Nothing Left to Learn: A Technique for Detecting Model Convergence in Big Data Exploration

Ryan Marcus
Brandeis University
rcmarcus@brandeis.edu

Vladimir Susaya
Brandeis University
vsusaya@brandeis.edu

Anna Yatskar
Brandeis University
aiy12@brandeis.edu

ABSTRACT

Since data exploration tasks are often labor-intensive, several systems have emerged that help steer users towards interesting data based on human-in-the-loop-feedback. These systems attempt to learn what regions of a dataset the user finds interesting by iteratively asking the user to label specific data items as interesting or uninteresting. Unfortunately, these systems lack reliable ways to detect when their models have converged, i.e., when the user's interests are well approximated. Thus, users must be sufficiently experienced with the data exploration tool to know when the system has converged. We discuss two algorithms for detecting exploration model convergence that can be easily added on top of many existing data exploration systems. The first algorithm relies on existing literature and uses a window of cross-validated F-score measurements to determine if the model's performance is improving. The second algorithm, Nothing Left to Learn (NLTL), uses a novel approach to more accurately and quickly detect convergence by testing if another round of human-in-the-loop interaction could possibly benefit the model.

1. INTRODUCTION

When dealing with big data, manually exploring data with traditional database management systems (DBMSes) can be cumbersome, tedious, and time-consuming. As such, Interactive Data Exploration (IDE) applications have come about in an attempt to improve on these drawbacks. IDE applications will generally require human input in their process, and can thus be considered "human-in-the-loop" systems. This human interaction is needed to evaluate what kind of results are relevant to the user's current interests. A process like this can be generalized as an active learning cycle in which samples are generated and then labeled or categorized by the user. Using this feedback, the application then creates or adjusts an existing model to predict the relevant area in the data. This cycle, depicted in Figure 1, is significant in that we can substitute different methodologies and classification models into this framework, while simultaneously minimizing the amount of work needed to be done by the user. This cycle continues until the user explicitly terminates the process or declines to provide any additional feedback, most likely as a result of the proposed model being good enough that further samples would not cause significant changes in the model. At this point, one would consider the model "converged", and the cycle can be halted.

The issue, of course, is that this assumes that the user, who is often assumed to not know exactly what they want, will be able to detect when the model has converged. Users unfamiliar with a dataset may waste time labeling additional data when the model already captures their interests, or users may make analytic errors based on data provided by under-fit models. Both results are undesirable because they impede analysis and potentially induce bad decision-making. Thus, an ideal IDE system would automatically terminate when the model reaches convergence.

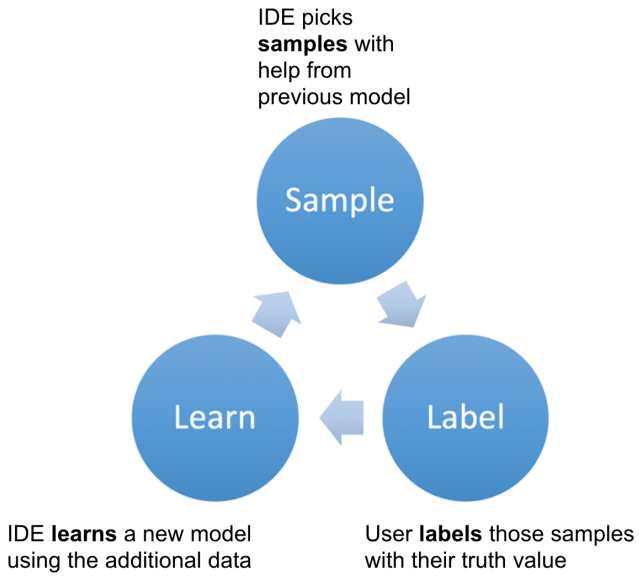

Figure 1: The IDE cycle

However, the problem of detecting convergence is difficult due to its subjective nature: how good is good enough? The definition of convergence may vary from application to application (a doctor may require a more accurate model than a retailer), from dataset to dataset (if there is significant noise in the dataset, model accuracy beyond a certain significance is useless), and even from day to day (what was good enough last year may not be good enough next year). Here, we present two techniques that can help remove the duty of evaluating whether a model has converged from the user. The first is based on ideas classically used in statistics and machine learning called buffered linear regression, and the second is a novel algorithm called "Nothing Left to Learn".

