A Visual Analytics Approach to Model Learning

Supriya Garg, I.V. Ramakrishnan, and Klaus Mueller

Computer Science Department, Stony Brook University

ABSTRACT

The process of learning models from raw data typically requires a substantial amount of user input during the model initialization phase. We present an assistive visualization system which greatly reduces the load on the users and makes the process of model initialization and refinement more efficient, problem-driven, and engaging. Utilizing a sequence segmentation task with a Hidden Markov Model as an example, we assign each token in the sequence a feature vector based on its various properties within the sequence. These vectors are then clustered according to similarity, generating a layout of the individual tokens in form of a node link diagram where the length of the links is determined by the feature vector similarity. Users may then tune the weights of the feature vector components to improve the segmentation, which is visualized as a better separation of the clusters. Also, as individual clusters represent different classes, the user can now work at the cluster level to define token classes, instead of labelling one entry at time. Inconsistent entries visually identify themselves by locating at the periphery of clusters, and the user then helps refine the model by resolving these inconsistencies. Our system therefore makes efficient use of the knowledge of its users, only requesting user assistance for non-trivial data items. It so allows users to visually analyze data at a higher, more abstract level, improving scalability.

KEYWORDS: Visual Knowledge Discovery, Visual Knowledge Representation, Data Clustering, Human-Computer Interaction.

INDEX TERMS: H.5.2 [Information Interfaces and Presentation]: User Interfaces—Graphical user interfaces; I.2.6 [Artificial Intelligence]: Learning—Concept Learning; I.5.3 [Pattern Recognition]: Clustering—Similarity Measures.

1 INTRODUCTION

With the tremendous growth in physical and online data collection technology, we are now experiencing an explosion of digital information. Since a large amount of these data are unstructured, various machine learning techniques have been developed to assign structure to these data to make them machine readable. This process can allow the machine to reason with and draw insight from data almost automatically. However, all such tasks depend heavily on large amounts of user-tagged data as the starting point, and use various semi-supervised learning methods [19]. Due to the high user input required, such tagged data is difficult to construct. Further, data is dynamic, and as a dataset grows and changes, we might need to supplement the tagged data from time to time. We propose to make this task simpler and interactive by designing a system where the user can obtain a visual overview of the dataset, and in that visual interface only tags those data elements that the system cannot easily resolve itself.

One crucial idea behind our system is that given good feature vectors to represent each data point, points that are similar will be close-by in the feature vector space. Here, we mean data-points which though rich in semantics, do not have an explicit high-dimensional feature vector automatically attached to them. In such cases we need to *design* feature vectors to represent the semantics and structure of the data-points. We aim to achieve this in our system by designing feature vectors which encompass a data point's structure, context, and location in the dataset. If some sort of semantic information is available, that can be added to the feature vectors as well.

Based on the above feature vectors, we design a visual interface where data is displayed in 2D space based on their feature vector similarity. This gives the users an overview of the dataset, and they can easily observe sets of data points which form spatial clusters. At this stage, we can apply clustering algorithms which arrange similar points closer together.

Clustering as we know is a general approach which can help users explore and analyze large data sets since it lets users work with groups of objects, rather than individual objects which can be large in number. Clustering associates objects in groups such that the objects in each group share some properties (similar feature vectors) that hold to a lesser degree for the other objects. Spatial clustering builds clusters from objects being spatially close or having similar spatial properties. However, clustering methods when run automatically can give non-intuitive results. Therefore, we allow the user to tweak the parameters of both the clustering algorithm and the data's feature vector to improve results.

We can use any well-known clustering algorithm to subdivide the points into clusters. This can help provide the initial biasing for any classification algorithm. However, the model learnt initially is a very rough estimation of the actual model, and needs fine-tuning to improve performance. For this purpose, we keep the user in the loop and utilize human insight to resolve inconsistencies. Overall, our visualization system allows the analyst to: (1) identify the classes in the data and communicate these to the model learning system; (2) assess the fitness of the generated model by the structure it imposes on the data; and (3) refine the model by communicating misrepresented patterns back to the model learning system.

The model learning system uses the visualization to *explain* the model to the analyst. While most communication of the analyst is gesture-driven, editing facilities are available to directly specify or refine the model. We call this iterative model building process data-driven, user-assisted model *sculpting* or model *debugging*.

After we have learnt an initial model, we apply it to the dataset to segment it. The user can then examine the visualization for possible misclassifications, and reclassify the data points. To guide the refinement of the model, we develop an interface similar to a step debugger in a common software development environment. Here, the model's structural graph represents the program script while the visualization is the program output. Users can step through the model's graph, examine the visual model explanations, and refine the corresponding rules if required.

The main contribution of our work is that it lightens the huge burden of individually hand tagging data, and allows users to tag

Email: {sgarg, ram, mueller}@cs.sunysb.edu

data at a higher level, with greater interactivity. Our approach can provide a solution to a large range of segmentation and classification problems in the presence of complex and ambiguous data. This includes the audio/video domain where we want to segment the input by speakers, scenes, moods, etc, and the image domain where we classify images by content type, or segment an image into objects.

On the other end of the spectrum are methods which try to learn important feature vectors for data classification automatically [16]. However, they are highly time intensive. By using our approach, we can let the user into the loop, and allow him to guide the system making the process much more efficient.

Our paper is structured as follows. Section 2 presents related work, Section 3 describes relevant theoretical aspects, Section 4 provides details on implementation, Section 5 presents results, and Section 6 ends with conclusions and pointers for future work.

2 Related work

This work follows the general Visual Analytics idea of combining human domain knowledge with automatic data analysis techniques by providing users with interactive visual interfaces, whereby 'interactive' means that users can actively participate in the analytical process as it evolves. Though our main focus is on allowing users to facilitate the process of model learning, visually guided data clustering is an integral part of it. There has been some work related to this field, including some in Visual Analytics. The approach described by Schreck et al. [17] allows users to leverage existing domain knowledge and user preferences, arriving at improved cluster maps. Zhang et al. [19] present a paradigm for visual exploration of clusters.Further, to make sense of the cluster results, visual representations are necessary. Projection-based approaches as presented in [8, 4] are common. We use a Multidimensional Scaling (MDS) approach in this paper.

