A Framework for Deep Quantification Learning

Lei Qi[1], Mohammed Khaleel[1], Wallapak Tavanapong[1], Adisak Sukul[1], and David Peterson[1]

Iowa State University, Ames, IA 50011, USA
{leiqi,mkhaleel,tavanapo,adisak,daveamp}@iastate.edu

Abstract. A quantification learning task estimates class ratios or class distribution given a test set. Quantification learning is useful for a variety of application domains such as commerce, public health, and politics. For instance, it is desirable to estimate the proportion of customer satisfaction in different aspects to improve customer relationships with less effort. We formulate the quantification learning problem as a maximum likelihood problem and propose the first end-to-end Deep Quantification Network (DQN) framework. DQN jointly learns quantification feature representations and directly predict the class distribution. Compared to classification-based quantification methods, DQN avoids three separate steps: classification of individual instances, calculation of the predicted class ratios, and class ratio adjustment to account for classification errors. We evaluated DQN on four public datasets, ranging from movie and product reviews to multi-class news. We compared DQN against six existing quantification methods and conducted a sensitivity analysis of DQN performance. Compared to the best existing method in our study, (1) DQN reduces Mean Absolute Error (MAE) by about 35%. (2) DQN uses around 40% less training samples to achieve a comparable MAE.

Keywords: Quantification Learning · Class Distribution Estimate · Deep Learning.

1 Introduction

In various problem domains, it is important to estimate class ratios (class prevalence) of a subject of interest. For instance, in commerce, knowing the prevalence of customer complaints in different aspects (e.g., packaging, durability, delivery and after sale service) of a product is key to improve the product and customer experience. In politics, knowing voters' proportional interest in different policy areas (e.g., healthcare, education) is useful to improve political candidates' campaign strategies to attract these voters. In healthcare, knowing the proportion of residents in different age groups affected by a specific disease is vital for determining appropriate prevention and treatment responses. Research in quantification learning has been conducted in various disciplines, leading to different terminologies, such as prior probability shift [1], prevalence estimation [2], or class ratio estimation [3]. The earliest work dated back to the application in

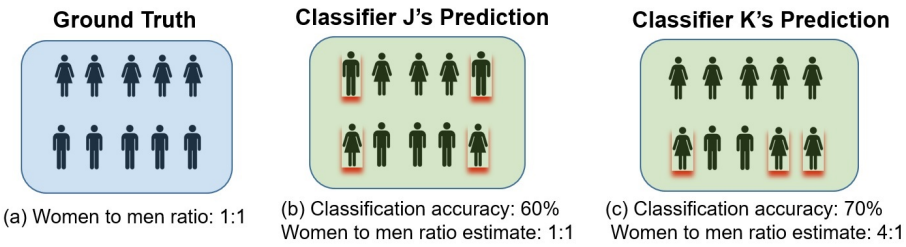

Fig. 1. Mismatch in performance when using classification for quantification

screening tests in epidemiology [4]. In this paper, we use class ratios and class distribution interchangeably.

Forman [5] gave the first formal definition of quantification learning: "Given a labeled training set, induce a quantifier that takes an unlabeled test set as input and returns its best estimate of the class distribution." Mathematically, a quantification problem can be defined as follows. Let x_i represent an instance i, and C is the set of class labels with the cardinality denoted as $|C|$. Given a training dataset $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, where $y_i \in C$ is the corresponding class label of x_i, and a test dataset $T = \{x_{n+1}, x_{n+2}, \ldots, x_{n+|T|}\}$, a quantifier outputs a vector $\widehat{P} = [\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_{|C|}]$ and $\hat{p}_i$ is the predicted ratio of the number of instances in class i to the total number of instances in the test dataset. An instance can be a text document, an image, or an audio file, depending on the applications.

With a large volume of data, manual estimation of class distribution is prohibitively time-consuming and impractical. Although quantification and classification learning are both supervised learning tasks, quantification learning has been relatively under-explored mainly because it is seen as a trivial and straightforward post-processing step of classification [6]. Quantification learning focuses on correct predictions at the aggregate level while classification learning aims at predicting the class of individual instances. When using a classifier to conduct quantification and the classifier is not completely accurate, there is a mismatch between classification and quantification performance. Fig. 1 illustrates the mismatch between classification performance and quantification performance. Fig. 1(a) shows the women to men ratio of 1:1 in the ground truth. Fig. 1(b) demonstrates that classifier **J** (with classification accuracy of 60%) mis-predicting two females as males and vice versa is able to give accurate class ratios of 1:1 as in the ground truth. However, classifier **K** with a 10% higher classification accuracy than classifier **J** gives a much worse estimate of four times the number of women to the number of men.

Another major issue is that classification learning generally assumes that training data and test data are independent and identically distributed (i.i.d) [7] [8] whereas quantification learning does not make the i.i.d. assumption. When there is a difference between the class distribution of the training data and that of the test data, an over-estimate or under-estimate of the class distribution in the

training data is likely to occur [5]. Therefore, quantification learning should be investigated in its own right. Existing works in quantification learning show some benefits of inducing a quantifier directly without classification [9][10], but none has used deep learning for quantification. In this paper, we design a framework for deep quantification learning to improve the effectiveness of quantification learning as it does for classification. We mainly focus on the quantification of text documents in this paper, but the framework is generic and can be extended to other modalities.

