Ordinal Text Quantification

Giovanni Da San Martino
Qatar Computing Research Institute
Hamad bin Khalifa University
Doha, Qatar
gmartino@qf.org.qa

Wei Gao
Qatar Computing Research Institute
Hamad bin Khalifa University
Doha, Qatar
wgao@qf.org.qa

Fabrizio Sebastiani[*]
Qatar Computing Research Institute
Hamad bin Khalifa University
Doha, Qatar
fsebastiani@qf.org.qa

ABSTRACT

In recent years there has been a growing interest in text quantification, a supervised learning task where the goal is to accurately estimate, in an unlabelled set of items, the prevalence (or "relative frequency") of each class c in a predefined set $\mathcal{C}$. Text quantification has several applications, and is a dominant concern in fields such as market research, the social sciences, political science, and epidemiology. In this paper we tackle, for the first time, the problem of *ordinal text quantification*, defined as the task of performing text quantification when a total order is defined on the set of classes; estimating the prevalence of "five stars" reviews in a set of reviews of a given product, and monitoring this prevalence across time, is an example application. We present OQT, a novel tree-based OQ algorithm, and discuss experimental results obtained on a dataset of tweets classified according to sentiment strength.

Keywords

Ordinal Quantification; Quantification; Sentiment Analysis;

1. INTRODUCTION

Ordinal classification (OC – also known as *ordinal regression*, or *rating inference*) is the problem of automatically labelling data items $\mathbf{x} \in X$ according to a set of classes $\mathcal{C} = \{c_1, ..., c_{|\mathcal{C}|}\}$ such that $|\mathcal{C}| > 2$ and there is a total order $\prec$ defined on the classes in $\mathcal{C}$. OC is usually viewed as a supervised learning problem, whereby the classifier $h : X \rightarrow \mathcal{C}$ needs to be generated from training data. OC is different from standard *single-label multi-class* (SLMC) classification, since in SLMC there is no order defined on $\mathcal{C}$.

OC is receiving increasing attention from the sentiment analysis and opinion mining community, due to the importance of managing increasing amounts of product reviews (and opinion-laden data in general) in digital form. An example of a totally ordered set of classes {VeryNegative, Negative, Fair, Positive, VeryPositive}, according to which customers may be asked to evaluate a product or service; the case of product reviews evaluated on a scale of 1 to 5 stars is another such example. More generally, OC is of key importance in the social sciences, since it is well-known that humans find it cognitively easier to express their judgments and evaluations on ordinal (i.e., discrete) scales.

In this paper we do not deal with ordinal classification, but with *ordinal quantification* (OQ). *Quantification* [6] is defined as the task of estimating the prevalence (i.e., relative frequency) $p_{Te}(c_i)$ of the classes $c_i \in \mathcal{C}$ in an unlabelled set Te, given a set Tr of labelled training items. Quantification finds its natural application in contexts characterized by *distribution drift*, i.e., contexts where the unlabelled data may not exhibit the same class prevalences as the test data. Distribution drift may be due to different reasons, including the inherent non-stationary character of the context, or class bias that affected the selection of the training data.

A naïve way to tackle quantification is via the "classify and count" (CC) approach, i.e., classify each unlabelled item independently and compute the fraction of the unlabelled items that have been attributed the class. However, a good classifier is not necessarily a good quantifier: assuming the binary case, even if $(FP + FN)$ is comparatively small, bad quantification accuracy might result if FP and FN are significantly different (since perfect quantification coincides with the case $FP = FN$). This has led researchers to study quantification as a task on its own right [1, 4, 6], rather than as a byproduct of classification.

However, while quantification has been studied both for the binary case and the SLMC case, no paper has tackled OQ yet. We do so for the first time by presenting an algorithm for OQ (dubbed OQT) based on "OQ-trees".

In §2 we describe OQT, while in §3 we discuss experiments we have run on a tweet sentiment analysis dataset.

2. OQ-TREES

In tackling OC our goal has been to design an algorithm that (a) leverages the information inherent in the class ordering $\prec$, and (b) performs quantification in the style of the *Probabilistic Classify and Count* (PCC) method [1], since this is the method that has proven the best performer in our SLMC text quantification experiments of [7]. We first introduce PCC, since OQT uses it as a subroutine; we will then move on to presenting OQT itself.

