Incremental Learning in Real Time Data Processing

Knowles Atchison, Jr.
Spring 2012
Johns Hopkins University
Design and Analysis of Algorithms
Research Paper
May 7, 2012

Abstract

In today's oversaturated information environment, it has become increasingly necessary to process data and arrive at decisions immediately without the data being at rest or retrieved from secondary storage. In the brief moment new information is available, it must be assimilated and the worthiness ascertained and applied to the decision making capabilities of the software in question. This is a paradigm shift from traditional batch processing and learning algorithms. With continuous data flow, the challenge is how to transform the vast amount of stream raw data into information and knowledge representation and accumulate experience over time to support future decision-making [Haibo et al. 2011]. This paper will investigate the concept of learning in a real time data environment and compare and contrast contemporary solutions proposed for the problem domain.

1 Introduction

Incremental learning has two base scenarios: either the processing node has training and/or prior knowledge of the desired solution space or it starts tabula rasa. This slight distinction plays a large role in the choice of algorithm and ultimately the function of the software. Regardless, once the data begins to flow, the behavior begins to run together with the following basic outline:

Data is received

Data is added to the learning model/framework

Useable information is ascertained and assimilated into the model

Some action is taken based on the prior step

Repeat

Algorithmic differences can vary from underlying data structures to decision behavior. Regardless of the underlying framework, an incremental learning algorithm must meet the following requirements [Zhao et al. 2010]:

It should learn from new data

It should not require access to the original data used for training (if applicable)

It should retain previously acquired knowledge

1.1 The Case for Frameworks

The vast majority of published works on incremental learning focus less on the algorithm at hand and more on the framework in which said algorithm exists. This is especially true in real time processing situations where the flow of data, the ordering of steps, and data outflow are generally mutually exclusive from the handiwork of the processing element in and of itself. However, the learning portion is not necessarily plug and play and thus there is a strong correlation between the framework and the learning algorithm/processing that takes place within.

1.2 Contributions

The goal of this paper is to investigate and analyze solutions that attempt to make real time data processing systems 'smarter' via incremental learning. Commercially available platforms and frameworks are also discussed to shed light on current approaches to this problem area. In the following sections, the paper will discuss modern problems in which incremental learning is deployed and the effects therein. Lastly, some future areas of research are introduced.

2 Text Document Classification

A problem instance of interest is text document classification. Text mining is becoming increasingly important to industry as it provides a means to discover patterns and trends in natural language and analyzes that information for use in application processing [Zhihang et al. 2007]. It is in the middle ground of incremental learning problems since it can operate both with and without training data. This presents several viable options from which to attack the problem, most research examples use a variation on neural networks as their modeling basis.

2.1 Text Classification Training

One option is to use a set of training data to initialize the system and set it up for future processing. Applications may have a large amount of such data available and new training often comes available over time [Zhihang et al. 2007]. The Incremental Learning of Text Classification (ILTC) framework proposed to address this problem functions in a slightly different way then traditional online learning. In a normal online learning situation, new data allows the system to adjust its hypothesis and act accordingly. ILTC operates by learning from the new data without reexaming the old training data, but retaining previously acquired knowledge.

The below figure outlines the framework:

