Non Uniform Sampling in Graph Convolution Networks

Author(s)
Rahul Jayan, Shantanu Singhal, Tarun Bansal
University of Wisconsin-Madison
jayan, singhal5, tbansal @ wisc.edu

Abstract *Machine Learning on graph networks is immensely valuable because of the ubiquity of graph representation in social networks and knowledge bases. The success of any machine learning algorithm depends largely on the underlying data representation and recently there has been renewed interest in designing graph embedding methods.In this paper, we implement a deep graph convolutional neural net for learning graph representations, which generates a sparse graph representation that supports attributed graphs and captures higher-order structures. We extend the Graph Convolution Network (GCN) model with GraphSage's Aggregator-Encoder architecture and propose new non uniform sampling techniques to inductively learn vertex embeddings. Our model AE-GCN achieved significant improvements over the skip-gram and other deep learning models.*

1 Introduction

Machine Learning on graph networks is immensely valuable because of the ubiquity of graph representation in numerous applications such as social networks and knowledge bases. Such supervised algorithms that make predictions over nodes and edges require a lower-dimensional vector representation of features that are independent and informative. Typically this is done by manually engineering features on a case-by-case basis using prior knowledge and domain-specific heuristics that don't generalize across different networks. Machine learning techniques that can learn such representations are favored over feature learning because they can be scaled up more easily.

Most of the works in learning node representations for graph-structured data have largely been based on the skip-gram model, which was originally introduced as a model for learning vector representations of words in the NLP domain. Popular examples like DeepWalk[10] and node2vec [4] use random-walk techniques to train the embedding model but require a large amount of CPU time to generate a sufficient number of walks. Such models are also unable to capture higher-order holistic features and produce only a coarse representation of the graph. In addition, features from such methods do not generalize to other networks, making it harder to use them for cross-network transfer learning tasks.

Deep learning models have proven to be highly successful for many analysis tasks on images and audio but haven't been well generalized well for domains like graphs or geometric data where nodes have varying connectivity patterns. In the last couple of years, a number of papers have re-visited this problem [1] [3] [6] [7], and many of them make use of graph convolutions known from spectral graph theory to define parameterized filters that work on arbitrarily structured graphs.

In this paper, we describe our implementation of a deep Graph Convolution Network (GCN) and qualitatively discuss the results on datasets of varying sizes to see how our model compares against other related methods in tasks such as node classification. We then analyze the efficacy of the GCN by monitoring its outputs during the training process and observing how the embeddings react. We also discuss the effects of network hyperparameters on the results. Finally, we describe the potential for future works with GCNs.

2 Background and Motivation

In this section, we describe the theory behind spectral formulation of Convolution Neural Networks (CNN) and provide a brief overview of the strengths and drawbacks of this approach and what motivated us to use this architecture for learning graph embeddings.

CNNs are so effective because they exploit the underlying structure of the data by learning locally aggregated features by applying kernel filter convolutions to local patches of the input, and the use of pooling operation that progressively reduces the dimensionality, allowing for hierarchical learning [8]. These operations work well in regular, grid-like Euclidean spaces because the data to be studied in those coordinates has translational invariance with respect to this grid. This allows for the use of global parameters, filters and subsampling, through strided convolutions and pooling. In their seminal paper "Spectral Networks and Locally Connected Networks on Graphs" [1] presented what they called a spectral construction of CNNs that works on graphs other than regular grids. To generalize the architecture, they drew on concepts from the graph signal processing domain where a convolution is computed as a simple point wise multiplication of the Fourier-transform of a signal. Despite the significance of their architecture their implementation suffered from computational drawbacks and have been subsequently been improved upon multiple times, most notably by Defferrard et al. [3] which proposed a technique for learning fast localized spectral filters with polynomial parameterization that can be represented as Kth-order polynomials of the Laplacian and have the same complexity as classical CNNs. These filters are exactly K-localized. which means it depends only on nodes that are at maximum K steps away from the central node (i.e Kth-order neighborhood).

