Project Title: Sports Video Annotation and Summarization using SigSpace

Team # 9

Members: Manikanta Maddula

Motivation:

The implementations and applications based on Big data tools especially real time applications are increasing day by day. Auto annotation, summarization, content analysis etc. from live video are interestingly growing in real time video analysis. Auto news article making, assisting for physically challenged people, task recommendations etc. can be said as applications of the video analysis. Combining video and audio analysis will have even more great applications which will be useful but it will be out of scope for this project. Annotation and summarization will be first step towards a better solution. Content analysis has wide range of applications in surveillance domain.

Objectives:

1. Categorize video into predefined sports categories
2. Identify specific format in that game
3. Object recognition in each video (People, places etc.).
4. Background and Foreground objects identification.
5. Real time analysis for all of the above. Fast response of within mille seconds.

System:

The system architecture will make use of apache Spark, Apache Storm, Apache Kafka, and video transferring client (android mobile or API). For the training purpose of model, spark's machine learning library is used. And to extract features from the video or image frames in video OpenImaj library is used. SIFT implementation is OpenImaj is used to achieve different objectives. Apache Storm is used to achieve parallel task distribution for real time data. Apache Kafka is used to communicate between different modules like Android and Storm, spark and storm, storm and android client. For a live video analysis, video stream data is sent to Strom using Apache Kafka as a Broker in between. Features of video stream is extracted using OpenImaj library. Using the model developed in spark machine learning library and with the extracted features, categorization, identification etc. are executed. Using Apache Kafka relevant information is sent to client application (e.g. Android mobile). SigSpace architecture can be used for the model training purpose. SigSpace architecture has advantages of space and time reduction, incremental learning, fuzzy matching, dimension reduction, distributed learning etc. YouTube videos of Olympics 2016 at Rio can be used for training purpose. For audio analysis, Android or web API's can be used to convert Speech to text and this text can be used but this application is trying to analyze significant events in audio data.

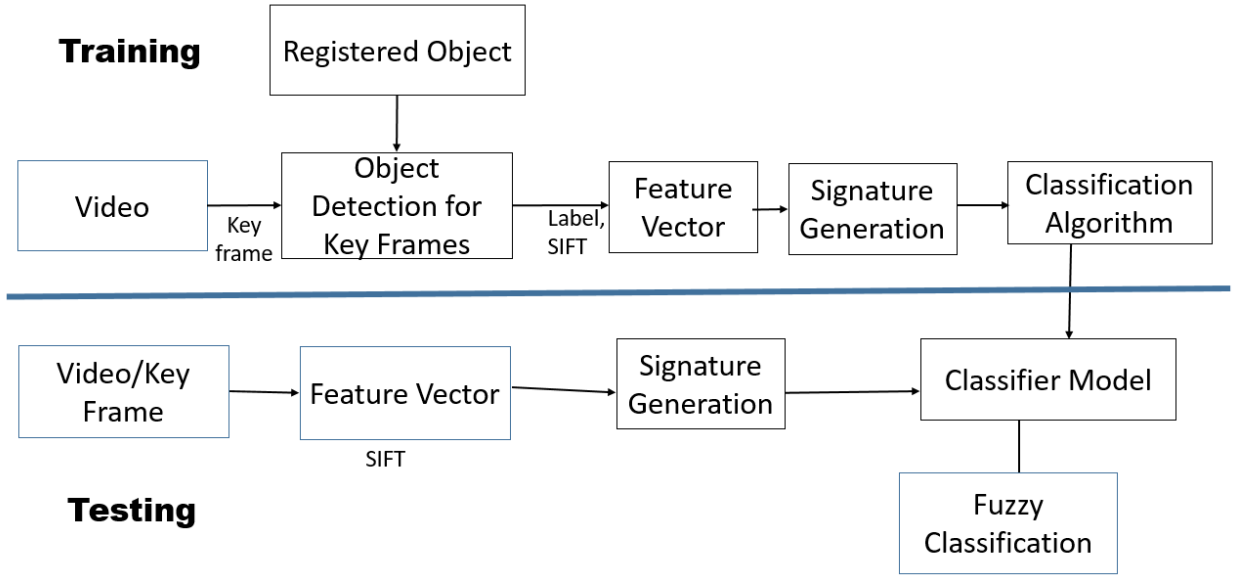
Update:

Based on suggestion from Dr. Yugyung Lee, goal is to work on one sport and focus on analyzing on it to identify different aspects – players (teams may be), movement of ball or stick that's used in sport, crowd identification, game statistics like who won the game or identifying when the game is over etc.