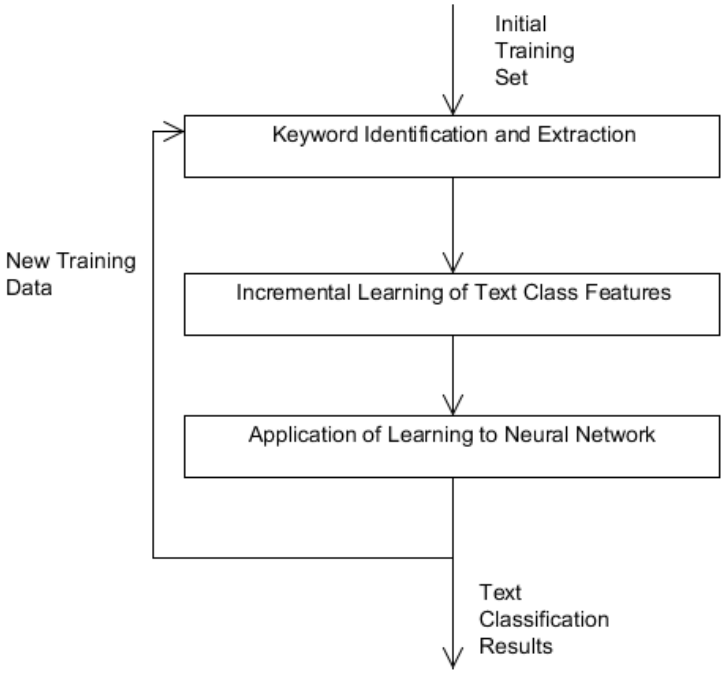

Figure 1: Framework of Incremental Learning of Text Classification [Zhihang et al. 2007]

The initial training set allows the system to begin operation under certain known conditions which are then adapted as new information becomes available.

Algorithm 1 Neural Network

Initialization

1. A list of initial terms and phrases are given as the first training set.

$$Tr = \{t_0 ... t_{m_0}\}$$, where m_0 is the total number of terms

2. Each term has an associated local and global weight.
3. The weights and terms form a feature vector that is used for classification.

Incremental Learning

1. Data [new documents] comes in and the weights of the current feature vector are adjusted.

$$document\ d = \bar{q} = \{q_0 ... q_{m_0}\}$$

$$q_i = frequency\ of\ i^{th}\ term$$

2. The old data is not reevaluated at any point in time; the current assessment is adjusted only.

3. The output to the neural network is a matrix $M \in R^{m_0 x k_0}$ where k_0 is the total number of text classes and m_0 is the total number of terms. Columns are feature vectors for a given text class and rows are feature vectors of the r^{th} term within the term list.

Application to Neural Network

1. Each node in the network represents a text class.
2. When new information is available the network must add nodes to account for this information.
3. Output may modify feature vectors.

$\bar{x} = \{x_0 ... x_{m_0}\}, x_i = q_i * g_i,$

Where g_i is the global weight of the i^{th} term and q_i is the frequency

4. Classification of the document is the main output. Secondary output is a new set of training information that is incorporated into the incremental learning in the same way the initial training set was assimilated.

Of particular importance is step number two in the initialization portion of the above algorithm. A word may have significant meaning within a phrase, even though it appears less frequently then other terms in that phrase. In the example of technical manuals, a verb will carry more contextual meaning even though it is found fewer times then other words, e.g. check memory leak, check and advise [Zhihang et al. 2007].

The incremental learning portion of the algorithm can handle nearly all cases where the information coming in is not "new", but rather a permutation on already known information. The neural network comes into play when previously unknown data is given as input and instances where data is not in the set of known items, but falls within similar text classes/features. Below is a diagram of the aforementioned neural network:

