f_i$
6:        if $class = l_i$ then
7:           $CT = CT + 1$ // wrongly classified
8:        end if
9:     end for
10:    if $CT > \lambda_{sim} \times |\mathcal{D}_{N+1}|$ then
11:       SC = SC $\cup\{l_i\}$
12:    end if
13: end for
14: Build $f_{N+1}$ and add it to $F_{N+1}$
15: for each $f_i$ of class $l_i \in SC$ do
16:    Update $f_i$
17: end for
18: Return $F_{N+1}$
```

In summary, the learning process uses the set *SC* of similar classes to the new class l_{N+1} to control both the number of binary classifiers that need to be built/updated and also the number of negative instances used in building the new classifier f_{N+1}. It thus greatly improves the efficiency compared to building a new multi-class classifier F_{N+1} from scratch.

Combining the above incremental learning process and the underlying classifier *cbsSVM* discussed in Section 5.2.3, the new learner, called *CL-cbsSVM* (*CL* stands for *Cumulative Learning*, a name used in Fei et al. [2016] for open-world learning) is able to tackle both challenges in incremental learning.

5.2.2 TESTING A CBS LEARNING MODEL

To test the new classification model $F_{N+1} = \{f_1, f_2, \ldots, f_N, f_{N+1}\}$, the standard technique of combining the set of 1-vs.-rest binary classifiers to perform multi-class classification is followed with a rejection option for the unknown. As output scores from different SVM classifiers are not comparable, the SVM scores for each classifier are first converted to probabilities based on a variant of Platt's algorithm [Platt et al., 1999], which is supported in LIBSVM [Chang and Lin, 2011]. Let $P(y|x)$ be a probabilistic estimator, where $y \in Y^{N+1}$ $(= \{l_1, l_2, \ldots, l_N, l_{N+1}\})$ is a class label and x is the feature vector of a test instance. Let θ $(= 0.5)$ be the decision threshold, y^* be the final predicted class for x, and C_0 be the label for the unknown. Classification of the test instance x is done as follows:

$$y^* = \begin{cases} \text{argmax}_{y \in Y^{N+1}} P(y|x) & \text{if } P(y|x) \geq \theta \\ C_0 & \text{otherwise} \end{cases} . \tag{5.1}$$

The idea is that for the test instance x, each binary classifier $f_i \in F_{N+1}$ is used to produce a probability $P(l_i|x)$. If none of the probabilities is greater than θ (= 0.5), the document represented by x is regarded as unseen/unknown and belonging to C_0; otherwise it is classified to the class with the highest probability.

5.2.3 CBS LEARNING FOR UNSEEN CLASS DETECTION

This subsection describes the core CBS learning method, which performs binary classification focusing on identifying positive class documents and also has the ability to detect unseen classes or classifying them as not positive. It provides the base learning method for open-world learning above [Fei et al., 2016]. The learning method is based on the idea of reducing the *open space risk* while balancing the *empirical risk* in learning. Classic learners define and optimize over empirical risk, which is measured on the training data. For open learning, it is crucial to consider how to extend the classic model to capture the risk of the unknown by preventing over-generalization. To tackle this problem, Scheirer et al. [2013] introduced the concept of *open space risk*. Below, we first discuss the open space risk management strategy in Fei et al. [2016], and then apply an SVM-based CBS learning method as the solution toward managing the open space risk. The basic idea of CBS learning is to find a "ball" (decision boundary) to cover the positive class data area. Any document falling outside of the "ball" is considered not positive. Although CBS learning only performs binary classification, applying the 1-vs.-rest method described in Section 5.2.2 gives a multi-class CBS classification model, which is called cbsSVM in Fei et al. [2016].

Open Space Risk

Consider the risk formulation for open image recognition in Scheirer et al. [2013], where apart from empirical risk, there is risk in labeling the open space (space away from positive training examples) as "positive" for any unknown class. Due to lack of information of a classification function on the open space, open space risk is approximated by a relative Lebesgue measure [Shackel, 2007]. Let S_o be a large ball of radius r_o that contains both the positively labeled open space O and all of the positive training examples; and let f be a measurable classification function, where $f_y(x) = 1$ means recognizing x as belonging to class y of interest and $f_y(x) = 0$ otherwise. In our case, y is simply any class of interest l_i.

