iHear – Mobile application which can recognize Speech to generate summary text data

CS5560: Knowledge Discovery and Management

Guided By

Dr. Lee Yugyung

By

Manikanta Maddula (15)

Motivation:

There are approximately 70 million deaf people in the world. For many, hearing aids, sign language, cochlear implants and subtitles are useful. Some develop Lip reading skill. However, access is limited to above mentioned technology for many people. So, a mobile application which can recognize speech from user and display summary text about what user is speaking would be very helpful and cost effective. The idea for the project came from Hearing Dog. It is an assistance dog specially trained to help deaf people. Dog is trained to alert the owner to important sounds like smoke alarms, doorbells, telephone ringing etc. A mobile application can be cost effective since a large amount of users are already using smart mobile systems. And this application serves even more than a hearing dog by recognizing speaker's audio and generates summary data (Key words) instead of displaying the whole text which is not comfortable for the user.

Objective:

There are some mobile applications which can recognize important sounds for users but this application serves more by making use of Machine Learning developments in recent years. There are several speech to text recognition API's available. These can be used to convert audio to text data. There has been a lot of development in natural language processing and semantic applications. Converted text can be summarized by using NLP, machine learning algorithms. But since computing power that is available in mobile is limited, light weight machine learning model can be used in client side to improve the performance and recognition time. Self-Organizing maps and sigspace feature modelling can be used to achieve light weight machine learning. Sample data is used for training (or learning) and model building. Speech to text API's are used to generate text data. From the converted text data, word embedding algorithms can be used to extract keywords. Here Ontology models are generated based on training data. To work with large models or large streaming data, apache spark can be used. Apache Spark is an open source big data processing framework which provides high performance and sophisticated analytics. Dynamic recognition can be used to generate topic data or summary data. A particular domain is selected for the purpose of project and later can be extended it to make it as domain independent.

The overall objective is to build an application which can recognize speech and generate topic data or summary data in text format.

Mobile Application:

Android is the selected mobile platform because by 2015 the amount of android phones brought are 81.61% globally. Application will have simple interface. A start button is used to start the speech recognition and stop can be used to make application stop listening. Most of the screen area is used to present the summary data. As this application deals with streaming data, complete analysis may be presented with some delay for better output. User will have option to save the output.

Domain Definition:

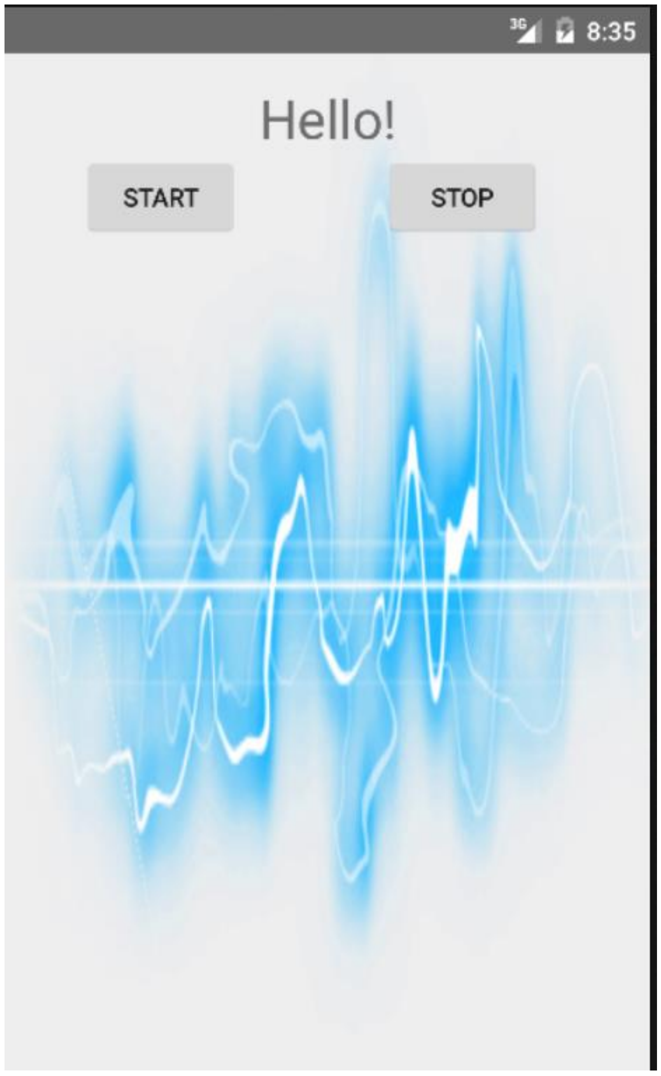
The application focuses on Sports domain. Application will be able to recognize the particular sport, players, event information and game commentary from the audio that the application is listening to. This could be from a speaker, a TV news, a recorded video etc.

Dataset:

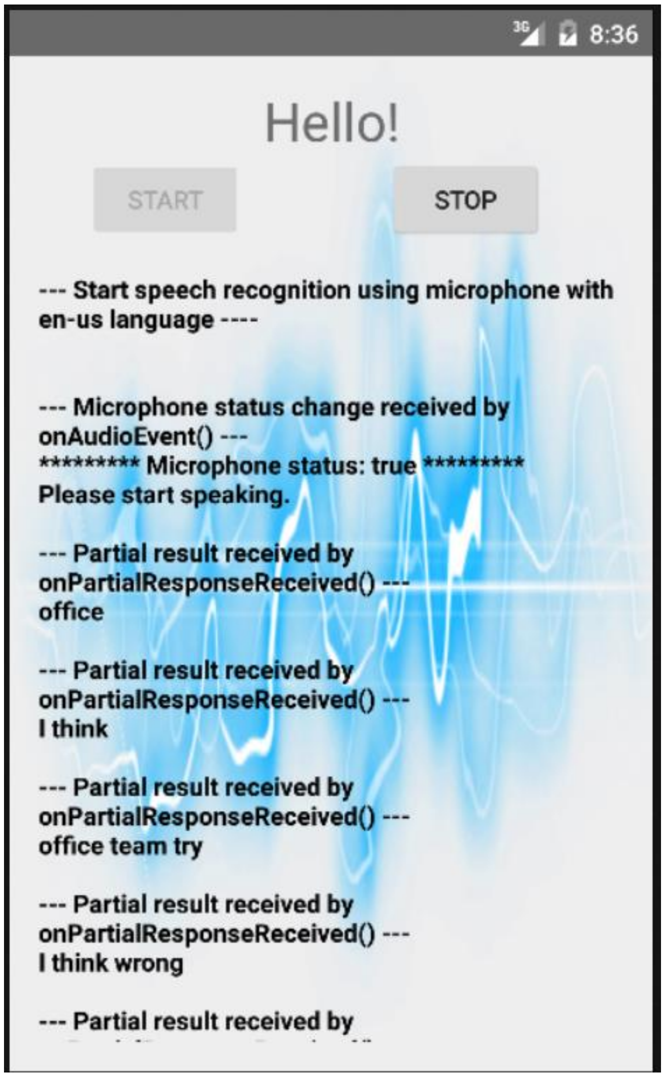
1. BBC Sport website corresponding to sports news in five topical areas from 2004-2005 with 737 documents and 5 natural classes. Source: http://mlg.ucd.ie/datasets/bbc.html
2. 20 Newsgroups data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. Source: http://qwone.com/~jason/20Newsgroups/
3. Wikipedia and DBpedia data API.