Our contributions are as follows: (1) We formulate the quantification learning problem as a maximum likelihood problem. We use a Jensen Shannon Divergence as the loss function to allow for use of gradient descent optimizers. (2) We propose the first end-to-end DQN framework that jointly learns effective feature representations and class distribution estimates. We introduce two strategies to select a set of documents (termed a *tuplet*) for training to investigate how well the induced quantifier generalizes to test sets with class distributions different from that of the training set. We examine five methods to extract the feature representations of tuplets. (3) We evaluated DQN variants on quantification tasks of text documents on four public datasets with 2, 4, and 20 classes using three metrics commonly used for measuring quantification errors: Mean Absolute Error, Relative Mean Absolute Error, and Kullback-Leibler Divergence. We performed a sensitivity analysis of DQN performance by varying tuplet sizes and training dataset sizes. (4) The highlights of our findings are as follows. Compared to the best existing method in our study, (i) DQN reduces Mean Absolute Error (MAE) by about 35%. (ii) DQN uses around 40% less training samples to achieve a comparable MAE. Therefore, DQN is more desirable since it reduces the manual labeling effort to create the training dataset significantly. We share the source code at https://github.com/****.

2 Related Work

We categorize existing methods for quantification learning into two categories based on whether the methods rely on the classification We further divide each category into sub-categories based on whether the methods take extracted features as input or they learn to extract features.

2.1 Classification-based Quantification

This category has two sub-categories: (a) Classify and Count and (b) Hybrid.

Classify and Count: Methods in this sub-category classify individual documents, count the number of documents predicted in each class, and calculate the class ratios. The methods differ in either the counting step or the post-processing steps.

Classify and Count (**CC**): Forman [5] and most authors published in this field used this basic CC method as the baseline. CC overestimates when the

prevalence of a class in the test dataset is lower than the prevalence of the same class in the training data and underestimates when the prevalence of the class in the test dataset is higher [5]. To address this drawback, several methods of adjusting the predicted class ratios after CC have been proposed by [11][12].

Adjusted Classify and Count (**ACC**): Forman [11] proposed Equation (1) to adjust the class ratios estimated by the CC method with *tpr* (true positive rate) and *fpr* (false positive rate) obtained from cross-validation for a binary class classification as follows:

$$p = \frac{\hat{p} - fpr}{tpr - fpr} \tag{1}$$

where $\hat{p}$ is the predicted proportion of the positive class from a CC method and p is the adjusted proportion. According to Equation (1), ACC is vulnerable to *tpr* and *fpr* values. For example, when *tpr* is close to *fpr*, the denominator is close to zero, which leads to a large value of p in Equation (1). Hopkins and King generalized Equation (1) for a multi-class classification task [13].

Probabilistic Classify and Count (PCC): Bella et al. [14] proposed probabilistic versions of Classify and Count (**PCC**) and adjusted Classify and Count (**PACC)** using the posterior probabilities output by a classifier for counting.

Hybrid: This sub-category adapts traditional classification algorithms for quantification tasks by either introducing a new combined loss function or using an ensemble of quantification results.

Milli et al. [15] proposed Quantification Trees, which use either Decision Tree or Random Forests for classification. They investigated two loss functions. Let FP_i and FN_i be the numbers of false positives and false negatives for a class i, respectively. One method calculates an error for the class i, $E_i = |FP_i - FN_i$. The other method computes $E_i = |FP_i - FN_i| \times |FP_i + FN_i|$ where the first term represents the quantification error and the second term represents the classification error. Other methods in this category include [16][17]. Esuli and Sebastiani's quantifier [16] used SVMperf [18] as a classifier for quantification. Barranquero et al. [17] proposed Q-measure as a loss function, an analogy to F-measure for classification. Pérez-Gállego et al. proposed the first ensemble quantifier for binary quantification [19], which is inspired by the idea of ensemble learning for classification. Although the methods in the Hybrid category reduce quantification errors, they still rely on the classification of individual documents.

2.2 Direct Quantification

King and Lu [20] presented a non-parametric approach (denoted as ReadMe) to estimate the distribution of the cause-of-death without training a classifier. Hopkins and King applied this method to estimate document category distributions [13]. The key assumption is that the proportions of the word patterns occurring in documents in each class are the same for both the test and labeled datasets. In simple terms, it means the same writing style for documents in each class in

both test and labeled datasets. ReadMe works in iterations until the number of iterations reaches the user-specified value. In each iteration, it randomly selects k words to form word patterns and calculates the proportion of the word patterns used in documents of different classes. ReadMe uses these proportions to estimate the class distribution of the labeled dataset. The error of the estimate and the truth gets smaller with more iterations. González et al. proposed methods based on Hellinger Distance (HD) [10] between two distributions and it was used for quantification of image data. The authors assume that given a class label y_i of document i, the probabilities of having the feature vector f_i in both the training and the test datasets are the same, i.e., $P_{train}\left(f_i|y_i\right) = P_{test}\left(f_i|y_i\right)$. The method repeatedly generates a validation dataset V from the training dataset with a given prior probability. Then, it uses the distribution of a validation dataset $\hat{V}$ with the least HD value to the distribution of the test dataset as the estimated distribution.

In summary, the classification-based methods have these major drawbacks. (1) They require separate steps (feature extraction, classifier training, counting, and adjustment of the class ratios) that are not jointly optimized to reduce the quantification error. (2) Except for the rare case of 100% accurate classification, more accurate classifiers do not always lead to more accurate class distribution estimates. See Fig. 1 example. (3) For the direct quantification category, the HD-based method requires good feature representations of instances to begin with. For ReadMe, the randomly chosen word patterns may not be good features for estimating class ratios. Motivated by the nature of the quantification learning problem, the major drawbacks of the existing methods, and the success of deep learning in several tasks [21], we proposed DQN.