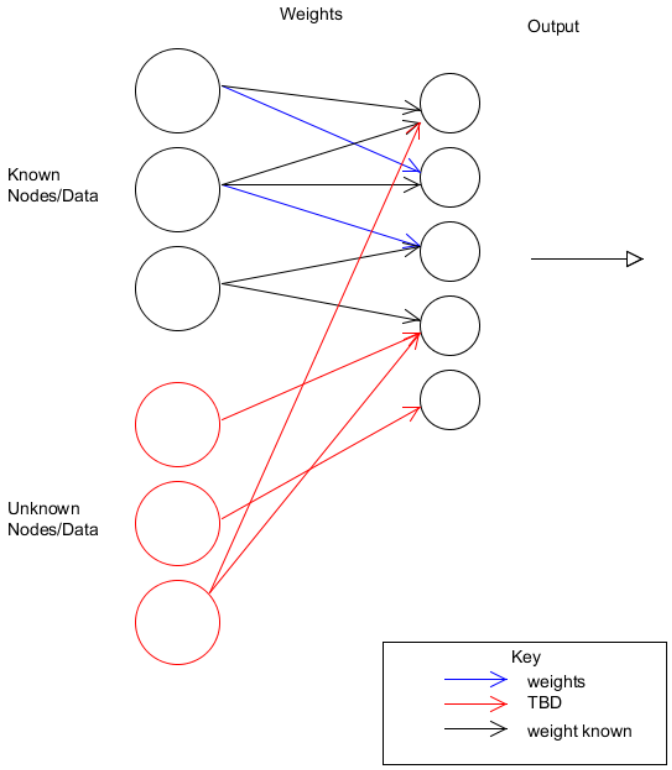

Figure 2: Incremental Neural Network Model [Zhihang et al. 2007]

The two major components of this approach are the incremental learning of text features (key terms and phrases) and the incremental neural network model. ILTC illustrates the ability to train a new system, acquire new knowledge, and assimilate it all without the need to neither refer to the original training data set nor forget information seen previously.

2.1.1 Training Result Analysis

The data sample of the experiment is a collection of diagnostic automobile documents that contain phrases from customers such as "there was a noise heard in the engine" and "engine runs rough" [Zhihang et al. 2007]. As a precursor to the computational results of the analysis, when we read the prior two sentences, our brain picks up on the relationship of the word engine to the rest of the sentence. The significance is present in our minds via sentence structure, word quantity, and common experience. The most important of which is the common experience, as we know that the engine is a crucial piece of the car and thus more weight is given to it when discussing things wrong with our automobiles. On the other hand, sentences and descriptions are often grammatically incorrect and may be entirely inaccurate as to the root cause of the mechanical problem (but that is outside the domain scope of this algorithm).

Starting with data set one, D1, training classifier F_1 was generated. Then for D2 through D4, F_{2-4} were generated. The nine categories of text documents were generated/classified via a process called 'Learning From Text' [Huang et al. 2006]. It is as follows:

Algorithm 2 Learning From Text

Initialization

1. Assume classification categories for $C_1, C_2, ..., C_n$ with training documents $T_1, T_2, ..., T_n$, where T_i contains the documents belonging to category $i, i = 1, ..., n$
2. Determine frequency threshold

Document Indexing

1. Generate a term list from terms in $T_r, T_r = T_1 \cup T_2 \cup T_n$
2. Filter terms
    a. Remove stopping words that will not be useful in classification (to, from, etc…)

b. Merge morphological variants of the a word into the same root (leak, leaks, leaking).
c. Track word frequency and note terms that exceed or are under the desired threshold.

Matrix Generation

Generate Term-Category Weight (TCW) Matrix [Huang et al. 2006]

TCW = M x N matrix where,

M = number of terms in the term index

N = number of diagnostic categories

$TCW_{ij} = tf_{ij} \bullet G_i$, multiplication of local and global weights,

tf_{ij} is the frequency of the ith terms in the term index occurring in T_{rj} for diagnostic category j

$$G_i = \log_2 \left[\frac{ndocs}{df_i} \right] + 1$$

ndocs, the number of documents in the entire training set

df_i, total number of documents in T_r that contain i

Each subsequent data group contains the same nine classes (remember that these are the categories that data can fall in and are represented by nodes in the neural network), but each data set has new terms for the algorithm to learn from. While the methodology for generating $C_1, C_2, ..., C_n$ is provided, the author never directly defines the categories or data sets.

The data sets are arranged as follows:

	D1	D2	D3	D4
Number of Samples	600	600	600	600
Number of new terms	346	123	59	139

Table 1: Data used in incremental experiments [Zhihang et al. 2007]

2.1.1.2 Initial Data Set

D1 is compared to Perceptron, which is a supervised learning algorithm that is a linear classifier, meaning it is a classification algorithm that makes predictions based on a linear predictor function combining a set of weights with the feature vector describing a given input. In the context of this problem, it is similar in the fact that it uses the weights of words within a phrase (the feature vector) to determine how to classify the text. Perceptron:

$$f(x) = \begin{cases} 1 & if\ w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

where:

w is a vector of real-valued weights

$w \cdot x$ is the dot product (computes a weighted sum)

b is possible bias

It is worth noting that the authors do not expound on any decision making process leading up to the choice of the Perceptron as the comparison tool to the initial training/classification of data.