Tasks Completed in Phase1:

1. Android Application UI:
   Homepage:



Start Listening:

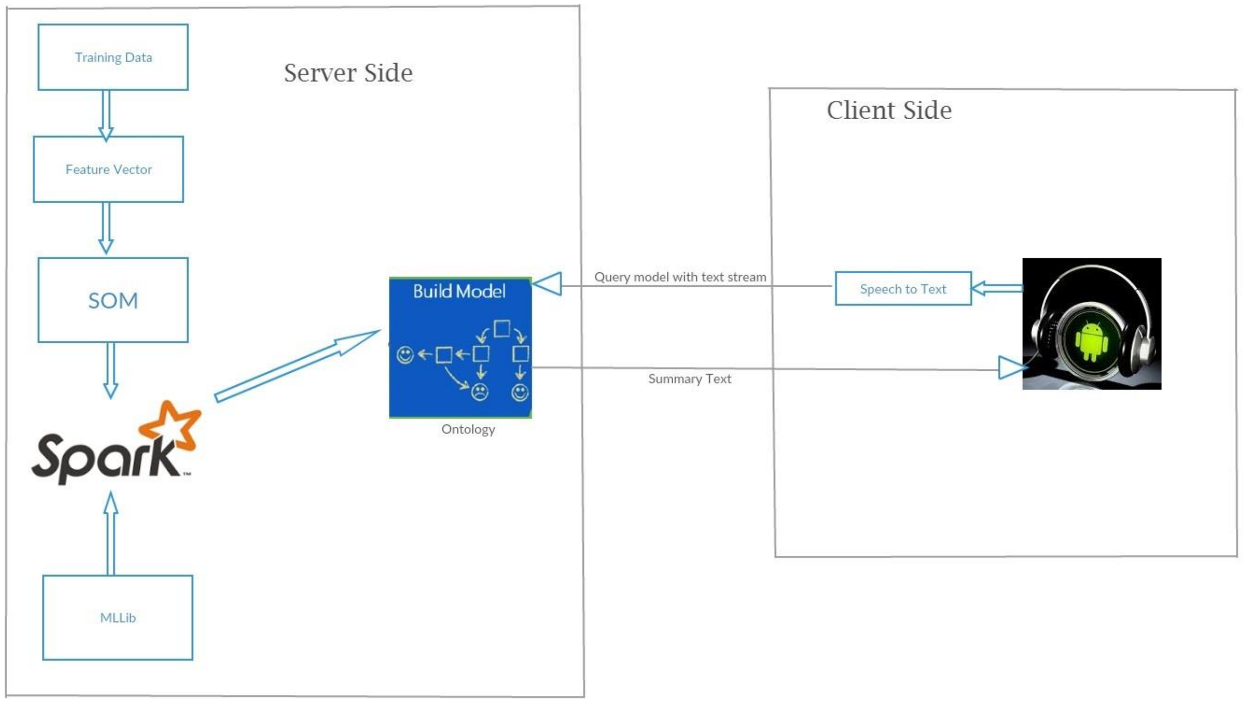


```
--- Start speech recognition using microphone with
en-us language ----

--- Microphone status change received by
onAudioEvent() ---
********* Microphone status: true *********
Please start speaking.

--- Partial result received by
onPartialResponseReceived() ---
office

--- Partial result received by
onPartialResponseReceived() ---
I think

--- Partial result received by
onPartialResponseReceived() ---
office team try

--- Partial result received by
onPartialResponseReceived() ---
I think wrong

--- Partial result received by
```

2. Microsoft Speech to text API implementation in android.
3. System Architecture Design.
4. Stanford Core NLP for NLP operations in Intellij using java.
5. TFIDF for some of the BBC dataset (Cricket category files) is done. Term document frequency is constructed.
   Example input: “Hayden sets up Australia win”
   Output for document1 (stripped):
   (1048576,[45,65,66],[0.9364934391916745,2.3814551551518304,7.751767917624546])

Implementation:

Architecture:

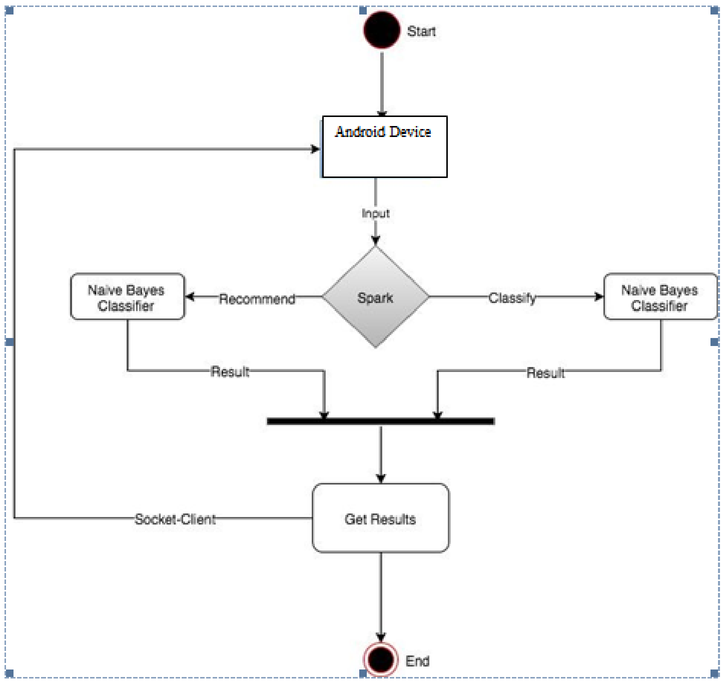


Raw data set is used to get feature vector by using NLP, TF-IDF, Word2Vec and LDA. By applying Machine learning algorithms (using Spark MLLib) to this feature vector a model is trained. Model is an ontology or knowledge graph related to domain. This model can be queried to get desired summary data by providing text stream as input. Features may need to be extracted from the input text data to query model for appropriate key words from input data.

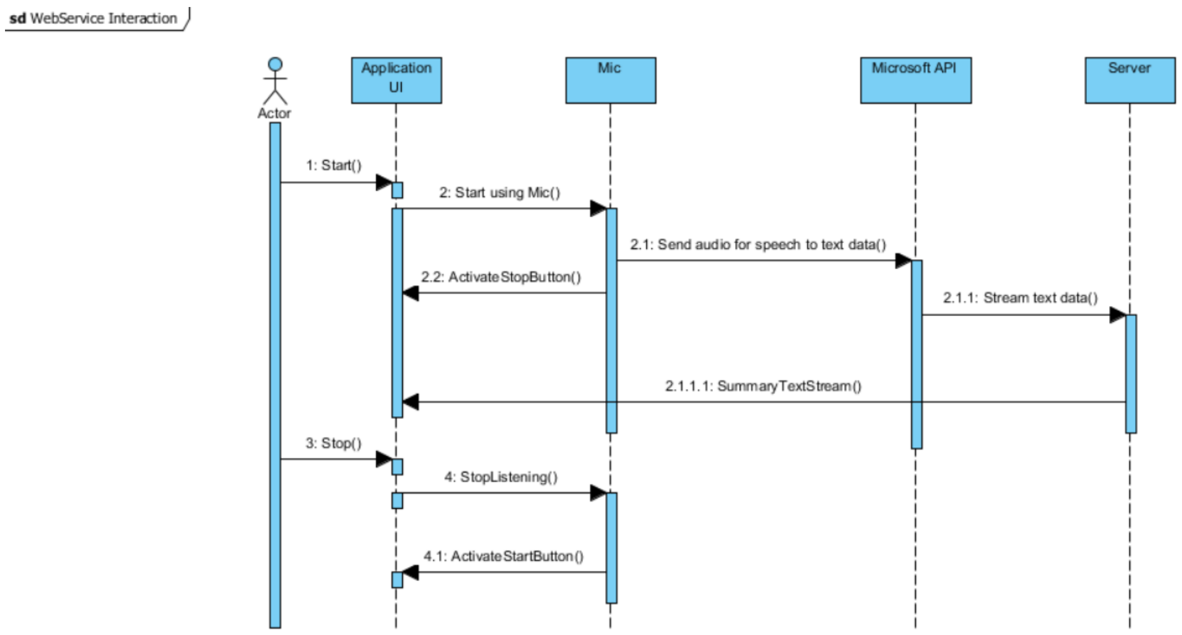
Audio data from android is converted to text using Microsoft Speech to Text conversion. Microsoft provides "Bing" (It's search engine) API and gives about 5000 transactions per month for free. This text stream is sent to server and summary text data is returned as result from server to android which is displayed in Android View.

Class Diagram:

This figure demonstrates the high level design architecture of the system.



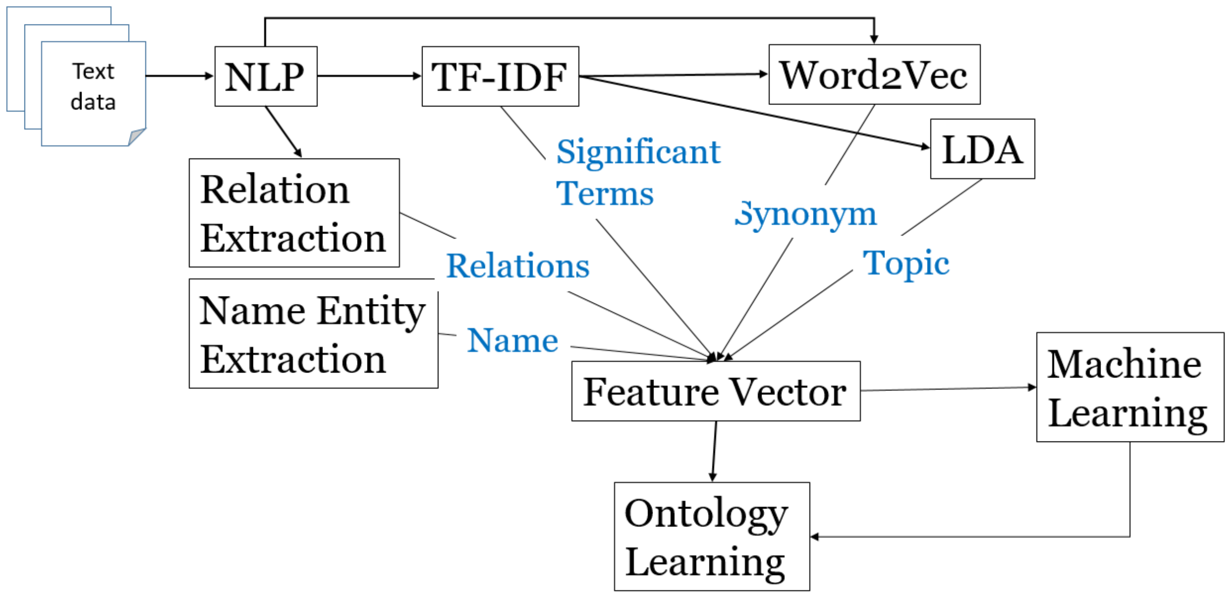
Sequence Diagram:


sd WebService Interaction
Actor
Application UI
Mic
Microsoft API
Server
1: Start()
2: Start using Mic()
2.1: Send audio for speech to text data()
2.2: ActivateStopButton()
2.1.1: Stream text data()
2.1.1.1: SummaryTextStream()
3: Stop()
4: StopListening()
4.1: ActivateStartButton()

The above figure is the sequence diagram for application interaction.

Workflow:

First phase of project is to develop Android UI. Then using API to convert speech from text.



Existing Services:

Microsoft speech to text API is used to convert Speech to text.

Stanford Core NLP API is used for NLP operations required for generating feature vectors. (Future)

Spark ML libraries are used for TF-IDF, Word2Vec operations. (Future)

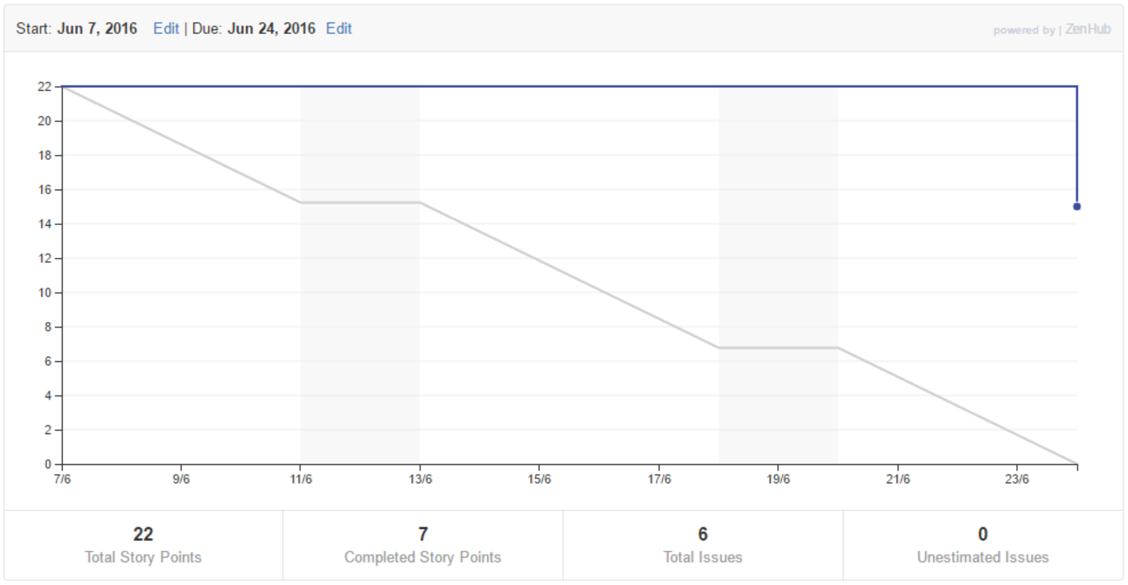
Spark ML Libraries are used for machine learning algorithms. (Future)

SparQL, OWL API, Ontology are used for model construction and querying. (Future)

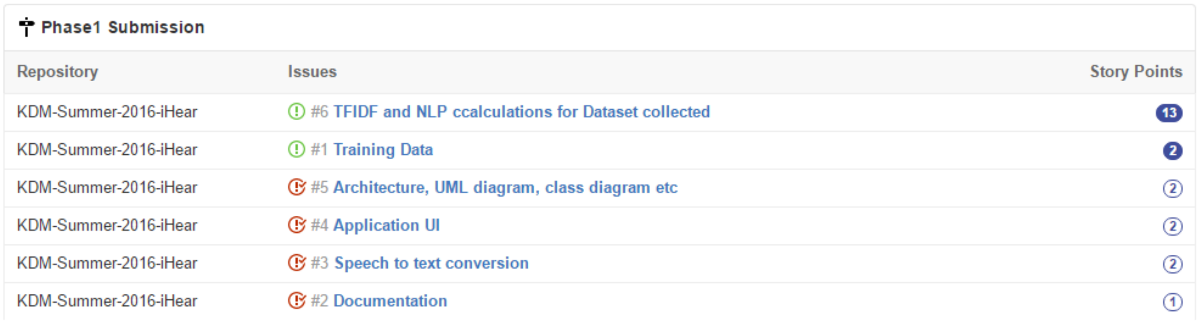
Android services for audio data generation.

Project Management:

Burndown Chart:

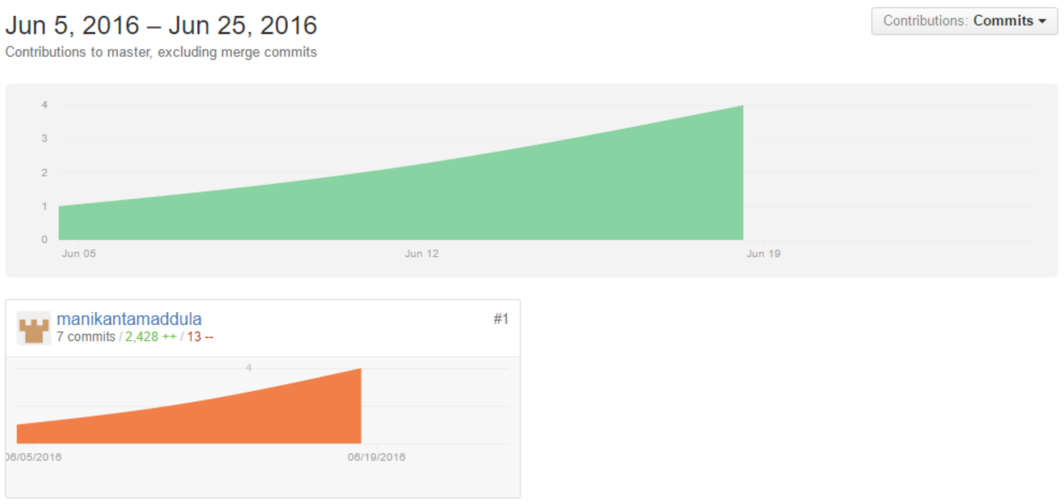

Start: Jun 7, 2016 Edit | Due: Jun 24, 2016 Edit

22	7	6	0
Total Story Points	Completed Story Points	Total Issues	Unestimated Issues


Phase1 Submission

Repository	Issues	Story Points
KDM-Summer-2016-iHear	#6 TFIDF and NLP ccalculations for Dataset collected	13
KDM-Summer-2016-iHear	#1 Training Data	2
KDM-Summer-2016-iHear	#5 Architecture, UML diagram, class diagram etc	2
KDM-Summer-2016-iHear	#4 Application UI	2
KDM-Summer-2016-iHear	#3 Speech to text conversion	2
KDM-Summer-2016-iHear	#2 Documentation	1

Contributions:


Jun 5, 2016 – Jun 25, 2016

Contributions to master, excluding merge commits

manikantamaddula #1

7 commits / 2,428 ++ / 13 --

Concerns/Issues:

1. Need to collect more datasets. Integrating datasets.
2. Need to figure out working with stream text data for querying server on summarization.
3. Need to know if feature extraction can be done in mobile client instead of server.
4. Can querying of model for summarization be implemented in mobile client.
5. Machine learning for categorization and classification.

Future Work:

Next phase of the project should complete until building model from datasets. This involves generating feature vectors, applying machine learning algorithms to build model.

Third phase of the project implements summarization from the model and audio data in application.