Learning models from patterns is an active research topic in various branches of computer vision. But there the pattern examples in most cases originate directly from image analysis, promoting unsupervised learning where subtle anomalies (the unexpected data) or new families of patterns which the model parameters cannot capture often go undetected. Visual analytics, on the other hand, aims to be more flexible in the data constellations encountered, appealing to the complex pattern recognition apparatus of humans and their intuition, creativity, and expert knowledge to point out unusual configurations for further testing and model refinement. Papers recognizing this are currently emerging. Janoos et al. [9] used a visual analytics approach to learn models of pedestrian motion patterns from video surveillance data, in order to distinguish typical from unusual behavior in order to flag security breaches in outdoor environments. Their semi-supervised learning approach in which users interact with video stream data improves upon the standard unsupervised learning schemes that are typically used in these scenarios.

There has been little work in the field of visually assisted machine learning. The last few years at VAST has seen some papers in this domain. Our own work on Model-Driven Visual Analytics [6] describes a visual analytics system for high-dimensional data analysis. In this system, users visually explore the data in a high-dimensional space, and mark example patterns to iteratively learn rules using Logic Programming. The paper on LSAView [3] provides a visual analytic framework to analyze the model parameters in Latent Semantic Analysis, hence promoting model learning and debugging. Andrienko et al. [1] present an approach to extracting meaningful clusters from large databases by combining clustering and classification, which are driven by a human analyst through a visual interface.

3 Theory

We shall use a text segmentation application as an example to illustrate our work. Important here are the concepts of *document* and *token*. In text segmentation we then have a collection of documents that contain the text and the tokens are the words appearing in these text items (each token is composed of a collection of letters). But we may just as well perceive a set of images (or even videos) as a collection of documents and coherent image regions as tokens (which are then further composed into individual pixels). So we see that these concepts are quite general.

The data segmentation task involves working on multiple documents, which are further divided into tokens to do any processing and calculations. A document is the unit which is subject to either classification or segmentation. For text, it can be a single string (an address), a story (news entry), or an entire web page. A token is usually the smallest semantically meaningful unit in the document, and can vary depending on your approach. For text, it is usually a word; for images, it can be a pixel, or a contiguous iso-value region. For video, the natural unit is a frame. We observe that if there is a well-known boundary, then it is easy to extract tokens. In cases where such boundaries are missing, a multi-scale approach can work well.

Given the tokenized dataset, we often need to start with a coarse segmentation. This segmentation gets refined as we apply the learning algorithm, and possibly involve the user in the loop. This *windowing* approach has been used in the segmentation of audio broadcast news into stories [18]. In the absence of a numeric way to define data tokens, these windows provide a simple way to define a feature vector.

Segmentation is the process of converting the data in a raw stream of information into structured records. Given a schema consisting of *n* attributes and an input string, the problem of segmenting the input string involves partitioning the string into contiguous sub-strings and assigning each sub-string a unique attribute from the *n* attributes. For example, given the address schema consisting of the five attributes <COMPANY, STREET, CITY, STATE, PHONE> and the input string "Adieu Travel 117 Franklin St Dansville NY (716) 335-2222", the task of segmentation is to convert the string into the address record: <Adieu Travel, 117 Franklin St, Dansville, NY, (716) 335-2222>. Further difficulties may emerge when the records appear in different order in different strings. The running example in our paper uses the BigBook business address dataset [3] which contains approximately 3000 business addresses from New York State.

Information on the web (such as product listings, audio/video collections, or newscast) exists in an unstructured format. Segmentation into structured records is necessary to facilitate efficient query processing and analysis. The same goes for images and video for content-based image retrieval.

Segmentation techniques either use rules for identifying attributes or employ statistical models. Rule-based approaches require domain experts to create and maintain a set of rules for each application domain. It is difficult to anticipate all possible variations in the documents to be segmented and design rules accordingly. Further, noise in the data can compound the difficulty. Therefore rule-based approaches are neither scalable nor robust. In contrast, statistical approaches automatically learn a statistical model for each application domain. The variability and noise in the input text data are elegantly dealt with by the statistical characteristics inherent in such approaches.

Table 1: Feature vector table. Each matrix entry represents the presence or absence of a token in a document. Each row represents the feature vector for the corresponding token.

	ABBAB	ABBCB	CDADC	DCCDC	EDFEF	FEEFF
A	1	1	1	0	0	0
B	1	1	0	0	0	0
C	0	1	1	1	0	0
D	0	0	1	1	1	0
E	0	0	0	0	1	1
F	0	0	0	0	1	1

3.1 Hidden Markov Models

Without loss of generality, we use a Hidden Markov Model (HMM) [15] to learn the segmentation model. The HMM is a widely used statistical model used for data segmentation. It is a generative model since it captures the probability distribution of observations (e.g. the input strings in ase of text segmentation). HMMs are commonly used to represent a wide range of phenomena in text, speech, and even images (using 2D HMM [10,12].). An HMM consists of a set of states *S*, a set of observations (in our case words or tokens) *W*, a transition model specifying $P(s_t|s_{t-1})$, the probability of transitioning from state s_{t-1} to state s_t, and an emission model specifying $P(w|s)$ the probability of emitting word *w* while in state *s*.

To compute hidden state expectations efficiently, we use the Baum Welch algorithm. Emission models are initialized using the approximate classification done using the visual interface. The transition model consists of a completely connected graph with uniform probabilities. Finally, we use the Viterbi algorithm with the learned parameters to label the test data.