Significance and Uniqueness:

There are several related projects in the domain of real time video analysis and sports. But this kind of architecture is never implemented at UMKC – CSEE. And usage of SigSpace is first to implement in Real time and video analysis. Sports domain is a billion-dollar industry which will have huge impact if application can do some useful analysis with the available data.

Work Flow:



Training:

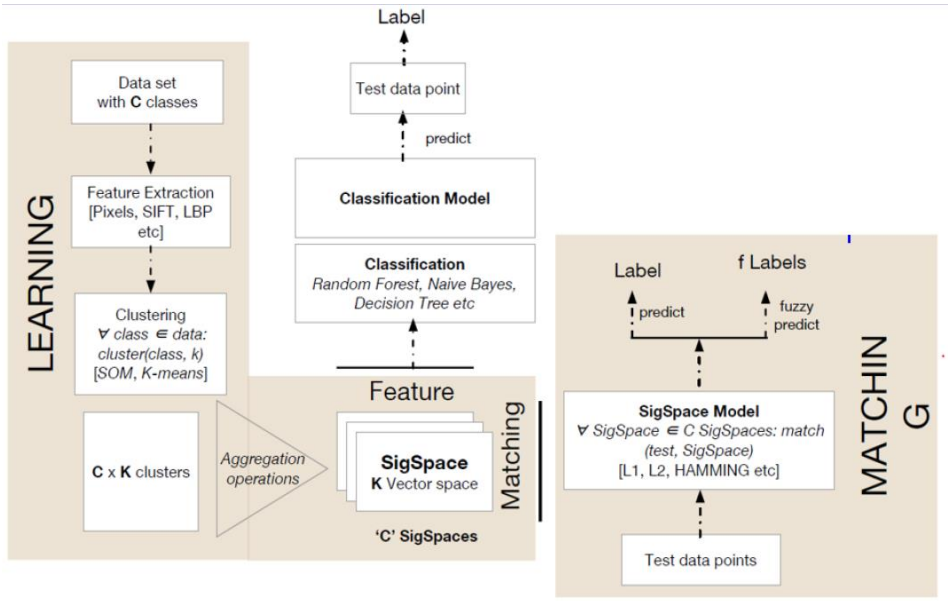
Object's that needs to be identified like person, ball, etc. are registered first. Then from video key frames are detected using OpenImaj or OpenCV libraries. SIFT features are extracted from each image. Signatures are generated from the SIFT features using SigSpace framework. Then these signatures for each class or each object are input to classification algorithms in Apache Spark Machine Learning Library. At the stage the generated model is saved for test input analysis.

Testing:

For a new input video (real time or file), SIFT feature vectors are extracted for each frame or image. Then signature is generated for each frame and signatures are input to classification model to classify the image. Here fuzzy classification can be used to generate confidence score for classification. Here different objects can be identified in video like player or person, ball, fences etc.

SigSpace:

SigSpace is a unique feature representation. In SigSpace, a class-based model was built by an evaluation and extension of existing machine learning models, i.e., K-means and Self-Organizing Maps (SOM). The Machine learning with SigSpace is modeled as a feature set with standard machine learning algorithms like Random Forests, Decision Tree etc., and a class model using L1 (Manhattan distance) and L2 (Euclidean distance) norms.



SigSpace Architecture (Image Clustering)

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. 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.

SIFT feature:

Features are the base for any Machine Learning task, good features should have four characteristics: discrimination, reliability, independence and be in small numbers. Models are generated based on the features given to the algorithm. We have worked on four kinds of features, three belongs to the image domain and one from an audio domain.

SIFT, which stands for Scale Invariant Feature Transform, is one of most popular feature extraction and description algorithms. Point features are very popular in many fields including 3D reconstruction and image registration. A good point feature should be invariant to geometrical transformation and illumination. A point feature can be a blob or a corner. SIFT extracts blob like feature points and describe them with a scale, illumination, and rotational invariant descriptor.