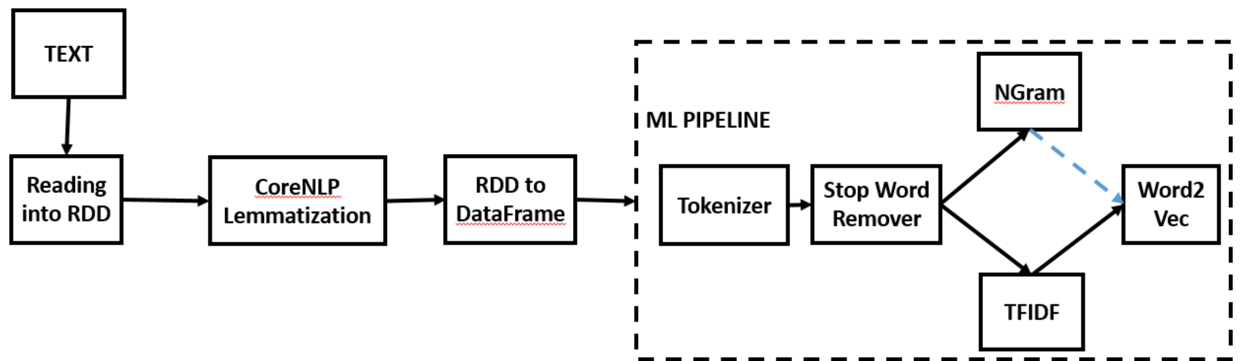
Project Phase 2

Word2Vec Vector:

Word2vec takes as its input a large corpus of text and produces a high-dimensional space (typically of several hundred dimensions), with each unique word in the corpus being assigned a corresponding vector in the space. Vector space is generated such that words in similar context are placed close to each other (e.g. Batsmen and bowler for Cricket sports class data). This vector can help in identifying missing terms and helps in clustering data of a particular class in the corpus. Spark Machine learning library provides this feature. Top N words will be chosen to form vector space. For each word from these N words, vector of fixed length is used to generate Word2Vec vector. Word2Vec gives synonyms (or similar in context) from the input text corpus.

Example:

```
luckhurst,(0.8266427357006286,programme),(0.8214192659449101,manager),(0.817240806
1576564,scotland),(0.8136894511675786,bit),(0.8130908702211868,security),(0.809329
6127207275,want),(0.8083818499540661,level),(0.8067136088558311,authority),(0.8064
129798041021,victory),(0.8063813083256627,short),
```
…



Cosine similarity score is calculated to find the words close to the context.

Since input to Word2vec is top TFIDF words, the output from word2vec is more relevant for any application. Word2vec produces context similar words and this can be used for prediction from input. And word2vec can be easily clustered since similar words are grouped. And this can be used for topic classification as well. For example, capitals and states can be grouped separately. And we can predict capitals of un known states after grouping (Prediction). TFIDF produces important words from a given data. This can be used for retrieving more similar documents and for semantic search but word2vec gives even more important information which can be applied for document retrieval and semantic search applications as well.

Reduction to 10% data is taken as parameter to estimate the size of the vectors. 100 word2vec synonyms are extracted for 1000 top TFIDF words.

Word2Vec after Ngrams did not give reliable or any useful synonyms because of the less data. This can be validated using larger dataset.

Named Entity Recognition:

Named Entity Recognition can be said to as classifying words in the documents to pre-defined categories like persons, location, organization etc. Stanford NER is used for this project. It provides near state-of-art labeling techniques. To generate a custom NER model for the domain under consideration, pre tagged training file is needed where labels are already available. Stanford's NERClassifier at command-line can be used to generate a custom model. The generated model can be tested based on F1 scores. And this model can be used with help of Stanford's NER classes to tag input queries. And standard Stanford NER along with human tagged categories (domain specific like "Sports") is used to generate training data.

Example Training Data:

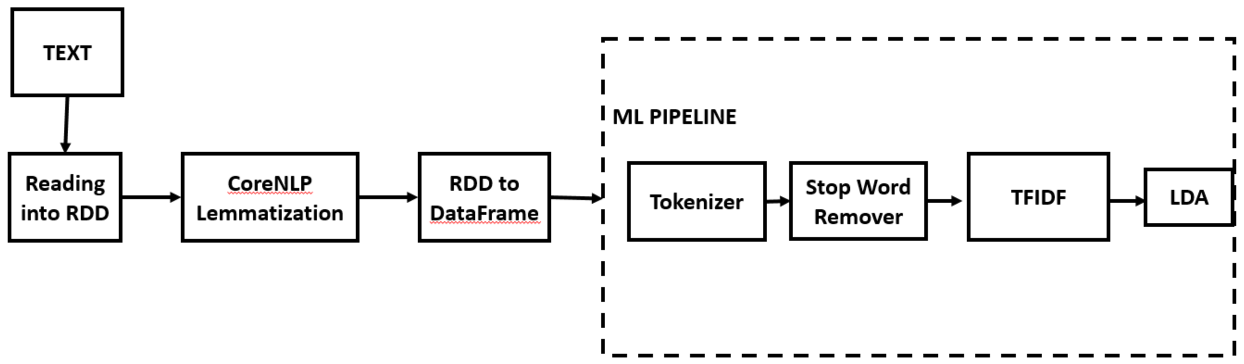
```
consist O
document    O
from   O
the    O
BBC    ORGANIZATION
Sport  O
website    O
correspond O
sport  O
news   O
article    O
five   NUMBER
```

Topic:

LDA (Latent Dirichlet Algorithm) is a statistical model used for topic modelling. It is one of the best available algorithms for topic modelling. It uses a probabilistic approach to identify various topics in a corpus or document. It can be thought of as a topic modelling algorithm. Spark Machine learning provides LDA in its machine learning library. We can get two types of vectors from LDA. Topic-word distribution and document-topic distribution but for text summarization needs, topic-word distribution is enough because we don't need to retrieve documents from corpora. Number of topics to be retrieved and number of words to be included in each topic depends on the vocabulary size and collection size.

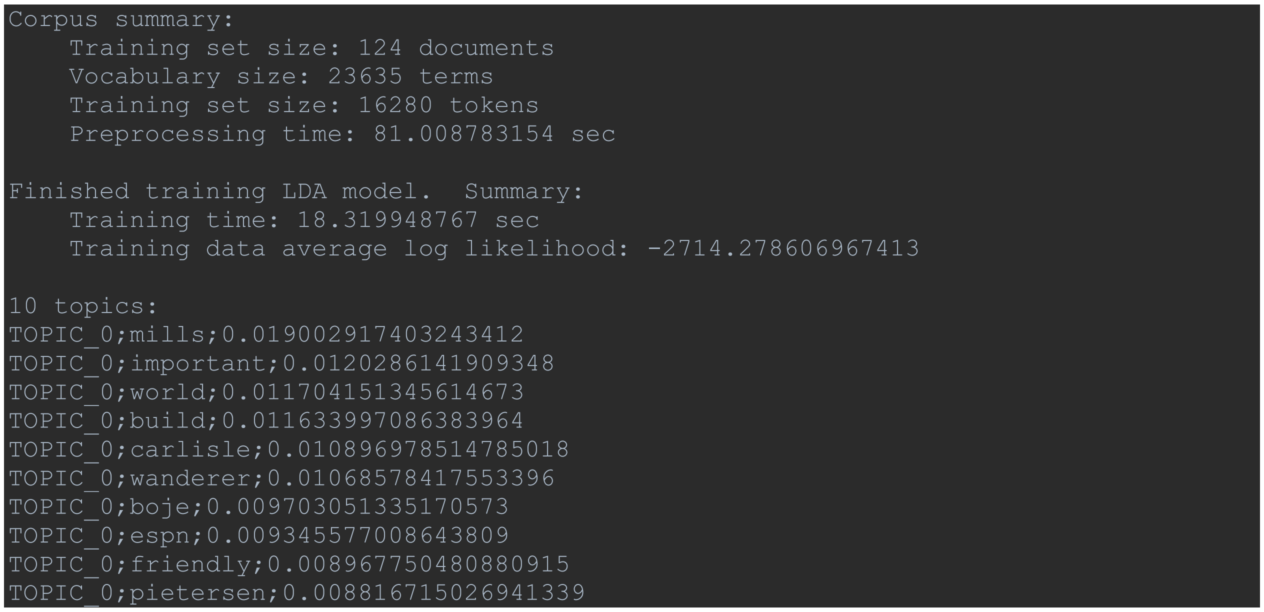
Number of topics=Number of documents / 10

Number of words in each vector=Vocabulary Size / (10 X Number of topics)



