Text Summarization using SigSpace

Manikanta Maddula[1]

***Abstract*— Text Mining is one of the research areas that has been gaining lot of interest these days. The increasing availability of online information has necessitated intensive research in the area of automatic text summarization within the Natural Language Processing (NLP) community. Summarization technique preserves important information while reducing the original document. Text summarization can be implemented using Text classification. Any text mining application needs a corpus and the data (or features) that is fed to classification machine learning algorithm is huge. This study focuses on optimizing computing of machine learning by using SigSpace model. SigSpace model uses signatures instead of raw features reducing data drastically and it implements other features like independent learning, distributed learning, optimized accuracy, fuzzy matching etc. Finally, evaluation is done comparing different methodologies. And Real-time implementation in a web-client using Twitter heron, Apache Kafka, Mongo DB.**

I. INTRODUCTION

In the era of big data, it is essential to explore the opportunities in discovering knowledge from big data. However, traditional machine learning approaches are not well fit to analyze the full value of big data. Specifically, current research and practice of machine learning does not fully support some important features for big data analytics such as incremental learning, distributed learning, and fuzzy matching. In this paper we use unique feature representation, named SigSpace. SigSpace has advantages like class level incremental learning, distributed learning and fuzzy matching. But I was able to implement only signature representation but not incremental learning for each class.

II. BACKGROUND AND RELATED WORK

A. SigSpace

As the name says SigSpace[13] uses signatures of a class as features to classification algorithm. In SigSpace, class based model was built by an evaluation and extension of existing machine learning models i.e., K-Means and Self Organizing Maps (SOM)[17]. The machine learning with SigSpace was modeled as a feature set with standard machine learning algorithms (e.g., Naive Bayes, Random Forests, Decision Tree) as well as a class model using L1 (Manhattan distance) and L2 (Euclidean distance) norms. Comparison with the famous feature representations[16] like Bag of Words and SigSpace is presented in Fig.1. SigSpace author Pradyumna, D. evaluated the model for Image Clustering and Audio classification. This paper is one of the first implementations of SigSpace to text mining and classification.

*This work was not supported by any organization

[1]M. Manikanta is a graduate student Computer Science Engineering, University of Missouri, Kansas City, USA

	Bag of Visual Words [1][2]	PCA [4]	SigSpace
Clustering	✅		✅
Code Book dependant	✅		
Keeps Original Data			✅
Data reduction		✅	✅
Easy of Model update			✅
Fuzzy Matching			✅
Incremental learning			✅
Distributed learning			✅

Fig. 1. Comparison between SigSpace and Others

B. Architecture

SigSpace is a class based modeling technique used to implement incremental and distributed learning approach. Architecture is presented in Fig.2. Architecture explains image clustering. So, features extracted here applies for image clustering. For each and every class in a data-set of C classes, Features for images are extracted using pixels, SIFT (Scale-invariant feature transform), LBP (Local Binary pattern). SIFT is an algorithm in computer vision to detect and describe local features in images. SIFT key points of objects are first extracted from a set of reference images and stored in a database. An object is recognized in a new image by individually comparing each feature from the new image to this database and finding candidate matching features based on Euclidean distance of their feature vectors. From the full set of matches, subsets of key points that agree on the object and its location, scale, and orientation in the new image are identified to filter out good matches. LBP is similar to SIFT but with binary patterns. The features extracted from image for each class clustered using SOM and kMeans to identify signatures for that class. Signatures from each class are aggregated to form SigSpace. This sigspace is used to classification model. And for test data points fuzzy matching is used for identification.

C. Self-Organizing Maps (SOM)

A self-organizing map (SOM)[17] or self-organizing feature map (SOFM) is a type of artificial neural network (ANN) that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional), discretized representation of the input space of the training samples, called a map. Self-organizing maps are different from other artificial neural networks as they apply competitive learn-

