Honours Dissertation: Semi-Supervised Learning on Data Streams

Michael Glenny

October 28, 2014

0.1 Introduction

0.2 Related Work

0.3 Label Stream: An Algorithm for Classifying Stream Data

The algorithm that I chose to attempt to improve was one developed by researchers at the University of Dallas. The Label Stream algorithm was published in 2009 by Clay Woolam, Mohammad M. Masud and Latifur Khan.[1]

The basic framework of the algorithm is to use an ensemble of models to predict which class a data point belongs to. The stream is broken into chunks, and each model is built from the data in a partially labelled chunk. When a new model is created, we simply need to update our ensemble by evicting the model which is performing worst on the new labelled data, as well as refining the current models in the ensemble based on the labelled data seen in the new chunk.

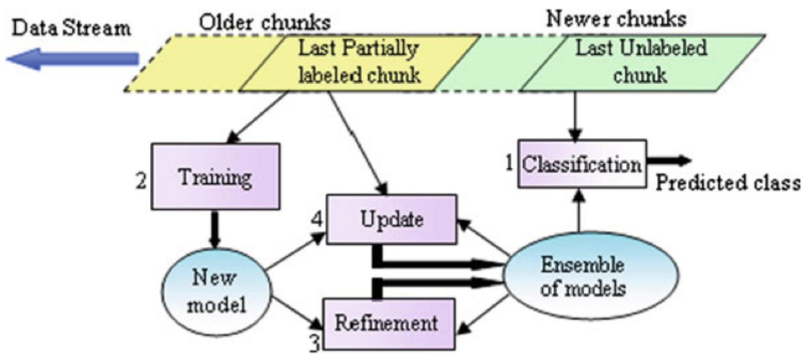


Fig. 1 Overview of ReaSC

[3]

0.4 Training a Model

We can train a model from a data chunk as soon as some arbitrary percentage of the data points in that chunk have been manually labelled. Experimental testing by Woolam et al. shows that Label Stream performs as well as many previous stream classification algorithms with as few as 10% labelled points.[1]

Training a model based on a chunk of data consists of three steps. Firstly, we cluster the data into large macro-clusters based on a modified version of K-Means clustering, in which we try not only to minimise intra-cluster distance like in standard K-means, but also minimise cluster impurity by taking into account the entropy of clusters as a dissimilarity measure in which clusters with points of different labels are punished.

Secondly, we then split those macro-clusters into class label pure micro-clusters. Some of these micro-clusters will be unlabelled, and hence the third step is to use a Label Propagation technique to label the unlabelled micro-clusters. Now that all the micro-clusters are labelled, they are ready to be used in the classification process.

0.4.1 The Macro-Clustering Step

MCI K-Means

In regular K-means clusters, we wish to minimise the intra-cluster distance in every cluster, which can be expressed as minimising an objective function:

$$O_{Dist} = \sum_{i=1}^{K} \sum_{x \in X_i} ||x - \mu_i||^2$$

This is an unsupervised technique, but we would like to take advantage of the fact that a certain percentage of our data chunk have labels associated. To do this, we will modify unsupervised K-means by adding an additional factor to the objective function. This is called by Woolam et al. MCIK-means, which stands for Minimising Cluster Impurity for K-means. [1] The objective function for this variation on K-means is:

$$O_{MCIKMeans} = \sum_{i=1}^{K} \sum_{x \in X_i} ||x - \mu_i||^2 (1 + Ent_i ADC_i)$$

Firstly, we need to define entropy. Entropy is a measure which is normally used to describe how many bits are necessary to convey information, but is useful in this context as a measure of purity of clusters. The formula for calculating the entropy of a cluster i is:

$$Ent_i = \sum_{c=0}^{C}(-p_i^c log(p_i^c))$$

where p_i^c is the prior probability of a point in the cluster i being of class c, i.e.:

$$p_i^c = \frac{|X_i(c)|}{|X_i|}$$

Entropy will give a higher penalty to impure clusters, however using only entropy will mean that every point has the same penalty related to impurity. This is a problem when we try and minimise our cluster impurity. Suppose for instance we have cluster A, with 9 points of label 1, and cluster B, with 3 points of label 1. Then, suppose we add 7 points of label 2.