[*]Fabrizio Sebastiani is currently on leave from Consiglio Nazionale delle Ricerche, Italy.

SIGIR '16, July 17 - 21, 2016, Pisa, Italy
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4069-4/16/07...$15.00
DOI: http://dx.doi.org/10.1145/2911451.2914749

```
Function GenerateTree ($\mathcal{C}, Tr, Va$);
/* Generates the OQ-tree */
Input  : Ordered set $\mathcal{C} = \{c_1, ..., c_{|\mathcal{C}|}\}$;
         Set $Tr$ of labelled items;
         Set $Va$ of labelled items;
Output: OQ-tree $T_\mathcal{C}$.
for $j \in \{1, ..., (|\mathcal{C}| - 1)\}$ do
    Train classifier $h_j$ from $\underline{Tr}_j$ and $\overline{Tr}_j$;
end
$T_\mathcal{C} \leftarrow$ Tree($\mathcal{C}, \{h_j\}, Va$);
return $T_\mathcal{C}$;

Function Tree ($\mathcal{C}, \mathcal{H}_\mathcal{C}, Va$);
/* Recursive subroutine for generating the
   OQ-tree */
Input  : Ordered set $\mathcal{C} = \{c_1, ..., c_{|\mathcal{C}|}\}$ of classes;
         Set of classifiers $\mathcal{H}_\mathcal{C} = \{h_1, ..., h_{(|\mathcal{C}|-1)}\}$;
Output: OQ-tree $T_\mathcal{C}$.
if $\mathcal{C} = \{c\}$ then
    Generate a leaf node $T_c$;
    return $T_c$;
else
    $h_t \leftarrow \arg\min_{h_j \in \mathcal{H}_\mathcal{C}} KLD(p, \hat{p}, h_j, Va)$;
    Generate a node $T_\mathcal{C}$ and associate $h_t$ to it;
    $\mathcal{H}'_\mathcal{C} \leftarrow \{h_1, ..., h_{(t-1)}\}$;
    $\mathcal{H}''_\mathcal{C} \leftarrow \{h_{(t+1)}, ..., h_{(|\mathcal{C}|-1)}\}$;
    $LChild(T_\mathcal{C}) \leftarrow$ Tree($\underline{\mathcal{C}}_t, \mathcal{H}'_\mathcal{C}, Va$);
    $RChild(T_\mathcal{C}) \leftarrow$ Tree($\overline{\mathcal{C}}_t, \mathcal{H}''_\mathcal{C}, Va$);
    return $T_\mathcal{C}$;
end
```

Algorithm 1: Function GenerateTree for generating an OQ-tree.

2.1 Probabilistic Classify and Count (PCC)

PCC, originally proposed in [1], consists of generating a classifier from Tr, classifying the items in Te, and estimating $p_{Te}(c)$ as the *expected* fraction of items predicted to belong to c. If (i) by $p(c|\mathbf{x})$ we indicate the posterior probability, i.e., the probability (as estimated by the classifier) of membership in c of the unlabelled item $\mathbf{x}$, (ii) by $p_{Te}(\hat{c})$ we indicate the fraction of items in Te predicted to be in c, and (iii) by $E[x]$ we indicate the expected value of x, this corresponds to computing

$$\hat{p}_{Te}(c) = E[p_{Te}(\hat{c})] = \frac{1}{|Te|} \sum_{\mathbf{x} \in Te} p(c|\mathbf{x}) \qquad (1)$$

where the "hat" symbol indicates estimation. The rationale of PCC is that posterior probabilities contain richer information than binary decisions, which are usually obtained from posterior probabilities by thresholding.