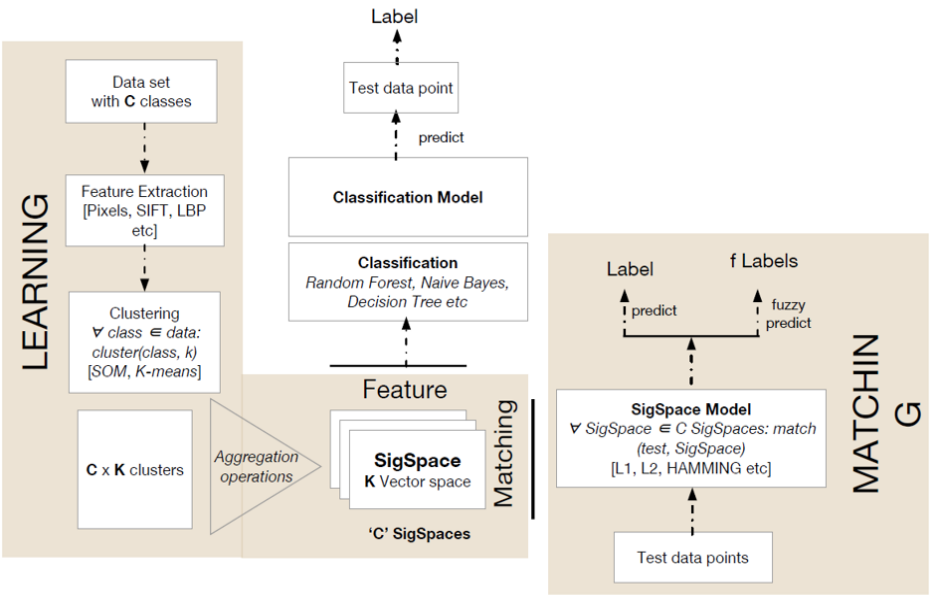

Fig. 2. SigSpace Architecture (Image Clustering)

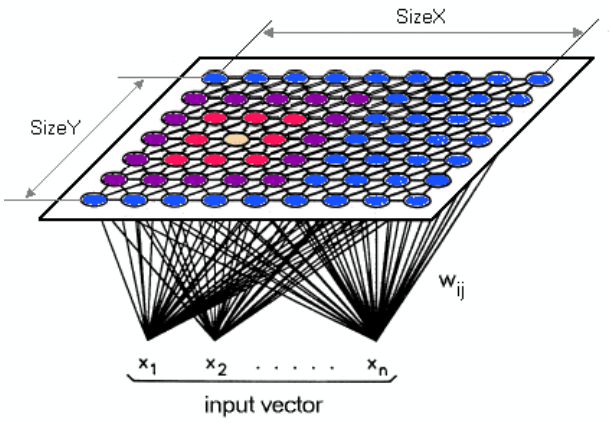

Fig. 3. Example SOM Map representation

ing as opposed to error-correction learning (such as back-propagation with gradient descent), and in the sense that they use a neighborhood function to preserve the topological properties of the input space. This makes SOM's useful for visualizing low-dimensional views of high-dimensional data, akin to multidimensional scaling. The SOM both projects a data set into a more illustrative form on a lower-dimensional space and reduces the number of data items into a smaller number of map locations. The representations become ordered according to their similarity relations in an unsupervised learning process. This organizing property makes it especially useful for analyzing and visualizing large data collections. The projection preserves the topology of the data so that similar data items will be mapped to nearby locations on the map. Refer Fig.3 for sample SOM explanation.

In this paper Encog Machine Learning Package[8] is used to implement SOM in Apache Spark. Word category maps[11] (also called self-organizing semantic maps) are SOM's that have been organized according to word similarities, measured by the similarity of the short contexts of the words. Word category maps are used to produce signatures.

III. Workflow

In this section, the architecture and over all work flow of the presented system is discussed. Architecture of system is given as in Figure3. Preprocessing is the first stage to clean text data and remove unnecessary data etc. Next, raw features are generated using Tokenizer, TFIDF, Word2vec. Using SOM data dimensions are reduced and then using KMeans, signatures of data are produced. These signatures are given to Naive Bayes Classification algorithm.

A. Preprocessing

Raw text data from articles is first read into an RDD in apache spark[1]. Stanford CoreNLP[7] provides a set of natural language analysis tools. It can give the base forms of words, their parts of speech, whether they are names of companies, people, etc., normalize dates, times, and numeric quantities, and mark up the structure of sentences in terms of phrases and word dependencies, indicate which noun phrases refer to the same entities, indicate sentiment, extract open-class relations between mentions, etc. Using Stanford CoreNLP, lemmatization is applied to the articles text data. Lemmatization is a process of returning base forms of words as in dictionary. This RDD is converted to Sparks Data Frame. Sparks data frame allows to do SQL operations as well as Sparks Machine Learning Pipeline architecture.