3.2 Overall Concept

The main idea behind this project is that humans with specific domain knowledge can easily spot the pattern in data, something which is very difficult for a machine to do. However, to make use of this expert knowledge, the data should be displayed such that the task for the user becomes easier. A good approach is to identify terms that exhibit similar characteristics, and display them together. This essentially means that we need to find appropriate feature vectors for each term and then cluster them together. In data segmentation and classification tasks, the location and neighborhood of a word has a great bearing on its classification. We observe two distinct cases:

I. Data belonging to the same class appears together *within* documents (e.g. when they belong to the same news topic), or

II. Data belonging to the same class appears in similar locations *across* documents (e.g. city names appearing in addresses).

We propose to treat each document as a sequence of tokens. For the case where each document contains one topic, the feature vector for the tokens is simply a binary vector with a 1 representing presence, and a 0 absence. An example is shown in Table 1 where each column is a document, and each row is a token. Here our vocabulary contains the letters (A,B,C,D,E,F), and we can clearly see the three clusters (A,B), (C,D) and (E,F). A better way to see patterns in a large dataset is to calculate the similarity between the feature vector of the tokens (using the dot product, or some other well-known method), and laying them out in 2D using Multidimensional Scaling (MDS) [11]. When tokens belong to exactly one topic, the classification task is quite simple. However the fact that most tokens belong to multiple topics makes the task harder, and often impossible to solve for the machine.

In Figure 1, we show an example to illustrate the value of this approach. In Figure 1(a), the tokens from our dataset are laid out randomly. Briefly looking at it makes it clear that these nodes represent male first names. However, in Figure 1(b) which shows the points laid out using the windowed approach, a user can see the further pattern that these names belong to different nationalities, and in fact this property (same nationality) causes nodes to be laid out close by. Figure 1(c) shows the words colored and circled according to their classification by the user. This task could not be done automatically since even though tokens in the same class occur close by, tokens belonging to different classes are also at a similar distance in multiple instances. Only the domain knowledge of a user can help resolve such ambiguities.

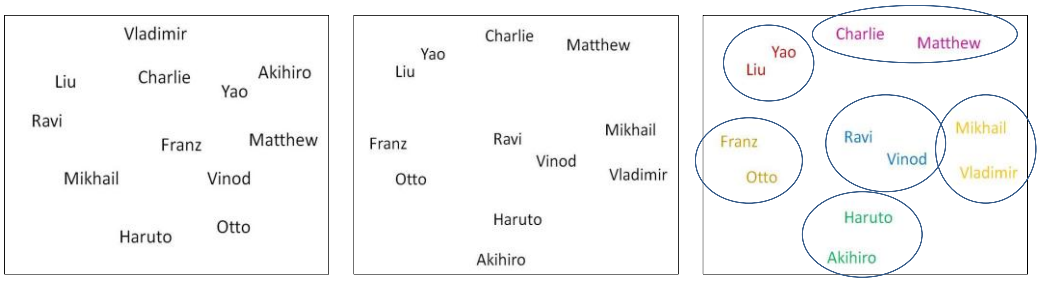
For case II, i.e. when tokens in the same class appear in similar locations *across* documents, we need to capture the locations they appear within all documents. To do so, we divide the tokens in each document into equal numbers of windows (N_w), i.e. each token is assigned a window according to its relative position within the document. Given document d_i , token tok_j, number of tokens in the document $num(d_i)$ and the location of the token $loc(tok_j)$, its window number is:

$$win\ \ i, tok_j\ \ = N_w \times (\frac{loc\ \ tok_j}{num\ \ d_i}) \quad (1)$$

$$\frac{tok_1\ tok_2|\ tok_3\ tok_4|\ tok_5\ tok_6}{win_1\quad\ |\quad win_2\quad |\quad\ win_3} \quad \text{(e.g.)}$$

The above example shows how a document with 6 tokens is divided into 3 windows. We construct the feature vector of each token as the histogram summarizing the frequency with which a token has appeared in different windows. An ideal window size is data and task dependent, and is best chosen while interacting with the visual layout of the data. For visualization, we can again calculate token similarity by taking a dot product between feature vectors, and display them in 2D using MDS.

An example of using windows for the initial segmentation of



(a) Random layout (b) Window-based layout (c) Clustered layout

Figure 1: This image shows the layout of points representing people's names (a) randomly, and (b) based on the window based approach. The user interacts with the graph in (b), and marks people who belong together (here based on nationality), giving us (c).

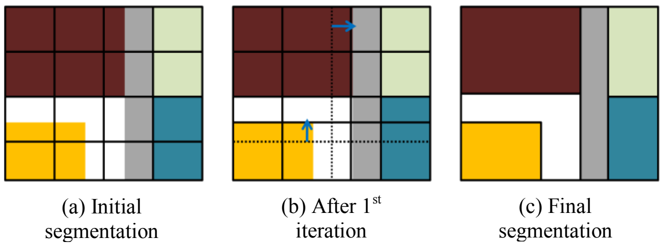


Figure 2: An example of using a coarse segmentation (windowing) to initialize the image segmentation process.