Unlike other histogram descriptors, SIFT algorithm does not give an overall impression of the image. Instead, it detects blob like features from the image and describes each and every point with a descriptor that contains 128 numbers. Point descriptors are given as output in the form of an array. Figure 3 shows accordion and airplane images and the SIFT keypoints detected on those. The radius of the circle on an image represents the strength of the keypoint.

Sample SIFT features generated:


```
C:/Users/Manikanta/Documents/Datasets/Caltech_101/101_ObjectCategories/accordion/image_0031.jpg,292.952270508,8913407,2.03150200844,28.0 52.0 54.0
21.0 5.0 3.0 5.0 18.0 16.0 11.0 23.0 45.0 37.0 12.0 30.0 41.0 16.0 1.0 1.0 2.0 12.0 16.0 59.0 56.0 0.0 0.0 0.0 0.0 10.0 13.0 15.0 1.0 61.0 37.0 20.0
20.0 27.0 12.0 3.0 4.0 128.0 26.0 5.0 5.0 4.0 15.0 104.0 128.0 17.0 4.0 4.0 23.0 51.0 116.0 125.0 40.0 1.0 5.0 3.0 5.0 11.0 18.0 45.0 11.0 92.0 26.0
4.0 36.0 78.0 10.0 1.0 4.0 128.0 128.0 85.0 43.0 5.0 5.0 8.0 28.0 18.0 37.0 121.0 128.0 59.0 29.0 11.0 7.0 25.0 57.0 20.0 26.0 9.0 2.0 3.0 2.0 14.0
21.0 6.0 18.0 22.0 0.0 0.0 1.0 24.0 113.0 84.0 17.0 0.0 0.0 0.0 1.0 10.0 45.0 111.0 53.0 0.0 0.0 0.0 0.0 44.0 90.0 14.0 12.0 6.0 0.0 0.0 0.0
C:/Users/Manikanta/Documents/Datasets/Caltech_101/101_ObjectCategories/accordion/image_0031.jpg,139.618850708,10617343,2.07910895348,75.0 19.0 1.0 0.0
1.0 3.0 4.0 2.0 191.0 91.0 0.0 0.0 0.0 0.0 0.0 0.0 79.0 23.0 0.0 0.0 0.0 0.0 0.0 1.0 7.0 2.0 0.0 0.0 0.0 0.0 0.0 2.0 73.0 11.0 4.0 1.0 0.0 1.0 0.0 5.0
191.0 49.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 152.0 11.0 0.0 0.0 0.0 0.0 0.0 0.0 7.0 10.0 2.0 0.0 0.0 0.0 0.0 0.0 0.0 2.0 70.0 3.0 0.0 0.0 0.0 0.0 0.0 0.0 13.0 191.0 21.0
0.0 0.0 0.0 0.0 0.0 14.0 181.0 12.0 1.0 0.0 0.0 0.0 0.0 3.0 3.0 3.0 3.0 0.0 0.0 0.0 0.0 2.0 50.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 17.0 191.0 5.0 0.0 0.0 0.0
0.0 0.0 17.0 149.0 4.0 1.0 0.0 0.0 0.0 0.0 0.0 3.0 1.0 1.0 3.0 1.0 0.0 0.0 1.0 0.0
C:/Users/Manikanta/Documents/Datasets/Caltech_101/101_ObjectCategories/accordion/image_0031.jpg,295.139892578,5964287,1.95088863373,84.0 57.0 0.0 0.0
4.0 29.0 12.0 8.0 155.0 134.0 0.0 0.0 0.0 3.0 4.0 9.0 35.0 8.0 0.0 0.0 0.0 25.0 58.0 18.0 1.0 0.0 1.0 2.0 6.0 61.0 81.0 6.0 48.0 7.0 0.0 0.0 43.0 52.0
7.0 15.0 155.0 32.0 0.0 0.0 0.0 0.0 1.0 3.0 77.0 90.0 4.0 0.0 0.0 0.0 0.0 15.0 49.0 52.0 4.0 0.0 0.0 0.0 0.0 16.0 30.0 38.0 8.0 34.0 6.0 0.0 1.0 88.0 34.0 3.0 8.0
155.0 72.0 0.0 0.0 0.0 0.0 0.0 1.0 29.0 155.0 24.0 0.0 0.0 0.0 0.0 9.0 35.0 25.0 1.0 0.0 0.0 0.0 0.0 1.0 33.0 62.0 3.0 31.0 2.0 0.0 0.0 0.0 107.0 33.0 0.0 4.0 155.0
7.0 0.0 0.0 2.0 1.0 0.0 64.0 155.0 7.0 2.0 0.0 0.0 0.0 0.0 2.0 31.0 8.0 7.0 1.0 0.0 0.0 1.0 4.0 11.0 4.0
```


