Proceedings of the 33rd European Modeling & Simulation Symposium ((EMSS)), 022
18th International Multidisciplinary Modeling & Simulation Multiconference

2724-0029 © 2022 The Authors.
doi: 10.46354/i3m.2022.emss.022

Distributed Classification - A Scalable Approach to Semi Supervised Machine Learning

Rainer Meindl[1], Simone Sandler[1], Elisabeth Mayrhuber[1,*] and Oliver Krauss[1]

[1]Research Group Advanced Information Systems and Technology, Research and Development Department, University of Applied Sciences Upper Austria

*Corresponding author. Email address: elisabeth.mayrhuber@fh-hagenberg.at

Abstract

Fitting real world data into a model for classification, is a challenging task. Modern approaches to classification are often resource intensive and may become bottlenecks. A microservice architecture that allows maintaining a model of real world data, and adding new information as it becomes available is presented in this paper. Updates to the model are handled via different microservices. The architecture and connected workflows are demonstrated in a use case of classifying text data in a taxonomy represented by a directed acyclic graph (DAG). The presented architecture removes the classification bottleneck, as multiple data points can be added independent of each other, and reading access to the model is not restricted. Additional microservices also enable a manual intervention to update the model.

Keywords: Text Classification; Distributed Environment; Microservice Architecture; Semi-Supervised Learning.

1. Introduction

With the emergence and adoption of Industry 4.0 and, as a result, the Internet of Things (IoT), software systems started producing large quantities of data that cannot be processed promptly by classic monolithic software. Instead, a software solution based on microservices, which can handle this volume of data by distributing the workload on different machines, is proposed. IoT applications provide large amounts of data, such as sensor and time-series data, and allow the generation and capture of user-specific information. Depending on the type of data, processing has to take different problems into account. Sensor and time-series data might contain data loss, invalid values, or an overall data drift, while user data must cope with missing values and input errors. By introducing microservices, it is possible to process different data sources in parallel without incurring priority over data management and processing. One or more connected microservices can handle each data source. In order to use the resulting processed data, due to its expected size, semi-supervised or unsupervised learning is employed. Other machine learning methods rely on a perfect reference data set, which is often not feasible in real-world scenarios. These methods are expensive in terms of system load and need access to all data, frequently resulting in a bottleneck for the system as a whole. As modern software solutions already use microservice architectures and distributed calculation, moving semi-supervised learning into a distributed environment is suggested.

In this publication, the use case focused on is the text classification of restaurant products. The information is already extracted and available in its basic textual form. Example input data include:

- Wiener Schnitzel mit Reis und Petersilerdäpfel
- Backhendlsalat mit Kernölpanade
- Spicy Chickenburger

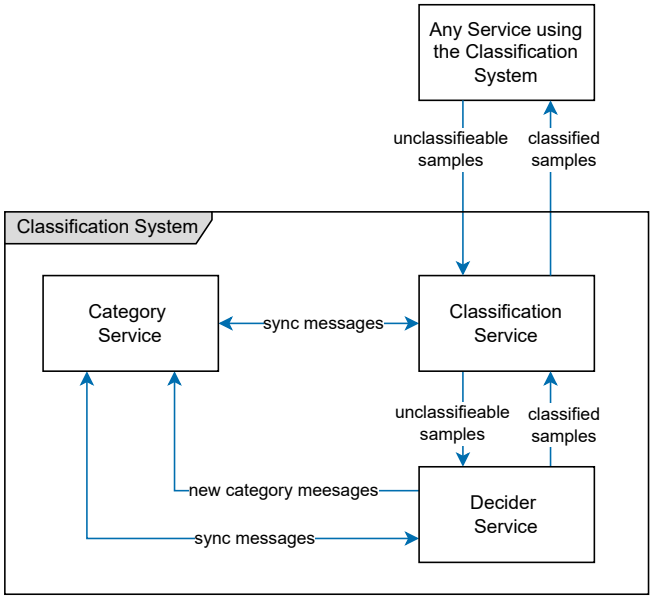

Figure 1. Overview of the suggested architecture, also containing message flow between the services.

The data has been generated by many different Austrian and German restaurant owners, resulting in heterogeneous descriptions for similar products. The aim is to process and categorize this data in order to create a basic data definition for German-speaking restaurants.