The best arrangement we can hope for is for cluster A to remained untouched, and cluster B to take all the points of label 2, resulting in an entropy of 0.26. However, what will happen when we incrementally add each point? Well, in the first instance, because cluster A is larger, a point of label 2 will be assigned to it. Now because cluster A is impure, we will continue adding points to cluster A, and our final configuration will be cluster A with 16 points in total, 9 of class 1 and 7 of cluster 2, and an entropy of 0.29, and cluster B as it was with an entropy of 0. This is not the optimal configuration!

To solve this problem, we introduce a measure of cluster impurity called Dissimilarity Count. We define the dissimilarity count of a point x in a cluster i to be:

$$DC(x, i) = \sum_{c=0}^{C}(|X_i| - |X_i c|)$$

Basically, the dissimilarity count of a single point is the number of points in the cluster with a class label other than the class of that point. In our previous example, using Dissimilarity Count, we would place all the points of label 2 into cluster B, because the dissimilarity count for all points of label 2 would stay the constant 3, for all the points of label 1 which were there before.

The Aggregated Dissimilarity Count, ADC, is simply the sum of all dissimilarity counts for each point in a cluster.

/* A graph demonstrating the above examples would help */

An outstanding issue is whether to consider unlabelled points as a distinct class or not. In the original Label Stream paper, it is clear that unlabelled points are not to be considered in the Dissimilarity count, as they explicitly state that if a point is unlabelled, $DC(x, i) = 0$.[1] However, in Khan et al. review of the same algorithm, it is more equivocal, and the paper seems to suggest that they did consider unlabelled points as a distinct class for the purposes of calculating the dissimilarity count. For my experiments, I have decided to uphold the convention of the dissimilarity count of an unlabelled point being zero.

Expectation Minimisation

In order to generate these clusters, firstly we need to initialise our clusters. This initialisation can be have a great effect on the quality of the subsequent clustering. Firstly, we take all the labelled points in our data chunk and calculate the prior probability of a certain class appearing in that chunk. Then, for each class, we calculate the proportion of the K clusters to be generated which should be initialised to a point of that class. This calculation is:

Number of clusters initialised to class $i = p_i^c K$

where K is the number of macro-clusters. Now, we use the furthest apart initialisation method to select the points which will be the centroids of our macro-clusters. That is, we find the point of class i which is furthest away from all the points in i which have already been selected as centroids for clusters. Masud et al. report that this method of initialisation, proportionate to the distribution of labelled data in the data chunk greatly improved their cluster quality.[3]

With the clusters initialised, we can now apply Expectation Minimisation to find a clustering. Firstly, we apply the E-step, in which we assign a point to the cluster which minimises the contribution to the global function, regardless of whether that point is already assigned to a cluster or not:

$$O_{MCIKMeans}(x) = ||x - \mu_i||^2(1 + Ent_i DC_i(x))$$

The value of the overall objective function depends on the order by which we add points from the chunk to the clusters. If we were to attempt to try all possible orderings of points, this step would be computationally intractable. Instead, we can randomly shuffle the points. This will not guarantee that the cluster converges to the global minimum for the objective function for MCI K-means, but the algorithm will converge in relatively few iterations.