In Section 2, we discuss works related to data exploration and convergence detection. In Section 3, we describe the buffered linear regression method. In Section 4, we introduce the Nothing Left to Learn algorithm and detail it formally. In Section 5, we present experimental results showing the performance of both algorithms in conjunction withe the AIDE [4] system. We give concluding remarks and discuss future work in Section 6.

2. RELATED WORK

Recently, several systems [3, 10, 5, 4, 13] have been conceived to guide users through large datasets. In [4], a decision tree model is built based on data labels provided iteratively by the user, and the authors seek to minimize the number of items to be labeled. However, there is no clear mechanism to determine when enough items have been labeled or when the model will cease to improve. In [3], the authors propose a database navigation system which guides the user through interesting parts of the dataset, taking into account the user's interests and the interests of previous users. Again, the authors want to minimize the number of iterations need to converge, but only suggest looking at previous query logs to determine when convergence has occurred. In [13], the authors use a probabilistic formulation to model the user's desires and to inform the iterative search project, but they do not suggest any techniques for detecting convergence. In [5], the authors do not build a model of the user's interests, but suggest other items that might be interesting to the user given a specific query. In [10], the authors structure data exploration as a data cube navigation task, and use pre-fetching to help the user navigate such cubes quickly. In [17], the authors propose a system that learns autonomously and continuously, learning the user's requirements over time. However, [17] is intended to find "interesting" data, not specific data required by a user at a specific time.

Several systems [9, 8, 1, 7, 10] seek to increase the performance of iterative exploratory workflows, while other systems [6] suggest specific SQL queries given a string of text. However, all of these systems still require the user to select or produce exact SQL queries, which may be inaccessible to non-expert users and require excessive manual searching [4].

Other work focusing on convergence detection have been too domain-specific [12], focused on eventual convergence (and thus not on detecting the first point where the model is converged) [18], or have used methods that cannot be easily adapted to human-in-the-loop systems [2, 16].

3. BUFFERED LINEAR REGRESSION

The buffered linear regression method for detecting convergence involves monitoring the cross-validated F-scores of the model after each "Learn" phase, as suggested in [19]. The motivation behind this approach lay in the importance of distinguishing between local plateaus and true convergence, in which no further iterations could produce a better model.

Initially, a buffer of size n is allocated containing null values. At each iteration of the cycle, the cross-validated F-score [15] of the model is added to the buffer. If there are more than n items in the buffer, the oldest F-score is removed, such that any given time there are no more than n items in the buffer. After adding an item to the buffer, we perform a linear regression [14] on the F-scores stored in the buffer. We then classify the slope of the line as a *small slope* or a *large slope*, depending on if the slope of the line is less than or greater than a fixed threshold ϵ. A small slope indicates that the average F-score has not significantly changed over the course of the last n samples, suggesting that the model may have converged, and a large slope indicates that the average F-score of the model has changed, and thus the model has likely not converged.

Of course, just because a model has relatively small changes in F-score for a set number of iterations does not mean the F-score will not substantially improve with additional information. A window size that is too small will result in plateaus being considered converged, even though a drastic improvement in F-score may happen at the next iteration. However, a window size that is too large will waste the user's time, causing them to label unnecessary data.

We evaluated windows of size 2, 4, 8, 16, and 32 points. Sizes were doubled in order to observe any correlation between window size and the corresponding iteration number chosen to declare a model converged. It was found that beyond window sizes of 2, 4, and 8 points, 16- and 32-point windows produced comparable results; hence, a window size of 8 and 16 points was chosen for experimentation and subsequent evaluation.

Note that in this method of buffered linear regression, a window of a given size was shifted (i.e. slid) one iteration forward, such that the analysis for a window of size 16 first began at iteration number 17, then 18, 19, and so on. Rather than using such a fixed-length sliding window technique, a sequentially increasing window size could have been implemented just as well; in that case, windows of sizes i, $i+1$, $i+2$, etc. where i is the number of iterations would have been compared amongst each other. This type of comparison, however, has essentially been accounted for by the size-doubling procedure, which revealed that once the window was of a respectable size (around 16 or 32 points), its length did not significantly affect the iteration number required for convergence. There are also computational benefits to only storing a fixed number of F-scores, as less memory is used and the linear regression procedure is performed on fewer data points.