GraphSAGE (Hamilton et al., 2017) [5] is a newly proposed architecture for generating vertex embeddings through aggregating neighborhood information. The FastGCN architecture proposed by Jie et al [2] is another closely related work that uses Monte Carlo approximation and importance sampling to handle memory challenges with growing neighborhoods in extremely large graphs. The two sampling schemes of GraphSage and FastGCN are fundamentally different in their approaches and cost but, both try to relax the requirement of simultaneous availability of data and use a batched training scheme to work with very large graphs.

3 Objectives

We draw inspiration from the related works of FastGCN and GraphSage and propose our own sampling strategy to reduce the computational and memory footprint of traditional GCN architecture and generalize the GCN model for inference.

The ability of CNNs to learn static local structures and aggregate them into high-level structural patterns has motivated our choice of extending GCN with GraphSage's Aggregator-Encoder architecture to achieve the following objectives

- Design a fast and scalable model that scales to large graphs with millions of edges.
- Learn a hierarchy of low-order features and higher-order subgraph structural features.
- Learn from both graph structure and node attributes, to support attributed graphs.
- Support across-network transfer learning tasks as much as possible.

4 Problem Statement

Given an input graph $G = (V, E)$ where V and E are the vertex and edge sets respectively and a pre-defined dimensionality F , where $F \ll |V|$; Our goal is to learn a function $f(X, A)$ that takes as input the signals/features from G given as

- A representation of the features of the graph as a matrix X of the shape N x D where N is the number of nodes, D is the number of input features and each node i is described by its feature vector x_i

- A representation of the graph structure as an adjacency matrix A of the shape $|V|$ x $|V|$ where each entry $A(u, v)$ denotes weight of the edge connecting u and v.

And generates a matrix output Z of the shape Nx F where N is the number of nodes and F is the number of output features, such this F-dimensional vector representation of each node best preserves the properties of the input graph G.

For this paper we have assumed G to be an undirected graph although it is possible to extend this approach to directed graphs as well.

5 Implementation and Proposed Techniques

We have drawn inspiration from architectures of GCN and Graphsage while designing our model architecture. Both of these models were designed with different objectives in mind. GCN worked really well for small and medium sized graphs while GraphSage was good for large graphs. Our model hopes to bridge this gap by introducing a model that can extend well for small or medium sized graphs while supporting large graphs as well.

Our main contribution involves extending the GCN model with GraphSage's Aggregator-Encoder architecture and using various sampling strategies to improve the quality of vertex embedding and the generalizability of these embeddings across networks. By making use of sampling and restricting the immediate neighbourhood size for each layer, we side step the computational cost incurred in traditional GCN implementations due to recursive neighbourhood expansion. This also handles the issue of memory bottlenecks that occur with very large graphs

Graphsage's Aggregator - Encoder architecture was designed for large sized graphs and as such performed relatively poor for small and medium sized graphs.The negative sampling for learning the weight matrices and the l2 normalization of the weight vectors were significant bottlenecks while extending for small graphs. This brought down classification accuracy while increasing computation. We could not extend GCN for large graphs as well as the layer wise weight updation was too computation intensive causing significant slowdowns.We then introduced our architecture by removing the above bottlenecks and introducing a layer wise sampling technique of our own. We also decided to stick with just mean aggregator for neighbourhood aggregation. This has proved to be computationally light while giving good performance.

The original GCN focussed on full batch gradient descent with neighbourhood sampling space spread across its entire neighbourhood. So we decided to use a batched sampling technique with batch size varying across different datasets. Instead of searching neighbours across its entire neighbourhood, we restrict those neighbours to within the current batch.This restriction of search space across batch size is used only for random sampling and hybrid sampling.

We briefly describe the algorithm for aggregating feature information from a node's local neighbourhood. This feature information mostly involves different text attributes per document. The embedding generation algorithm generates embedding for nodes assuming that the model parameters have already been learned.