B. ML Pipeline

Apache Spark[1] provides machine learning pipeline and several machine learning algorithms. Tokenization is the

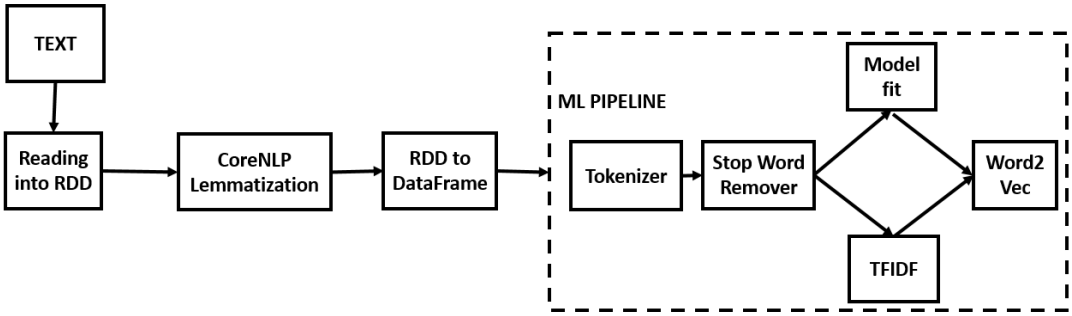

Fig. 4. Pre-processing and ML Pipeline

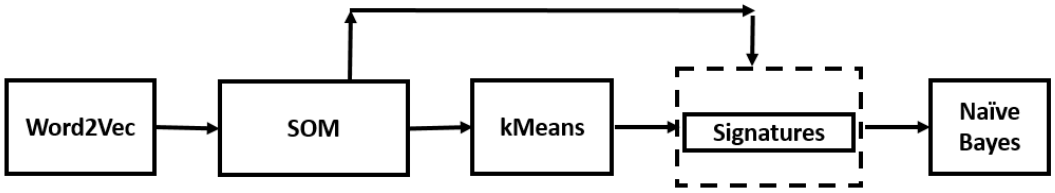

Fig. 5. Signature Generation and classification

process of taking text (such as a sentence) and breaking it into individual terms (usually words). A simple Tokenizer class provides this functionality. Stop words are words which should be excluded from the input, typically because the words appear frequently and dont carry as much meaning.

StopWordsRemover takes as input a sequence of strings (e.g. the output of a Tokenizer) and drops all the stop words from the input sequences. In this paper default sparks stop word list is used and no additional words were added to the list. Word2vec model is used to generate features from text data. Word2Vec computes distributed vector representation of words. The main advantage of the distributed representations is that similar words are close in the vector space, which makes generalization to novel patterns easier and model estimation more robust. Distributed vector representation will be useful for clustering and etc. Word vectors of 100 dimensions is used in this experiment. Sample word vector representation: [[0.13914690911769867,-0.09770306199789047, 0.19433113932609558,.....,0.10426576435565948],incident]

Top TF-IDF words are generated for each class individually and word vector representation of these top words is taken as signature for each class. 10000 top TF-IDF words for 4 classes are used in this experiment.

C. Signature Generation

Word vector representations generated from word2vec are of high dimensions ranging from 100 to thousands. Size of data at this stage will be high due to this. So, SOM is used to reduce the dimensions of data typically to one or two, so that they can be visually represented in maps. Example, reduced dimension map is presented in Fig.6. Word map categories will be present in this map. One of the other main advantages of SOM is it wont disturb the data topologically. Input neurons which are close together in data will also be close together or at same output neurons. SOM weight is saved at this stage.

This word map categories[9][11][12][14] are then applied to kMeans to identify different signatures. Say we want to identify sub topics in Cricket class, we can get more number of clusters as needed. In this experiment, 10 sub topics are identified for each class so, a total of 40 categories from whole data. An example clustered visualization for 4 classes is presented in Fig.7. So, signatures of 40 categories are identified at this stage.