2. Architecture

A microservice architecture consisting of three microservice types is presented in this section. The first service, named *Category Service*, is a central repository for the possible classes. It provides unified and robust access to the classification targets consumed by the other services. The *Classification Service* contains the algorithm to assign a text sample to a class. It is dependent on the *Category Service* since it needs the predefined classes for the classification. The third service, referred to as *Decider Service*, is optional. Its purpose is to support the *Classification Service* when an assignment of a sample to a class is not certain. In such cases, it helps to decide the class using manual labelling, hence the name *Decider Service*. The following sections describe the architecture, focusing on communication interfaces and protocols.

Figure 1 depicts an overview of the orchestration of the microservices, including their messaging interfaces, inputs, and outputs. Each of these microservices is described in-depth in the following subsections.

2.1. Category Service

In order to enable distributed computation, there needs to be an external data storage holding the meta-information necessary for said computation (Salza et al., 2017). In this use case, the meta-information describes the possible categories for a given product of a restaurant. It is represented as a directed acyclic graph (DAG), persisted in a graph database. Each of these nodes is referred to as a category or cluster. Leaf nodes are the most discrete classification possible. This is unfeasible in a production environment, though, as text data is inherently convoluted and unreliable (Klein, 2001). Hence, hierarchical clustering using DAGs, where the path from the root to the leaf describes the possible classification is proposed. As seen in Figure 2, an abstract root node encapsulating all items in the DAG is assumed. Each edge describes a specification of the item class relative to the parent node. The full path from parent to leaf describes the actual classification. This path is pruned when the classification would not be confident enough in a lower level. The resulting incomplete path is defined as a *sub DAG* which can be used for further manual classification or refinement. Note that each sample can only be part of one cluster when manual classification is applied. This fundamental design decision has been taken since, otherwise, the evaluation for system users would have been deemed convoluted to allow proper product information management.

Access to this DAG is managed via RabbitMQ (VMware) messaging. Each service needing the DAG subscribes to the classification service. There are two types of messages, sync messages and update messages. Sync messages are commonly sent during the subscription of a service, as the category service shares the whole DAG using this message type. Update messages describe changes to the current DAG, that need to be reflected to all subscribers. Adding, deleting and replacing of both leaf nodes and subtrees in the DAG is supported. Whenever changes are applied to the DAG, update messages are sent and, given inconsistency in some service cache, sync messages may be requested.

Depending on its quality, the DAG might evolve during the runtime of the system, when used in conjunction with the *Decider Service* (see subsection 2.3). The system, as well as any end users, can suggest changes to the current DAG. As stated above, messaging can trigger changes and syncs of the DAG, but changes can be made directly to the DAG using the REST API of the *Category Service*. Any confirmed changes, i.e., changes authored or accepted by domain experts using a *Decider Service*, are synced to active services.

Initially, the goal is to fit all provided samples in each cluster of the DAG. During a starting period, no further clusters are proposed. After reaching a certain number of classified items inside a cluster, the accuracy of the classification service might suffer as the coherence of the cluster itself decreases (Song and Park, 2009). Furthermore, historical changes in data might inadvertently change the initial meaning of the cluster (Lu et al., 2018). Hence, advanced DAG management, like splitting a cluster into two, is recommended by the *Category Service*. Though, no such action is taken without confirmation of the user.