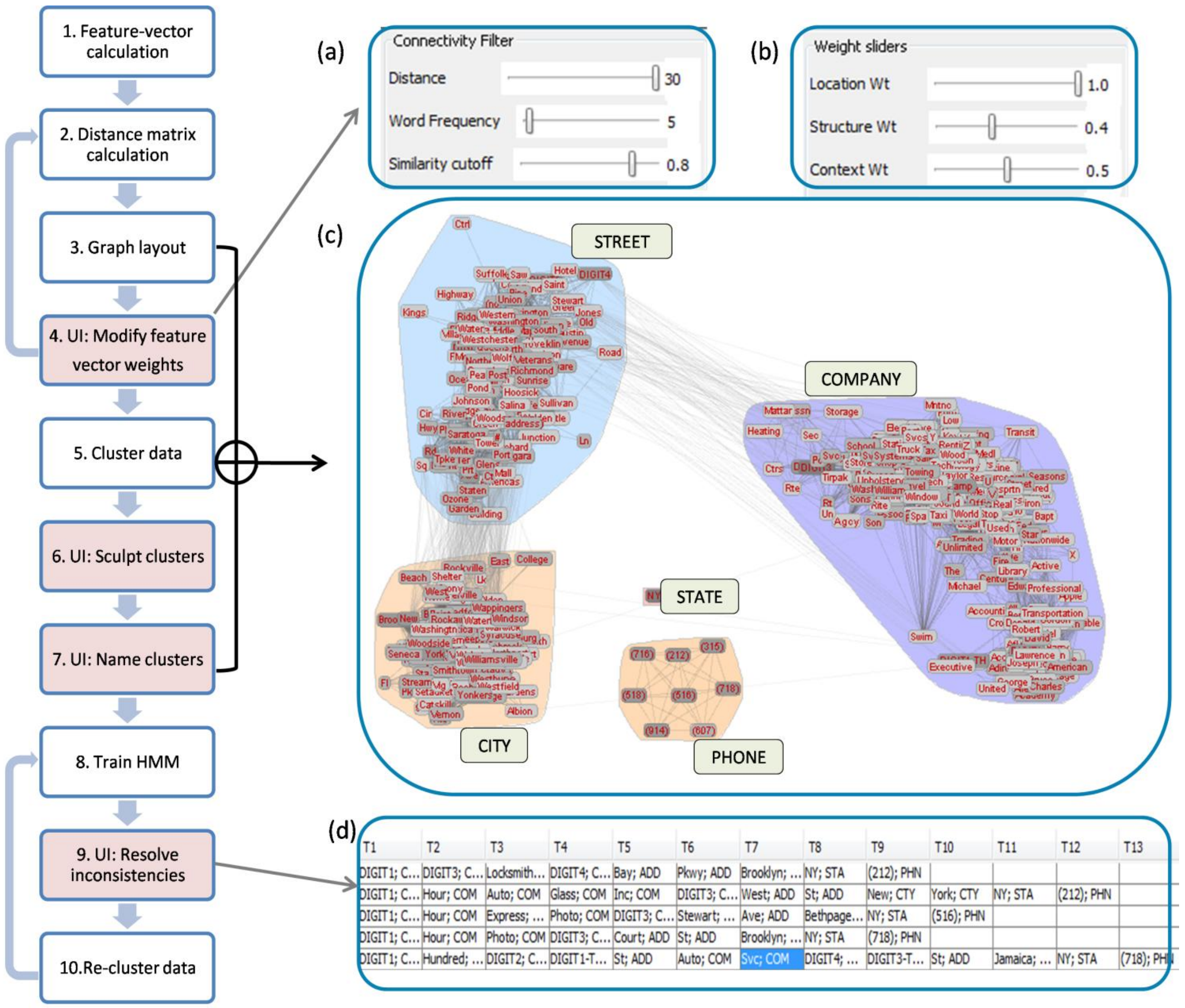
data is shown in Figure 2. Here our task is to segment the colored image into its constituent blocks (Fig 2(c)). Initially we divide the image into 4×4 sub-images. This starts off the learning algorithm, and after a few iterations gives the required result. Note that the window sizes are at a similar scale to the final segments. If we start off with windows too large (larger than the largest segment), then the results might not be even close to expected. On the other hand, windows that are too small increase the problem size, making it longer (and harder) to solve.

This windowing approach is similar to the one used in [14] which use bigrams and trigrams as basic units for document visualization.

4 Implementation

The basic approach of our system is as follows: we take tokenized data documents, calculate the relevant feature vectors, and allow the user to help initialize and refine models to cluster the data. As mentioned, we use HMMs as an example of a learning approach. An overview of our system is shown in the flowchart in Figure 3.

Initially the data is preprocessed, and tokenized. So each document will contain a sequence of tokens which need to be labeled. We consider all identical tokens to be instances of the same *entity*, and our feature vector calculations are based on these entities.



Connectivity Filter
Distance 30
Word Frequency 5
Similarity cutoff 0.8

Weight sliders
Location Wt 1.0
Structure Wt 0.4
Context Wt 0.5

T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13
DIGIT1; C...	DIGIT3; C...	Locksmith...	DIGIT4; C...	Bay; ADD	Pkwy; ADD	Brooklyn; ...	NY; STA	(212); PHN				
DIGIT1; C...	Hour; COM	Auto; COM	Glass; COM	Inc; COM	DIGIT3; C...	West; ADD	St; ADD	New; CTY	York; CTY	NY; STA	(212); PHN	
DIGIT1; C...	Hour; COM	Express; ...	Photo; COM	DIGIT3; C...	Stewart; ...	Ave; ADD	Bethpage...	NY; STA	(516); PHN			
DIGIT1; C...	Hour; COM	Photo; COM	DIGIT3; C...	Court; ADD	St; ADD	Brooklyn; ...	NY; STA	(718); PHN				
DIGIT1; C...	Hundred; ...	DIGIT2; C...	DIGIT1-T...	St; ADD	Auto; COM	Svc; COM	DIGIT4; ...	DIGIT3-T...	St; ADD	Jamaica; ...	NY; STA	(718); PHN

Figure 3: Overall system. (Left): Flowchart outlining our approach. The red boxes represent user-interaction steps. (Right): An overview of the system. (c) The entities are laid out in a node-link diagram, along with the clusters; (a) and (b) show sliders used by the user in step 4. (a) Sliders to filter the graph, (b) Sliders to adjust the weights of the feature vector components, (d) The interface showing segmentation results that fulfil certain inconsistency criteria – the user then interacts with the system (step 9) to resolve these inconsistencies.

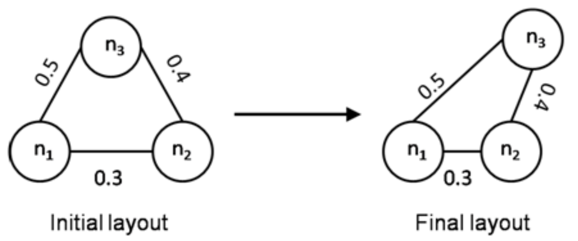


Figure 4: Graph evolution based on node dissimilarities (as shown along the edges)

Observing the datasets led us to the realization that there are three important properties which can help us classify a token –

(a) Its structure. For text this includes – is it a word, is it a number, its length etc.
(b) Its context, e.g. the tokens that appear before and after it,
(c) Its location in the document.