5.1 Algorithm

We extend this algorithm from Graphsage paper [5]. Here we assume that we have learned parameters from k aggregator functions denoted by MEAN_AGGREGATE$_k$ which aggregate informations from the neighbors as well as a set of learnable weight matrices for all layers from 1 to K.Weight matrices learned in each layer is fed to a softmax function and RELU before passing as input to the next layer.This layer wise aggregation of neighborhood information directly translates to aggregation of node information across a search depth of size K. The final output h_v^K is then fed to loss function.

Loss Function In order to learn the predictions we apply a cross entropy loss function to the output representation, Z. The weight matrices and parameters of the aggregator function are learned via stochastic gradient descent.

Algorithm 1 Algorithm for forward propagation

Require: Graph $G(V, E)$ with feature vector X; depth K
Require: Weight matrix $W^k \forall k \in \{1..K\}$
$h_v^0 = X_v$, $\forall v \in V$
for $k = 1 \dots K$ **do**
  for $v \in V$ **do**
    $h_{neigh}^k =$ MEAN_AGGREGATE$_k(\{h_u^{k-1}, \forall u \in$ SAMPLE_NEIGH$(v)\}$
    $h_v^k = \sigma$ (W^k.CONCAT(h_v^{k-1}, h_{neigh}^k))
  end for
end for
Output: h_v^K

5.2 Sampling

The neighborhood of the nodes is defined by the sampling technique we use. This sampling technique can vary across layers. There are three sampling techniques which we have used widely - Random sampling , priority sampling, hybrid sampling. Priority sampling involves choosing high priority nodes from the current node's neighborhood. This is based on the intuition that highly connected nodes must have more weightage than any other nodes while aggregating neighborhood information. Hybrid sampling is a combination of random sampling and priority sampling. The ratio of priority to random in hybrid sampling is a tunable parameter that we fix by grid search.Mixed sampling involves choosing a combination of sampling technique for the two layers. We used Priority sampling in the first layer to learn more from its immediate neighbours while randomly sampling neighbours in the second layer to learn from neighbours which is at a distance of 2 nodes from the current one.

Through mixed/hybrid sampling across layers we have tried to build a more inductive training scheme that learns a model based on a representation of its neighbourhood in the training samples so that it generalizes better to unseen (test) data points from the same graph or other related graphs.This ensures sufficient generalization while not overfitting to the graphs seen in the training phase.This intuition was supported by results from the early experiments that we conducted, more on that in Section 7, Results.

5.3 Architecture

The architecture is dependent on the dataset and hierarchical structure of the graph .The sampling techniques across layers , the neighborhood size , the search depth k and batch size are the tunable parameters of our model. To perform model evaluation we have stuck to search depth of 2 as it has been proved to be a good choice [3] [4]. How the depth of neural network affects the classification accuracy is something which we have not explored much and is an interesting future work.

Since the model hyperparameters varies across datasets, the architecture can be explained easily by taking example of any of the commonly used datasets like that of technical paper citations. In these citation network datasets Citeseer, Cora and Pubmed, the nodes are documents and edges are reference links. The datasets contain sparse feature vectors for each document, a list of citation links between documents and each document has a class label. Statistics are summarized in Table 1. Note that the citation links are considered as undirected edges and all labels of the training examples are used for training unlike the semi-supervised setting of GCN [7] where only a small portion of the labels are used.

Figure 1 shows the network architecture of a 2-layer deep Aggregator-Encoder GCN with mixed sampling when used for Cora dataset.

In the below architecture diagram, the first layer, uses Hybrid Sampling with a wide window of 7 to create and compresses the input embedding from 1433 (in case of Cora) to 512 (in this example). The second layer, uses Random Sampling to pick 5 random neighbours in its immediate neighbourhood and further compresses their averaged representation to the final size of 128. The successive layers work to expand the neighbourhood from which a vertex learn its encoding, so a two layer deep architecture will learn from across its 2-hop neighbours. The configurable model design allows for different strategies at each hop. Currently, we're using our understanding of the network topology to