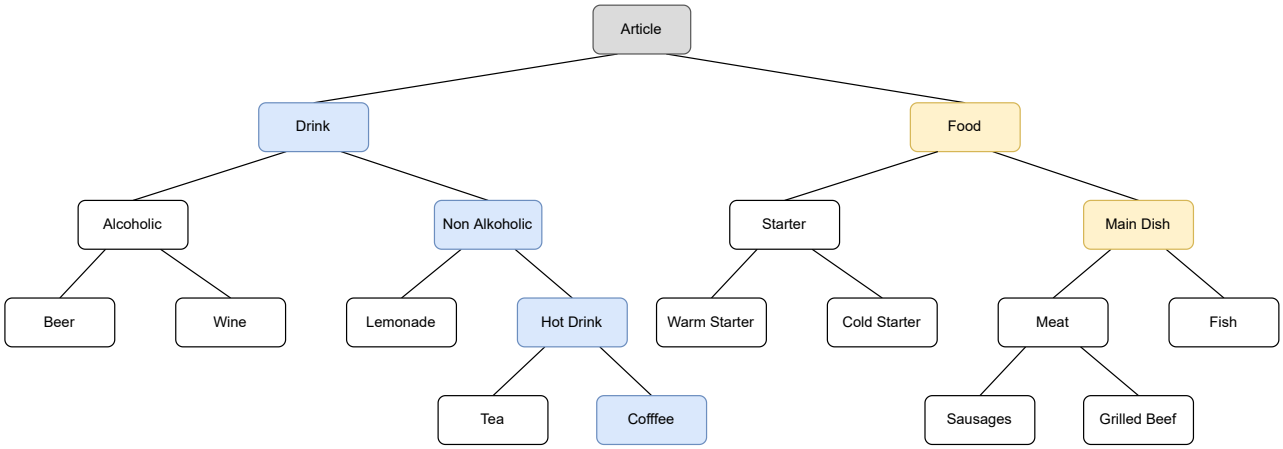

Figure 2. Example DAG describing food categories. Marked in blue is a possible complete classification path. The yellow path is an example of an unfinished classification path, allowing for further manual classification

2.2. Classification Service

The classification service encapsulates logic to process text and cluster it accordingly. Relative Levensthein Distance (Cheatham and Hitzler, 2013; Yujian and Bo, 2007) with substring classification (Zhang and Lee, 2006) is applied to product descriptions of restaurants. The data enters the system using asynchronous message queuing and undergoes standard string preprocessing steps, such as normalization of characters and number extraction (Petz et al., 2012). The results are comparable tokens in their corresponding normal form, that are split into contextual substrings. A sequence of substrings can then be used to match certain product names, which is in turn used as a feature for the classification service. This transformation increases robustness of the clustering, as it mitigates the effect of typographical error in clustering (Zhang and Lee, 2006). The goal is to aggregate the given data into hierarchical, cohesive clusters, whereas each cluster represents one possible category in the hierarchy for a given sample.

During startup of the classification service, it requests the DAG from the category service using a sync message (see subsection 2.1). The DAG describes the currently defined cluster, in which data samples can be classified into. In this service, the DAG is considered read-only and only modified, when the category service explicitly shares update messages. Thus, each instance of the classification service is dependent on the category service.

Given a text sample, the *Classification Service* tries to classify it into the current node's children. Depending on the cohesion of the clusters the children of the selected node are checked recursively, till one of two cases happen: (1) Leaf nodes of the DAG are reached, meaning the item can be clearly classified as one specific category, marked in Figure 2 in blue. (2) The confidence of the classification falls below a certain threshold, resulting in the classification of the sample in the current node. In Figure 2, this path is marked in yellow. This uncertain classification can manually be refined using the *Decider Service* (see subsection 2.3), in turn increasing the classification accuracy for future samples.

During the system's lifetime, classification results may not be absolute. Clusters may be added and removed, new data may make a cluster more attractive for a subset of samples of an already established cluster, or new datasets might describe different domains entirely (Lu et al., 2018). Hence, reevaluating the classified samples during the system lifecycle to ensure they are still part of the most fitting cluster, if any changes to the DAG are registered, is suggested. These changes are suggested by domain experts using a *Decider Service*. Each sample affected by the change is considered unclassified again and reintroduced to the message queue which holds the input data, i.e., the samples yet to classify. The classification starts from the beginning for each of these samples.

2.3. Decider Service

Many machine learning algorithms are strongly dependent on labelled data. In terms of clustering, the lack of labels is not as severe, but still an issue for the correct execution of the system as a whole. Due to the lack of context, data might be classified with high confidence into wrong clusters, as there are no references yet. This dilemma is called the *cold start problem* (Darshna, 2018).