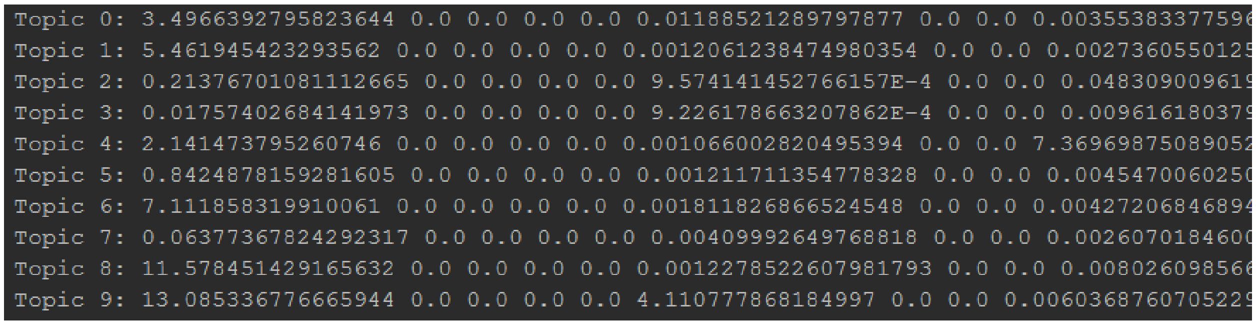
LDA model development using NLP->TFIDF->LDA:

Sample for Cricket Class:



```
Corpus summary:
    Training set size: 124 documents
    Vocabulary size: 23635 terms
    Training set size: 16280 tokens
    Preprocessing time: 81.008783154 sec

Finished training LDA model.  Summary:
    Training time: 18.319948767 sec
    Training data average log likelihood: -2714.278606967413

10 topics:
TOPIC_0;mills;0.019002917403243412
TOPIC_0;important;0.0120286141909348
TOPIC_0;world;0.011704151345614673
TOPIC_0;build;0.011633997086383964
TOPIC_0;carlisle;0.010896978514785018
TOPIC_0;wanderer;0.01068578417553396
TOPIC_0;boje;0.009703051335170573
TOPIC_0;espn;0.009345577008643809
TOPIC_0;friendly;0.008967750480880915
TOPIC_0;pietersen;0.008816715026941339
```

A Topic Word matrix is also generated:



```
Topic 0: 3.4966392795823644 0.0 0.0 0.0 0.0 0.01188521289797877 0.0 0.0 0.00355383377596
Topic 1: 5.461945423293562 0.0 0.0 0.0 0.0 0.0012061238474980354 0.0 0.0 0.0027360550125
Topic 2: 0.21376701081112665 0.0 0.0 0.0 0.0 9.574141452766157E-4 0.0 0.0 0.048309009619
Topic 3: 0.01757402684141973 0.0 0.0 0.0 0.0 9.226178663207862E-4 0.0 0.0 0.009616180379
Topic 4: 2.141473795260746 0.0 0.0 0.0 0.0 0.001066002820495394 0.0 0.0 7.36969875089052
Topic 5: 0.8424878159281605 0.0 0.0 0.0 0.0 0.001211711354778328 0.0 0.0 0.0045470060250
Topic 6: 7.111858319910061 0.0 0.0 0.0 0.0 0.001811826866524548 0.0 0.0 0.00427206846894
Topic 7: 0.06377367824292317 0.0 0.0 0.0 0.0 0.00409992649768818 0.0 0.0 0.0026070184600
Topic 8: 11.578451429165632 0.0 0.0 0.0 0.0 0.0012278522607981793 0.0 0.0 0.008026098566
Topic 9: 13.085336776665944 0.0 0.0 0.0 0.0 4.110777868184997 0.0 0.0 0.0060368760705229
```

WordNet Synonyms:

Word2Vec gives synonyms (or similar in context) for the input text corpus. But Wordnet produces similar items from the existing dictionary of corpus but not relevant to the corpus under consideration.

This may better suit for domain independent systems. But still applying WordNet synonyms for TFIDF and comparing them with Word2Vec results may yield good features for application under consideration.

Feature Vector:

Text Summarization requires to build a Feature Vector based on the existing corpus for the domain under consideration. This feature vector can be a matrix created from different features extracted using TFIDF, Relation Extraction, Named Entity Recognition, Word2Vec vector and LDA model for topic identification. Or the final feature vector can be a tuple of the mentioned vectors.

TFIDF:

Term frequency – Inverse Document frequency is a numerical statistical analysis which is intended to identify important words in a document in a collection of corpus. This is one of the simple ranking systems to identify important words that make up the particular class in collection. TFIDF is generally applied after tokenization, lemmatization, stop word removal etc.

Example input: “Hayden sets up Australia win”
Output for document1 (stripped):

```
(1048576,[45,65,66],[0.9364934391916745,2.3814551551518304,7.751767917624546])
```

So, we will have a term-score vector in which score shows the importance of the word to a particular class. Length of the required TFIDF vector can be approximated based on the size of the corpus or size of collection in terms of number of words.

Word2Vec Vector:

Sample Feature vector for Word2Vec:

```
(100000,[4907,6369,14444,20035,97395,129154,138356,140586,159152,181938,237276,299224,30090
9,329556,334532,347585,360210,409280,429905,457042,476413,529669,579456,598759,612143,6121
87,621525,694261,710299,734459,787968,792248,842071,850056,859041,879256,884388,918666,924
359,926926,971660,978074,978809,1001982,1035612],[1.0986122886681098,1.0986122886681098,1.
0986122886681098,1.0986122886681098,1.0986122886681098,2.1972245773362196,1.09861228866
81098,1.0986122886681098,1.0986122886681098,1.0986122886681098,1.0986122886681098,1.0986
122886681098,0.6931471805599453,1.0986122886681098,1.0986122886681098,1.098612288668109
8,0.6931471805599453,1.0986122886681098,0.6931471805599453,1.0986122886681098,1.09861228
86681098,1.0986122886681098,1.0986122886681098,1.0986122886681098,1.0986122886681098,1.0
986122886681098,1.0986122886681098,1.0986122886681098,1.0986122886681098,1.09861228868
1098,1.6218604324326575,1.0986122886681098,1.0986122886681098,1.0986122886681098,1.38629
43611198906,1.0986122886681098,1.0986122886681098,1.0986122886681098,0.6931471805599453,
1.0986122886681098,1.0986122886681098,1.0986122886681098,0.6931471805599453,1.0986122886
681098,0.0])
```

Relation:

Information extraction refers to the extraction of structured relation triples from plain text. Stanford provides a great relation extraction software. OpenIE gives structured triplets in the form of (Object, relation, Subject). The system first splits each sentence into a set of entailed clauses. Each clause is then maximally shortened, producing a set of entailed shorter sentence fragments. These fragments are then segmented into OpenIE triples.

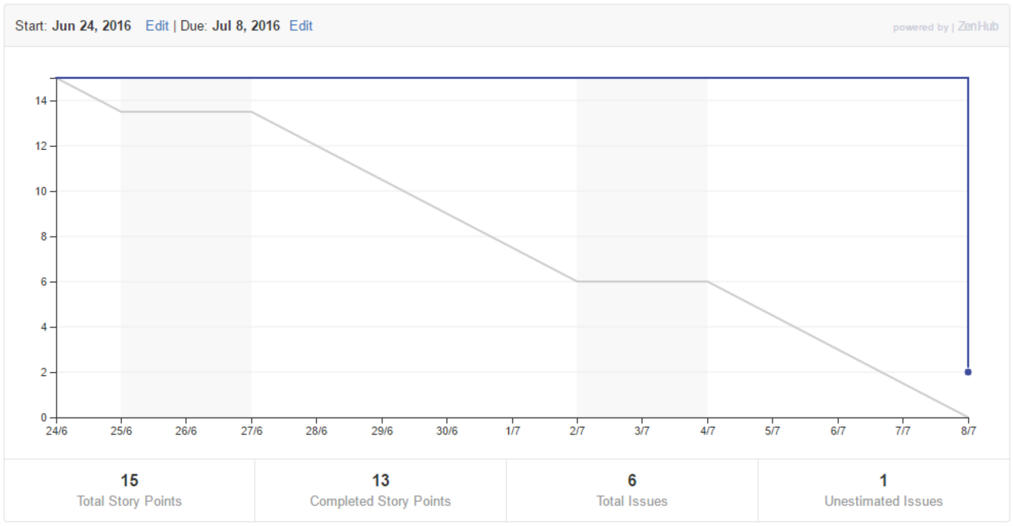
For e.g. (Systems, based on, hand-written rules)-extracted clause → (Systems, based on, rules)

Generating Triplets for the other feature vectors can be helpful in designing ontology for the corpus. But this would also be helpful after clustering the data based on algorithms to make use of it efficiently. Generated triplets can be used for building ontology. Here local class ontologies can be built using this feature.

Developed feature vector is sent to classification or clustering algorithms – Self Organized Maps and K-means. This reduces the data dimensionally which helps in classifying incoming queries and building semantic or ontology model for the corpus. Building a Feature Matrix in terms of type of feature as row and particular vector as column would be correct choice.