3 Problem Formulation of Quantification Learning

Recall in the introduction that given a labelled dataset D and a test dataset T, a quantifier estimates the class distribution of the entire dataset T. Therefore, we design DQN that partitions the input D during training or T during testing) into a number of tuplets. A tuplet is a set of m instances where $m >= 1$. This enables the prediction of the class distribution for each tuplet and gives a reliable class distribution estimate using multiple tuplets. For ease of presentation, we consider a binary quantification problem and formulate it as a maximum likelihood problem as follows. Let the class ratio for a positive class be r; the ratio of the negative class is then $1 - r$. Tuplets are generated from original dataset using one of methods introduced in Section 4.2, and corresponding class distributions of tuplets are computed from class labels of instances in a tuplet. As an analogy, a tuplet for DQN is same as an individual training data point for classification, and the class distribution of the tuplet for DQN is same as class label of the training data point for classification. Let N and M be the number of tuplets in the training set and test set, respectively. Given a set of tuplets with its corresponding class distribution, $DT = \{(t_1, r_1), (t_2, r_2), \ldots, (t_N, r_N)\}$ where $r_i \epsilon [0, 1]$ is the corresponding class ratio of the positive class of the tuplet t_i, and

a test dataset $TT = \{t_{N+1}, t_{N+2}, \ldots, t_{N+M}\}$, a quantifier outputs a value $\hat{p}$ that is the predicted ratio for the positive class of TT. We assume the conditional probability of the class ratios given DT is the same as TT. Therefore, we can see DQN as a function F that learns a complex mapping from t_i to r_i. We write it as $r \sim F(t, \Theta)$, where F is parameterized by the parameter set Θ. The conditional probability of r given t is written as $P(r|t, \Theta)$. The likelihood function under F is in Equation (2) and the log-likelihood function $L(\Theta)$ is given in Equation (3).

$$\prod_{k=1}^{N} P\left(r_k|t_k,\ \Theta\right) \tag{2}$$

$$LL(\Theta) = \sum_{k=1}^{N} \log\left[P\left(r_k|t_k, \Theta\right)\right] \tag{3}$$

Maximizing the log-likelihood is the same as minimizing the negative log-likelihood. We rewrite Equation (3) as the negative log-likelihood as in Equation (4).

$$NLL(\Theta) = -\sum_{k=1}^{N} \log\left[P\left(r_k|t_k, \Theta\right)\right] \tag{4}$$

We use Jensen-Shannon Divergence (JSD) as our loss function. JSD is a measure of similarity between two probability distributions [22] and is defined as in Equation (5).

$$JSD(P\|\hat{P}) = \frac{1}{2}\left(D(P\|M) + \frac{1}{2}(D(\hat{P}\|M)\right) \tag{5}$$

where $D(P\|\hat{P})$ is Kullback–Leibler Divergence (KLD) [23] between the two distributions and $M = \frac{1}{2}(P + \hat{P})$. We use $JSD \in [0, 1]$ because it has several good properties compared to KLD. JSD is differentiable and has the symmetric property where $JSD(P\|\hat{P}) = JSD(\hat{P}\|P)$. JSD has been applied in Generative Adversarial Networks (GAN) [22] as well as in several other research fields. In this problem formulation, P represents the class distribution of a tuplet from the training data and $\hat{P}$ represents the predicted class distribution of the tuplet. Equation (4) with the JSD as the loss function can be solved to obtain an estimated parameter set $\hat{\theta}$ using an optimizer such as gradient descent optimizers.

4 Deep Quantification Network (DQN) Framework

We present our DQN framework, the loss function, and the algorithms for training and testing. Fig. 2 shows the overall framework.

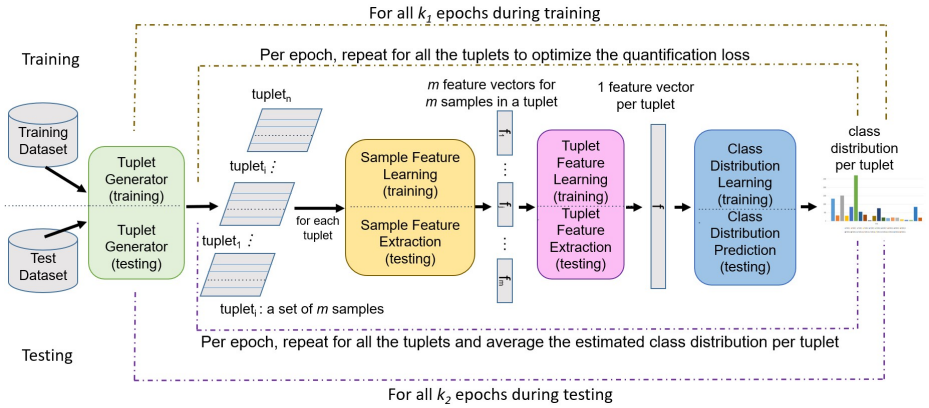

Fig. 2. DQN framework; the top half of the diagram shows the training process. The bottom half illustrates the test process; m is a DQN hyper-parameter; k_1 and k_2 are the numbers of epochs for training and testing, respectively.

4.1 DQN Framework

Our problem formulation defines a tuplet as a set of m instances; m is the tuplet size. Therefore, the first component of the framework is the tuplet generator that assigns input instances to tuplets. Ideally, a tuplet should be large enough to include samples of instances from all the classes in order to get a reasonable class distribution estimate from a single tuplet. The tuplet size is a hyper-parameter of DQN. On one extreme, when the tuplet size m is one (i.e., a tuplet with only one instance), DQN degrades to a classification-based method for quantification. On the other extreme, if the tuplet size is as large as the size of the training dataset, we are restricted to only one tuplet with a fixed class distribution, which makes the trained model unable to perform well for other datasets.

We describe our tuplet generation strategies in detail in Section 4.2.

During training (illustrated in the top half of Fig. 2), the tuplet generator generates all the tuplets using all labeled instances in the training dataset as well as calculates the corresponding class distribution for each tuplet. In each epoch, the tuplet generator passes each tuplet to the sample feature learning component. This component is a layer of neurons to learn parameters of a function that extracts feature representation for each sample in a tuplet. Any deep learning architecture that is effective for the modality of the instances (e.g., LSTM for text documents, CNN for images) can be used. Since each tuplet has m instances, this component outputs m feature vectors $f_1, \ldots, f_m$. These vectors become the input to the tuplet feature learning component to extract a tuplet feature vector f. The feature vector is passed to one or more layers of neurons trained to estimate the class distribution of a tuplet such that the quantification loss between the predicted and the ground truth is optimized. During back-propagation, the weights and biases are updated based on a gradient descend algorithm as done

in classification learning. The entire process is repeated until the desired number of epochs is reached.