Structure and context can also include information beyond the information contained in the tokens themselves. This includes meta-data which comes attached. Webpage data though not necessarily machine readable already exhibits some structure via html meta-data. The non-visual web browser presented in [13] uses the meta-data to segment webpages into semantically related regions.

We numerically capture the above properties in each entity's feature vector. This means that a feature vector contains multiple high-dimensional vectors. When we calculate the similarity between any two entities, we need to calculate similarities based on each property separately, and then combine them into a final value.

A survey on cluster data mining techniques [2] concludes that data specific attribute selection has yet to be invented. Recent work on unsupervised feature learning [16], confirms that user interaction can greatly reduce the learning time. To help alleviate this problem to some extent, we give the power of assigning the weights of the feature vector elements to the user. The similarities due to different elements are all normalized to ranges between [0, 1]. Given two entities *x* and *y*, and similarity values *sim$_i$(x, y)* and weights w_i due to the three properties, the final similarity between them is:

$$sim\ x,y\ = \frac{{}_i w_i\ *\ sim_i\,(x,y)}{{}_i w_i} \tag{2}$$

4.1 Initialization Stage

The entities are displayed as a node-link diagram using a force-directed layout algorithm [7]. This algorithm considers a spring-like force for every pair of nodes *(x, y)* where the ideal length δ_{xy} of each spring is proportional to the graph-theoretic distance between nodes x and y. Minimizing the difference between Euclidean and ideal distances between nodes is equivalent to a metric multidimensional scaling problem. Here we use the *dissimilarity* between nodes as the ideal distance between two nodes. The dissimilarity is simply the inverse of similarity:

$$dissimilarity\ x,y\ = 1 - sim(x,y) \tag{3}$$

Figure 4 shows the graph evolution, and finally patterns emerge showing the overall structure of the dataset. At this stage, the user can modify the weights of the feature vectors. This updates the node similarities as well as the graph layout.

Note that any pair of nodes will have a similarity measure, but keeping all the edges will give us an extremely dense graph with no discernible structure. Removing edges between nodes with low similarity (< 0.5) helps this structure to emerge better. The optimal cutoff value depends on the data density and the feature vector selection (i.e. weights). We allow the user to select this cutoff value to find the appropriate optimal value. Figure 5 shows an illustration of how the graph layout changes with the removal of edges representing low similarity. The one disadvantage is that some nodes become isolated, i.e. are not connected to any other node. We can handle this by either allowing the user to assign them to a cluster *after* we call the clustering algorithm. Else, the HMM can classify them during the learning stage, and they get displayed in the corresponding cluster during the refinement stage.

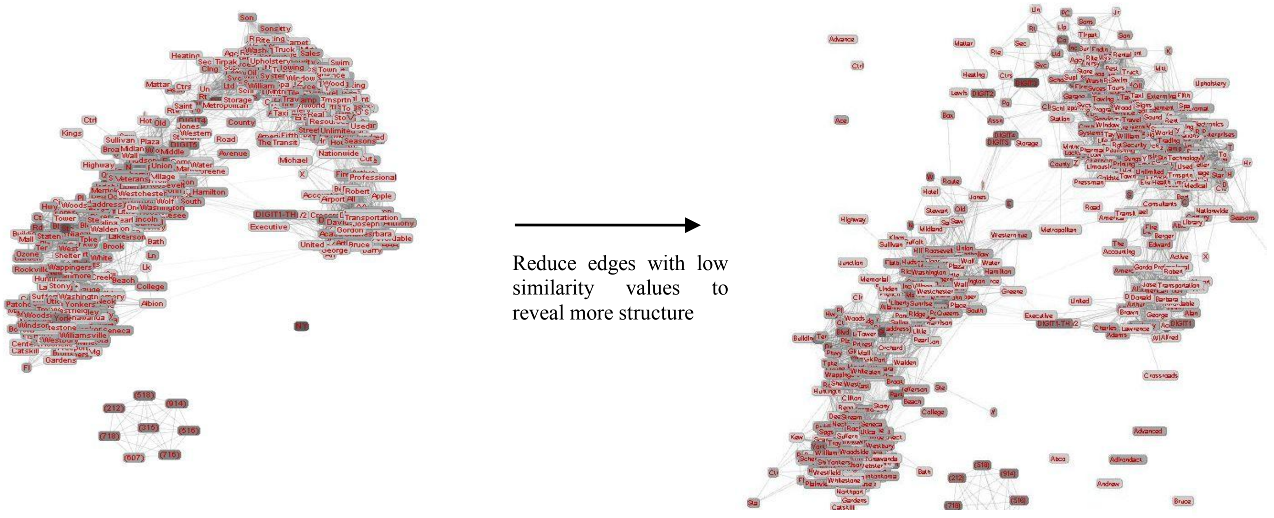

Reduce edges with low similarity values to reveal more structure

Figure 5: Graph structure: As we change the cutoff values for node-pair similarity from 0.5 to 0.85, the graph goes from being almost a single mass of nodes on the left, to one displaying more clusters on the right

Users can affect the graph layout, and the clustering using the following inputs:

(a) Modifying the weights of the feature vector elements
(b) Modifying the similarity cutoff – this removes the edges representing low similarity, and also reveals a clearer structure of the dataset.
(c) Modifying the frequency cutoff of the data points. This leaves only the most prominent nodes behind, which act as representative nodes, and give the users a good idea of the classes present in the dataset.
(d) Modifying the "fineness" level of clustering – this controls how many clusters the data gets split into. Getting the most appropriate number of clusters will reduce the load on the user during the *sculpting* stage. (Step 6 in Figure 3).