In Fei et al. [2016], O is defined as the positively labeled area that is sufficiently far from the center of the positive training examples. Let $B_{r_y}(cen_y)$ be a closed ball of radius r_y centered around the center cen_y of positive class y, which, ideally, should tightly cover all positive examples of class y only; S_o be a larger ball $B_{r_o}(cen_y)$ of radius r_o with the same center cen_y. Let classification function $f_y(x) = 1$ for $x \in B_{r_o}(cen_y)$, and $f_y(x) = 0$ otherwise. Also let q be the positive half space defined by a binary SVM decision hyperplane Ω obtained using positive and negative training examples. We also define the size of ball B_{r_o} to be bounded by Ω, $B_{r_o} \cap q = B_{r_o}$. Then the positive open space is defined as $O = S_o - Br_y(cen_y)$. S_o needs to be determined during learning for the positive class.

This open-space formulation greatly reduces the open space risk compared to traditional SVM and 1-vs.-Set Machine in Scheirer et al. [2013]. For traditional SVM, classification function $f_y^{SVM}(x) = 1$ when $x \in q$, and its positive open space is approximately $q - B_{r_y}(cen_y)$, which is only bounded by the SVM decision hyperplane Ω. For 1-vs.-Set Machine in Scheirer et al. [2013], $f_y^{1-vs-set}(x) = 1$ when $x \in g$, where g is a slab area with thickness δ bounded by two parallel hyperplanes Ω and Ψ $(\Psi || \Omega)$ in q. And its positive open space is approximately $g - g \cap B_{r_y}(cen_y)$. Given open-space formulations of the traditional SVM and 1-vs.-Set Machine, we can see that both methods label an unlimited area as the positively labeled space, while Fei et al. [2016] reduces it to a bounded area of a "ball."

Given the open space definition, the question is how to estimate S_o for the positive class. Fei et al. [2016] used the center-based similarity space learning (CBS learning), which transforms the original document space to a similarity space. The final classification is performed in the CBS space. Below, we introduce CBS learning and briefly discuss why it is suitable for the problem.

Center-Based Similarity Space Learning

Let $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ be the set of training examples, where x_k is the feature vector (e.g., with unigram features) representing a document and $y_k \in \{1, -1\}$ is its class label. This feature vector is called a document space vector (or *ds-vector*). Traditional classification directly uses $\mathcal{D}$ to build a binary classifier. However, CBS learning transforms each ds-vector x_k (no change to its class label) to a center-based similarity space feature vector (CBS vector) $cbs\text{-}v_k$. Each feature in $cbs\text{-}v_k$ is a similarity between a center c_j of the positive class documents and x_k.

To make CBS learning more effective by generating more similarity features, multiple document space representations or feature vectors (e.g., one based on unigrams and one based on bigrams) can be used to represent each document, which results in multiple centers for the positive documents. There can also be multiple document similarity functions used to compute similarity values. The detailed learning technique is as follows.

For a document x_k, we have a set R_k of p ds-vectors, $R_k = \{d_1^k, d_2^k, \ldots, d_p^k\}$. Each ds-vector d_j^k denotes one document space representation of the document x_k, e.g., unigram representation or bigram representation. Then the centers of positive training documents can be computed, which are represented as a set of p centroids $\mathcal{C} = \{c_1, c_2, \ldots, c_p\}$. Each c_j corresponds to one document space representation in R_k. The Rocchio method in information retrieval [Manning et al., 2008] is used to compute each center c_j (a vector), which uses the corresponding ds-vectors of all training positive and negative documents:

$$c_j = \frac{\alpha}{|\mathcal{D}_+|} \sum_{x_k \in \mathcal{D}_+} \frac{d_j^k}{\left\| d_j^k \right\|} - \frac{\beta}{|\mathcal{D} - \mathcal{D}_+|} \sum_{x_k \in \mathcal{D} - \mathcal{D}_+} \frac{d_j^k}{\left\| d_j^k \right\|} \ , \tag{5.2}$$

where $\mathcal{D}_+$ is the set of documents in the positive class and $|.|$ is the size function. α and β are parameters. It is reported that using the popular *tf-idf* (*term frequency and inverse document frequency*) representation, $\alpha = 16$ and $\beta = 4$ usually work well [Buckley et al., 1994]. The subtraction is used to reduce the influence of those terms that are not discriminative (i.e., terms appearing in both classes).