During testing, for each epoch, the tuplet generator generates tuplets from all unlabeled instances in the test dataset. The second component extracts a feature vector for each sample in a tuplet using the pre-trained weights and biases. The third component extracts the tuplet feature vector for the last component that outputs the class distribution estimate of a tuplet. This process repeats for all the generated tuplets in this epoch and the arithmetic mean of the estimated class ratios for each class from all the tuplets is used as the class ratio estimate for the class. Sub-sequent epochs follow the same process and averages all the estimated class distributions of all the epochs as the final class distribution estimate.

4.2 Tuplet Generation

We introduce two strategies for assigning training instances to tuplets: the random selection strategy and the Zipf distribution selection strategy. We use the random selection strategy to establish the baseline performance. We introduce the Zipf distribution [24] selection strategy to generate a variety of class distributions that might be much different from the underlying class distribution in the training dataset to enhance the generalization ability of the model for test datasets with different class distributions.

Random Selection Strategy: This strategy randomly selects m instances without replacement from the training dataset to form a tuplet. The already selected instances are not eligible for assignment to subsequent tuplets in this epoch. That is, an instance is assigned exclusively to one tuplet in each epoch. For example, for the training dataset with 1,000 instances, if the tuplet size is 100, we have a total of 10 tuplets per epoch. In each epoch, we randomly choose 100 instances from the 1,000 instances and assign them to the first tupletcalculate the class ratio for each class for this tuplet based on the class labels of the instances. Next, we randomly choose 100 instances from the 900 remaining instances and assign them to the second tuplet. We repeat this process to create more tuplets until the number of remaining instances is less than the tuplet size. Finally, we get 10 tuplets for this epoch. For each subsequent epoch, we get ten different tuplets due to random selection. Although random sampling with replacement can also be considered, we are against it for two reasons. First, some instances may be selected many more times than other instances, creating a bias in training. Second, we cannot guarantee that all training instances are selected per epoch unless we implement more constraints. We recommend using more epochs with our random selection without replacement strategy than using random selection with replacement.

Zipf Distribution Selection Strategy: Our goal is to prevent DQN from overfitting the class distribution in the training dataset and to generalize DQN for different class distributions of future test datasets. We propose to generate

tuplets with different class distributions synthetically. Because many types of data in physical and social sciences can be approximated well with Zipf distribution [24], we chose the Zipf distribution. That is, we use Equation (6) to calculate num_i, the number of instances in a class $i$in a tuplet.

$$num_i = \frac{m}{i^Z * \sum_{j=1}^{|C|} j^Z} \tag{6}$$

where $|C|$ is the number of classes and $z \in [0, 1]$ is the skew factor. When the skew factor value is zero, all classes have the same number of instances, i.e., uniform distribution. When the skew factor is one, a few classes have many instances while several classes have very few instances. After calculating the number of instances for each class i for a tuplet, we randomly select num_i instances without replacement for each class i from the training dataset and assign them to the tuplet. The already assigned instances are not eligible for other tuplets in this epoch. The difference between the two strategies is the class distribution of each tuplet. With the Zipf distribution selection strategy, the class distributions of tuplets can vary significantly from the class distribution of the training dataset.

4.3 Sample Feature Learning/Extraction

To obtain feature representation of a tuplet, we first extract feature representation of each sample in the tuplet. To apply the DQN framework for a specific application, it is necessary to choose an appropriate neural network (NN) architecture suitable for the modality of the data and the application (e.g., CNN for images, 3D-CNN for videos). As we focus on the application of DQN on text documents in this paper, we choose Long Short-Term Memory (LSTM) [25] to learn effective feature representations for each sample (document) in a tuplet during training. LSTM can deal with variable length documents and is good at feature extraction of sequence data like text. This step outputs m fixed-length vectors: $f_1, \ldots, f_m$, where $f_i \in R^d$ and d is the number of elements in the feature vector f. During testing, the learned LSTM parameters are used to extract sample feature vectors.

4.4 Tuplet Feature Learning/Extraction

We study five alternatives to obtain a tuplet feature vector f from the feature vectors $f_1, \ldots, f_m$ of the samples in the tuplet. Let $|f|$ denote the dimension of the tuplet feature vector f.

1. Concatenation (CAT): We concatenate $f_1, \ldots, f_m$ one after another; therefore, $|f| = d * m$.
2. Average (AVG), Median (MED) and Maximum (MAX): we compute the column-wise arithmetic average (or median or maximum) value for each dimension of each feature vectors for m samples in a tuplet to obtain a unified feature vector for the tuplet.

3. Additional neural network (NN) layers: We feed f_1 ,. . . , f_m to the additional NN layers such as a dense layer, or a convolutional layer. For these NN layers, we do not use any architecture that is impacted by the order of the samples in a tuplet (e.g., LSTM) since the order should not impact quantification results. The dimension of the tuplet feature vector f depends on the chosen NN architecture. During training, the parameters of the NN layers are learned. During testing, the learned parameters are used to extract one tuplet feature vector per tuplet.

4.5 Class Distribution Learning

The last component in Fig. 2 is a fully connected layer followed by a soft-max layer. During training, this component takes the tuplet feature vector to learn a probability-valued vector, which is the estimated class distribution of the tuplet. As we mentioned in Section 2, we use JSD as the loss function between the estimated class distribution and the true class distribution during training. During testing, this component uses the learned parameters to estimate the class distribution.

4.6 Train and Test Algorithms

Fig. 3 outlines the algorithms for training and testing DQN, respectively.