The results are as follows:

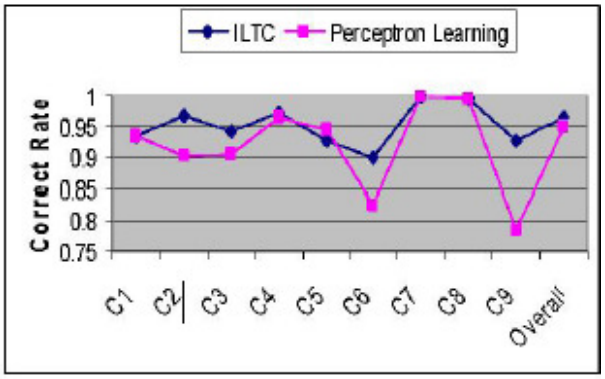

Figure 3: Performance of ILTC on data set DI with comparison to the standard Perceptron learning. [Zhihang et al. 2007]

While it appears that this method is preferable, it is unfortunate that the classes C1 through C9 are not explicitly defined in the experiment and thus the revelation that ITLC outperformed in all classes except for C5 raises more questions then it answers. For example, what in the data in C5 allows a simple linear classifier to have more correct output then a new framework specifically designed for such a purpose? Can data be hand crafted to give better initial results for certain algorithms? How was correctness measured? Regardless, the takeaway from part one of the experiment is that a proper choosing of framework and algorithm will allow for a better initial result which lays the groundwork for the remainder of the procedure.

2.1.1.3 Incremental Learning Results

"Three iterations of incremental learning were conducted on data sets, D2, D3 and D4. During each iteration, the feature space is expanded, which is implemented subsequently in the expansion of the input nodes in the neural network system: 123 new input nodes were added during the incremental learning from D2, 59 new input nodes were added during the incremental learning from D3, 139 new terms from D4" [Zhihang et al. 2007].

In this portion of the experiment, the competing algorithm is a stochastic weight update scheme, which a neural network that also implements incremental learning, but does not retain previously acquired knowledge.

The results are as follows:

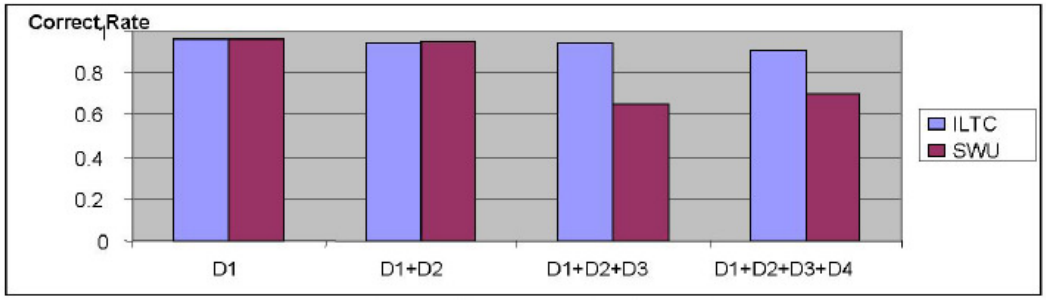

Figure 4: Performance Comparison of ILTC and SWU [Zhihang et al. 2007]

Again the ITLC framework outperforms its counterpart; however there are several issues with the above portion of the experiment. First and foremost, SWU does not retain prior information. In this manner, comparing the two is akin to comparing apples and oranges. ITLC will be more correct, but SWU may outperform, computationally speaking, since it is doing "less work". ITLC strikes the balance between new and old information learnt, which is arguably one of the most important characteristics of an incremental learning system doing real time data processing.

2.2 Text Classification sans Training

On the other end of this spectrum is the setup that does not require an initial training set. This situation reflects a real world scenario where the initial training data may be cost prohibitive or otherwise somehow unobtainable. One solution for the lack of labeled data required is to develop algorithms that can learn from a small number of

labeled examples augmented with a large number of unlabeled examples [Liu et al. 2011].