Further, we notice that the visualization reveals more structure when we display entities from the entire dataset, rather than from a smaller subset. This is because in the smaller dataset most low frequency entities do not cluster well. As we increase the number of documents, some of these frequencies improve, and the entities become better defined.

When the various settings work together to produce a semantically meaningful structure, the user can call the clustering algorithm. A good choice of weights at this early stage will minimize the work required from the user at later stages – especially at the cluster sculpting stage, and relabeling the tokens at the refinement stage. In most cases, a range of values gives decent results, hence carefully updating the weights and observing the visualization can take care of this problem.

We use the Markov Cluster Algorithm (MCL), an unsupervised cluster algorithm for graphs based on the simulation of stochastic flow in graphs [5]. Only those similarity values that are above the cutoff specified by the user are passed to the clustering algorithm. This has a similar effect as it has on the visualization – it helps segment the data better, i.e. divides them into more classes. Once the clusters are calculated, they are displayed by forming a convex-hull boundary around the data points they contain. Further, we relax the strength of edges across clusters to reduce their spatial overlap. (See Figure 6)

The physical boundaries between clusters, might lead the user to discover some semantic discrepancies – this usually occurs due to entities which are ambiguous and can occur in multiple classes, or in cases where the feature vector is not able to classify an entity correctly. In this case, a user can visually *sculpt* the clusters – this involves splitting and merging clusters, dragging nodes from one cluster to another, or duplicating nodes into different clusters (for example, the token 'York' may appear in 'Street', 'State', 'City', and even 'Company' depending on context). The visualization assists the user at this stage by highlighting the nodes that have more than 99% of edges to nodes within the same cluster. At the end of this stage, the number of states in the HMM is assumed to be the number of clusters in our visualization. Further, the emission probabilities are calculated based on these clusters – entities with a higher frequency in the original dataset have a higher emission probability. To allow the model learning algorithm to assign a word to a different class from that in the above cluster, we assign a small emission probability value to all the words in all the states other than the one they appear in. This approach also allows the Baum-Welch algorithm to learn fairly accurately for words which appear in multiple classes. We go through the first round of HMM training to learn our initial model, and then use Viterbi algorithm to segment the strings.

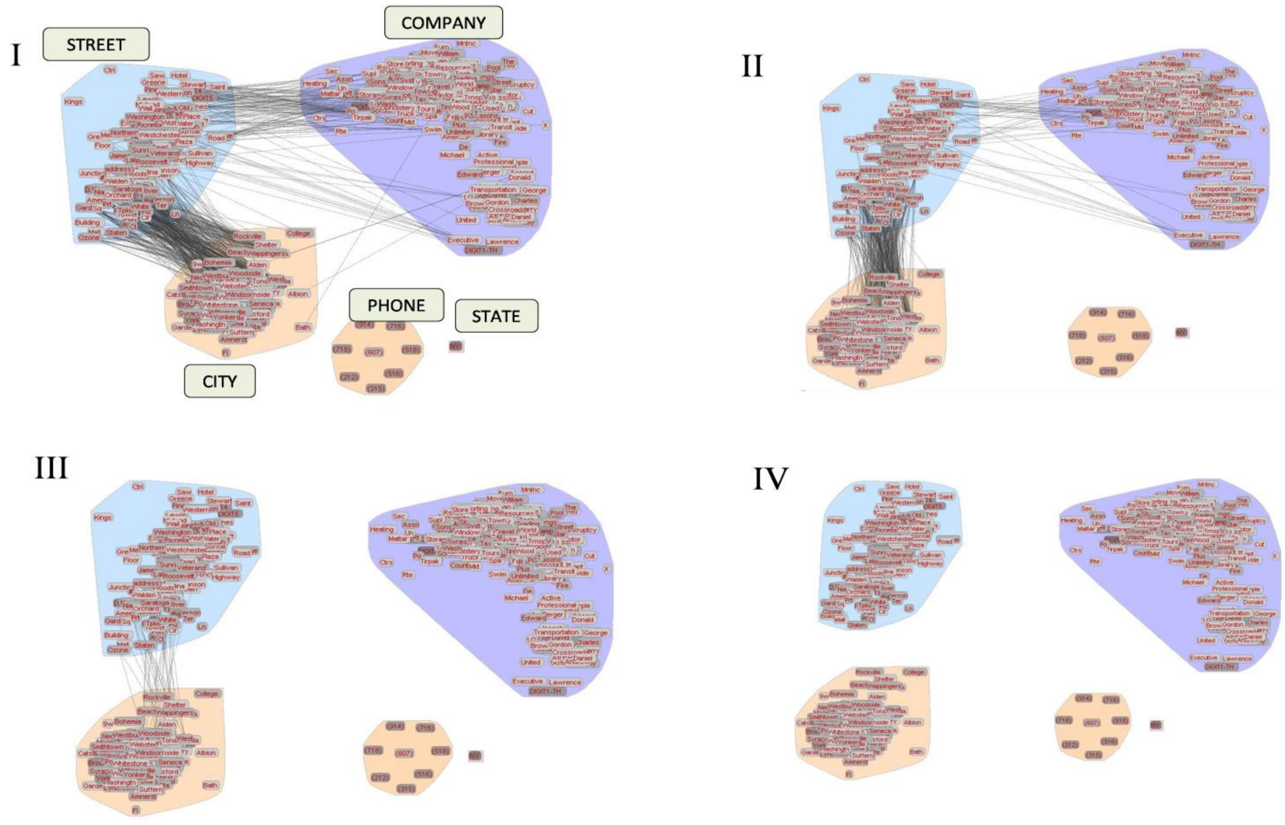


Figure 6: Result. The four panels show the evolution of the clusters as the user reclassifies the segmentation of some address entries, and the HMM relearns the model taking this into account

4.2 Refinement Stage

At this stage the user can help resolve inconsistencies in the segmentation, and help debug the model. To make this more intuitive, we first need the user to give the clusters semantic names. This is done by highlighting the nodes in all clusters which has a very low similarity to nodes in other clusters. In case the presence of some entity makes the identity of the class ambiguous, the user can choose to see some strings where the entity appears. For e.g. if Washington appears in a cluster with what mostly appears to be street names, seeing it in context will help the user be sure that the given cluster indeed contains street names, not cities or states. Two examples are given below:

1. D Sacilotto 399 *Washington* St New York NY (212) 966-7274
2. Burke & Casserly PC 255 *Washington* Avenue Ext Albany NY (518) 452-1961

At this stage, we have named clusters and a trained HMM. Now, we can involve multiple users in the refinement stage. Just as interaction with users can quickly help define models, misclassifications at any stage can slow the process down. Hence involving multiple users at the refinement stage can help remove mistakes made by a user due to either lack of knowledge, or uncertainty.