2.2 Generating an OQ-tree

We will tackle OQ by arranging the classes in $\mathcal{C}$ into an *OQ-tree*; we thus dub our algorithm OQT. Given any $j \in \{1, \ldots, (|\mathcal{C}| - 1)\}$, $\underline{\mathcal{C}}_j = \{c_1, \ldots, c_j\}$ will be called a *prefix* of $\mathcal{C}$, and $\overline{\mathcal{C}}_j = \{c_{j+1}, \ldots, c_{|\mathcal{C}|}\}$ will be called a *suffix* of $\mathcal{C}$. Given any $j \in \{1, \ldots, (|\mathcal{C}| - 1)\}$ and a set S of items labelled according to $\mathcal{C}$, by $\underline{S}_j$ we denote the set of items in S whose class is in $\underline{\mathcal{C}}_j$, and by $\overline{S}_j$ we denote the set of items in S whose class is in $\overline{\mathcal{C}}_j$. Our algorithm for training such a tree is described in concise form as Algorithm 1, and goes as follows.

Assume we have a training set Tr and a held-out validation set Va of items labelled according to $\mathcal{C}$. The first step (Line 3) consists of training $(|\mathcal{C}| - 1)$ binary classifiers h_j, for $j \in \{1, \ldots, (|\mathcal{C}| - 1)\}$. Each of these classifiers must separate $\underline{\mathcal{C}}_j$ and $\overline{\mathcal{C}}_j$; for training h_j we will take the items in $\underline{Tr}_j$ as the negative training examples and the items in $\overline{Tr}_j$ as the positive training examples. We require that these classifiers, aside from taking binary decisions (i.e., predicting if a test item is in $\underline{\mathcal{C}}_j$ or in $\overline{\mathcal{C}}_j$), also output posterior probabilities, i.e., probabilities $p(\underline{\mathcal{C}}_j|\mathbf{x})$ and $p(\overline{\mathcal{C}}_j|\mathbf{x}) = (1 - p(\underline{\mathcal{C}}_j|\mathbf{x}))$.

The second step (Line 5) is building the OQ (binary) tree. In order to do this, among the classifiers h_j we pick the one (let us call it h_t) that displays the highest quantification accuracy (Line 12) on Va, and we place it at the root of the binary tree. Here, quantification is performed according to the PCC method described in §2.1. We measure the quantification accuracy of h_j via the standard measure for evaluating SLMC quantification, i.e., *Kullback-Leibler Divergence* (KLD)[1], defined as

$$KLD(p, \hat{p}) = \sum_{c_j \in \mathcal{C}} p(c_j) \log \frac{p(c_j)}{\hat{p}(c_j)} \qquad (3)$$

where $\hat{p}$ is the distribution estimated via PCC using the posterior probabilities generated by h_j. We then repeat the process recursively on the left and on the right branches of the binary tree (Lines 14 to 17), thus building a fully grown quantification tree.

2.3 Estimating class prevalences via OQT

The algorithm for estimating class prevalences by using an OQ-tree is described in concise form as Algorithm 2, and goes as follows. Essentially, for each item $\mathbf{x} \in Te$ and for each class $c \in \mathcal{C}$, we compute (Line 6) $p(c|\mathbf{x})$; the estimate $\hat{p}_{Te}(c)$ is computed as the average, across all $\mathbf{x} \in Te$, of $p(c|\mathbf{x})$. The posterior probability $p(c|\mathbf{x})$ is computed in a recursive, hierarchical way (Lines 13 to 18), i.e., as the probability that, in a SLMC setting, the binary classifiers that lie on the path from the root to leaf c, would classify item $\mathbf{x}$ exactly in leaf c (i.e., that they would route $\mathbf{x}$ exactly to leaf c). This probability is computed as the product of all the posterior probabilities returned by the classifiers that lie on the path from the root to leaf c.

An example quantification tree for a set of $|\mathcal{C}| = 6$ classes is displayed in Figure 1; for brevity, classes are represented by natural numbers, the total order defined on them is the

[1]Note that KLD is a measure of error, and not of accuracy; i.e., lower values are better. Note also that KLD is undefined when one of the $\hat{p}(c_j)$ is 0. To solve this problem, in computing KLD we smooth both $p(c_j)$ and $\hat{p}(c_j)$ via additive smoothing, i.e., we compute

$$p^s(c_j) = \frac{p(c_j) + \epsilon}{(\sum_{c_j \in \mathcal{C}} p(c_j)) + \epsilon \cdot |\mathcal{C}|} \qquad (2)$$