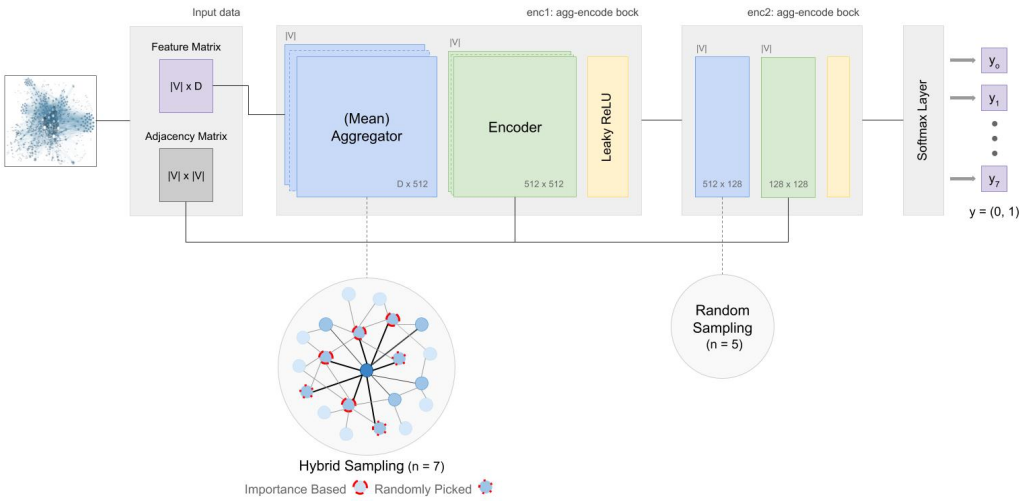

Figure 1: Architecture of a 2-layer GCN with Mixed Sampling

tune the network neighbourhood and sampling strategy parameters for each dataset but in a future implementation we would use candidate sampling to evaluate different options over a small subsample and pick the best one programatically.

6 Experiments

We test our model on graph datasets with increasing number of nodes and higher degree of connectivity to evaluate its performance when compared with the original GCN model and popular skip-gram models like DeepWalk and node2vec. We loosely follow the experimental set-up of FastGCN and evaluate these models on three benchmark tasks: classification of research topics with the Cora dataset,classification of research topics with the Citeseer dataset , categorizing academic papers with the Pubmed dataset. The statistics of these datasets are summarized in Table 2

The experiments have been designed to independently evaluate the impact of the design changes we made to the existing models along with that of the sampling strategies. The impact of neighborhood size or sampling size on the classification accuracy has also been evaluated to decide on the optimal parameter value. We adjusted the training/validation/test split of all datasets to align with the experimentation setup of other architectures.

6.1 Experimental setup

To contextualize the empirical results on our benchmark tasks we compare against four baselines: GCN , FastGCN, Deepwalk and our AE-GCN. We also compare our results using different sampling methods against each other. We compare the results against various combinations of three different sampling techniques we primarily use.

Three sampling strategies are used in our model namely Random sampling , priority sampling and hybrid sampling. All the combinations of these three sampling techniques across the different layers have also been evaluated for each of the three datasets.The choice of sampling technique is more dependent on the type of classification problem, the average connectivity of the graph and how generalizable is the learning across different subgraphs of similar hierarchical structure.

Our model was implemented in pytorch with vanilla Stochastic Gradient Descent.The set of hyper-parameters used for testing was determined on early validation tests using different subsets of each dataset.

Dataset	Nodes	Edges	Classes	Features
Citeseer	3,327	4,732	6	3,703
Cora	2,708	5,429	7	1,433
Pubmed	19,717	44,338	3	500

Table 1: Dataset Statistics

6.2 Results

The empirical results which we have achieved is tabulated in this section.We achieved significant improvements in classification accuracy across all the three datasets.The below table gives a comparison of our model against all the previous work in terms of test set accuracy.