The Minimisation step is to simply recalculate the centroid of each cluster, μ_i, by averaging all the points now in that cluster. The calculation for that step is:

$$\mu_i = \frac{\sum_{x \in X_i} x}{|X_i|}$$

We then repeat these steps until the convergence condition is met. We consider the algorithm to have converged if no single point has changed cluster after the point re-assignment phase. This idea of shuffling points randomly and this convergence condition is called the Iterative Control Method, which is guaranteed to converge.[4]

0.4.2 The Micro-Clustering Step

Once the macro-clustering step is complete, we will have K macro-clusters which may or may not be class pure. Our next step is to break these macro-clusters into micro-clusters which are class pure. The procedure is rather simple based on what is considered 'pure':

1. Firstly, if a macro-cluster only contains labelled points from a single class, then it is already pure, and we can consider that macro-cluster as a micro-cluster.
2. Secondly, if a macro-cluster contains labelled points from two different classes, then we split that macro-cluster into two micro-clusters where each micro-cluster will only contain points from one class.
3. Thirdly, if a macro-cluster contains labelled points from one class as well as unlabelled points, we will split these points into two micro-clusters, one with the labelled points, and with unlabelled points.
4. Fourthly, if we have a micro-cluster which only contains unlabelled points, we will further subdivide that micro-cluster into micro-clusters based on the predicted class labels of those unlabelled points.

The reason we split these macro-clusters into micro-clusters is so that we can make definitive statements about the labels of the labelled micro-clusters. If this is the goal of the splitting, we might ask why we split unlabelled micro-clusters based on their predicted classes. If an unlabelled micro-cluster is constituted of two underlying classes, which have some geometric division, hopefully our predictions from previous models will be able to detect this. If we can successfully split this micro-cluster, the quality of our label propagation will increase greatly. An example is given in this figure taken from Woolam et al.[1]

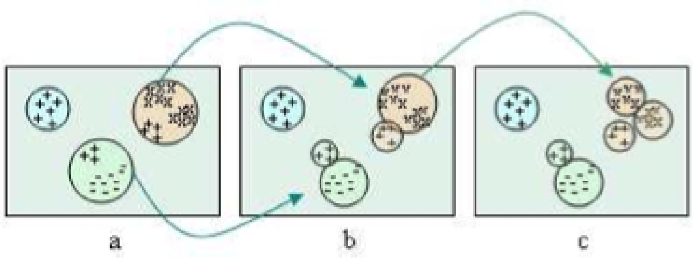


Fig. 1. Illustrating micro-cluster creation. '+' and '-' represent labeled data points and 'x' represents unlabeled data points. (a) Macro-clusters created using the constraint-based clustering. (b) Macro-clusters are split into micro-clusters. (c) Unlabeled micro-clusters are further split based on the predicted labels of the data points.

Once we have finally decomposed our macro-clusters into micro-clusters, we won't store the complete micro-cluster, because we have no use for the complete listing of points in the micro-cluster. In the Label Stream algorithm, three basic pieces of information are stored about the micro-cluster. However, at this point, we will calculate a measure of cluster quality based on the LoOP measure, explained later:

1. The centroid of the micro-cluster, μ
2. The weight of the micro-cluster, which is simply the number of points it contains, ρ
3. The label of the micro-cluster. If the micro-cluster contains only unlabelled points,
4. In addition, we will store the LoOP score of the cluster.

This summary of the micro-cluster is called a pseudo-point. However, at the moment, all the micro-clusters which had no labelled points in them have no label! Hence, we move on to the third phase of our label propagation.

0.4.3 Label Propagation

In order to label our unlabelled micro-clusters, we will use a label propagation technique originally devised by Zhou et al.[5]. Their basic technique is to build a complete graph where each micro-cluster is and each edge is a measure of weight or influence that one micro-cluster has on another. The influence in that technique that i exhibits on j is:

$$W_{i,j} = e^{-(\frac{||x_i - x_j||}{2\sigma^2})}$$

0.5 Using LoOP to evaluate cluster quality

0.6 Experiments

Experiments were conducted on a variety of datasets. Since the aim of the alterations to the LabelStream algorithm was to increase the classification accuracy of the algorithm, no experiment tracked the relative point classification rate of the algorithm with and without the LoOP density score. Subsequently, the hardware the tests were performed on is not particularly relevant, however for completeness they were performed on a MacBook Pro with a 2.5GHz processor and 4GB of memory.

Since the LabelStream algorithm is stochastic, for each of these results, the test was run twenty times each, and the cumulative accuracy results for each averaged.