Since there is a lack of pre-labeled data, the key aspect to bootstrap into a useful system is to employ the use of clustering, specifically a highly modified version called the fuzzy partition clustering method (FPCM) [Liu et al. 2011]. FPCM divides the dataset via basic features of the initial sample data (i.e. simple feature vectors of the document):

$$\text{Given a document set: } X = \{x_1, x_2, ...x_n\}$$
$$\text{Sample } x_k \text{is afeature vector } p(x_k) = (x_{k1}, x_{k2}, ...x_{ks}), x_{kj}(1 \le j \le s)$$

Samples are divided into fuzzy subsets via some comparability measurement (this is a notable comparison to the Fuzzy C-Means method of achieving similar results.) A sample illustration:

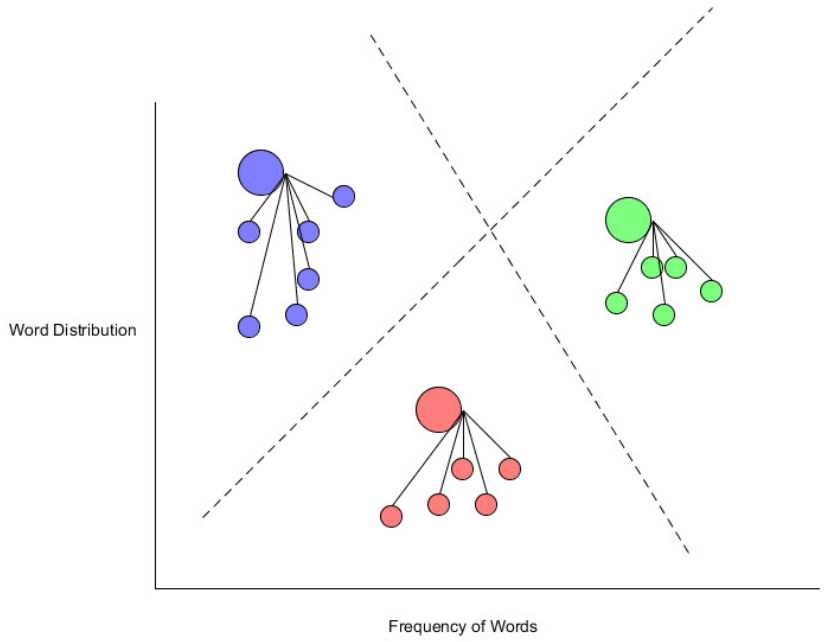

Figure 5: Visualization of Fuzzy Clustering

Based on the criteria of the frequency and distribution of certain words within a sample data file, three clusters of "similar" documents have been partitioned without the need for initial training data. The sample data in fact becomes, in essence, training data. The fuzzy clustering algorithm is substituted for pre-labeled, known training data.

With the training aspect now complete, the next key part of the system is the underlying model in which knowledge is stored. In this instance a naïve Bayesian model

is used as the framework for incremental learning. In contrast to the neural network presented in section 2.1, the naive Bayesian model is an applied principle of the keep it simple mantra. Rather then build up an overly complicated system with a set of predefined data, a simple classifier allows for [0,1] decision process similar to Perceptron. This moves the decision making capabilities from a relationship oriented structure to one of mere probability.