Based on R_k for a document x_k (in both training and testing) and the previously computed set $\mathcal{C}$ of centers using the training data, we can transform a document x_k from its document space representations R_k to one center-based similarity space vector $cbs\text{-}v_k$ by applying a similarity function Sim on each element d_j^k of R_k and its corresponding center c_j in $\mathcal{C}$:

$$cbs\text{-}v_k = Sim(R_k, \mathcal{C}) \ . \tag{5.3}$$

Sim can contain a set of similarity measures. Each measure m is applied to p document representations d_j^k in R_k and their corresponding centers c_j in $\mathcal{C}$ to generate p similarity features (cbs-features) in $cbs\text{-}v_k$.

For ds-features, unigrams and bigrams with tf-idf weighting were used as two document representations. The five similarity measures in Fei and Liu [2015] were applied to measure the similarity of two vectors. Based on the CBS space representation, SVM is applied to produce a binary CBS classifier f_y.

Why Does CBS Learning Work?

We now briefly explain why CBS learning gives a good estimate to S_o. Due to using similarities as features, CBS learning generates a boundary to separate the positive and negative training data in the similarity space. Since similarity has no direction (or it covers all directions), the boundary in the similarity space is essentially a "ball" encompassing the positive class training data in the original document space. The "ball" is an estimate of S_o based on those similarity measures.

5.3 DOC: DEEP OPEN CLASSIFICATION

This section describes a deep learning based classification method called DOC [Shu et al., 2017a], which performs only classification and unseen class instance rejection, but does not do incremental learning of new classes. DOC is based on CNN [Collobert et al., 2011, Kim, 2014] and is augmented with a 1-vs.-rest final Sigmoid layer and Gaussian fitting for classification. This algorithm has been shown to perform better than many existing methods in both open-world text classification and open-world image classification.

5.3.1 FEED-FORWARD LAYERS AND THE 1-VS.-REST LAYER

The DOC system (given in Figure 5.1) is a variant of the CNN architecture [Collobert et al., 2011] for text classification [Kim, 2014].[1] The first layer embeds words in document $\mathbf{x}$ into dense vectors. The second layer performs convolution over dense vectors using different filters of varied sizes. Next, the max-over-time pooling layer selects the maximum values from the results of the convolution layer to form a k-dimension feature vector $\boldsymbol{h}$. Then $\boldsymbol{h}$ is reduced to a N-dimension vector $\boldsymbol{d} = d_{1:N}$ (N is the number of training/seen classes) via two fully connected layers and one intermediate ReLU activation layer:

$$\boldsymbol{d} = \boldsymbol{W}'(\mathrm{ReLU}(\boldsymbol{W}\boldsymbol{h} + \boldsymbol{b})) + \boldsymbol{b}' \ , \tag{5.4}$$

where $\boldsymbol{W} \in \mathbb{R}^{r \times k}$, $\boldsymbol{b} \in \mathbb{R}^{r}$, $\boldsymbol{W}' \in \mathbb{R}^{N \times r}$, and $\boldsymbol{b}' \in \mathbb{R}^{N}$ are trainable weights; r is the output dimension of the first fully connected layer. The output layer of DOC is a 1-vs.-rest layer applied to $d_{1:N}$, which allows rejection. We describe it next.