4. NLTL

Next, we present the Nothing Left to Learn (NLTL) algorithm. The NLTL algorithm happens in between the "Sample" and "Label" phase of the IDE cycle (as depicted in Figure 1). Given a set of samples that have yet to be shown to the user, NLTL detects convergence by checking to see if the *maximum possible change in the model* is significant. Intuitively, the algorithm can be described as:

1. After a "Sample" phase, we have a set of samples for the user to classify and a model representing the system's current best guess of the user's interests.

2. Use the current model to predict the results of each sample, and then invert the results. In other words, the system makes its best guess as to what the user will label each new data point, and then flips that guess.

3. Train a new model with the addition of these new, inverted guesses. This can be accomplished by adding the inverted guesses to the set of labeled data and then using whatever supervised training technique is employed by the system to generate a new model.

4. Check to see if the difference between the new and old model is significant. If it is not, then the model has likely converged.

We next explain the NLTL algorithm formally. Before the "Label" phase, an IDE has access to two pieces of data: the most up-to-date model (which was trained on the newly labeled data in the previous "Learn" phase), and the data points that were just sampled. Let the entire set of currently labeled data be represented by L, where each $l \in L$ is a tuple of $(data, label)$. We assume that all labels are boolean (true or false), although extending NLTL to handle probabilistic labels would not be difficult. We call the set of data most recently sampled (that has not yet been labeled by the user) U, where each $u \in U$ is a single data point.

We represent the model trained on all the labeled data as $M(L)$, and we represent the data-labeling function of a model as $\sigma(M(L), d)$, which takes an unlabeled data point to a label. We

represent the user as a model which always provides the correct label, and thus we write the user-provided label for a piece of data as $\sigma(Truth, d)$.

We additionally assume that, at the current iteration, all of the data in L is correctly labeled, although this requirement can be relaxed in exchange for a decrease in accuracy. Formally,

$$\forall (d, l) \in L \quad (l = S(d))$$

We will also take advantage of a *model delta function* Δ that measures the differences between two models. For two models trained on two different data sets, L and L', we expect $\Delta(M(L), M(L'))$ to be zero if $M(L)$ and $M(L')$ are extremely similar, and we expect $\Delta(M(L), M(L')$ to be one if $M(L)$ and $M(L')$ are substantially different. Since any given model $M(L)$ can be thought of as a distribution of the entire dataset, normalized Earth Mover's Distance (EMD) [11] is a good choice that can efficently computed. We use EMD in our experiments.

The fundamental principle of this method is as follows: before each "Label" phase, the labels of U that will cause the greatest change in the model are the opposite of what the current model predicts each label of U to be.

Consider the set of labels for U predicted by the current model.

$$U' = \{(d, \sigma(M(L), d) \mid d \in U\}$$

If we trained the model again using L and these new data points, the model would not change substantially, because no additional information has been added. Formally:

$$\Delta(M(L), M(L \cup U')) \approx 0$$

This is intuitively clear in the case of decision trees. At each stage of decision tree construction, the decision tree training algorithm selects the split that maximizes entropy [15] with respect to the label of each data point. So, adding U' to L will not change any of the splits chosen by the training algorithm, since the labels for U' were selected based on the same entropy model as L.

Consider, however, the set $\neg U'$, which is the same as U', but with all the labels flipped.

$$\neg U' = \{(d, \neg\sigma(M(L), d) \mid d \in U\}$$

The set $\neg U'$ represents the *maximum possible change in entropy* from the labeling of the data in U. In other words, $\neg U'$ is the labeling that will produce the greatest possible change in the model. Therefore, for any labeling x, we know that $\neg U'$ weakly dominates x:

$$\Delta(M(L), M(L \cup \neg U') \geq \Delta(M(L), M(L \cup x)$$

Intuitively, this will be the case for many other machine learning algorithms as well. Since most models are trained to minimize a loss function related to the labels of the training set [14], the largest change in the model likely occurs when new data that contradicts previous assumptions are encountered.