Algorithm 3 Bayesian Classifier

Initial Sample Data Clustering

1. Select representative points $v_1, v_2, ...v_c$ as initial centroids based on some criteria (the larger circles in Figure 3).
2. Determine fuzzy membership based on the distance from each element to the centroid.
3. Select samples close to the centroids as now labeled samples/training sets

$U_A : U_A = \{x_k \mid h_k < \omega\}, \omega \text{is the threshold}$; remainder of elements are in set U_B

Build Bayesian Classifier

1. Train with U_A
2. Classify unlabeled documents from set U_B

$s \leftarrow 1, s \text{ is size of } U_B$

$x \leftarrow x_s, x_s \in U_B$

$\min\{\sum_{j=1}^{c} L_{ij}(x) * P(c_j \mid x)\}$

$x \text{ belongs to class } c_i, \text{but was classified in } c_j. P \text{ is the probability of } x \text{ being clustered into } c_i \text{ and } L_{ij} \text{ is the loss of } x \text{ if moved to cluster } j$

Runtime Execution

Classifier is determined by the posterior probability. It will attempt to determine the mutual information relationship between known and new data. Evaluates the feature vector (incoming data) with the classifier set (known data) [Liu et al. 2011]:

$X = \{x_1, x_2, ...x_m\}, C = \{c_1, c_2, ...c_n\}$

$foreach\, x_j \, in\, X$

$s[j] = QMEI(c_j, x_j)$

Rank $s[]$, export to classifier

$p(x_i) = \sum_{i=1}^{n} Gauss(x_i - x_j, \sigma I)$ //probability density

//mutual information relationship

$$QMEI(c_j, x_j) = \log \frac{(\sum_{i=1}^{n} \int_x p(c_i, x_j)^2 dx_j)(\sum_{i=1}^{n} \int_c p(c_i)^2 p(x_j)^2 dx_j)}{(\sum_{i=1}^{n} \int_x p(c_i, x_j) p(c_i) p(x_j) dx_j)^2}$$

More responsibility is placed on the initial labeling of sample data, as it drives the remainder of the algorithm during execution. If the "seed" is not done correctly, the rest is ultimately irrelevant.

2.2.1 Results Analysis

The data set for this experiment was a collection of 4,800 web pages, pre categorized into six categories of 800 documents each. The Bayesian classifier is combined with FPCM and two other clustering methods for comparison. The results can be seen in the below figure:

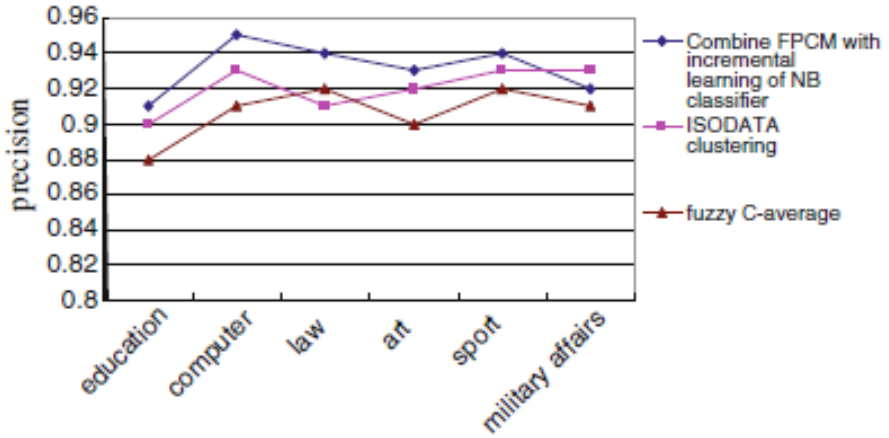

Figure 6: The proposed comprehensive classifier is compared with ISODATA, clustering, and fuzzy C-average [Liu et al. 2011]

We can immediately see that the data sets used in the experiment are well defined and can easily be recreated. Additionally, the comparison of the proposed classifier is done combined with other external clustering methodologies. As the above diagram illustrates, the new classifier clearly has an advantage in certain text categories. Interestingly enough, the category in which it is outperformed is military affairs, the jargon of which is often populated with acronyms and short hand that often requires prior knowledge to comprehend correctly. In this context, the lack of a predefined training set most likely worked against FPCM.