D. Naive Bayes

Now signatures are used as input to Naive Bayes classification algorithm. In machine learning, Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features. Apache Spark provides an implementation of Naive Bayes classifier by taking labeled point data as input. The signatures generated for all categories or classes is converted to labeled point data and is sent to Naive Bayes algorithm. Naive Bayes model is saved to predict class for future text articles. Sample Naive Bayes Input:

(33.0,[13.0,87.0])
(12.0,[3.0,80.0])
(24.0,[35.0,46.0])

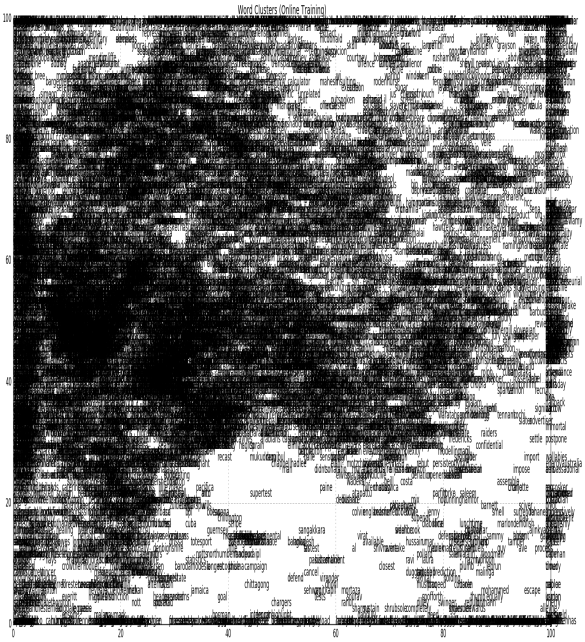

Fig. 6. Sample Clustering into 4 classes

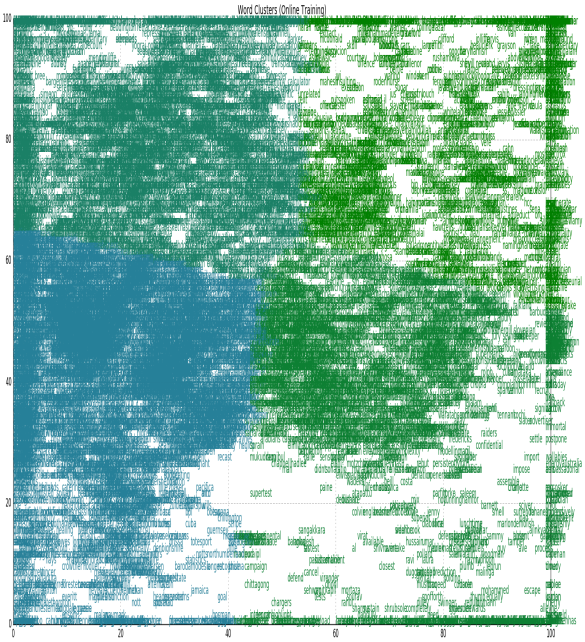

Fig. 7. SOM Dimension reduction

E. Text Summarization

Now for an input article the same workflow is applied using saved models. And features will be generated for input document in similar manner. And classification is done for individual sentences in the put document. So, sentences fall into different topics or classes. Sentences which group into same class will hold the key point information of the article. And on top of this a threshold of Naive Bayes predicted probability is used to filter more sentences.

IV. Evaluation

Having described the full pipeline, we now evaluate it in this section. We evaluate the overall workflow by calculating various performance measures like precision, recall, F-Measure and data reduction. And also the presented model is compared with several other approaches for same data.