If the greatest possible change in the model for another iteration is small (below some threshold α), we can argue that the model has converged, since it is impossible for the information provided by the next step of the algorithm to make a substantial change to the model. If this is the case, the model cannot possibly gain any additional information from an additional "Label" phase. We thus say that the model has *nothing left to learn.*

Of course, it might be the case that more iterations would create a substantial change in the model, but the next iteration certainly will not. Regardless of if future iterations will make a difference, if $\Delta(M(L), M(L \cup \neg U') < \alpha$, there is no reason to ask the user to label U. Further, we can use the same technique multiple times: we can look at what would happen if the user labeled all of U against the model, and then labeled all of U_2 (the next set of samples produces). If the labeling of U and U_2 against the model is insufficient to produce a significant change, we can either stop or check up until U_n. Of course, this becomes computationally expensive, since the model is very unlikely to incorrectly classify 100% of the sampled data. Large values of n requires potentially expensive sampling of the data space that might not be worthwhile. This is especially true if the model is making predictions that are mostly correct: inverting those predictions and sampling from new areas of the database based on those inverted predictions will likely not produce results that can be reused in any meaningful way.

In the next section, we show how NLTL performs against the buffered linear regression approach explained in the previous section.

5. EXPERIMENTS

We now present our experimental results, based on target queries on the Sloan Digital Sky Survey (SDSS). We evaluated our convergence detection technique on AIDE, an automative user navigation system for interactive data exploration [4]. AIDE exemplifies the Sample-Label-Learn cycle depicted in Figure 1. It uses advanced sampling techniques to select relevant samples from an arbitrary database, then asks the user to classify them. AIDE then trains a decision tree on the labeled results, and uses that decision tree to inform its sampling in the next cycle.

To begin, we generated 21 sample queries over the SDSS dataset, which were run through AIDE. Based on these results, we manually determined the point of convergence by considering the model generated at each iteration.

The 21 sample queries only look at the attributes ra and dec of SDSS, which determine the section of space in which we expect to find celestial objects. The values of the attributes were chosen by looking at the distribution of these objects, and choosing queries that return approximately 300 to 500 tuples. This range was selected to be large enough for AIDE to detect, but also small enough so that multiple iterations would be required to refine the model around the area. It is also worth noting that these target queries only cover single, contiguous area of data, and do not take advantage of AIDE's ability to model multiple relevant areas.

We compared the hand-selected convergence points with the convergence points detected by the buffered linear regression method with window sizes 8 and 16, and a cutoff value of $\epsilon = 0.001$. These window sizes was chosen because they represent good tradeoffs between window size and window accuracy, as determined by cross-validated search. For NLTL, we chose a significant threshold if 5%, which was again selected via cross-validated search. The raw values for each query are shown in Table 1, where "Truth" represents the hand-selected convergence point, "NLTL" represents the Nothing Left to Learn algorithm, "BLR8" represents a buffered linear regression with a window size of 8, and "BLR16" represents buffered linear regression with a window size of 16.

Using these points of convergence, we evaluated the root-mean-squared-error (RMSE) of these points with the predicted points of convergence as determined by our two methods to evaluate the error, as well as the percent accuracy. The results are shown in Table 2.

We found that buffered linear regression resulted in an error of 34.51% and 27.93% for window sizes of 8 and 16 respectively. Both window sizes had a tendency to overestimate the correct point

Query	Truth	NLTL	BLR8	BLR16
Q1	63	56	28	34
Q2	30	34	22	28
Q3	27	31	28	36
Q4	39	47	23	40
Q5	28	40	19	27
Q6	39	29	20	26
Q7	36	31	37	37
Q8	28	24	16	20
Q9	33	39	20	26
Q10	42	28	22	42
Q11	18	18	28	35
Q12	38	34	41	18
Q13	34	32	34	18
Q14	38	30	29	20
Q15	41	37	18	40
Q16	39	43	24	44
Q17	33	9	7	8
Q18	38	28	20	28
Q19	36	30	20	33
Q20	22	20	27	18
Q21	43	40	26	20