Model	Cora	Citeseer	Pubmed
DeepWalk	67.2	43.2	65.3
GCN	81.5	70.3	79.0
FastGCN	81.8	-	77.6
GraphSage	82.9	77.2	84.9
AE-GCN-random	87.4	80.3	88.7
AE-GCN-priority	**88.5**	80.6	89.3
AE-GCN-hybrid	87.9	**81.3**	**89.9**
AE-GCN-mixed	87.6	81.1	89.1

Table 2: Performance Comparison: Test Set Accuracy

We achieved considerable improvement over existing techniques like graphsage and GCN for the given datasets.From the above results we can see that priority sampling worked exceedingly well for Cora dataset. This can be due to the fact that the dataset is relatively small and there are few nodes with degree much greater than the average node degree of the graph. This made it possible to aggregate neighborhood information using only those few highly connected nodes. For citeseer and pubmed datasets, the hybrid sampling worked best. We think the gap between priority and hybrid would widen as the graph size increases.

The impact of different sampling strategies across the two layers for each of the threee datasets is shown in Figure 2 3 4 We plot the learning curve to reach a validation accuracy of 90%.For all the datasets we can see that this is achieved more or less around 20 epochs.

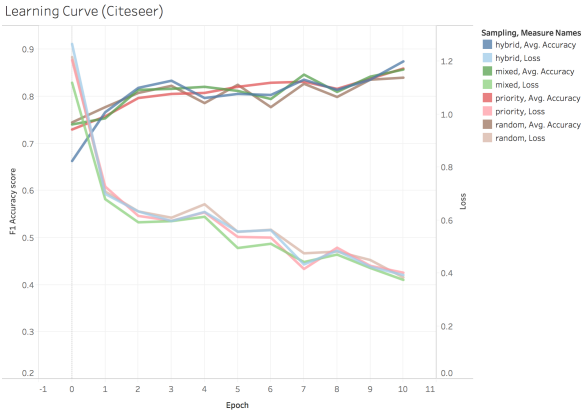

Figure 2: Learning curve during Training (Accuracy and Loss on dual axis) for Citeseer

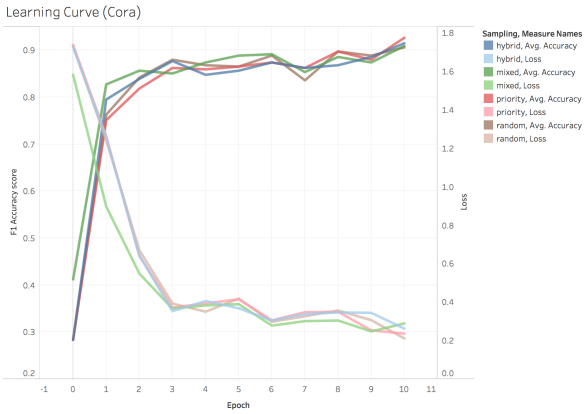

Figure 3: Learning curve during Training (Accuracy and Loss on dual axis) for Cora dataset

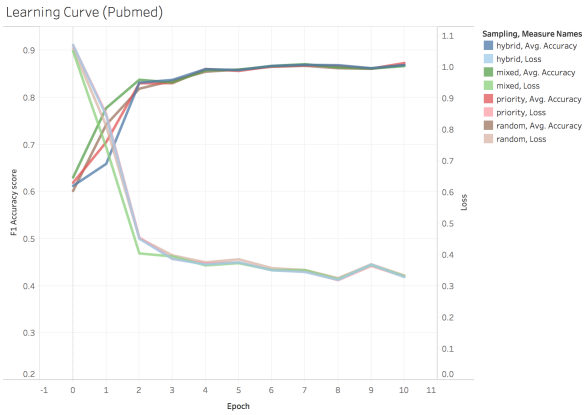

Figure 4: Learning curve during Training (Accuracy and Loss on dual axis) for Pubmed

On further evaluation, we can see that the validation accuracy achieved for layerwise sampling is good but not a significant difference is achieved across different datasets. The mixed sampling technique where we use two different techniques in the two layers also shows lot of promise. We think that these techniques can achieve optimum accuracy for very large graphs.