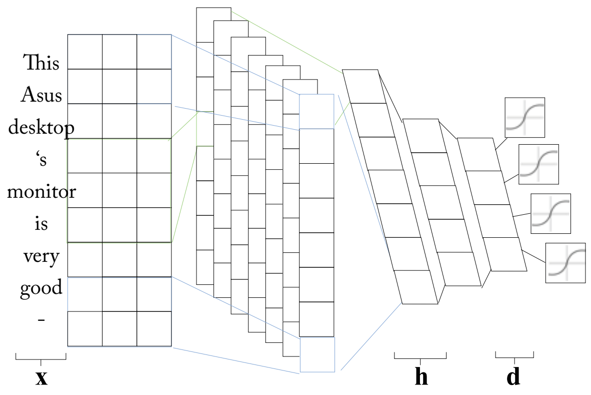

Figure 5.1: Overall network of DOC.

Traditional multi-class classifiers [Bendale and Boult, 2016, Goodfellow et al., 2016] typically use softmax as the final output layer, which does not have the rejection capability since the

[1] https://github.com/alexander-rakhlin/CNN-for-Sentence-Classification-in-Keras

probability of prediction for each class is normalized across all training/seen classes. Instead, a 1-vs.-rest layer is built containing N Sigmoid functions for N seen classes. For the i-th Sigmoid function corresponding to class l_i, DOC takes all examples with $y = l_i$ as positive examples and all the rest examples for $y \neq l_i$ as negative examples.

The model is trained with the objective of summation of all log loss of the N Sigmoid functions on the training data D:

$$\begin{aligned} \text{Loss} = \sum_{i=1}^{N}\sum_{j=1}^{n} & -\mathbb{I}(y_j = l_i)\log p(y_j = l_i) \\ & -\mathbb{I}(y_j \neq l_i)\log(1 - p(y_j = l_i)) \ , \end{aligned} \tag{5.5}$$

where $\mathbb{I}$ is the indicator function and $p(y_j = l_i) = \text{Sigmoid}(d_i^j)$ is the probability output from ith sigmoid function on the jth document's ith-dimension of $\boldsymbol{d}$.

During testing, we reinterpret the prediction of N Sigmoid functions to allow rejection, as shown in Equation (5.6). For the i-th Sigmoid function, we check if the predicted probability Sigmoid(d_i) is less than a threshold t_i belonging to class l_i. If all predicted probabilities are less than their corresponding thresholds for a test example, the example is rejected; otherwise, its predicted class is the one with the highest probability:

$$\hat{y} = \begin{cases} \textit{reject}, \ \text{if Sigmoid}(d_i) < t_i, \forall l_i \in \mathcal{Y}; \\ \text{argmax}_{l_i \in \mathcal{Y}} \text{Sigmoid}(d_i), \ \text{otherwise} \ . \end{cases} \tag{5.6}$$

When DOC was published, OpenMax [Bendale and Boult, 2016] was the state-of-the-art. It uses a classification network and add to it the rejection capability by utilizing the logits that are trained via the closed-world softmax function. One assumption of OpenMax is that examples with equally likely logits are more likely from the unseen or rejection class, which can be examples that are hard to classify. It also requires validation examples from the unseen/rejection class to tune the hyperparameters. In contrast, DOC uses the 1-vs.-rest sigmoid layer to provide a representation of all other classes (the rest of the seen classes and unseen classes), and to enable the 1 class to form a good boundary. Experimental results in Shu et al. [2017a] show that this basic DOC is already better than OpenMax. DOC is further improved by tightening the decision boundaries, which we discuss next.