Table 1: Values for each method and ground truth

Method	RMSE	% Err
BLR8	15.11	34.51%
BLR16	12.69	27.93%
NLTL	8.46	18.91%

Table 2: Root mean squared error and percent error for various methods

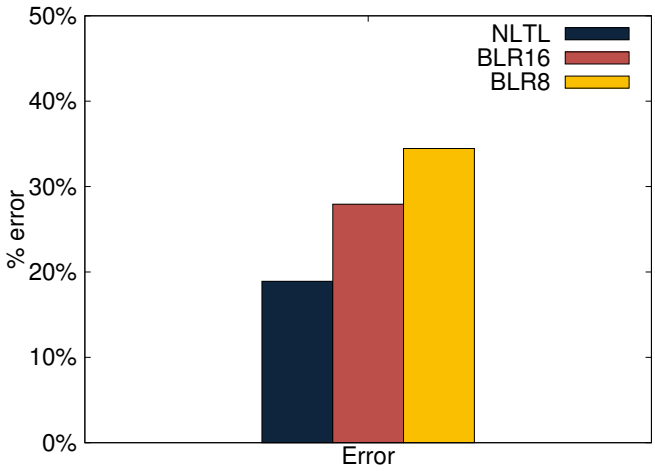

Figure 2: Percent error of each method

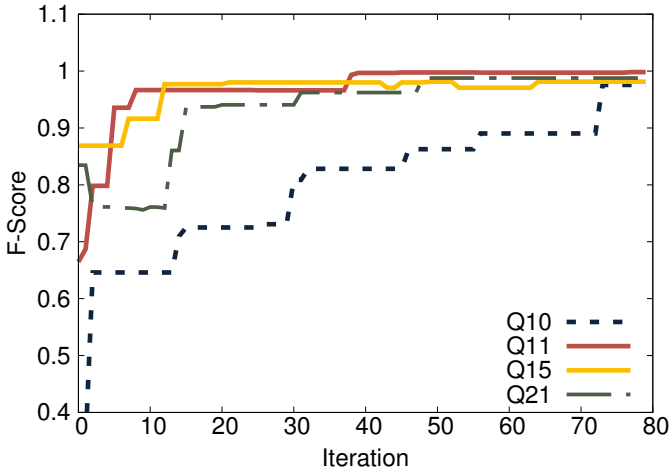

Figure 3: Cross-validated F-scores for model at each iteration

of convergence. In other words, this method tends to predict a converged iteration number that is lower than the actual iteration number. This is because the buffered linear regression method will detect convergence at the first plateau large enough to fill the buffer, despite the fact that a buffered linear regression with window size n will iterate past the actual point of convergence by at least n in order to calculate subsequent F-Scores. As expected, larger window sizes are more capable of detecting plateaus, but come with potential risk of overestimating the result (see query 3).

The slopes of query number 10 (see Figure 3) exhibits a kind of staircase behavior. The iteration at which convergence was declared depended highly on the chosen window size. When examined under a shorter window, such as of length 8, the numerous, short plateaus of query 10 led to the premature conclusion of convergence at iteration 22, while the use of a longer window size brought convergence to iteration 42. Note that the larger the difference in iteration numbers declared for convergence, the more the window choice matters; query 10 demonstrates exactly this. The larger the window size, the higher the regression line's slope, due to its factoring in the raw data's consistently small plateaus in F-score values, and finally, the later the convergence point.

Query number 15 converged at iteration 40 using a window size of 16. Further examination of the slope following iteration number 40 revealed that its value never again exceed the 0.001 threshold, and so the model can be deemed to have indeed converged by that iteration number. Note that with a window size of 8, the model would have been declared converged at iteration 18, which in retrospect is easy to identify as premature, since the slope does take on values exceeding the threshold following iteration number 20.

Queries number 11 and 21 both suffered from the fact that 16 was too small a window size to detect the spikes in slope that each query exhibited at later iterations following their supposed convergence point. While a trivial solution to this issue would be to simply increase the window size (to 32, for instance), this approach is not practical, as it could potentially require increasing the size to encompass all of the iterations of a query – hardly a solution to the problem of automatic convergence. Further, such a large window size sets too high a lower bound on the minimum number of iterations that must occur before convergence can be detected.

Table 1 shows the detected convergence iteration for NLTL. The NLTL algorithm showed a substantial improvement over buffered linear regression, achieving an error of 18.91%, which represents an improvement of 33%. The error percentages are graphed in Fig-

ure 2. Again, NLTL was likely to underestimate the point of convergence due to a combination of the 5% chosen threshold and the methodology chosen for manually determining the point of convergence, but the underestimation was significantly less than for buffered linear regression.