Sample Signatures Generated:


```
37 11 6 12 42 10 4 8 136 25 4 6 25 7 3 22 36 7 4 23 122 22 3 8 29 11 6 11 41 10 4 9 47 10 5 15 56 13 4 10 148 19 2 6 33 9 3 28 48 5 3 23 140 35 3 10
40 13 6 12 49 12 6 14 48 8 2 11 58 16 5 10 148 24 1 7 35 6 2 18 51 9 2 29 142 28 2 6 43 14 6 9 49 15 6 16 38 7 2 10 46 16 5 9 142 20 2 6 28 6 2 25 38
7 3 21 129 24 3 7 31 10 5 9 41 12 5 12
38 10 6 18 69 15 5 4 162 38 3 3 10 2 2 23 96 19 6 6 9 6 6 18 13 8 6 10 27 11 7 5 45 9 6 24 93 23 4 5 167 26 1 3 12 3 1 33 121 14 4 8 14 8 8 26 18 8 5
10 32 14 8 8 45 7 5 22 89 26 4 6 167 34 1 3 11 4 1 23 125 23 7 8 12 7 6 14 17 9 7 11 30 12 7 8 37 6 4 14 65 24 5 8 165 23 1 2 9 4 2 36 97 19 7 5 8 4 4
16 12 7 8 10 25 8 5 4
29 28 21 18 25 23 16 15 51 47 22 16 28 29 19 22 41 31 14 15 52 55 22 21 26 22 9 16 56 49 11 8 38 38 22 17 26 30 23 22 46 54 35 30 47 43 19 14 99 64 22
14 41 43 16 29 24 17 7 19 89 81 14 9 45 37 16 12 25 32 26 24 47 26 13 15 47 70 44 30 108 50 13 12 31 35 22 46 23 19 11 30 96 69 11 8 33 30 15 10 25 29
16 14 43 42 21 15 32 33 20 19 46 50 24 20 37 31 17 19 23 27 18 29 56 34 10 9
37 23 22 19 22 33 48 39 38 36 35 29 42 42 42 30 39 36 45 39 38 30 33 30 27 21 19 18 30 39 49 31 35 44 63 49 39 30 27 20 84 44 44 35 36 34 50 57 42 30
28 32 58 72 64 37 40 45 53 39 35 26 27 27 36 24 26 24 41 57 61 40 79 64 56 32 41 38 39 38 45 45 67 60 56 37 32 31 35 26 26 24 37 48 54 40 34 40 52 31
24 20 19 20 39 28 35 32 42 39 43 35 52 37 34 25 34 38 40 35 26 30 47 39 34 24 20 19
14 8 9 11 20 8 6 6 72 19 8 7 12 6 7 16 127 21 4 5 29 10 8 38 20 5 4 23 115 27 6 8 17 10 9 11 24 10 8 8 93 25 12 10 16 8 7 15 143 33 4 8 39 10 5 30 24
5 3 31 141 32 3 6 17 9 8 10 23 10 10 10 94 15 7 8 16 11 12 26 142 27 4 11 40 8 4 34 24 6 4 36 141 28 2 5 14 7 6 7 20 10 8 8 74 17 6 5 12 7 8 20 127 34
```


Datasets:

Video data can be downloaded from youtube.com for any sports which we are interested in doing video annotation and analysis. For now, I am working on Sigspace implementation.