Training Algorithm (Algorithm 1 in Fig. 3): Given a set of labeled instances, denoted as D, the tuplet generator generates tuplets from D using one of the methods proposed in Section 4.2. Either method generates the number of tuplets per epoch of $\lfloor \frac{|D|}{m} \rfloor$.

The tuplets and their corresponding class distribution are given as input to the sample feature-learning component in mini-batches. We use Equation (7) to calculate the overall loss L for all the tuplets in a mini-batch.

$$L = \sum JSD(P \| \hat{P}) + \lambda \sum_{\theta \in \Theta} \theta^2 \tag{7}$$

where P is the real class distribution of a tuplet from the training data, and $\hat{P}$ is the estimated class distribution of the same tuplet in a mini-batch. We update the hyper-parameters of Q using a gradient descent method. Finally, we obtain the trained quantifier Q. The last term $\lambda \sum_{\theta \in \Theta} \theta^2$ in Equation (7) is the regularization term [21] to prevent overfitting and obtain a smooth model. λ is the weight decay and Θ is the set of all parameters in the model.

Testing Algorithm (Algorithm 2 in Fig. 4): Given the trained quantifier Q and a test dataset T, we use the random selection without replacement method introduced in Section 4.2 to generate tuplets from T and input them into the pre-trained quantifier Q. The test algorithm runs in multiple epochs. For each epoch,

Algorithm 1: DQN Training Algorithm
Input:
D: Training set of documents with class labels
M: Tuplet generation method
m: Tuplet size
k: Number of epochs
b: Number of tuplets per mini batch
Output: Q: Quantifier
Algorithm:
1: **for** epoch 1 to k:
2: generate $\left\lfloor \frac{|D|}{m} \right\rfloor$ tuplets from D using M
3: assign b tuplets per mini-batch
4: **for each** mini-batch:
5: train Q using the tuplets in the mini-batch
6: compute the loss L per Equation (7)
7: update Q parameters to minimize the loss L using gradient descent
8: **return** Q

Algorithm 2: DQN Test Algorithm
Input:
T: Test dataset without labels
Q: Trained quantifier from Algorithm 1
m: Tuplet size
k: Number of epochs
Output: $\frac{\overrightarrow{res}}{k}$: estimated class distribution of T
Algorithm:
1: $\overrightarrow{res} \leftarrow \vec{0}$
2: **for** epoch 1 to k:
3: generate $\left\lfloor \frac{|T|}{m} \right\rfloor$ tuplets from T using the random selection without replacement in 4.2
4: $\vec{e} \leftarrow \vec{0}$
5: **for each** tuplet t:
6: $\vec{e} \leftarrow \vec{e} + Q(t)$
7: $\overrightarrow{res} \leftarrow \overrightarrow{res} + \frac{m*\vec{e}}{|T|}$
8: **return** $\frac{\overrightarrow{res}}{k}$

Fig. 3. Training and test algorithms for DQN. ***res*** and ***e*** are vectors; ***0*** is a zero vector.

Lines 4-7 find the average estimate of class distributions of all the tuplets for the epoch and store it in the vector variable ***res***. The average is averaged again over the number of epochs in Line 8. Notice that in different epochs, the same instance has a chance to be assigned with other tuplets. Hence, using a sufficiently large number of epochs will produce different combinations of instances in tuplets. This is to have a reliable estimate of the underlying class distribution in the test data.

5 Experiments

5.1 Datasets

We used four public balanced datasets. The details are in Fig. 4. To evaluate the performance of a quantifier on test dataset with different class ratios as done in previous works [11], we created test datasets artificially from the original test datasets. For binary quantification tasks, we extracted samples with a prevalence of the positive class varying from 0.1 to 0.9 with the interval of 0.1. For instance, in a test dataset with the prevalence of the positive class of 0.1, the prevalence of the negative class is 0.9 (1-0.1). To generate a test dataset with n documents, we randomly selected $0.1*\ n$ positive samples and $0.9*\ n$ negative samples from the positive and negative classes, respectively, in the original test dataset. We repeated this process ten times for this prevalence and reported the average of the quantification errors from the ten experiments. Hence, each reported error for each method is the average of the errors from 90 experiments. For multi-class quantification tasks, we synthetically created the test datasets with different class ratios using the Zipf distribution in Equation (5) with different skew factors. We varied the skew factors from 0 (uniform distribution) to

0.9 (highly skewed distribution) with the interval of 0.1. For each skew factor, we created ten different test datasets and reported the average of the quantification errors from the experiments on these datasets. Each reported error for each method is the average of the errors from 100 experiments.

Dataset Description	#classes	Total #training instances	Total #test instances	Average #words per instance
Stanford Large Movie Review Dataset (IMDB) [26]	2	25,000	25,000	231
Yelp Polarity Reviews (YELP) [27]	2	560,000	76,000	133
AG News (AG-NEWS) [27]. Only news titles used	4	120,000	7,600	8
20 Newsgroups (20-NEWS) [28]	20	~16,000	~2,000	285

Fig. 4. Datasets used for performance evaluation.

5.2 Compared Methods

We compared DQN with six existing methods across all the categories of existing works: Classify and Count (**CC**), Adjusted Classify and Count (**ACC**), Probabilistic Classify and Count (**PCC**), Probabilistic Adjusted Classify and Count (**PACC**), Hybrid (**HA**), and Direct Quantification using ReadMe [13]. HA denotes a Hybrid method that uses the weighted sum of equally weighted classification loss and quantification loss defined as $\frac{1}{|C|}\sum_{c\in C}|FPR_c - FNR_c|$ with a similar idea as that of Quantification Tree [15]. FPR_c and FNR_c are false positive and false negative rates for a class i, respectively. The compared techniques were introduced in Section 2. ReadMe is the only existing direct quantification method. Both ReadMe and DQN do not require features to be known in advance. Therefore, we chose ReadMe to represent the existing work in the direct quantification category. For all the compared methods except ReadMe, we used the same LSTM architecture. This is to ensure that any performance difference does not come from different types of classifiers used. We used Keras and Scikit-learn to implement the proposed method and all the other methods except ReadMe. For ReadMe, we ran the original code [13].