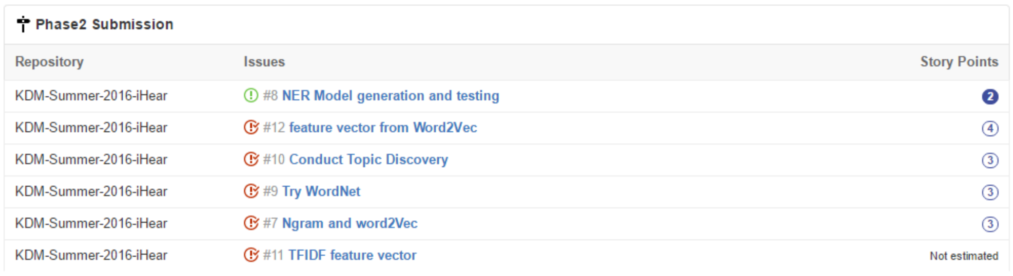
Project Management:

Burndown Chart:


Start: **Jun 24, 2016** Edit | Due: **Jul 8, 2016** Edit

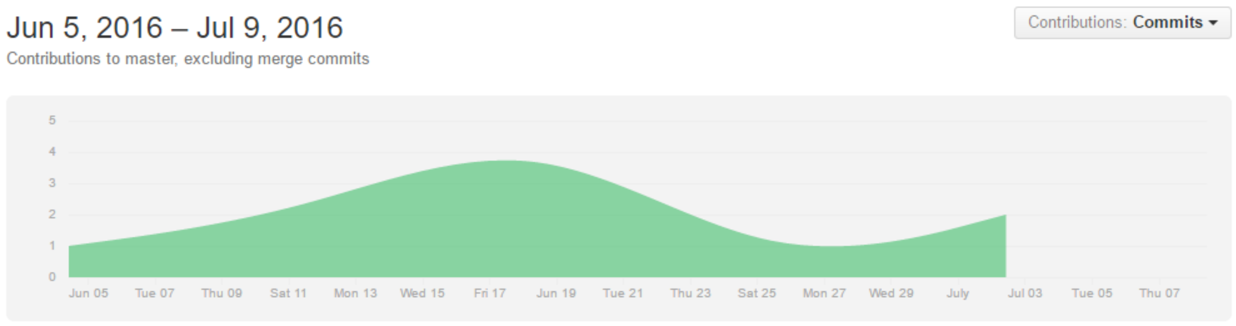
powered by | ZenHub

15	13	6	1
Total Story Points	Completed Story Points	Total Issues	Unestimated Issues

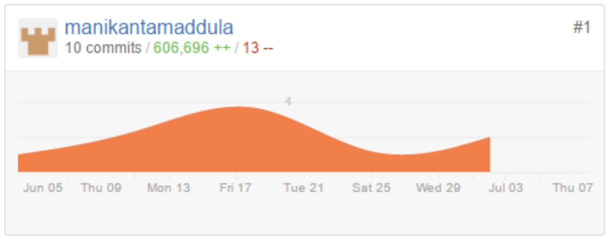

Phase2 Submission

Repository	Issues	Story Points
KDM-Summer-2016-iHear	#8 **NER Model generation and testing**	2
KDM-Summer-2016-iHear	#12 **feature vector from Word2Vec**	4
KDM-Summer-2016-iHear	#10 **Conduct Topic Discovery**	3
KDM-Summer-2016-iHear	#9 **Try WordNet**	3
KDM-Summer-2016-iHear	#7 **Ngram and word2Vec**	3
KDM-Summer-2016-iHear	#11 **TFIDF feature vector**	Not estimated

Contributions:


Jun 5, 2016 – Jul 9, 2016

Contributions to master, excluding merge commits


manikantamaddula #1

10 commits / 606,696 ++ / 13 --

Concerns/Issues:

1. Need to collect more datasets. Integrating datasets.
2. Need to figure out working with stream text data for querying server on summarization.
3. Need to know if feature extraction can be done in mobile client instead of server.
4. Can querying of model for summarization be implemented in mobile client.
5. Machine learning for categorization and classification. Self-Organized Maps.
6. Meta level clustering using SOM
7. Automatic generation of Ontology

Future Work:

Next phase of the project should complete until building ontology model from Feature Vectors. This involves applying machine learning algorithms to build model and learning about OWL Ontology.

Next phase also implements summarization from the model and audio data in application.

Project Phase 3

Dataset:

1. BBC Sport website corresponding to sports news in five topical areas from 2004-2005 with 737 documents and 5 natural classes. Source: http://mlg.ucd.ie/datasets/bbc.html
2. 20 Newsgroups data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. Source: http://qwone.com/~jason/20Newsgroups/
3. Wikipedia dataset for 4 classes is collected (Cricket, Rugby, Tennis, Football) and parsed using python program
   https://petscan.wmflabs.org/
   https://en.wikipedia.org/wiki/Special:Export

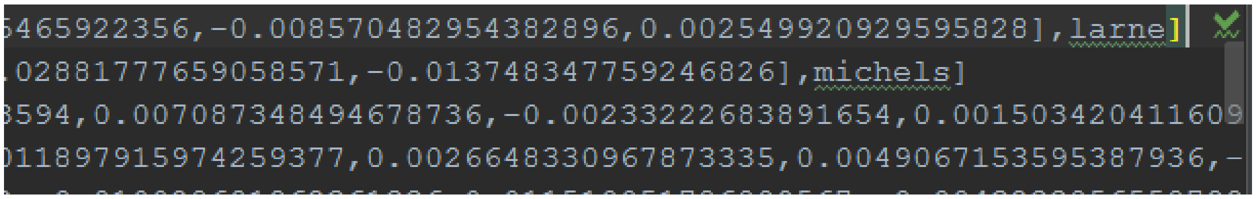
Workflow:

The application is intended to use sigspace model for Machine Learning. Sig Space, that was designed for a class-level incremental learning while supporting distributed learning and fuzzy matching. It is also scalable since it uses signatures to do machine learning where signatures will be about only 1% of the raw data input to sig space. Sig Space is tested on image and audio. This would be first implementation of Sig space for text mining.

	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			✓

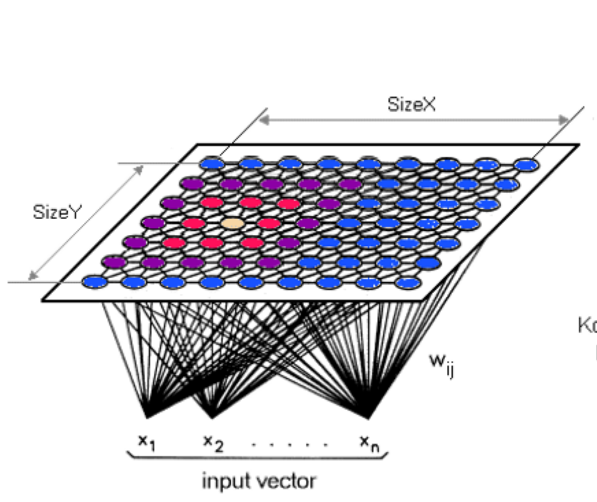
But in text mining global code book is needed with apache spark implementation. So, a Global SOM(Self Organizing Maps) approach is used to generate features which makes it code book dependent.

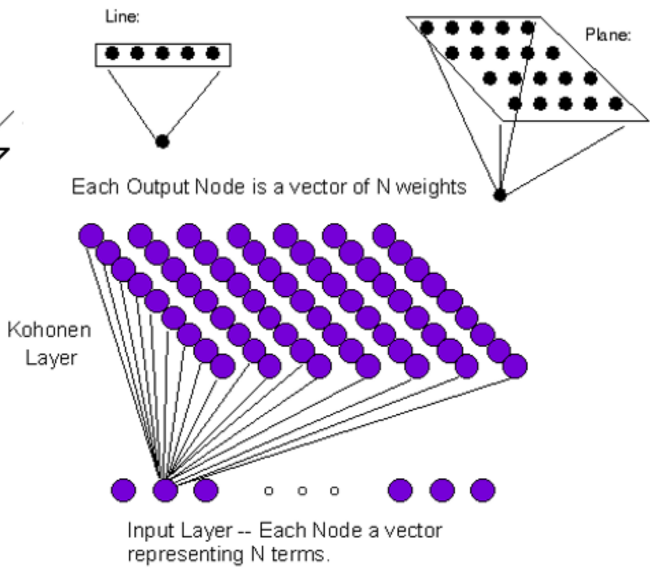
Word2Vec Feature Vector:
Word2vec vectors are given to Self-Organizing Maps for data reduction. Each vector is of 100 dimensions as input to the SOM.
Example input to SOM:



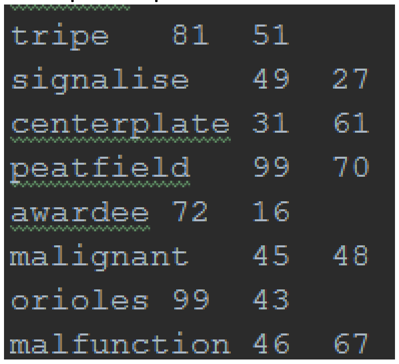
```
5465922356,-0.008570482954382896,0.002549920929595828],larne]
.0288177659058571,-0.013748347759246826],michels]
3594,0.007087348494678736,-0.0023322268389165 4,0.00150342041160 9
0118979159742593 77,0.0026648330967873335,0.004906715359538 7936,-
```

SOM (Self-Organizing Map): (Unsupervised learning)
A SOM is a special type of neural network that classifies data. Typically, a SOM will map higher resolution data to a single or multidimensional output. This can help a neural network see the similarities among its input data. Dr. Teuvo Kohonen of the Academy of Finland created the SOM. Because of this, the SOM is sometimes called a Kohonen neural network.