We pick documents that contain tokens that did not cluster well – i.e. had a lot of highly similar nodes in other clusters. These are the entities which belong to multiple classes in a dataset. The user resolves this inconsistency, and if such a token is indeed classified into more than one class by the user, we duplicate it to represent various identities of a single entity. This also reduces the edges across clusters, making the model better defined. We ask the user to reclassify a few documents at a time, and then retrain the HMM to reflect these changes.

During this debugging stage, the user can highlight the consecutive tokens in the graph layout. When a specific token is selected, the corresponding node(s) in the graph are highlighted. Also, the cluster to which it has been classified is also highlighted. This helps the user see the possible classes the token could belong to. The user can traverse forwards and backwards along a given document to establish if there is any misclassification.

The visualization where the clusters are marked often excludes entities with low frequency – either because they were filtered out by the user, or they were not assigned a cluster due to low similarity to any cluster. However, after the HMM is trained, all the entities in the dataset will be assigned emission probability values. Further, if we want to display duplicates for entities belonging to multiple clusters, the visualization will become too dense. To overcome this problem, after the initialization stage, we only display the entities which have very few high similarity edges to nodes in other clusters. These are called the *inner* nodes, and are the representatives of their classes. When a certain token is selected during debugging, its corresponding node(s) are also visualized. The user can still use our interface to display more nodes if required.

5 Results

The system we described can help in the classification of varied types of datasets. As a starting step, we worked with text-based datasets since they require minimal preprocessing, and can show the utility of our method. In this section we will see a running example of learning a model (HMM) for the business address dataset. The BigBook business address dataset contains approximately 3000 business addresses from New York State. This dataset contains a large number of distinct integers, and each separate integer does not have a special meaning, but usually just represents a street or building number or is a part of a business name. To make sure that the HMM is able to learn well, and apply the model to addresses with new tokens, we replace each integer with the term DIGIT-<*i*> where *"i"* is the length of the integer.

After the initial preprocessing, we calculate feature vectors based on the three properties of structure, context and location:

- The feature vector on *structure* contains the following information: does it contain a letter, does it contain a digit, does it contain a non-alphanumeric symbol, does it begin with a capital letter, is it all caps, and its length. Except for the last element, the others are binary.

- The feature vector on *context* contains a summary (i.e. histogram) of the structure based feature vectors of the tokens that appear immediately before and after the given entity in all the documents.

- Finally, for the feature vector on *location*, we use the windowing approach as presented in Section 3.2. Initially, if we know the number of classes (N_c) in the dataset, we use its multiples as the number of windows. In the absence of this information, the window size is decided by the document lengths (i.e. number of tokens in the document).

Now the data is displayed on the screen, and as the user plays with the similarity cutoff slider, the graph goes from being one main cluster with all nodes interconnected, to one which shows the inbuilt structure of the data. At this point, the user clusters the dataset. Since there is a lot of overlap between the terms in ADDRESS and CITY (see Figures 4 and 6), there is an extra cluster which contains these common tokens. The user organizes and splits this cluster to the best of his knowledge, and merges the corresponding halves with the two clusters.

Next we highlight the *inner* nodes in the dataset, which shows the user the representative terms for each cluster, and he is able to name them. Figure 7 shows an example of these inner nodes. Now we initialize the emission probabilities for all states based on the clusters. If an entity belongs to a cluster, its emission probability

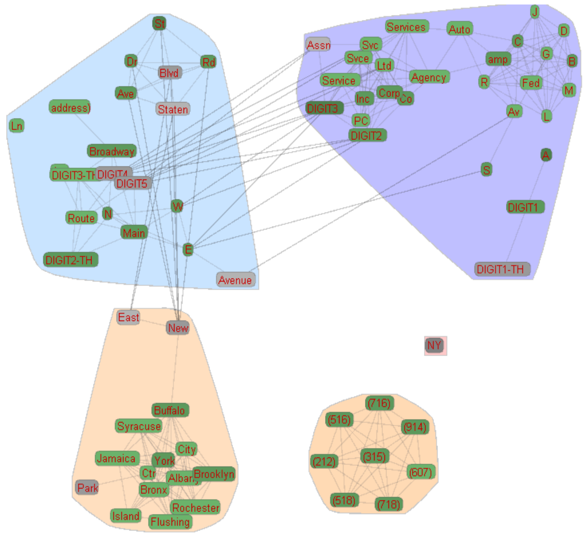


Figure 7: Graph showing the inner entities/nodes in green. These nodes help the user both during the cluster sculpting stage, and during the cluster naming stage.

is the ratio between frequency of the entity and the total frequency of the entities in the cluster. As mentioned in Section 4.1, we assign a small probability value to all the entities *not* belonging to the cluster. At this stage the Baum-Welch algorithm is called to learn an initial model.

Now the user is shown strings with one token that has high similarity to nodes in multiple clusters. Given that the token has high similarity to entities in classes c_1 and c_2, we:

(a) Assign a positive value if the classification was correct
(b) If the classification in class c_i is deemed incorrect, and reclassified to class c_j, then we assign a heavy penalty to c_i and a heavy bonus value to c_j.