Chicken Salad is used as an example, as shown in Figure 3. Given a cluster *Salad* and a cluster *Chicken Dish* chances are that the sample is classified into salad, which is initially correct, as both the Levensthein Distance as whole and relative to the substring matches quite well. But this raises the issue, that further samples containing the phrase *Chicken*, such as *Garlic Chicken*, can be considered *Salad*, as the resulting *Salad* cluster would be more cohesive than the *Chicken Dish* cluster, when relying on the Levensthein metric with substring classification. The same applies the other way round. If *Chicken Salad* is classified as *Chicken Dish*, *Caesar Salad* might follow the same principle. This problem, caused by insufficient data, will

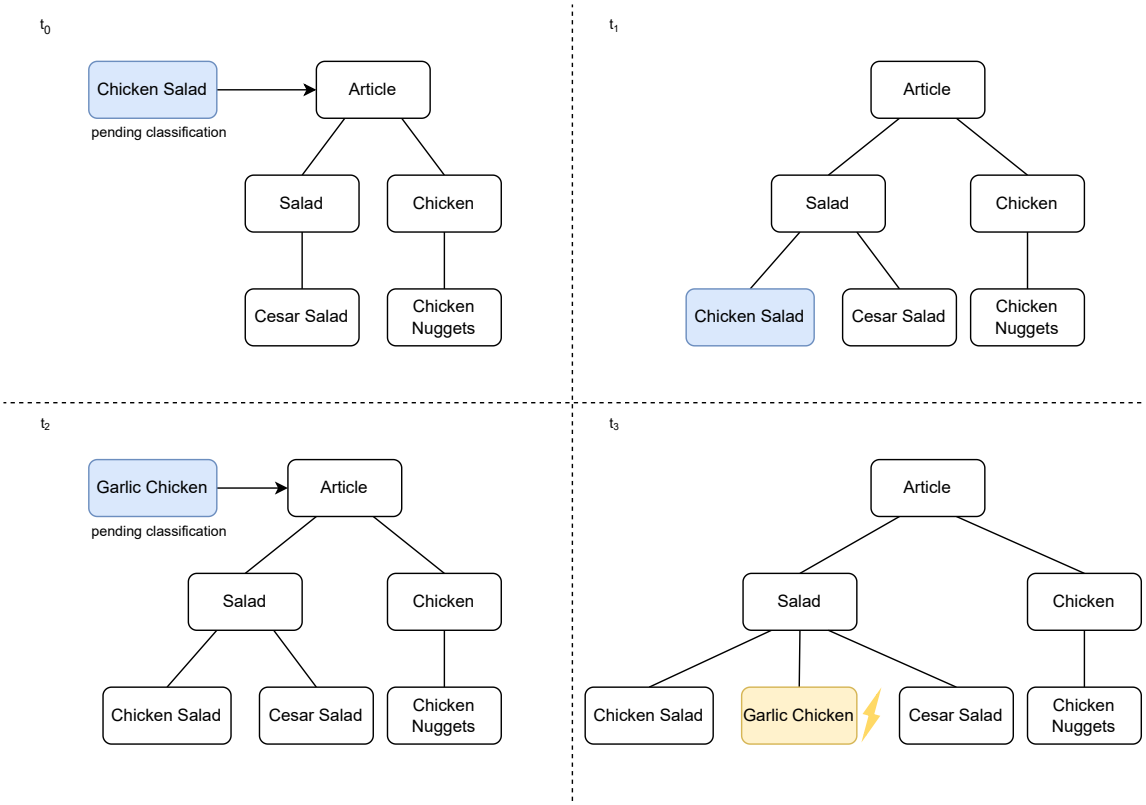

Figure 3. Example for possible misclassification during application start with little to no labeled data.

be mitigated by introducing a limited supervision component into the system. Each classification is denoted with a confidence for each possible cluster. The cluster deemed most suitable for a sample in terms of confidence still has to surpass a predefined threshold for automatic classification. Otherwise, manual classification by domain experts is necessary.

A *Decider Service* is a microservice, that acts as an interface for domain experts to manually classify samples. The goal is to reduce the necessity of such services to a minimum, as manual classification is slow, costly, and biased (Ur-Rahman and Harding, 2012). But, as described above, certain situations may need additional support during classification.

One other important purpose of this service is increasing the amount of articles that can be classified. As can be seen in Figure 4, the amount of classifiable items with the original labelled items has a certain limit (larger areas with blue background). In order to increase the number of classifiable items, manually labelled items is added in multiple intervals (smaller areas with yellow background). This results in more labelled items that can be used during automatic classification. The ratio of manually labelled items to automatically classified items is minimized, when performing manual classification as soon as the amount of automatically classified items stagnates.