0.6.1 SynD Dataset

The Dataset

The primary purpose of the SynD dataset is to examine how each algorithm deals with conceptual drift in the data stream. SynD is a synthetic dataset generated in MOA which is based on a shifting hyperplane. The equation of the hyperplane is:

$$\sum_{i=1}^{d} a_i x_i = a_0$$

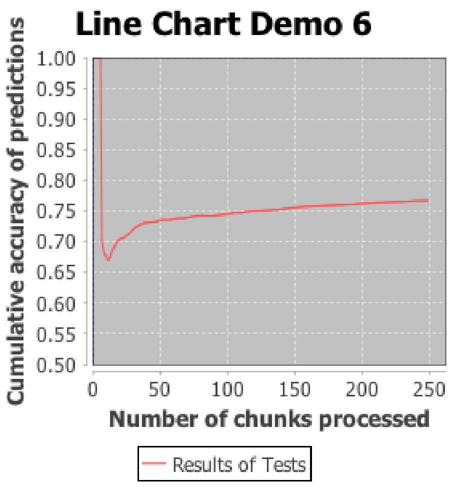
where d stands for the number of dimensions, a_i is the weight associated with dimension i, and x_i is the value of the ith dimension of data point x. Each point can be classified by considering whether $\sum_{i=1}^{d} a_i x_i <= a_0$; if it is, then the point is placed in class 1, otherwise it is placed in class 2. Each point is generated vector of length $d\{x_1, ..., x_d\}$ where $x_i \in [0, 1]$. The weight vector is also a vector of length $d\{a_1, ..., a_d\}$ where each weight is in the range of $[0, 1]$. The value of a_0 is set to generate roughly the same number of points from each class. We can do this easily if we set a_0 as the average of the weights, $\frac{1}{2} \sum_{i=1}^{d} a_i$. Also, we swap the class label on some points to generate noise in the dataset.

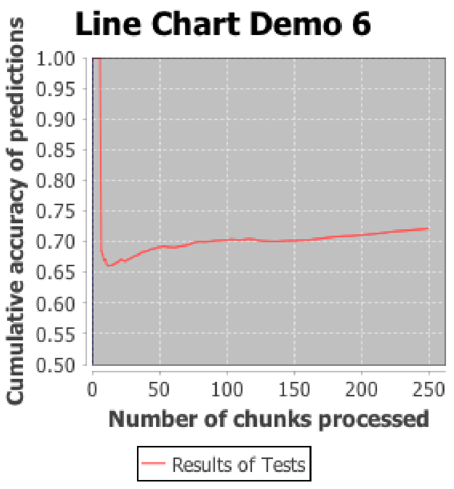
We can introduce concept drift by choosing a subset of the weights and gradually varying them over a certain number of data points in the stream. We simply define a magnitude of drift t, a vector of drift direction $\{s_1, ..., s_d\}$, and over N data points a certain subset of the weights so that a_x changes by $s_x t/N$. Finally, after N data points have been generated, we can generate a new subset of weights, a new magnitude of change, and randomly change the direction of change of weight for some weights.

For our experiments, we followed the experimental parameters chosen by the developers of the LabelStream algorithm for comparative reasons. The parameters are:

Instances = 250,000
$d = 20$
Number of drifting attributes = $0.2d$
Magnitude of drift $\in [0.1, 1]$
Chance of drift direction change = 10%
Number of points before drift recalculation = 1000
Noisy points = 5%

Results





As we can see from this clearly labelled graph, using the cluster weighting in the label prediction slightly improved the cumulative accuracy of the classifier. One of the observed properties of this dataset was that the classification accuracy was increased by excluding originally unlabelled microclusters from the ensemble point classification; in short, the label propagation onto unlabelled microclusters degraded the performance of the algorithm.

Incidentally, the clusters which had the largest outlier factor were the large, originally unlabelled microclusters. These would often have an LoOP score of 0.7, rather than the very small labelled microclusters, which would have a score approaching one. These microclusters tended to be small, due to the paucity of labelled data in the training chunk, and hence tended to have lower outlier scores. Consequently, the algorithm which included the LoOP scores emphasised these small labelled microclusters in the labelling process and marginalised the larger, unlabelled microclusters, resulting in a higher classification accuracy.