5.3.2 REDUCING OPEN-SPACE RISK

Sigmoid function usually uses the default probability threshold of $t_i = 0.5$ for classification of each class i. But this threshold does not consider potential open space risks from unseen (rejection) class data. We can improve the boundary by increasing t_i. We use Figure 5.2 to illustrate. The x-axis represents d_i and y-axis is the predicted probability $p(y = l_i|d_i)$. The sigmoid function tries to push positive examples (belonging to the i-th class) and negative examples (belonging to the other seen classes) away from the y-axis via a high gain around $d_i = 0$, which serves as the default decision boundary for d_i with the probability threshold $t_i = 0.5$. As demonstrated by those three circles on the right-hand side of the y-axis, during testing, unseen class examples (circles) can easily fill in the gap between the y-axis and those dense positive (+) examples, which may reduce the recall of rejection and the precision of the i-th seen class prediction. Obviously, a better decision boundary is at $d_i = T$, where the decision boundary more closely "wrap" those dense positive examples with the probability threshold $t_i \gg 0.5$. Note that only t_i is used in classification decision making in this work and T is not used.

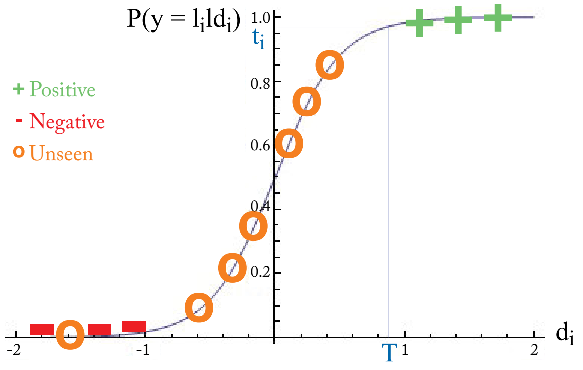

Figure 5.2: Open-space risk of sigmoid function and desired decision boundary $d_i = T$ and probability threshold t_i.

To obtain a better t_i for each seen class i-th, we use the idea of outlier detection in statistics.

1. Assume the predicted probabilities $p(y = l_i|\mathbf{x}_j, y_j = l_i)$ of all training data of each class i follow one half of the Gaussian distribution (with mean $\mu_i = 1$), e.g., the three positive points in Figure 5.2 projected to the y-axis (we don't need d_i). We then artificially create the other half of the Gaussian distributed points (≥ 1): for each existing point $p(y = l_i|\mathbf{x}_j, y_j = l_i)$, we create a mirror point $1 + (1 - p(y = l_i|\mathbf{x}_j, y_j = l_i)$ (not a probability) mirrored on the mean of 1.

2. Estimate the standard deviation σ_i using both the existing points and the created points.

3. In statistics, if a value/point is a certain number (α) of standard deviations away from the mean, it is considered an outlier. We thus set the probability threshold $t_i = \max(0.5, 1 - \alpha\sigma_i)$. The commonly used number for α is 3, which also works well in our experiments.

Note that due to Gaussian fitting, different class l_i can have a different classification threshold t_i.

5.3.3 DOC FOR IMAGE CLASSIFICATION

DOC was originally proposed for open-world text classification. It was also later experimented for image classification and shown to perform very well [Shu et al., 2018], better than OpenMax [Bendale and Boult, 2016], which was designed for open image classification.

The evaluation used two publicly available image datasets: MNIST and EMNIST.

(1) **MNIST**[2]: MNIST is a well-known database of handwritten digits (10 classes), which has a training set of 60,000 examples, and a test set of 10,000 examples. In the experiment, 6 classes were used as the set of seen classes and the rest 4 classes were used as unseen classes.

(2) **EMNIST**[3] [Cohen et al., 2017]: EMNIST is an extension of MNIST to commonly used characters such as English alphabet. It is derived from the NIST Special Database 19. In the evaluation, EMNIST Balanced dataset with 47 balanced classes were used. It has a training set of 112,800 examples and a test set of 18,800 examples. 33 classes were used as the set of seen classes and another 10 classes were used as the unseen classes.

In Shu et al. [2018], DOC is compared with OpenMax [Bendale and Boult, 2016].[4] Both systems are based on deep learning. The results shown that DOC markedly outperforms OpenMax.