Line:
Plane:
Each Output Node is a vector of N weights
Kohonen Layer
Input Layer -- Each Node a vector representing N terms.

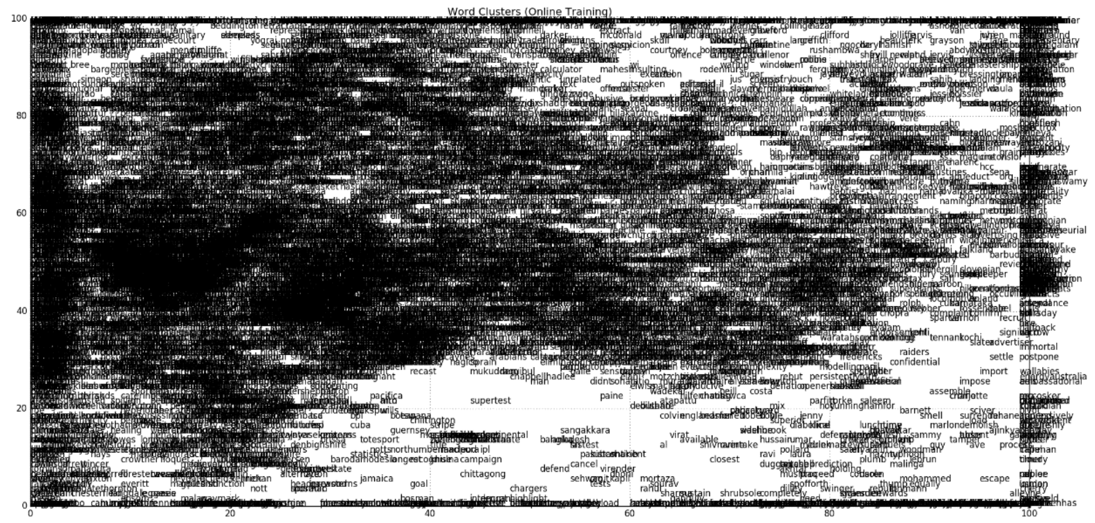
SOM will output two-dimension data for this application. So, we can visualize the data to see the clusters that are formed.
Example output:



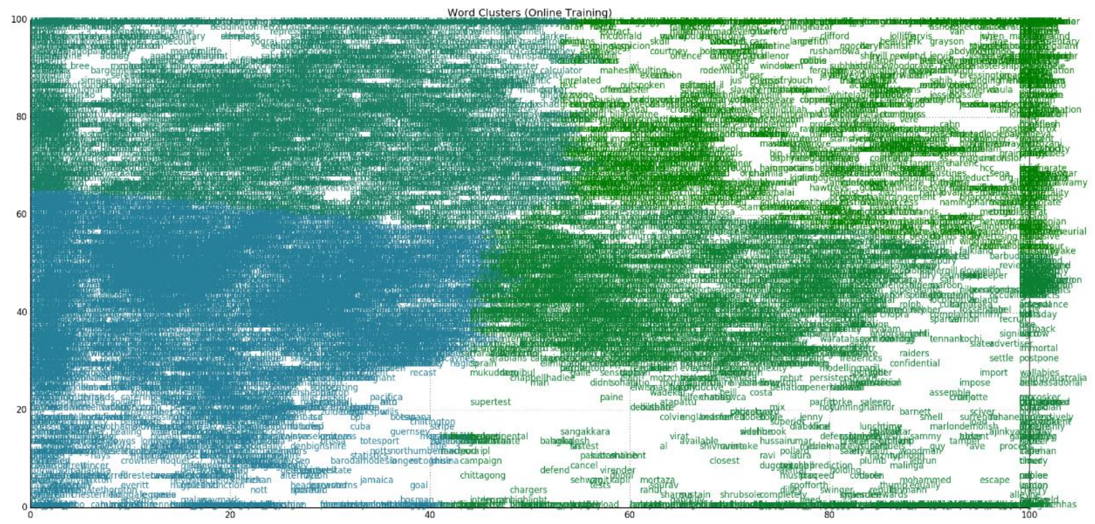
```
tripe   81   51
signalise    49   27
centerplate 31   61
peatfield    99   70
awardee 72   16
malignant    45   48
orioles 99   43
malfunction 46   67
```

Lot of data reduction happens due to this dimension reduction. Some accuracy is lost at this stage but SOM groups the closest neurons from input to closest neurons or same neuron in output. So, the contextual information is not lost much. kMeans is used to cluster this dimensionally reduced data.

SOM Dimension Reduction:
10000 output neurons are there in output. matplotlib.pyplot is used to visualize the data.



Data Visualization after clustering data using kMeans:

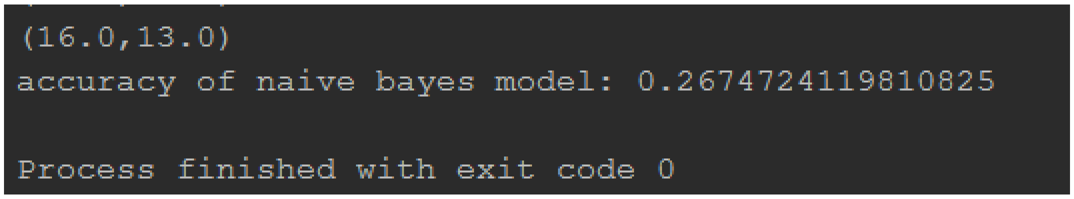


Naïve Bayes Classification:

Naive Bayes is a simple multiclass classification algorithm with the assumption of independence between every pair of features. Naive Bayes can be trained very efficiently. Within a single pass to the training data, it computes the conditional probability distribution of each feature given label, and then it applies Bayes' theorem to compute the conditional probability distribution of label given an observation and use it for prediction.
With the cluster numbers from kMeans as labels to data, Naïve Bayes classification is performed.

Accuracy:

The Dataset is randomly split into 80% for training and 20% for testing and for Naïve Bayes classification on signature data produced the following accuracy:



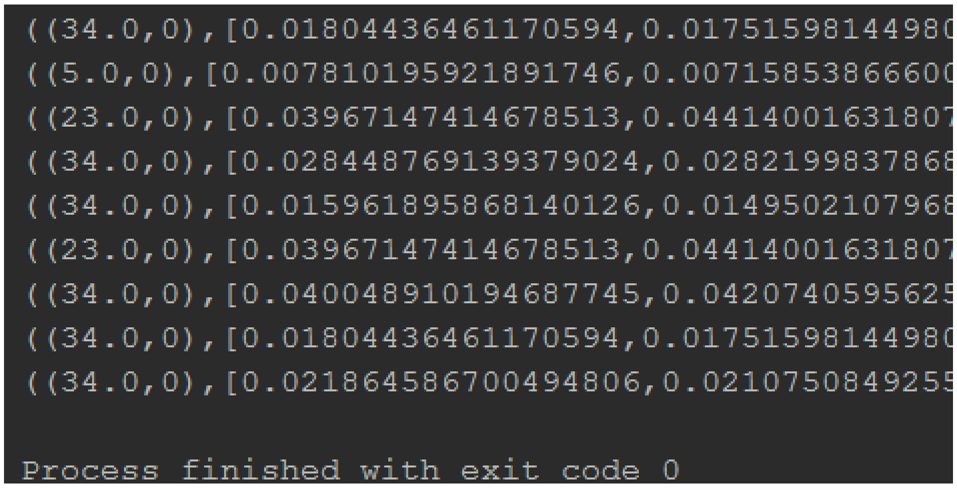
```
(16.0,13.0)
accuracy of naive bayes model: 0.2674724119810825

Process finished with exit code 0
```

Testing Stage:
In the testing phase, same workflow is followed. But for Word2vec, SOM, kMeans, Naïve Bayes, existing models from training stage is used to classify the input data.