0.6.2 KDDCup Network Intrusion Dataset

The Dataset

The KDDCup Network Intrusion dataset is an extremely common test dataset for Machine Learning problems. Each data point contains 41 categorical or real value attributes relating to the status of a network connection, and then a classification value which states whether the network connection was an attack, and if so, what kind of attack it was. The classification problem is hence to identify whether the connection is normal or whether it is an attack of a certain variety. However, for my implementation, all categorical data except the class of network attack was removed for purposes of comparison with the Label Stream algorithm. The actual set used was the fully labelled ten percent subset freely available from most data repositories [1] which has approximately 500,000 data points.

It is worth noting that this data was not originally a stream, it was simply interpreted as a stream. This means that extremely sudden concept drift can occur in the data which results in large changes to the ensemble. It is also worth noting that although there are 24 classes in the dataset, 98% of the data points fall into one of three classes: Normal, Neptune attack, or Smurf attack. This means the success of the classification depends largely on predicting these three classes.

Results

0.6.3 Forest Cover Dataset

The Dataset

The Forest Cover Dataset is a dataset which attempt to classify what type of forest cover is in a particular area of forest based on certain attributes. Each data point has 54 attributes, however most of these are binary variables relating to particular characteristics of soil type, so if you count them as one attribute, there are only 10 attributes. All of these attributes are integers or real values. There are then 7 types of forest cover into which the data point can be classified.

The dataset is large, containing 500,00 data points. These are predominantly made up of two classes: Spruce-Fir with 210,000 records and Lodgepole Pine with 280,000 records. The remaining classes number between 2,500 and 40,000 records. It is also worth noting this dataset is not a stream, it is simply interpreted as a stream.

[1]`www.kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`

Results

0.6.4 Power Usage Dataset

The Dataset

The Power Usage dataset is a very simple data stream. There are only two attributes for each data point; the amount of power on the main grid and the amount of power transformed from other grids. The class to be predicted is the hour of the day which these power measurements came from. This means there are 24 classes. The interesting property of this dataset is that many of the classes are extremely similar. In fact, when this data stream was used for experimentation by Zhang et al.[2] they transferred the data set into a binary classification problem, possibly to avoid this problem.

The other interesting aspect of this dataset is that there is natural concept drift, caused by factors like the day of the week the data is measured on and the season. The downside to this dataset there are only 30,000 records, which is very small. Simply reducing the size of the chunks will leads to less labelled data in to train each model on, so for this dataset we increased the proportion of data in each chunk which is labelled.

0.6.5 Sensor Stream Dataset

The Dataset

The sensor stream dataset is a real world data stream generated by the Intel Berkeley Research Lab. They planted sensors throughout their offices which every few minutes took measurements of temperature, humidity, light and voltage. The classification problem is to decide which of the 54 sensors the data point came from. One of the interesting features of this data is the sheer volume, since it has over 2,000,000 records. However, we must consider that per class, the amount of data is probably in line with some of the other data sets we have tested here.

Results

0.7 Conclusion

Bibliography

[1] Woolam, Clay, Mohammad M. Masud, and Latifur Khan. "Lacking labels in the stream: classifying evolving stream data with few labels." Foundations of Intelligent Systems. Springer Berlin Heidelberg, 2009. 552-562.

[2] Zhang, Peng, Xingquan Zhu, and Li Guo. "Mining data streams with labeled and unlabeled training examples." Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on. IEEE, 2009.

[3] Masud, Mohammad M., et al. "Facing the reality of data stream classification: coping with scarcity of labeled data." Knowledge and information systems 33.1 (2012): 213-244.

[4] Besag, Julian. "On the statistical analysis of dirty pictures." Journal of the Royal Statistical Society. Series B (Methodological) (1986): 259-302.

[5]