A. Dataset

The dataset used for this paper is collected from 20 Newsgroups. 20 Newsgroups[3] data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. BBC Sport[2] website corresponding to sports news in five topical areas from 2004-2005 with 737 documents and 5 natural classes. Out of this data only 4 classes of data are taken for experiment. Cricket, Football, Rugby and Tennis. Wikipedia data is collected for these classes using following steps:

- Get the titles of required pages using : https://petscan.wmflabs.org/. PetScan[4] can generate lists of Wikipedia (and related projects) pages or Wikidata items that match certain criteria, such as all pages in a certain category, or all items with a certain property. PetScan can also combine some temporary lists (here called ”sources”) in various ways, to create a new one.
- Get the Actual article data as XML dump from titles using: https://en.wikipedia.org/wiki/Special:Export[5]
- Parse the XML data dump individual text articles using python.[10]

Total raw text data is of 53 MB size. The following table2 explains different performance metrics for the presented system:

TABLE I
Evaluation Table

Metrics	Value
Accuracy	0.307136715
Weighted Precision	0.18779641
Weighted Recall	0.307136715
Weighted F-Measure	0.218288242

The size of the word vector representation sent as input to SOM is of 60.4 MB (60400 KB). And the size of signatures of all the classes combined together is 538 KB which is just 0.89 % of raw feature data and 1 % of raw text data. Here data reduction is done drastically.

TABLE II
EVALUATION TABLE2

Metrics	SOM-kMeans-NV	SOM-SOM-NV	kMeans-NV
Accuracy	0.307136715	0.230311277	0.456447771
Weighted Precision	0.18779641	0.121611037	0.208344568
Weighted Recall	0.307136715	0.230311277	0.456447771
Weighted F-Measure	0.218288242	0.152954293	0.286099608
Input to Naive Bayes	538KB	544KB	54.9MB
Raw word vectors	60.4 MB	60.4 MB	54.5 MB
Raw text data size	50MB	50MB	50MB

B. Other approaches

Several other approaches can be implemented for generating signatures for Classification. And also a traditional text mining approach is implemented for comparison.

1) SOM-SOM: Instead of SOM-kMeans, we can implement SOM and SOM to generate signatures. Word vector representations of top TFIDF words is inputted to SOM to reduce the dimensions of data. This two-dimension data is inputted to another SOM to generate signatures for 40 classes (or 10 sub topics for each class).

2) K-Means and NV: To compare between traditional text mining approach, this is implemented. K-Means is applied on top of word vector representations of top TF-IDF words to generate 40 clusters. This is converted to labeled data points as input to Naive Bayes algorithm. Comparison of different approaches is presented in Figure.

Refer to Table2. Here we can easily say that though accuracy is less for traditional text mining (kMeans-NV) approach, the data reduction is so good that it helps for performance. And accuracy for SOM-kMeans is more than SOM-SOM. And it can be seen that using signatures is efficient in storage space and computing.

C. No. of Classes

An evaluation of comparing performance of SOM-kMeans-Naive Bayes methodology for different number classes is presented in table3. Here we can see that as the number of classes increased the accuracy and other parameters of Naive Bayes algorithm decreased proportionally. Here it can also be seen that as the number of classes increased, the size of the signatures increased but it is minimal.

D. Different Classifiers

An evaluation of different classifiers available in Apache spark is presented in Table4. Comparison is made between Naive Bayes, Decision Tree and Random Forest classification algorithms. Here it can be seen that Random Forest performed better with an accuracy of 0.58 even for 40 classes. Random Forest Performed better than Naive Bayes but not better than Decision Tree.

E. Summarization

Sample output for a test document:

((25.0,The wicketkeeper claimed immediately.),[0.052532050353851875, 0.00805542112760521, ... , 0.023203159922015 97]) This vector contains predicted class, actual sentence, predicted probabilities of all the classes. This kind of vectors is generated for each and every sentence in the test article. Largest number of sentences classified to same class are extracted. This is outputted as summarized text.

V. REAL-TIME IMPLEMENTATION

The test scenario for real-time text summarization can be implemented using Apache kafka - a pub/sub message system, Apache Spark, Twitter heron - a distributed realtime computation system, Mongo DB. The methodology used in implementation is SOM-kMeans-DT.

A. Architecture

The high level architecture is presented in Fig. 8. From a web-client data or in this case the text to be summarized is sent to Apache Heron using a Kafka Producer. Inside heron there is a Kafka Consumer to receive text from the client. In heron, spark is used to extract features, generate signatures, to classify the sentences to sub-topics. And then summarization is done using these results. Summarized sentences are saved in mongo db which is used to display results in web-client.

B. Web-client

In the web-client, user is able to paste the text to be summarized and clicks on send. Text that needs to be summarized should belong to one of the classes "Cricket, Football, Rugby, Tennis". And then user can see summarized article by clicking on "Get Summarized article". Here a new screen with summarized text is presented. This is implemented in a Java servlet program using Eclipse IDE.