Depending on the available resources, meaning data, computational power, and workforce, multiple *Decider Services* can be deployed, as they are independent of each other. Two strategies are supported when dealing with multiple domain expert inputs:

First come first serve The first manual classification is assumed correct, and the sample is removed from further classification. Any other input is dropped.

Cooperative weights Given more than one registered *Decider Service*, a manual classification corresponds to a positive weighting for a cluster in the classification process. The sum of these weights determine the cluster this sample is classified into. Given two manual classifications in different clusters, the *Classification Service* chooses the one manual classification with the higher weight to break the tie.

Each *Decider Service* receives the samples to manually classify via asynchronous message queues and also syncs the DAG of the *Category Service*. This also allows independent horizontal scaling of these services, by adding one or more instances. The received samples are cached locally, and during each manual classification step, a random subset of the cache is selected for the domain expert to resolve. Each subset is annotated with the possible clusters and confidences, as determined by the *Classification Service*, to support the decision-making process of the domain expert. The resulting classified sample is shared throughout the system via message queues and removed from all *Decider Services*.

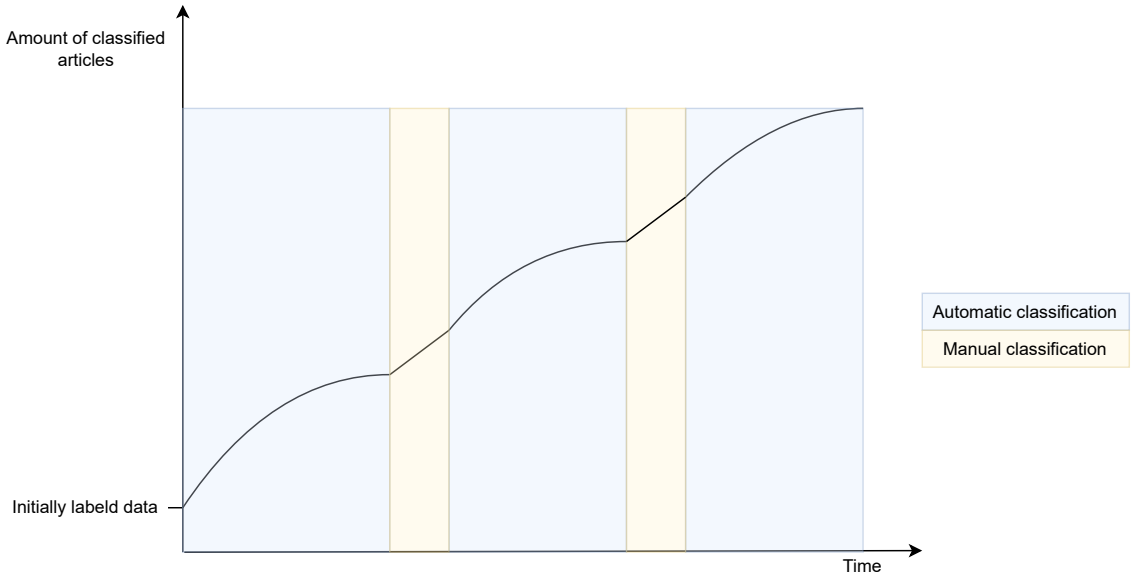

Figure 4. Schematic visualization of the amount of classified articles over time from Sandler (2021).

In the case that a sample does not fit into any cluster, an interface for suggesting a new category to the *Category Service* is provided. The domain expert has to designate the name of the category as well as its position in the DAG. Additionally, depending on the relative position of the new category in the DAG, they may trigger the re-clustering process of all samples in the affected DAG changes. Changed, or removed categories trigger the same process, while newly introduced categories only cause the samples of all parent nodes to be reclassified. This behaviour for new categories can be deactivated by the domain expert per new category.