Data for sigspace implementation:

Type	Categories	OverallSize	Count	References
UEC Food	256	4.3GB	31,907	http://foodcam.mobi/dataset256.html
Object Categories	257	1.2GB	30,867	http://resolver.caltech.edu/CaltechAUTHORS:CNS-TR-2007-001
Caltech 101 Object Categories	101	122MB	8,778	https://www.vision.caltech.edu/Image_Datasets/Caltech101/

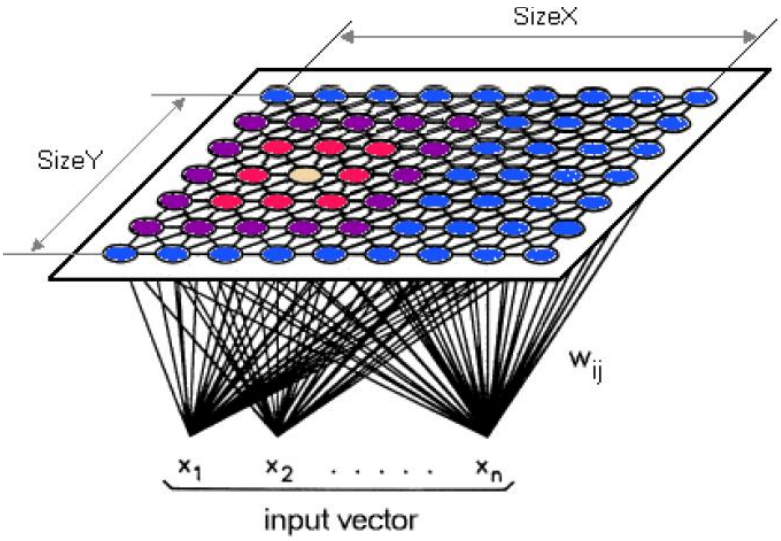
Algorithms:

For the workflow we require several machine learning algorithms: Self Organizing Maps and k-Means for signature generation. And Naïve Bayes, Random forest and Decision Tree for classification model. Analysis on EM algorithm is yet to done for signature generation.

Self-Organizing Maps:

A self-organizing map (SOM) 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 learning as opposed to error-correction learning (such as backpropagation 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 Figure for sample SOM explanation.

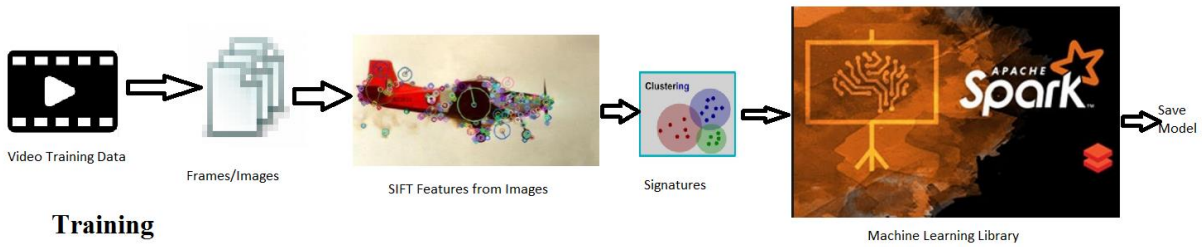


Example SOM Map representation

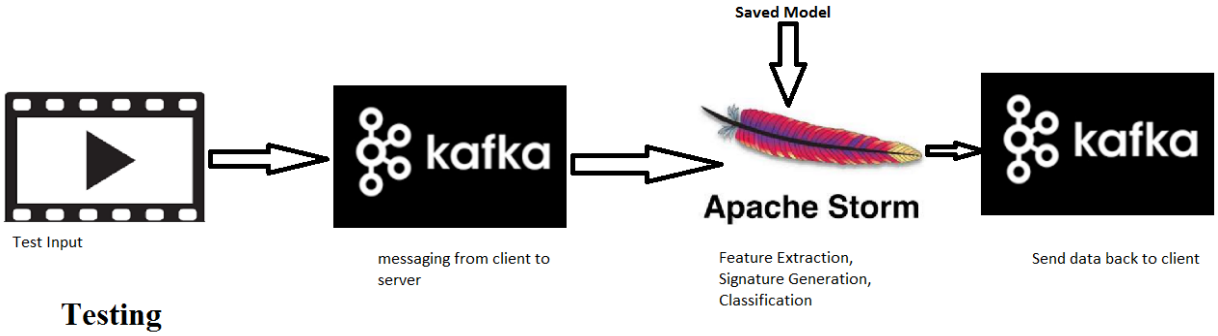
Instead of SOM we can use k-Means or EM (Expectation Maximization) algorithms to generate the signatures for individual classes since all of them are clustering algorithms.