C. Apache-Kafka

Apache Kafka is an open-source stream processing platform for handling real-time data feeds. It is a scalable pub/sub message queue architected as a distributed transaction log. A kafka producer is implemented in the web-client using java libraries. Text to be summarized is sent as ¡Key, Message¿ pair with key as "article id" and actual text as Message. And these messages are consumed in Twitter Heron by implementing a kafka consumer. Multiple consumers can be subscribed to same topic, so this can be implemented in a distributed system.

TABLE III
EVALUATION BASED ON NO. OF CLASSES

Metrics (SOM-kMeans-NV)	4 Classes	20 Classes	40 Classes
Accuracy	0.705467675	0.395460798	0.307136715
Weighted Precision	0.599410381	0.301215389	0.18779641
Weighted Recall	0.705467675	0.395460798	0.307136715
Weighted F-Measure	0.642733022	0.317656779	0.218288242
Input to Naive Bayes	528KB	532KB	538KB

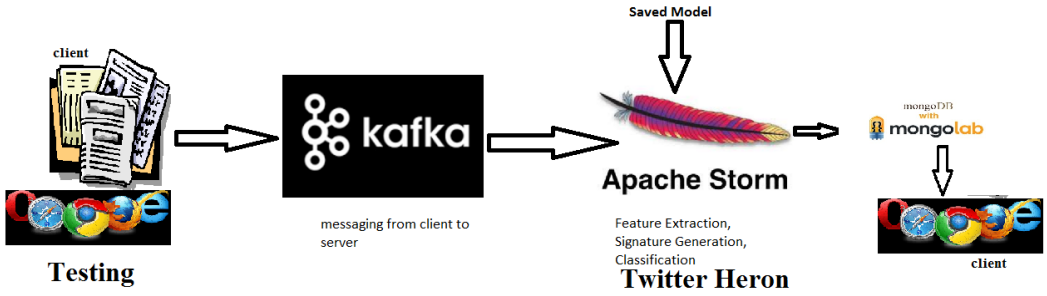

Fig. 8. Real-Time Architecture

D. Twitter-Heron

Twitter-Heron is a distributed stream processing computation framework. It uses custom created ”spouts” and ”bolts” to define information sources and manipulations to allow batch, distributed processing of streaming data. It is designed as a ”topology” in the shape of a directed acyclic graph (DAG) with spouts and bolts acting as the graph vertices. Edges on the graph are named streams and direct data from one node to another. Together, the topology acts as a data transformation pipeline. Here we can implement this in Apache Storm as well, but Twitter Heron is better than storm in performance. Data is exchanged between spouts and bolts in tuple based format. A kafka consumer is implemented inside spout to consume the text to be summarized as messages. And using Stanford nlp libraries, sentences are generated and emitted as tuples. These are consumed in a bolt. In the first bolt, a spark environment is implemented. Features are extracted for individual sentences using the process explained in training. Lemmatization, Tokenization, stop-word remover and then using the saved Word-to-vector model from training phase, a vector represented feature for the sentence is created. It is then dimensionally reduced using the Self-Organizing Map saved in training phase. It is then classified to one of the 40 sub-topics. This process is repeated for all the sentences in text. In the next bolt, using class predictions text is summarized. Not all sentences from text get classified to same class. So, the sentences which get classified to same class are grouped and the group with highest number of sentences is outputted as a summarized text. This summarized text is saved in Mongo DB. Using the article id, the summarized text is displayed to user in the web-client.

E. Result

For the test condition an article based on Cricket is selected. source: https://cricket.yahoo.com/news/india-peg-england-back-jennings-ton-debut-133017554.html. This article contained 28 sentences and after summarization, it contained 21 lines. This summarization efficiency can be improved by considering individual words instead of sentences while classifying to sub-topics. There was more latency than expected in real-time implementation because of two reasons: 1. Because of single machine, spout and bolt processing was serial. 2. Spark is implemented inside heron which is sub-second latency system but the latency in spark itself is in seconds. To implement the whole process in a more fast processing system needs training phase to be also implemented in the new system because text feature extraction process is code book dependent.