There were eight queries in which BLR16 had a better prediction than NLTL (queries 2,4,5,7,10,15,17,19). For queries 5,10, and 17 (the three queries with the largest difference between the two methods), the samples suggested by AIDE after just a few iterations became extremely similar, highlighting that NLTL depends on the underlying IDE system for sample selection. The fact that BLR16 outperformed NLTL in several queries suggests that an ensemble method might produce good results.

6. CONCLUSIONS

We have presented Nothing Left to Learn (NLTL), a new algorithm for detecting model convergence in big data exploration. Previously, users of interactive data exploration (IDE) tools were assumed to be knowledgable enough to tell when the IDE's model had converged, which is a questionable assumption given the complexity of models generated and the appeal of IDEs to non-expert users. NLTL allows for interactive data exploration tools to detect when they have arrived at the user's desired query or interests. The NLTL algorithm shows substantial improvements over previous techniques like buffered linear regression. Better estimation of how many iterations are needed to capture an analyst's interest means that analysts can process large datasets more efficiently, as fewer iterations will be wasted, and more effectively, as the analyst will be able to pose additional questions in the same amount of time.

While the problem of convergence detection will always be somewhat subjective (a doctor may require a more accurate model than a retailer, what was good enough last year may not be good enough next year), both techniques presented in this paper represent a significant improvement over pushing the problem of convergence detection fully on the user.

In the future, we would like to investigate ensemble methods that combine NLTL with buffered linear regression in order to provide a more accurate estimate of the convergence point. We will also experiment with different types of buffered models to detect when F-scores fluctuate wildly within a window but still have a relatively flat slope. Finally, we will test NLTL on other interactive data exploration tools as well as human-in-the-loop active learning tools unrelated to data exploration.

7. REFERENCES

[1] I. Alagiannis, S. Idreos, and A. Ailamaki. H2o: a hands-free adaptive store. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1103–1114. ACM, 2014.

[2] M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002.

[3] U. Cetintemel, M. Cherniack, J. DeBrabant, Y. Diao, K. Dimitriadou, A. Kalinin, O. Papaemmanouil, and S. B. Zdonik. Query steering for interactive data exploration. In *CIDR*, 2013.

[4] K. Dimitriadou, O. Papaemmanouil, and Y. Diao. Explore-by-example: An automatic query steering framework for interactive data exploration. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 517–528. ACM, 2014.

[5] M. Drosou and E. Pitoura. Ymaldb: exploring relational databases via result-driven recommendations. *The VLDB Journal?The International Journal on Very Large Data Bases*, 22(6):849–874, 2013.

[6] J. Fan, G. Li, and L. Zhou. Interactive sql query suggestion: Making databases user-friendly. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 351–362. IEEE, 2011.

[7] S. Idreos, I. Alagiannis, R. Johnson, and A. Ailamaki. Here are my data files. here are my queries. where are my results? In *Proceedings of 5th Biennial Conference on Innovative Data Systems Research*, number EPFL-CONF-161489, 2011.

[8] S. Idreos, S. Manegold, H. Kuno, and G. Graefe. Merging what's cracked, cracking what's merged: adaptive indexing in main-memory column-stores. *Proceedings of the VLDB Endowment*, 4(9):586–597, 2011.

[9] A. Kalinin, U. Cetintemel, and S. Zdonik. Interactive data exploration using semantic windows. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 505–516. ACM, 2014.

[10] N. Kamat, P. Jayachandran, K. Tunga, et al. Distributed and interactive cube exploration. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 472–483. IEEE, 2014.

[11] H. Ling and K. Okada. An efficient earth mover's distance algorithm for robust histogram comparison. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(5):840–853, 2007.

[12] F. Provost, D. Jensen, and T. Oates. Efficient progressive sampling. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 23–32. ACM, 1999.

[13] B. Qarabaqi and M. Riedewald. User-driven refinement of imprecise queries. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 916–927. IEEE, 2014.

[14] S. Rogers and M. Girolami. *A First Course in Machine Learning*. Chapman & Hall/CRC, 1st edition, 2011.

[15] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[16] S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308, 2000.

[17] A. Wasay, M. Athanassoulis, and S. Idreos. Queriosity: Automated data exploration. In *Big Data (BigData Congress), 2015 IEEE International Congress on*, pages 716–719. IEEE, 2015.

[18] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[19] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.