Naïve Bayes, Decision Tree and Random Forest are used as classification models. Apache Spark machine learning library provides an implementation of all these.

Implementation:



Training Stage of work flow is implemented mostly in apache spark. Frames are extracted from video data and signatures are generated from SIFT features in Apache Spark. SIFT features are extracted from images using OpenCV or OpenImaj libraries. Classification models are build using Apache Spark machine learning libraries and models are saved at this stage.



Testing

For test scenario or for new documents, video data is sent to Apache storm server using apache kafka. Apache Kafka is the publisher/ subscriber messaging system. Real time processing is done in apache Storm. Here features extraction, signature generation, classification. And the results from storm are sent to client using Apache Kafka.

Project Management:

Work Completed:

Sigspace implementation:

Feature Extraction from Image data using SIFT and Signature generation from features using SOM and k-Means.

Time Taken: 40 hours approximately

Work to be completed:

SigSpace - classification implementation. Image extraction from video. Training workflow for video data. Test case workflow implementation in Kafka, Storm.

Project Phase -2

Implementation:

Signatures:

So far signatures are generated using Self Organizing-Maps and kMeans. Now I implemented signature generation using Gaussian Mixture Model available in Apache spark MLLib. Spark's GMM uses Expectation maximization to calculate prior probability distribution to classes.

A Gaussian Mixture Model represents a composite distribution whereby points are drawn from one of *k* Gaussian sub-distributions, each with its own probability. The `spark.mllib` implementation uses the expectation-maximization algorithm to induce the maximum-likelihood model given a set of samples. The implementation has the following parameters:

- *k* is the number of desired clusters.
- *convergenceTol* is the maximum change in log-likelihood at which we consider convergence achieved.
- *maxIterations* is the maximum number of iterations to perform without reaching convergence.
- *initialModel* is an optional starting point from which to start the EM algorithm. If this parameter is omitted, a random starting point will be constructed from the data.

Screen Shots of code Implementation:

```
def EMSignature(sc: SparkContext, globalNbIter: Int, exp: Experience): Unit = {
  // Read data from file
  println("\n***** READ : " + exp.name)
  //val datas: RDD[LabelVector] = SparkReader.parseSIFT(sc, exp.dataPath).cache()

  val datas = SparkReader.parseForKMeans(sc, exp.dataPath).cache()

  /**
    * Calculate the no of clusters based on the no of data rows in a Data set/Class
    */
  val K = clusterCount(datas.count(), exp.nbTotProto)

  println(exp.name + " : " + exp.nbTotProto + " = " + K)
  println("Total data" + datas.count())

  val gmm = new GaussianMixture().setK(K).run(datas)
  gmm.save(sc,"data/EMModel")
  val mapClusterIndices =gmm.predict(datas)
  val x: Array[MultivariateGaussian] = gmm.gaussians

  // output parameters of max-likelihood model
  for (i <- 0 until gmm.k) {
    println("weight=%f\nmu=%s\nsigma=\n%s\n" format
      (gmm.weights(i), gmm.gaussians(i).mu, gmm.gaussians(i).sigma))
    val clusterCenter = gmm.gaussians(i).mu;
  }

  // Save results
  val sfilename = exp.outDir + "/" + exp.name + ".clustering.em_centers_" + exp.nbTotProto


  if (!Files.exists(Paths.get(sfilename))) {
    mapClusterIndices
      .map(d => d)
      .saveAsTextFile(sfilename)
  }
```

Classification Algorithms:

Used Decision Tree and Random Forest to implement classification model for 20 classes of Image Data. And this is implemented on two data sets 101 Object Categories and 256 Object categories. All of this is implemented in Apache Saprk.

Decision Tree:

Decision trees and their ensembles are popular methods for the machine learning tasks of classification and regression. Decision trees are widely used since they are easy to interpret, handle categorical features, extend to the multiclass classification setting, do not require feature scaling, and are able to capture non-linearities and feature interactions. Tree ensemble algorithms such as random forests and boosting are among the top performers for classification and regression tasks.