5.3 Hyper-Parameters Setting

Recall that DQN has four components (Fig. 2). For the tuplet generator using the Zipf distribution selection (Section 4.2), during training, we varied the Zipf skew parameter z value from 0.1 to 1.0 with an interval of 0.1. For the sample feature learning component (Section 4.3), we used one shared LSTM layer with 128 neurons and ReLU as the activation function. For the tuplet feature learning component that uses additional layers, we used one dense layer with 256 neurons

to extract a high-level representation for a tuplet. Finally, the class distribution learning component (Section 4.4) had 256 nodes for the fully connected layer with the sigmoid function as the activation function and $|C|$ nodes for softmax layers where $|C|$is the number of classes. We chose the hyper-parameter values for our training empirically. For DQN, we set the mini-batch size to 8 and the tuplet size to 100. We used stochastic gradient descent as the optimizer. We used the dropout rate and the recurrent dropout rate of 0.2. The learning rate was 1.0E-5. For the classification-based quantification methods, CC, ACC, PCC, PACC and HA, to make the comparison fair, we used LSTM as the classifier; we kept the same number of neurons, dropout rate and leaning rate as that of DQN, but used cross-entropy as the loss function for the classifier. The mini-batch size of the classifier was 64. The word embedding size was 150, the same for all the methods.

5.4 Performance Metrics

We used three commonly used metrics: Mean Absolute Error (MAE), Relative Mean Absolute Error (RMAE), and Kullback-Leibler Divergence (KLD) to quantify the errors [6]. Techniques that offer the lowest errors are most desirable.

$$MAE(P, \hat{P}) = \frac{1}{|C|} \sum_{c \in C} |P(c) - \hat{P}(c)| \tag{8}$$

$$RMAE(P, \hat{P}) = \frac{1}{|C|} \sum_{c \in C} \left| \frac{P(c) - \hat{P}(c)}{P(c)} \right| \tag{9}$$

$$KLD(P \| \hat{P}) = \sum_{c \in C} P(c) \log \frac{P(c)}{P(c)} \tag{10}$$

where P is the class distribution truth, and $\hat{P}$ is the predicted class distribution.

5.5 Experimental Results & Discussion

Fig. 5 presents quantification errors on the four datasets when the quantifiers were trained on their respective entire training dataset. In all the experimental results, we used the following legends. DQN-R and DQN-Z denote DQN using the random selection and the selection with Zipf distribution to generate tuplets as previously discussed, respectively. The suffixes, CON, AVG, MED, MAX, NN denote the tuplet feature learning/extraction method.

We have five findings as follows. (1) Our DQN variants perform better than all the other compared methods regardless of binary and multi-class quantification tasks. On average across the four datasets, DQN-Z-NN gives 35% lower MAE achieved by the best existing methods. See finding 5 below. DQN learns good feature representations for quantification and the tuplet generation strategy is able to utilize combinations of individual training instances to generate many

tuplets for optimizing parameters to avoid overfitting with the training class distribution. (2)DQN-Z consistently gives lower quantification errors than DQN-R does in all four datasets. We trained DQN-Z with different class distributions of training tuplets so that it learns feature representations and parameter values for diverse class distributions that may occur in the test dataset. We recommend using DQN-Z to predict class distributions, especially in the applications that expect the class ratios to change significantly or periodically. (3) Fig. 5 also shows that DQN-Z-NN (with the dense layer of 256 neurons for tuplet fea-

	Binary quantification						**Multi-class quantification**					
	IMDB			**YELP**			**AG-NEWS**			**20-NEWS**		
	MAE	RMAE	KLD	MAE	RMAE	KLD	MAE	RMAE	KLD	MAE	RMAE	KLD
CC	0.135	0.316	0.047	0.128	0.284	0.043	0.101	0.221	0.038	0.054	0.142	0.024
PCC	0.154	0.361	0.064	0.146	0.325	0.059	0.116	0.253	0.051	0.062	0.162	0.032
ACC	0.077	0.242	0.022	0.076	0.237	0.028	**0.051**	**0.150**	**0.014**	0.031	0.109	0.011
PACC	**0.068**	**0.214**	**0.017**	0.073	0.218	0.020	0.058	0.169	0.018	0.027	0.096	0.009
HA	0.074	0.233	0.020	**0.070**	**0.210**	**0.018**	0.056	0.163	0.016	0.030	0.105	0.010
ReadMe	0.072	0.226	0.019	0.075	0.236	0.028	0.052	0.153	0.013	**0.026**	**0.094**	**0.008**
DQN-R-CON	0.062	0.195	0.013	0.059	0.176	0.012	0.047	0.137	0.010	0.025	0.088	0.007
DQN-R-AVG	0.065	0.204	0.014	0.062	0.184	0.013	0.049	0.143	0.011	0.026	0.092	0.007
DQN-R-MED	0.064	0.201	0.014	0.061	0.181	0.013	0.048	0.141	0.011	0.026	0.090	0.007
DQN-R-MAX	0.055	0.173	0.010	0.052	0.156	0.009	0.041	0.121	0.008	0.022	0.078	0.005
DQN-R-NN	0.050	0.157	0.008	0.047	0.141	0.007	0.038	0.110	0.006	0.020	0.071	0.004
DQN-Z-CON	0.053	0.167	0.009	0.050	0.150	0.008	0.040	0.117	0.007	0.021	0.075	0.005
DQN-Z-AVG	0.056	0.176	0.011	0.052	0.156	0.009	0.042	0.123	0.009	0.022	0.079	0.006
DQN-Z-MED	0.055	0.173	0.010	0.053	0.158	0.010	0.041	0.121	0.008	0.022	0.078	0.005
DQN-Z-MAX	0.047	0.147	0.007	0.045	0.132	0.006	0.035	0.103	0.006	0.019	0.066	0.004
DQN-Z-NN	***0.044***	***0.138***	***0.006***	***0.042***	***0.124***	***0.005***	***0.033***	***0.097***	***0.005***	***0.018***	***0.062***	***0.003***
Error reduced	35%	36%	53%	40%	41%	72%	35%	35%	64%	31%	34%	63%