3. Related Work

Distributing machine learning algorithms has been tried and tested several times using different explicit algorithms like Support Vector Machines, or Relevance Vector Machines (Scardapane et al., 2016; Silva et al., 2010).The difference to the approach of this paper is, that with the approach of Silva et al. (2010) the classification tasks are modelled with a DAG but the items which need to be classified are static. In the case of this publication, the items which need to be classified can be added dynamically with message queues. Most previous works have studied the benefit of having multiple nodes prepared and trained the same algorithms for improved accuracy and reliability (Nedelkoski et al., 2019). The mentioned publications show an increase of performance when using distributed classification, in this paper the goal an increase in availability due to the fact that the distribution enables read access to the current DAG while there are still new items coming in.

Salza et al. (2017) present cCube, a solution for ensemble machine learning tailored to evolutionary machine learning classifications. It employs a microservice architecture to train multiple models in parallel and generate an optimal algorithm for a given problem. Other than cCube, this approach gives the possibility for manual classification combined with an automatic one.

In recommendation systems, clustering of user interests is used to determine further interests of this and similar users. Darshna (2018) applies k-Means clustering on musical features and user ratings to determine good suggestions for different users. Additionally, a mitigation of the cold start problem by enriching small datasets with currently popular data samples is proposed. This approach is only feasible, when a metric for popular items exists, which, for most non-marketing related use cases, does not exist as it is in this case. In the proposed approach, clustering with k-Means is not possible in the first step because there are no explicit metrics that can be used. This is why specific algorithms were developed for this use case. Sandler (2021) described these algorithms in her Master Thesis. k-Means is used by the described system to cluster restaurants based on the products they offer.

4. Conclusion and Future Work

This distributed approach of classifying items allows scaling performance fluently by adding more machines for manual labellings, like the *Decider Service*. Also, the classification can be distributed to several machines, which enables querying the graph while classifications are processed. Also, the quantity and quality of the labelling can be improved, as it can be seen in Figure 4. In the future, this approach's performance enhancements will be tested precisely in several test settings to determine which bottlenecks can be eliminated.

References

Cheatham, M. and Hitzler, P. (2013). String similarity metrics for ontology alignment. In *International semantic web conference*, pages 294–309. Springer.

Darshna, P. (2018). Music recommendation based on content and collaborative approach & reducing cold start problem. In *2018 2nd International Conference on Inventive Systems and Control (ICISC)*, pages 1033–1037. IEEE.

Klein, B. D. (2001). User perceptions of data quality: Internet and traditional text sources. *Journal of computer information systems*, 41(4):9–15.

Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., and Zhang, G. (2018). Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363.

Nedelkoski, S., Cardoso, J., and Kao, O. (2019). Anomaly detection and classification using distributed tracing and deep learning. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 241–250.

Petz, G., Karpowicz, M., Fürschuß, H., Auinger, A., Winkler, S. M., Schaller, S., and Holzinger, A. (2012). On text preprocessing for opinion mining outside of laboratory environments. In *International Conference on Active Media Technology*, pages 618–629. Springer.

Salza, P., Hemberg, E., Ferrucci, F., and O'Reilly, U.-M. (2017). Ccube: A cloud microservices architecture for evolutionary machine learning classification. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '17, page 137–138, New York, NY, USA. Association for Computing Machinery.

Sandler, S. (2021). Classification of Restaurant Articles into a Taxonomy. Technical report.

Scardapane, S., Fierimonte, R., Di Lorenzo, P., Panella, M., and Uncini, A. (2016). Distributed semi-supervised support vector machines. *Neural Networks*, 80:43–52.

Silva, C., Lotric, U., Ribeiro, B., and Dobnikar, A. (2010). Distributed text classification with an ensemble kernel-based learning approach. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(3):287–297.

Song, W. and Park, S. C. (2009). Genetic algorithm for text clustering based on latent semantic indexing. *Computers & Mathematics with Applications*, 57(11-12):1901–1907.

Ur-Rahman, N. and Harding, J. A. (2012). Textual data mining for industrial knowledge management and text classification: A business oriented approach. *Expert Systems with Applications*, 39(5):4729–4739.

VMware. Rabbitmq. https://www.rabbitmq.com/, accessed on, 13 May 2022.

Yujian, L. and Bo, L. (2007). A normalized levenshtein distance metric. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1091–1095.

Zhang, D. and Lee, W. S. (2006). Extracting key-substring-group features for text classification. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 474–483.