`spark.mllib` supports decision trees for binary and multiclass classification and for regression, using both continuous and categorical features. The implementation partitions data by rows, allowing distributed training with millions of instances.

Ensembles of trees (Random Forests and Gradient-Boosted Trees) are described in the Ensembles guide.

Evaluation Results:

	101 Object Categories			
	Decision Tree			
	som_centers_1x200	**som_centers_1x100**	**som_centers_1x300**	**som_centers_1x500**
Time to train the Decision Tree model:	58633	58633	137705	192548
Accuracy	0.103614458	0.103614458	0.109412711	0.092261905
Weighted Precision	0.108936616	0.108936616	0.117295731	0.093236857
Weighted Recall	0.103614458	0.103614458	0.109412711	0.092261905
Weighted FMeasure	0.104197508	0.104197508	0.10972056	0.092103895

	256 Object categories			
	Decision Tree			
	som_centers_1x10	**som_centers_1x20**	**som_centers_1x30**	**som_centers_1x50**
Time to train the Decision Tree model:	27310	201302	366434	303136
Accuracy	0.142857143	0.103092784	0.181818182	0.175675676
Weighted Precision	0.123323615	0.099400514	0.211810412	0.16342607
Weighted Recall	0.142857143	0.103092784	0.181818182	0.175675676
Weighted FMeasure	0.126889437	0.095719076	0.181084583	0.167148928

With KMeans:

	101 Object categories			
	Decision Tree			
	kmeans_centers_1(	**kmeans_centers_2(**	**kmeans_centers_3(**	**kmeans_centers_50**
Time to train the Decision Tree model:	38683	65838	87772	128234
Accuracy	0.069230769	0.075903614	0.076427997	0.084821429
Weighted Precision	0.073752654	0.079946099	0.080330136	0.089280093
Weighted Recall	0.069230769	0.075903614	0.076427997	0.084821429
Weighted FMeasure	0.069871115	0.076110061	0.076927559	0.085829516

	256 Object categories			
	Decision Tree			
	kmeans_centers_1(	**kmeans_centers_2(**	**kmeans_centers_3(**	**kmeans_centers_50**
Time to train the Decision Tree model:	45776	55525	77372	131732
Accuracy	0.152173913	0.108910891	0.118881119	0.136546185
Weighted Precision	0.134057971	0.118733302	0.121578804	0.132767639
Weighted Recall	0.152173913	0.108910891	0.118881119	0.136546185
Weighted FMeasure	0.138026225	0.097109491	0.110200135	0.133205656

Random Forest:

Random forests are ensembles of decision trees. Random forests are one of the most successful machine learning models for classification and regression. They combine many decision trees in order to reduce the risk of overfitting. Like decision trees, random forests handle categorical features, extend to the multiclass classification setting, do not require feature scaling, and are able to capture non-linearities and feature interactions.

`spark.mllib` supports random forests for binary and multiclass classification and for regression, using both continuous and categorical features. `spark.mllib` implements random forests using the existing decision tree implementation. Please see the decision tree guide for more information on trees.

Evaluation Results:

	101 Object Categories			
	Random Forest			
	som_centers_1x100	**som_centers_1x200**	**som_centers_1x300**	**som_centers_1x500**
Time to train the Decision Tree model:	317735	66600	98674	145590
Accuracy	0.094986807	0.132220796	0.122098022	0.146610615
Weighted Precision	0.166216707	0.152498023	0.138769863	0.167169597
Weighted Recall	0.094986807	0.132220796	0.122098022	0.146610615
Weighted FMeasure	0.059301764	0.135543096	0.12216164	0.147813608

	256 Object Categories			
	Random Forest			
	som_centers_1x10	**som_centers_1x20**	**som_centers_1x30**	**som_centers_1x50**
Time to train the Decision Tree model:	33448	45304	50176	73639
Accuracy	0.224489796	0.234042553	0.230769231	0.182222222
Weighted Precision	0.183503401	0.220376432	0.222023609	0.180138251
Weighted Recall	0.224489796	0.234042553	0.230769231	0.182222222
Weighted FMeasure	0.189167975	0.219570223	0.213488027	0.170878412

From these results it can be observed that depending on data to get best results or accuracy, SOM map size needs to be tuned. And apache spark model parameters need to be tuned.