Fig. 5. Quantification errors on different quantification tasks.

ture learning) achieves the lowest quantification errors among all the variants. The additional dense layer can extract better tuplet feature representations. 4) Among the remaining tuplet feature extraction methods, MAX consistently gives the lowest error below those of CAT, AVG, and MED. 5) The best existing methods are as follows: PACC for IMDB, HA for YELP, ACC for AG-NEWS, and ReadMe for 20-NEWS. The 20-NEWS dataset has long formal documents similar to the datasets ReadMe was originally investigated [13]. Nevertheless, both DQN-R and DQN-Z gave the lowest quantification errors in terms of three metrics among all the compared methods across all four datasets. The p-value of paired t-test (between best model of existing methods and DQN-Z-NN) results are 9.64E-7, 3.34E-7, 3.38E-6, 3.31E-6 for IMDB, YELPS, AG-NEWS and 20-NEWS, respectively.

5.6 Sensitivity Analysis

We demonstrate the impact of training dataset sizes and tuplet sizes on DQN using IMDB (binary quantification) and AG-NEWS (multi-class quantification).

We used the same strategy as mentioned in Section 5.3 for creating test datasets artificially from the original test datasets to evaluate the performance of a quantifier on different class ratios and report the average.

Impact of the training dataset size on DQN Our goal is to determine robustness of different methods to training data, which has practical impact on minimizing time-consuming manual labeling effort. Fig. 6 shows that ACC and PACC perform consistently much better than CC and PCC, and DQN-Z-NN gives the lowest quantification errors. Therefore, we selected these methods, ACC, PACC, HA, ReadMe, and DQN-Z-NN to evaluate their performance under different training set sizes. We varied the percentage of the training dataset used to train the five quantifiers. We randomly selected $p\%$ documents from the entire training dataset to create the limited training dataset according to the specified percentage $p \in \{10, 20, \ldots, 100\}$. We trained each of the quantifiers using the limited training dataset and calculated MAE. Fig. 6 shows that DQN consistently offers the lowest MAEs. For IMDB, ACC and PACC suffer the most from the limited training data. For AG-News with short text documents, HA and PACC suffer the most. These methods rely on correct prediction of individual documents to estimate the class distribution. Both direct quantification methods (ReadMe and DQN) are less impacted by the decrease in the training dataset size. With around 60% of the entire training data, DQN reduces MAE significantly down to around 0.07 and 0.051 for IMDB and AG-NEWS, respectively. To offer the similar MAE, the other methods need 90% or more of the training data.

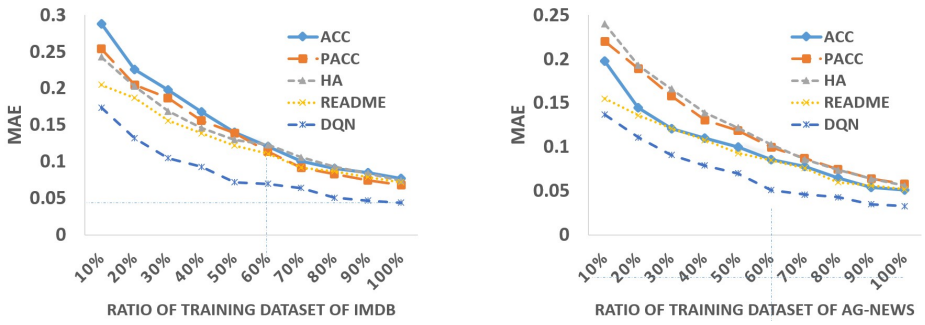

Fig. 6. Impact of training dataset size on MAE on IMDB and AG-NEWS.

Impact of the tuplet size on DQN We investigated DQN-R and DQN-Z with the NN tuplet feature learning when varying tuplet sizes on two datasets. We varied the tuplet size as $2^n * |C|$ where n is { 0, 1, 2, . . . , 7} and $|C|$ is the number of classes, which is 2 and 4 for IMDB and AG-NEWS, respectively. We

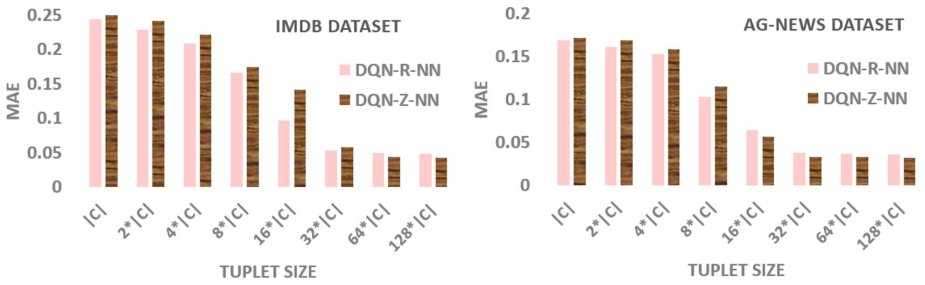

Fig. 7. Impact of tuplet size on DQN; —C—: Number of classes.

have following findings. (1) Fig. 7 shows that, for these DQN variants, MAEs reduce as the tuplet size increases. It is because that as tuplet size increases, there are two main benefits. First, we have more combinations of instances to generate a variety of tuplets. Second, the class ratios predicted from a large tuplet are more reliable than those predicted from a small tuplet. (2) MAEs of DQN reduce slowly at first and then faster after the tuplet size is $8 * |C|$. However, after the tuplet size is around $32 * |C|$, there is not much improvement. Intuitively, each tuplet should have a sufficient number of documents for each class to give an accurate prediction. We believe that when the tuplet size is too small, it is difficult to extract patterns for quantification because there is not much information about class ratios and word usage of documents in each class. As we mentioned in the Section 1, when the tuplet size is 1, this special case makes DQN degrade to classification (except the different loss function) and counting. (3) DQN-R-NN performs a litter bit better than DQN-Z-NN when the tuplet size is small and vice versa as the tuplet size increases. It is because DQN-Z-NN needs a larger tuplet size to ensure that at least one training instance belongs to each class in the tuplet. Recall DQN-Z-NN is trained on the tuplets generated according to the Zipf distribution with different skew factors. In other words, most of tuplets have a severe class imbalance. Therefore, we recommend using a large tuplet size when expecting a severe imbalance.