5.3.4 UNSEEN CLASS DISCOVERY

At the beginning of this chapter, we saw that in the second task of open-world learning a system or a human user identifies the hidden unseen classes in the rejected instances before they are incrementally learned in the third task. In Shu et al. [2018], an attempt was made to solve this problem automatically. In all previous works, this was done manually. The task is called *unseen class discovery*. The idea in Shu et al. [2018] is to transfer the class similarity knowledge learned from the seen classes to the hidden unseen classes. The transferred similarity knowledge is then used by a hierarchical clustering algorithm to cluster the rejected instances/examples to discover the hidden classes in the rejected instances. Note that this transfer of knowledge is from supervised learning to unsupervised learning.

This proposed transfer is warranted because we humans seem to group things based on our prior knowledge of what might be considered similar or different. For example, if we are given two objects and are asked whether they are of the same class/category or of different classes given some context, most probably we can tell. Why is that the case? We believe that we have learned in the past what are considered to be of the same class or of different classes in a knowledge context. The knowledge context here is important. For example, we have learned to recognize some breeds of dogs, which forms the knowledge context. When we are given two new/unseen breeds of dogs, we probably know that they are of different breeds. If we are given many different dogs from each of the two breeds, we probably can cluster them into two clusters. However, if

[2]http://yann.lecun.com/exdb/mnist/
[3]https://www.nist.gov/itl/iad/image-group/emnist-dataset
[4]https://github.com/abhijitbendale/OSDN

our previous knowledge only has classes such as dog, chicken, pig, cow, and sheep and we are given two different but unseen breeds of dogs, we probably will say that they are of the same kind/class and are dogs. However, if we are given a tiger and a rabbit, we will probably tell that they are from different classes.

To solve this problem, Shu et al. [2018] proposed a Pairwise Classification Network (PCN) to learn a binary classifier to predict whether two given examples are from the same class or different classes, i.e., $g(\mathbf{x}_p, \mathbf{x}_q)$. The positive training data of PCN consists of a set of pairs of intra-class (same class) examples, and the negative training data consists of a set of pairs of inter-class (different classes) examples all from seen classes. A hierarchical clustering method then uses the function $g(\mathbf{x}_p, \mathbf{x}_q)$ (which can be regarded as a distance/similarity function) to find the number of hidden classes (clusters) in the unseen/rejected class examples. Further details can be found in Shu et al. [2018].

5.4 SUMMARY AND EVALUATION DATASETS

As AI systems such as self-driving cars, mobile robots, chatbots, and personal intelligent assistants are increasingly working in real-life open environments and interacting with humans and/or automated systems, open-world learning is becoming increasingly important. An open-world learner should be able to detect new things that it has not seen before and learn them incrementally to become more and more knowledgeable. To some extent, we can regard such a learner as self-motivated because it actively identifies unseen objects and learns them to become more and more knowledge. Open-world learning is still highly challenging and needs a great deal of future research.

Although in this chapter we only discussed open-world learning in the supervised learning setting, it can be viewed from a broad perspective of detecting things unknown and learning the unknown things. It thus can be applied to any type of learning. For example, the learning method presented in Chapter 8, which continually spots and learns new knowledge in human-machine conversations, can also be seen as a form of open-world learning.

Fei et al. [2016] evaluated their method using the 100-products Amazon review dataset created by Chen and Liu [2014b][5] and the popular text classification dataset 20-Newsgroup.[6,7] The 100-products Amazon review dataset contains Amazon reviews from 100 different types of products. Each type of product (or domain) has 1,000 reviews. The 20-Newsgroup dataset contains news articles of 20 different topics. Each topic has about 1,000 articles. Shu et al. [2018] used image datasets MNIST[8] and EMNIST.[9] ImageNet[10] and its derivative datasets can also be used.

[5]https://www.cs.uic.edu/~zchen/downloads/KDD2014-Chen-Dataset.zip
[6]http://qwone.com/~jason/20Newsgroups/
[7]https://archive.ics.uci.edu/ml/datasets/Twenty+Newsgroups
[8]http://yann.lecun.com/exdb/mnist/
[9]https://www.nist.gov/itl/iad/image-group/emnist-dataset
[10]http://image-net.org/