where $p^s(c_j)$ denotes the smoothed version of $p(c_j)$ and the denominator is just a normalizing factor (same for the $\hat{p}^s(c_j)$'s); the quantity $\epsilon = \frac{1}{2 \cdot |Te|}$ is used as a smoothing factor. The smoothed versions of $p(c_j)$ and $\hat{p}(c_j)$ are then used in place of the non-smoothed versions in Equation 3; as a result, KLD is always defined.

```
1  Function QuantifyViaHierarchicalPCC ($Te, T_{\mathcal{C}}$);
   /* Estimates class prevalences on $Te$ using
      the quantification tree */
   Input  : Unlabelled set $Te$;
            Quantification tree $T_{\mathcal{C}}$;
   Output: Estimates $\hat{p}(c)$ for all $c \in \mathcal{C}$;
2  for $c \in \mathcal{C}$ do
3  |  $\hat{p}(c) \leftarrow 0$
4  end
5  for $\mathbf{x} \in Te$ do
6  |  CPost($\mathbf{x}, T_{\mathcal{C}}, 1$); /* Compute the $\{p(c|\mathbf{x})\}$ */
7  |  for $c \in \mathcal{C}$ do
8  |  |  $\hat{p}(c) \leftarrow \hat{p}(c) + \frac{p(c|\mathbf{x})}{|Te|}$;
9  |  end
10 end
11 return $\{\hat{p}(c)\}$

12 Procedure CPost ($\mathbf{x}, T_{\mathcal{C}}, SubP$);
   /* Recursive subroutine for computing all the
      posteriors $\{p(c|\mathbf{x})\}$ */
   Input  : Unlabelled item $\mathbf{x}$;
            Quantification tree $T_{\mathcal{C}}$;
            Probability $SubP$ of current subtree;
   Output: Posteriors $\{p(c|\mathbf{x})\}$;
13 if $T_{\mathcal{C}} = \{c\}$ then
   |  /* $T_{\mathcal{C}}$ is a leaf, labelled by class $c$ */
14 |  $p(c|\mathbf{x}) \leftarrow SubP$;
15 else
16 |  CPost($\mathbf{x}, LChild(T_{\mathcal{C}}), p(\underline{\mathcal{C}}_t|\mathbf{x}) \cdot SubP$);
17 |  CPost($\mathbf{x}, RChild(T_{\mathcal{C}}), p(\overline{\mathcal{C}}_t|\mathbf{x}) \cdot SubP$);
   |  /* $p(\underline{\mathcal{C}}_t|\mathbf{x})$ and $p(\overline{\mathcal{C}}_t|\mathbf{x})$ are the posteriors
   |     returned by the classifier associated
   |     with the root of $T_{\mathcal{C}}$  */
18 end
```

Algorithm 2: Function QuantifyViaHierarchicalPCC for estimating prevalences via an OQ-tree.

order defined on the natural numbers, and sets of classes are represented by sequences of natural numbers. Note that, as exemplified in Figure 1, OQT generates trees for which (a) there is a 1-to-1 correspondence between classes and leaves of the tree, (b) leaves are ordered left to right in the same order as the classes in $\mathcal{C}$, and (c) each internal node represents a decision between a suffix and a prefix of $\mathcal{C}$.

Point (c) is interesting, and deserves some discussion. In Figure 1, internal node "1234 vs. 56" is trained by using items labelled as 1, or 2, or 3, or 4, as negative examples and items labelled as 5, or 6, as positive examples; however, by looking at Figure 1, it would seem intuitive that items labelled as 6 should *not* be used, since the node is root to a subtree where class 6 is not an option anymore. The reason why we do use items labelled as 6 (which is the reason why the node is labelled "1234 vs. 56" and not "1234 vs. 5") is that, during the classification stage, the classifier associated with the node might be asked to classify an item whose true label is 6, and which has thus been misclassified high up in the tree. In this case, it would be important that this item be classified as 5, since this minimizes the contribution of this item to misclassification error; and the likelihood that this happens is increased if the classifier is trained to choose between 1234 and 56, rather than between 1234 and 5. Note also that this is one aspect for which OQT is a true *ordinal*