6 Conclusion and Future Work

We present the first attempt to use deep learning for quantification tasks to estimate the class distribution of a given dataset. We introduce DQN—a framework for deep quantification learning applicable for various quantification tasks. We present extensive evaluation results of DQN on four public datasets against six existing methods in all categories in the literature. DQN outperforms these methods on different types of text data and tasks, especially when the training dataset is small. We performed sensitivity analyses on important parameters of DQN. However, our work has some limitations. We did not evaluate DQN performance on other data modalities such as audio, image, and graph data. Nevertheless, we expect DQN to perform well when using an appropriate deep

model for learning effective feature representations of the data. Second, interpretation of DQN models in reaching the predicted class distribution is challenging and critical to increase trust and transparency of the black-box quantification model. We will explore these issues in our future work.

References

1. Quionero-Candela, J., Sugiyama, M., Schwaighofer, A. and Lawrence, N.D., 2009. Dataset shift in machine learning. The MIT Press.
2. Barranquero, Jose, et al. "On the study of nearest neighbor algorithms for prevalence estimation in binary problems." Pattern Recognition 46.2 (2013): 472-482.
3. Asoh, Hideki, Kazushi Ikeda, and Chihiro Ono. "A fast and simple method for profiling a population of twitter users." The Third International Workshop on Mining Ubiquitous and Social Environments. 2012
4. Buck, A.A. and Gart, J.J., 1966. Comparison of a screening test and a reference test in epidemiologic studies. II. A probabilistic model for the comparison of diagnostic tests. American Journal of Epidemiology, 83(3), pp.593-602.
5. Forman, George. "Quantifying trends accurately despite classifier error and class imbalance." SIGKDD. ACM, 2006.
6. González, P., Castaño, A., Chawla, N.V. and Coz, J.J.D., 2017. A review on quantification learning. ACM Computing Surveys (CSUR), 50(5), p.74.
7. Bishop, C.M., 2006. Pattern recognition and machine learning. springer.
8. Hofer, Vera, and Georg Krempl. "Drift mining in data: A framework for addressing drift in classification." Computational Statistics & Data Analysis 57, no. 1 (2013): 377-391.
9. King, G. and Lu, Y., 2008. Verbal autopsy methods with multiple causes of death. Statistical Science, 23(1), pp.78-91.
10. Alaiz-Rodrıguez, R., et al. "Estimating Class Proportions in Boar Semen Analysis using the Hellinger Distance.". Industrial, Engineering and Other Applications of Applied Intelligent Systems, 2010
11. G. Forman, "Counting positives accurately despite inaccurate classification," presented at the European Conference on Machine Learning, 2005, pp. 564–575
12. G. Forman, "Quantifying counts and costs via classification," Data Min. Knowl. Discov., vol. 17, no. 2, pp. 164–206, 2008.
13. Hopkins, D.J. and King, G., 2010. A method of automated nonparametric content analysis for social science. American Journal of Political Science, 54(1), pp.229-247.
14. Bella A, Ferri C, Hernández-Orallo J, et al. Quantification via probability estimators[C]//2010 IEEE International Conference on Data Mining. IEEE, 2010: 737-742.
15. Milli L, Monreale A, Rossetti G, et al. Quantification trees, IEEE, 2013 ICDM. pp. 528-536.
16. A. Esuli and F. Sebastiani, "Optimizing Text Quantifiers for Multivariate Loss Functions," ACM Trans Knowl Discov Data, vol. 9, no. 4, pp. 27:1–27:27, Jun. 2015.
17. J. Barranquero, J. Díez, and J. José del Coz, "Quantification-oriented Learning Based on Reliable Classifiers," Pattern Recogn, vol. 48, no. 2, pp. 591–604, Feb. 2015.
18. T. Joachims, "A Support Vector Method for Multivariate Performance Measures," ICML2005.

19. P. Pérez-Gállego, J. R. Quevedo, and J. J. del Coz, "Using ensembles for problems with characterizable changes in data distribution: A case study on quantification," Inf. Fusion, vol. 34, pp. 87–100, 2017.
20. King, G. and Lu, Y., 2008. Verbal autopsy methods with multiple causes of death. Statistical Science, 23(1), pp.78-91.
21. Goodfellow, I., Bengio, Y. and Courville, A., 2016. Deep learning. MIT press.
22. Goodfellow, Ian, et al. "Generative adversarial nets." NIPS. 2014, pp. 2672-2680.
23. Kullback, S. and Leibler, R.A., 1951. On information and sufficiency. The annals of mathematical statistics, 22(1), pp.79-86.
24. Zipf, G.K., 1949. Human behavior and the principle of least effort.
25. Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. Neural computation, 9(8), pp.1735-1780.
26. Maas, Andrew L., et al. "Learning word vectors for sentiment analysis." ACL, 2011.
27. Zhang, Xiang, et al. "Character-level convolutional networks for text classification." NIPS 2015.
28. Lang, Ken. "Newsweeder: Learning to filter netnews." In Machine Learning Proceedings 1995, pp. 331-339. 1995.