The accumulated values at the end of a reclassifying round are used to modify the emission tables learned by the Baum-Welch. Further, based on this, we also split the nodes if they are classified into different classes in the cluster. The user helps in the debugging process till the clusters become better defined, as shown in Figure 6. After four rounds of refinement based on the tokens with high presence in multiple clusters, we get a good model for classification. The time taken between iterations depends on the complexity and implementation of the algorithms used to learn the model, and segment the strings – in this case the Baum Welch, and Viterbi algorithm respectively.

6 Conclusions

In this paper we have presented a general approach to visualization-assisted model learning for data segmentation and classification tasks. The driving motivation for our approach is that the typical manual tagging of data is very resource intensive. Further, even if one uses just a small dataset for tagging to boot-strap the learning process, if the chosen subset is not a good representative of the entire dataset, then the models learnt might not be robust. On the other hand, completely automated methods which take a brute force approach by using large feature vectors for classification have the drawback of being very time intensive, and small misclassifications can cause the model learnt to be less than ideal. In this case user interaction at various stages can help the machine stay on track, and so will improve the speed as well.

As part of our future work, we aim to extend this approach to varied data types and with different clustering and classification algorithms. This promises to give further insight into semi-automatic design of feature vectors for other domains. In this paper, we have demonstrated the use of our system for a fairly constrained problem – the segmentation of business addresses using HMMs. Work is currently underway that extends the use of our system to other model building tasks, such as classifiers, using the categorization of large image collections and documents as a driving application.

In future work, we plan to optimize the usability aspects if the system by ways of focused user studies. In particular, we would like to study and precisely quantify how much faster a model of equal quality can be derived with our framework, as opposed to more traditional non-interactive approaches. Further, we also wish to study if the visual analytics approach we have proposed here allows domain experts to produce models that are more accurate than those derived automatically since the user is actively involved in the model-sculpting task.

Acknowledgements

This research was supported by NSF grants CCF-0702699 and CNS-0627447. We also thank all reviewers for their useful suggestions.

References

[1] G. Andrienko, N. Andrienko, S. Rinzivillo, M. Nanni, D. Pedreschi and F. Giannotti. Interactive visual clustering of large collections of trajectories. *Proc. IEEE Symp. Visual Analytics Science and Technology (VAST)*, pp. 3-10, 2009.

[2] P. Berkhin. A survey of clustering data mining techniques. *Grouping Multidimensional Data*. pp. 25–71, 2006.

[3] P. Crossno, D. Dunlavy, T.Shead. LSAView: A Tool for Visual Exploration of Latent Semantic Modeling. *Proc. IEEE Symp. Visual Analytics Science and Technology (VAST)*, pp. 83-90, 2009.

[4] I.S. Dhillon, D.S. Modha and W.S. Spangler. Class visualization of high-dimensional data with applications. *Computational Statistics and Data Analysis*. 41(1): 59–90, 2002.

[5] S.V. Dongen. *Graph clustering by flow simulation*. PhD Thesis, University of Utrecht, The Netherlands. 2000.

[6] S. Garg, J.E. Nam, I.V. Ramakrishnan, K. Mueller. Model-Driven Visual Analytics. *Proc. IEEE Symp. Visual Analytics Science and Technology (VAST)*, pp. 19-26, 2008.

[7] J. Heer, S. Card, J.A. Landay. Prefuse: a toolkit for interactive information visualization. *Proc. SIGCHI Conference on Human Factors in Computing Systems*, pp. 421–430, 2005.

[8] A. Hinneburg, D. Keim, M. Wawryniuk. HD-Eye: Visual mining of high-dimensional data. *IEEE Computer Graphics and Applications*. 19(5): 22–31, 1999.

[9] F. Janoos, S. Singh, O. Irfanoglu, R. Machiraju, R. Parent. Activity Analysis Using Spatio-Temporal Trajectory Volumes in Surveillance Applications. *Proc. IEEE Symp. Visual Analytics Science and Technology (VAST)*, pp. 3-10, 2007.

[10] J. Jiten, B. Merialdo. Semantic Image Segmentation with a Multidimensional Hidden Markov Model. *Advances in Multimedia Modeling*. 4351: 616–624, 2007.

[11] J.B. Kruskal and M. Wish. *Multidimensional scaling*. Sage Publications, Inc. 1978.

[12] J. Li, A. Najmi, R. Gray. Image classification by a two-dimensional hidden Markov model. *IEEE Trans. on Signal Processing*. 48(2): 517–533, 2000.

[13] J.U. Mahmud, Y. Borodin, I.V. Ramakrishnan. Csurf: a context-driven non-visual web-browser. *Proc. 16th International Conference on World Wide Web*, pp. 31-40, 2007.

[14] Y. Mao, J. Dillon. G. Lebanon. Sequential Document Visualization. *IEEE Trans. on Visualization and Computer Graphics*. 13(6): 1208-1215, 2007.

[15] L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*. 77(2): 257–286, 1989.

[16] R. Raina, A. Battle, H. Lee, B. Packer, A. Ng. Self-taught learning: transfer learning from unlabeled data. *Proc. 24th International Conference on Machine Learning*, pp. 766, 2007.

[17] T. Schreck, J. Bernard, T. Tekusova, J. Kohlhammer. Visual cluster analysis of trajectory data with interactive Kohonen maps. *Proc. IEEE Symp. Visual Analytics Science and Technology (VAST)*, pp. 3–10, 2008.

[18] A. Vinciarelli, S. Favre. Broadcast News Story Segmentation using Social Network Analysis and Hidden Markov Models. *Proc. 15th International Conference on Multimedia*, pp. 264-267, 2007.

[19] K.B. Zhang, M. Huang, M. Orgun, Q. Nguyen. A Visual Method for High-Dimensional Data Cluster Exploration. *Proc.16th International Conference on Neural Information Processing*, pp. 699–709, 2009.

[20] X. Zhu. Semi-Supervised Learning Literature Survey. *Technical Report 1530*, 2005.