Text Summarization:

Classification and predicted probabilities are generated for each sentence in the input. Sentences fall into different topics inside each class. More number of classes (predicted) decide the class to be considered. After that a threshold of predicted probability is used to summarize the data. Summarization effectiveness may be small because of naïve approach in reducing sentences. Same analysis can be performed at word level instead of sentence level to get good summarization.

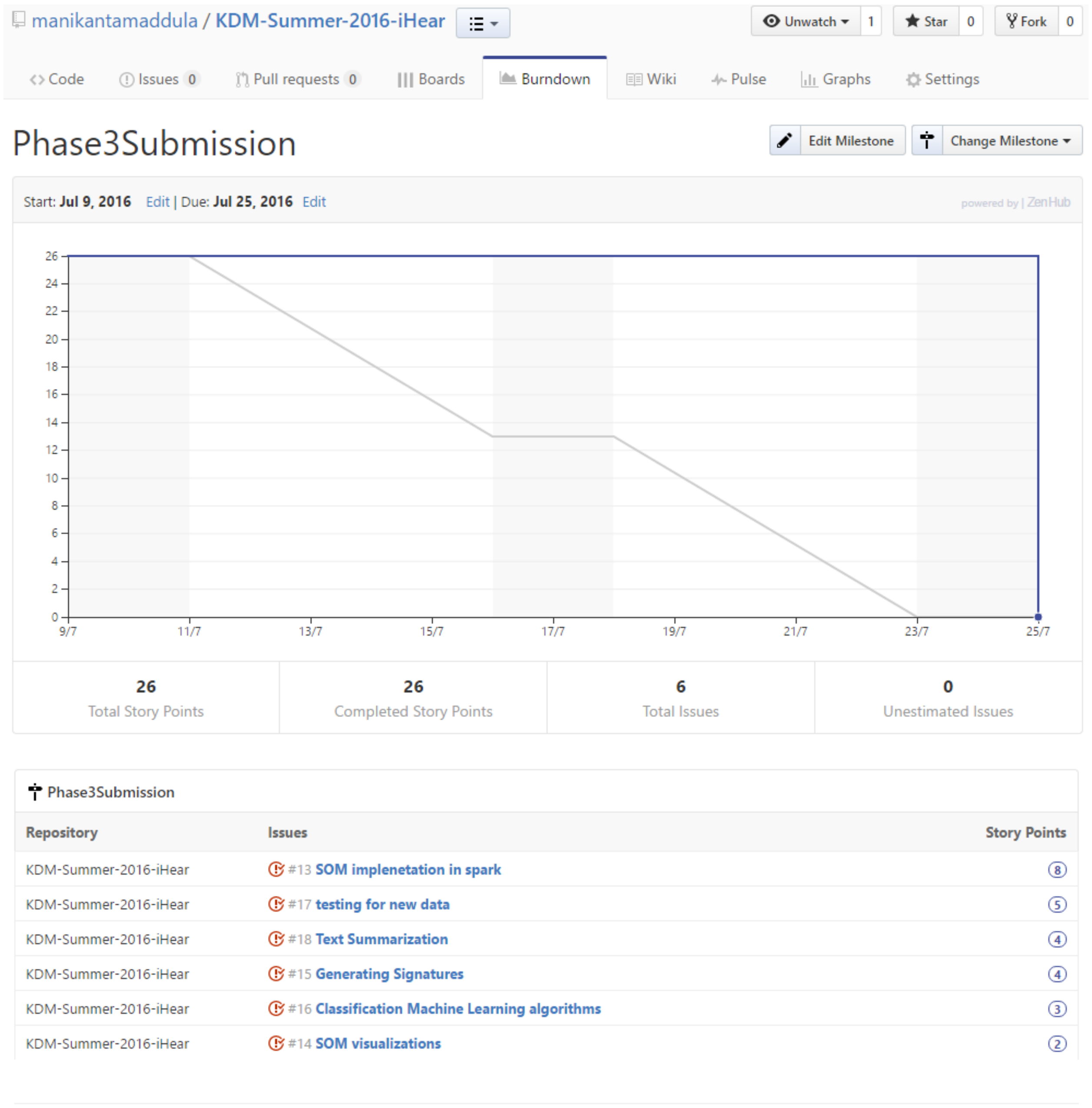
Sample output:



```
((34.0,0),[0.01804436461170594,0.0175159814498
((5.0,0),[0.007810195921891746,0.00715853866600
((23.0,0),[0.03967147414678513,0.044140016318O7
((34.0,0),[0.028448769139379024,0.0282199837868
((34.0,0),[0.015961895868140126,0.0149502107968
((23.0,0),[0.03967147414678513,0.044140016318O7
((34.0,0),[0.040048910194687745,0.0420740595625
((34.0,0),[0.01804436461170594,0.0175159814498
((34.0,0),[0.021864586700494806,0.0210750849255

Process finished with exit code 0
```

Project Management:

Burndown Chart:



Phase3Submission

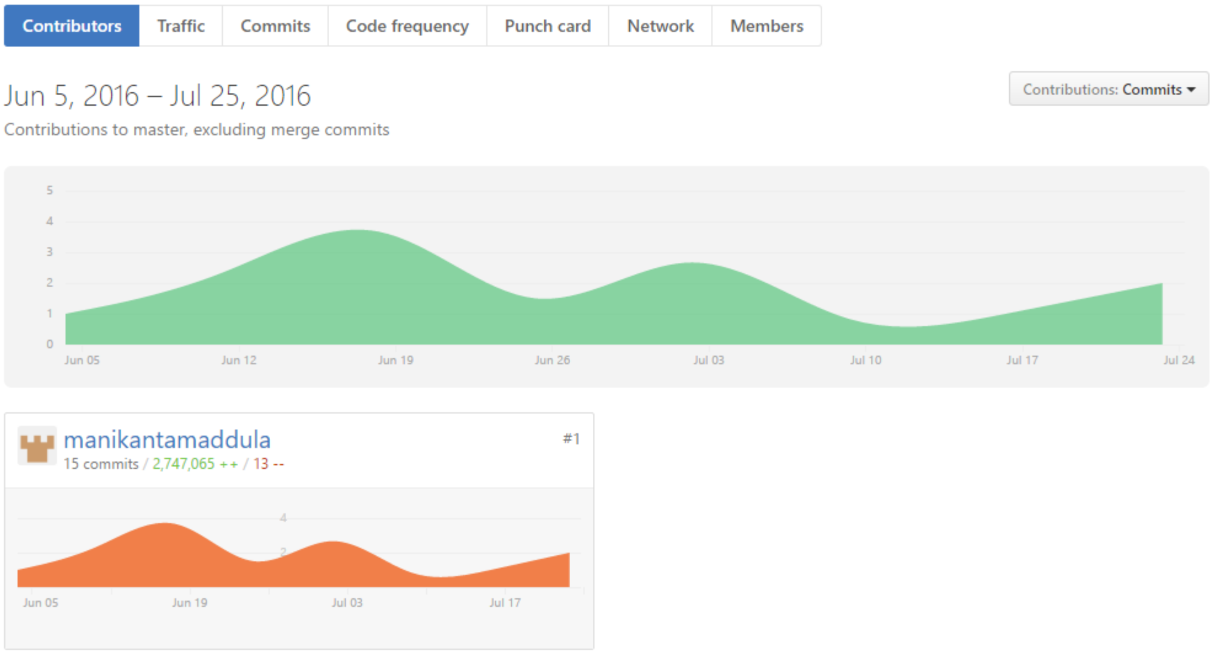
Start: **Jul 9, 2016** Edit | Due: **Jul 25, 2016** Edit

26	**26**	**6**	**0**
Total Story Points	Completed Story Points	Total Issues	Unestimated Issues

Phase3Submission

Repository	Issues	Story Points
KDM-Summer-2016-iHear	#13 **SOM implenetation in spark**	8
KDM-Summer-2016-iHear	#17 **testing for new data**	5
KDM-Summer-2016-iHear	#18 **Text Summarization**	4
KDM-Summer-2016-iHear	#15 **Generating Signatures**	4
KDM-Summer-2016-iHear	#16 **Classification Machine Learning algorithms**	3
KDM-Summer-2016-iHear	#14 **SOM visualizations**	2

Contributions:



Jun 5, 2016 – Jul 25, 2016

Contributions to master, excluding merge commits

manikantamaddula

15 commits / 2,747,065 ++ / 13 --