3 Stream Computing

Most tools in place for "big data" today exist as batch processing systems [Satzger et al. 2011]. MapReduce, Hadoop, et al. operate on the data as a final set for the given run since the problem is divided out as sub-problems. While highly efficient, this fails to solve similar problems where data is constantly arriving over time. The fundamental difference in the two approaches can be illustrated with the following diagram:

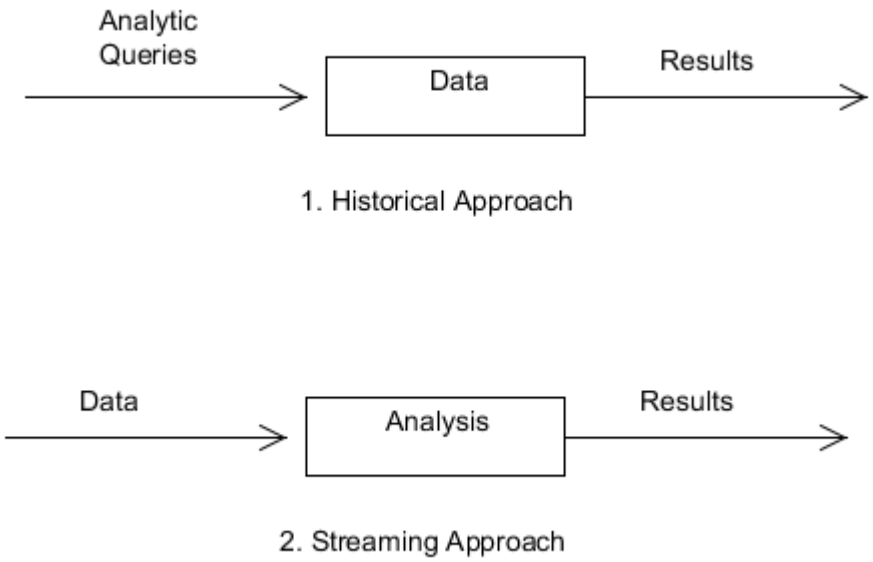

Figure 7: Batch Processing vs. Stream Computing

It comes down to data at rest or data in motion. How does a system not only receive data, but provide meaningful analysis before said data is gone? Two such solutions to this issue are the Elastic Stream Computing framework and IBM's InfoSphere.

3.1 Correlation to Incremental Learning

While not all stream computing frameworks implement learning algorithms, the concept of the stream computing framework is closely tied with the concept of machine learning in general. When data is arriving over a period of time, eliminating the ability to

process a single batch as one unit, stream computing combines the flexibility and power of cloud computing with the opportunity to implement incremental learning solutions within the framework.

3.2 Elastic Stream Computing

Written in Erlang, Elastic Stream Computing (ESC) is a stream computing engine designed for real time data computation [Satzger et al. 2011]. It is similar to MapReduce in the fact that it uses key/value pairs, but diverges with the design for distributed online processing of event streams with unknown/varying rates of data.

The overall data structure of ESC is that of a directed acyclic graph (DAG). A DAG defines the data flow through the system and vertices represent operations to be done on the data at that given point in time. The ideal environment to set up ESC is the cloud. This allows the elasticity in the name since the platform itself has zero control over data rates and should scale up and down when appropriate. The diagram below outlines the overall architecture of ESC. Note the similarities between the diagram below and the neural network put in place to solve the text classification problem in section 2.1.

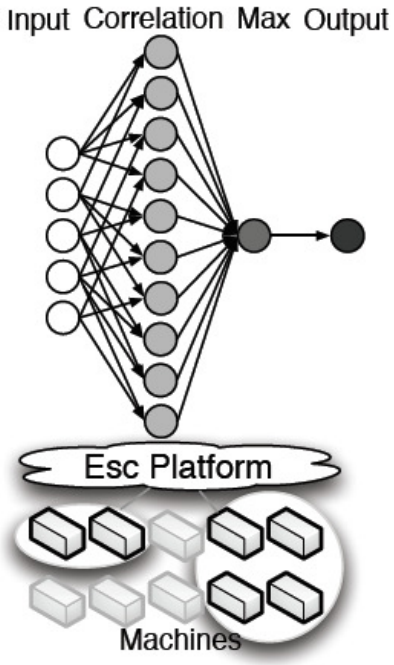

Figure 8: Esc Platform Overview [Satzger et al. 2011]

While the description of programmatic options at vertices is brief at best, the authors [Satzger et al. 2011], do leave the door open for interpretation. In these critical points within the framework, incremental learning algorithms would be of excellent use given the DAG nature of ESC.