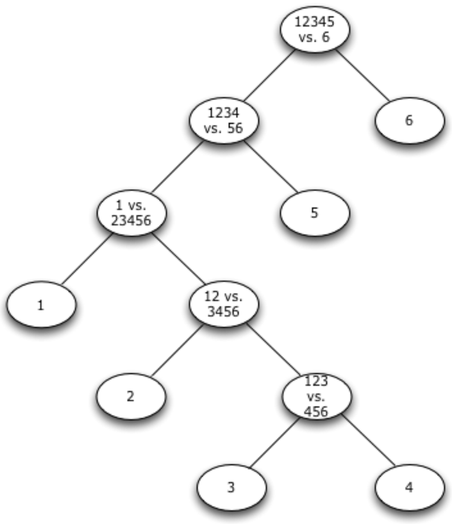

Figure 1: An example OQ-tree.

quantification algorithm: if there were no order defined on the classes this policy would make no sense.

A second reason why OQT is a truly OQ algorithm is that the groups of classes (such as 1234 and 56) between which a binary classifier needs to discriminate are groups of *contiguous* classes. It is because of this contiguity that the trees we generate makes sense: if, say, our classes represent degrees of positivity of product reviews, with 1 indicating most negative and 6 indicating most positive, group 56 may be taken to represent the positive reviews (to different degrees), while 1234 may be taken to represent the reviews that are not positive; a group such as, say, 256, would instead be very hard to interpret, since it is formed of non-contiguous classes that have little in common with each other[2].

3. EXPERIMENTS

The evaluation measure we adopt for our experiments is the only one known for OQ, i.e., the *Earth Mover's Distance* (EMD) [9], a measure well known in the field of computer vision and whose use in OQ was proposed in [4]. When there is a total order on the classes in $\mathcal{C}$, EMD is defined as

$$EMD(\hat{p}, p) = \sum_{j=1}^{|\mathcal{C}|-1} |\sum_{i=1}^{j} \hat{p}(c_i) - \sum_{i=1}^{j} p(c_i)| \qquad (4)$$

and can be computed in $|\mathcal{C}|$ steps from the estimated and true class prevalences. Like KLD in Equation 3, EMD is a measure of error, so lower values are better; EMD ranges between 0 (best) and $|\mathcal{C}| - 1$ (worst).

The dataset we use is the one released within SemEval 2016 Task 4 "Sentiment Analysis in Twitter" [8], and consists of tweets labelled according to five degrees of sentiment, from VeryNegative to VeryPositive. The dataset comes broken down into subsets TRAIN (6000 tweets), DEV (2000 tweets), and DEVTEST (2000 tweets); we use them for training, parameter optimization, and testing, respectively.

[2]The code that implements our method is available from http://alt.qcri.org/tools/quantification/

Table 1: *EMD* values obtained by the six methods (lower is better) and percentage of deterioration with respect to the best performer.

PCC(SVM)	CC(SVORIM)	ACC(SVORIM)	CC(ϵ-SVR)	ACC(ϵ-SVR)	OQT(SVM)
0.222 +5.71%	0.337 +60.48%	0.653 +210.95%	0.325 +54.76%	0.675 +221.43%	**0.210**

Each of these three sets is broken down into "topics" consisting of 100 tweets each; i.e., the task is to quantify the prevalence of a certain class (say, VeryPositive) among the comments *about a certain topic*. One thus needs to compute EMD separately on each topic, and then average the results.

For lack of space (and because it is not the main focus of this paper) we do not describe the preprocessing steps we adopt to generate the vectorial representations for our tweets; the details are given in [7, §4.1].