VI. FUTURE WORK

The presented methodology cannot implement code book independence, so, the best future work should be to make it code book independent. It is difficult to make it code book independent in text domain because features generated will depend on domain being used. But an evaluation can be done by implementing class based approach for same methodology to test the extent of independence that can be achieved.

Text Summarization can be implemented by classifying each word instead of each sentence. In this way, a more effective summarization can be achieved.

The current approach is limited to having only one type of SigSpace, which is entirely based on the features used, but the recognition problem has a huge range of features that needs to be addressed. Meta data of SigSpaces can be used for better matching. This can even help SigSpaces

TABLE IV
EVALUATION BASED ON DIFFERENT CLASSIFIERS

Metrics	SOM-kMeans-NV	SOM-Kmeans-RF	SOM-Kmeans-DT
Accuracy	0.307136715	0.439133425	0.58218707
Weighted Precision	0.18779641	0.30136235	0.465711247
Weighted Recall	0.307136715	0.439133425	0.58218707
Weighted F-Measure	0.218288242	0.333877829	0.493593283
Input to Classification algo	538KB	542KB	541KB

to be organized hierarchically which helps very large scale classification/recognition problems to be achievable.

VII. CONCLUSIONS

The presented approach can be said as Global SigSpace since the workflow is not implemented for each class individually. And models at different stages had be saved, so, it is code book dependent. The data reduction has been done drastically that it adds value in computing power and storage. Original data is not lost during process and distributed learning is also implemented. This model can be extended to implement incremental learning and code book independent. Since the implemented approach is code book dependent, sub-second latency system was not possible because of latency created by Apache spark.

Source Code: https://github.com/manikantamaddula/KDM-Summer-2016-SigSpace_Text_Data,
https://github.com/manikantamaddula/RealTime-Fall2016-Sports-Video-Annotation-and-Summarization/tree/master/SourceCode/Text%20Summarization

ACKNOWLEDGMENT

I would like to thank Dr. Yugyung Lee[3] for her kindness, mentoring and valuable suggestion of different approaches for work flow, real-time implementation.

[3]Dr. Yugyung Lee is Faculty of Computer Science Engineering, SCE, University of Missouri, Kansas City, USA

REFERENCES

[1] https: spark.apache.org/
[2] http://mlg.ucd.ie/datasets/bbc.html
[3] http://qwone.com/ jason/20Newsgroups/
[4] https://petscan.wmflabs.org/
[5] https://en.wikipedia.org/wiki/Special:Export
[6] http://sujitpal.blogspot.com/2014/10/clustering-word-vectors-using-self.html
[7] http://stanfordnlp.github.io/CoreNLP/
[8] http://www.heatonresearch.com/
[9] Kaski, Samuel, et al. "WEBSOMself-organizing maps of document collections." Neurocomputing 21.1 (1998): 101-117.
[10] http://www.evanjones.ca/software/wikipedia2text.html
[11] Honkela, Timo. "Self-organizing maps of words for natural language processing applications." Proceedings of the International ICSC Symposium on Soft Computing. 1997.
[12] Shaikh, A. J., and V. L. Kolhe. "Framework for web content mining using semantic search and natural language queries." Computational Intelligence and Computing Research (ICCIC), 2013 IEEE International Conference on. IEEE, 2013.
[13] Pradyumna Doddala,SigSpace: Class-base Feature representation for Scalable and Distributed Machine Learning, Thesis Spring 2016 at University of Missouri-Kansas City.
[14] Guru Teja Mannava,iHear: Lightweight Machine Learning Engine with Context Aware Recognition Model, Thesis Spring 2016 at University of Missouri-Kansas City.
[15] Kraska, Tim, et al. "MLbase: A Distributed Machine-learning System." CIDR. Vol. 1. 2013.
[16] Onan, Aytu, Serdar Korukolu, and Hasan Bulut. "Ensemble of keyword extraction methods and classifiers in text classification." Expert Systems with Applications 57 (2016): 232-247.
[17] https://en.wikipedia.org/wiki/Self-organizing_map
[18] https://en.wikipedia.org/wiki/Apache_Kafka
[19] storm.apache.org/
[20] https://twitter.github.io/heron/
[21] https://mlab.com/