The test set accuracy for the three datasets for different sampling technique is given on Figure5. In this context, same sampling technique is applied across both the layers except in the case of mixed sampling where we perform priority sampling in the first layer and random sampling in the second layer.This means that the result for random sampling plotted has random sampling being used for both layers.The results show that our hybrid sampling approach gave good results for Pubmed dataset where the dataset is comparatively large compared to much smaller Cora and Citeseer whereas Mixed sampling worked much better for Citeseer.

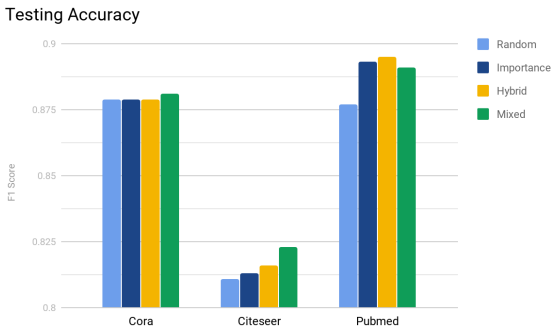

Figure 5: Accuracy on test data

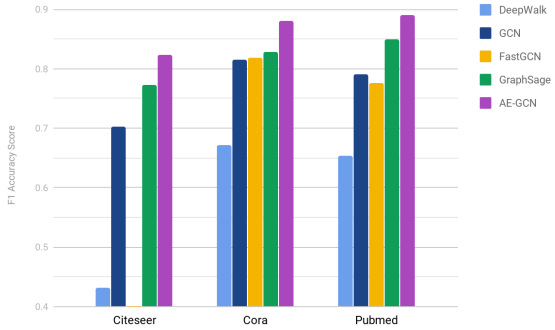

Figure 6: Accuracy comparison with related works

6.3 Setbacks

In this paper, till now we have focussed only on the results we have achieved , but not much on the various techniques we have explored to improve the performance of GCN for node classification.

We initially hoped to use poincare embedding [9] to achieve much higher classification accuracy. We aimed to utilise the inherant abilty of poincare embeddings to learn hierarchical representations of the graph by embedding into an n dimensional poincare ball. This theoretically would have given us faster convergence and improved classification. We found that the computation required for negative sampling and embedding onto hyperbolic space by comparing across the samples is too large. The poincare model [9] achieved attractive results by leveraging asynchronous lock free updates of the weight vector using multiple threads. Since the primary aim of this project was not to accelerate the gradient computation aspect, we decided to focus on the impact of sampling.

Another aspect of the project we explored is the generalization across the graphs. This requires learning more about the roles of each node in a graph rather than its community structure. We tried to perform these experiments with PPI dataset which consists of 20 graphs, 2400 average number of nodes and 28.8 as average degree. PPI is a dataset which classifies protein roles using various protein-protein interaction graphs. We could not perform experiments well with these datasets as we were restricted due to the limited support of pytorch for sparse matrices. We then performed experiments using smaller graph datasets but the results were not conclusive enough to evaluate how well our embeddings generalize across related but unseen graphs

7 Conclusion

Our intuition coupled with reading prior work on using deep learning for graph embedding led to significant improvements in prediction accuracy. Also, the aggregator encoder model first suggested

in Graph-Sage paper helped us in scaling the implementation. This approach combined with non uniform sampling strategies helped in generalizing GCN even more.

In our current approach we have primarily focused on single graph datasets. In future we would like to extend our approach to multi-graph datasets and enable transfer learning. Transfer learning helps in leveraging knowledge acquired in a source domain to improve the accuracy and speed of learning in a related target domain. This might lead to significant gains in certain datasets like PPI. We would also try to couple non uniform negative sampling with our approach to test its effect on prediction accuracy. Another interesting direction of future work could be the use of much deeper layered networks and alternative aggregators like LSTM and Max-Pooling.

References

[1] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[2] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

[3] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[4] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[5] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1025–1035, 2017.

[6] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[7] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[8] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[9] Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In *Advances in Neural Information Processing Systems*, pages 6341–6350, 2017.

[10] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.