We evaluate OQT against the following baselines. The 1st is a PCC SLMC quantifier; it does not take order $\prec$ into account but, as shown in [7] it is a very strong SLMC classifier anyway. The 2nd is a CC quantifier (see §1) that uses the SVORIM OC learning algorithm [2]. The 3rd is an ACC ("Adjusted Classify and Count" – see [6, §2]) quantifier that also relies on SVORIM. The 4th and 5th are again a CC quantifier and an ACC quantifier, this time using an adaptation to ordinal regression of the ϵ-SVR metric regression algorithm [3]. While OQT and PCC are not inherently SVM-based, we choose SVMs as their base learner, also for better experimental uniformity with the other four methods, which are inherently SVM-based; from now on the two former methods will thus be called OQT(SVM) and PCC(SVM). We use the implementations of (i) standard SVMs, (ii) SVORIM, and (iii) ϵ-SVR, as available in the LIBSVM suite[3]. For each learning algorithm we have optimized the C parameter (which sets the tradeoff between the training error and the margin) on the DEV set, performing a grid search on all values of type 10^x with $x \in \{-6, ..., 7\}$. Note that both PCC(SVM) and OQT(SVM) (which depends on PCC(SVM)) require as input not the classifier's binary decisions but its posterior probabilities. Since SVMs do not natively generate posterior probabilities, we use the `-b` option of LIBSVM, which converts the scores originally generated by SVMs into posterior probabilities according to the algorithm of [10].

The results of our experiments are given in Table 1. Our OQT(SVM) method is the best performer, with PCC(SVM) the second best, with a +5.71% deterioration with respect to OQT(SVM). Two aspects are interesting to discuss here.

The first is that the two best methods are the ones that make use of posterior probabilities, i.e., where class prevalences are computed as *expected values* of the binary class assignments; the other 4 methods are based on CC and ACC, which do not make use of posterior probabilities and use the binary class assignments instead. This seems a further (albeit indirect) confirmation of the results of [7], where PCC(SVM) emerged as the best of 8 methods in SLMC tweet quantification. The good result of PCC(SVM) is even more striking if we consider that the other 4 methods, which PCC(SVM) beats by a wide margin, leverage the $\prec$ class order while PCC(SVM) does not; this seems further evidence of the value of posterior probabilities in quantification.

A second observation is that ACC, a quantification method that has been consistently reported to perform better than CC [1, 5, 7, 6], here performs much worse than CC, in the context of both SVORIM and ϵ-SVR. This is an aspect that will deserve further investigation.

In Subtask E of the SemEval 2016 Task 4 shared task (a subtask which deals with ordinal tweet quantification by sentiment – see [8]), the system described in this paper obtained an EMD score of 0.243, ranking 1st in a set of 10 participating systems, with a high margin over the other ones (systems from rank 2 to rank 8 obtained EMD scores between 0.316 and 0.366). Overall, these results are encouraging and preliminary at the same time. For the future we plan to run more experiments, using more datasets (e.g., product review datasets) and more baseline systems from the world of ordinal classification and SLMC quantification.

[3]LIBSVM is available from http://www.csie.ntu.edu.tw/~cjlin/libsvm/

4. REFERENCES

[1] A. Bella, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. Quantification via probability estimators. In *Proceedings of the 11th IEEE International Conference on Data Mining (ICDM 2010)*, pages 737–742, Sydney, AU, 2010.

[2] W. Chu and S. S. Keerthi. Support vector ordinal regression. *Neural Computation*, 19(3):145–152, 2007.

[3] H. Drucker, C. J. Burges, L. Kaufman, A. Smola, and V. Vapnik. Support vector regression machines. In *Proceedings of the 11th Annual Conference on Neural Information Processing Systems (NIPS 1997)*, pages 155–161, Denver, US, 1997.

[4] A. Esuli and F. Sebastiani. Sentiment quantification. *IEEE Intelligent Systems*, 25(4):72–75, 2010.

[5] A. Esuli and F. Sebastiani. Optimizing text quantifiers for multivariate loss functions. *ACM Transactions on Knowledge Discovery and Data*, 9(4):Article 27, 2015.

[6] G. Forman. Quantifying counts and costs via classification. *Data Mining and Knowledge Discovery*, 17(2):164–206, 2008.

[7] W. Gao and F. Sebastiani. From classification to quantification in tweet sentiment analysis. *Social Network Analysis and Mining*, 2016. Forthcoming.

[8] P. Nakov, A. Ritter, S. Rosenthal, F. Sebastiani, and V. Stoyanov. SemEval-2016 Task 4: Sentiment analysis in Twitter. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval 2016)*, San Diego, US, 2016. Forthcoming.

[9] Y. Rubner, C. Tomasi, and L. J. Guibas. The Earth Mover's Distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.

[10] T.-F. Wu, C.-J. Lin, and R. C. Weng. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5:975–1005, 2004.