3.3 InfoSphere

Every day, consumers and businesses generate data at a pace that would have seemed unprecedented just a few years ago. Each hour, retail mammoth Wal-Mart handles more than one million customer transactions, feeding databases estimated at over 2.5 petabytes in size [IBM 2011]. InfoSphere is a framework to provide real time analytics on data that is in motion. Akin to ESC, it places an additional abstraction atop the streaming framework by allowing programmers to write in their Streams Processing Language (SPL).

Aggregation of data sources coupled with the flexibility and power of functional programming paradigms is a welcoming environment for machine learning, incremental learning in particular.

3.4 Practical Usage

In the two aforementioned platforms, the usage of the classification algorithms from section 2 would not necessarily be universally plug and play. For example, in IBM's InfoSphere, we could have a process that does not need to be initially trained, while creating and retaining information seen previously. In this instance, the Bayesian classifier would a better fit. Additionally, external constraints weight heavily on design choices such as the ability to create and maintain a set of initial training data of which to feed a system.

A framework's architecture may also lend itself to one particular algorithm. In the instance of ESC, the network topology of the processing nodes is, in and of itself, a neural network. It would follow that the use of an algorithm that closely mimics the

container's format would be of great use and possibly easier to implement, considering how often economic factors weigh in on the decision making process.

4 Future Research

The few scenarios we have touched upon in this paper have not even scratched the surface of what is out there, both in academia and the workplace. As readily evidenced in the discussion of text classification of section 2, the permutations for a given problem domain are numerous: training data, underlying data structure, so on and so forth. The concept of incremental learning has been applied in situations of traffic flow to ecommerce.

As cloud computing continues to take off, we expect incremental learning solutions within cloud/streaming frameworks to rapidly increase, as companies bring new solutions into the marketplace. While the ground has been thoroughly treaded, the depth is something to be desired and a possible avenue of exploration going forward.

5 Conclusions

In this paper, we discussed the concept of incremental learning, laid out the basic qualifications of an incremental learning algorithm, examined the results of each algorithm's experiments, and explored text classification as a problem area in which incremental learning could be applied. We saw how the existence or lack of training data can affect the underlying data structures for a given solution. Current frameworks and commercially available solutions were also presented, highlighting some of the ways incremental learning can be implemented in the real world.

7 References

Published Books and Papers

HAIBO HE, SHENG CHEN, KANG LI AND XIN XU. 2011. Incremental Learning From Stream Data. *Neural Networks, IEEE Transactions on* 22, 1901-1914.

HUANG, L. AND MURPHEY, Y. 2006. Text Mining with Application to Engineering Diagnostics. *Advances in Applied Artificial Intelligence* 4031, 1309-1317.

LIU, L. AND LIANG, Q. 2011. A high-performing comprehensive learning algorithm for text classification without pre-labeled training set. *Knowledge and Information Systems* 29, 727-738.

SATZGER, B., HUMMER, W., LEITNER, P. AND DUSTDAR, S. 2011. Esc: Towards an Elastic Stream Computing Platform for the Cloud. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on,* Anonymous, 348-355.

ZHAO, Q., JIANG, Y. AND XU, M. 2010. Incremental Learning by Heterogeneous Bagging Ensemble. In *Advanced Data Mining and Applications,* L. CAO, J. ZHONG AND Y. FENG, Eds. Springer Berlin / Heidelberg, 1-12.

ZHIHANG CHEN, LIPING HUANG AND MURPHEY, Y.L. 2007. Incremental Learning for Text Document Classification. In *Neural Networks, 2007. IJCNN 2007. International Joint Conference on,* Anonymous, 2592-2597.

WWW References

IBM InfoSphere Product Page. http://www-01.ibm.com/software/data/infosphere/