Project Management:

Work Completed:

Sigspace implementation:

Feature Extraction from Image data using SIFT and Signature generation from features using SOM and k-Means. Signature Generation using GMM(in progress). Classification on SOM signatures using Decision Tree and Random Forest.

Time Taken: 60 hours approximately

Work to be completed:

SigSpace - classification implementation on kMeans and GMM clustering signatures, and on SIFT features. Image extraction from video. Training workflow for video data. Test case workflow implementation in Kafka, Storm and Spark.

Project Phase -3

Random Forest:

kMeans:

	101 Object categories			
	Random Forest			
	kmeans_centers_10	**kmeans_centers_20**	**kmeans_centers_30**	**kmeans_centers_50**
Time to train the Decision Tree model:	1980390	1757413	137621	198644
Accuracy	0.050131926	0.07830552	0.084264832	0.096689438
Weighted Precision	0.053494158	0.088257086	0.098839629	0.094490817
Weighted Recall	0.050131926	0.07830552	0.084264832	0.096689438
Weighted FMeasure	0.049272428	0.081033844	0.086669288	0.092374875

	256 Object Categories			
	Random Forest			
	kmeans_centers_10	**kmeans_centers_20**	**kmeans_centers_30**	**kmeans_centers_50**
Time to train the Decision Tree model:	13888	26698	54323	117411
Accuracy	0.255319149	0.173469388	0.261437908	0.253112033
Weighted Precision	0.308308004	0.208728254	0.26349006	0.250178769
Weighted Recall	0.255319149	0.173469388	0.261437908	0.253112033
Weighted FMeasure	0.23157717	0.166012559	0.246839638	0.240649172

From all the above analysis we can say that, for 101 Object Categories data, the best accuracy was 14.6% for 20 categories and is with SOM 1x500 centers and Random forest. And for 256 Object Categories data, the best accuracy was 26.1% for 20 categories and is with kMeans 30 centers and Random forest.

EM Clustering:

A Gaussian Mixture Model represents a composite distribution whereby points are drawn from one of *k* Gaussian sub-distributions, each with its own probability. The `spark.mllib` implementation uses the expectation-maximization algorithm to induce the maximum-likelihood model given a set of samples. The implementation has the following parameters:

- *k* is the number of desired clusters.
- *convergenceTol* is the maximum change in log-likelihood at which we consider convergence achieved.
- *maxIterations* is the maximum number of iterations to perform without reaching convergence.
- *initialModel* is an optional starting point from which to start the EM algorithm. If this parameter is omitted, a random starting point will be constructed from the data.

EM clustering can be an alternatively competitive for SOM and kMeans to get signatures. EM clustering can also be compared for Sigspace architecture.

But EM is a probabilistic approach with Gaussian mixture model. So, it won't have center's as in SOM. So, for signature we need to consider something else other than center for e.g. weights, mean, sigma distribution etc. Tried this but not a complete documentation of Spark Implementation is available so, some research study is still needed.

Global SOM (SOM on SOM):

In the above approach signatures are generated using SOM and kMeans for individual categories. Now we can generate one more SOM on these signatures to more distinctively identifying categories i.e. to form more clear clusters to increase the accuracy. We can add more SOM layers to increase accuracy further more. SOM implementation used so far generates signatures from features i.e. data is reduced in dimensionality and no. of features as well. But for the global SOM, the number of signatures should not decrease. We are not trying to find signatures on top of signatures so, a modified implementation of SOM at Spark is needed.

Work to be Done:

Global SOM: SOM modified implementation. SigSpace - GMM clustering signatures. For video summarization: Image extraction from video is implemented for sample files. Training workflow for video data is implemented for sample files. Test case workflow implementation in Kafka, Storm and Spark.

Work Done Time: 60 hours approximately.

Bibliography:

1. Pradyumna Doddala,SigSpace: Class-base Feature representation for Scalable and Distributed Machine Learning, Thesis Spring 2016 at University of Missouri-Kansas City.
2. https: spark.apache.org/
3. http: storm.apache.org/
4. http://kafka.apache.org/
5. http://openimaj.org/
6. http://www.infoworld.com/article/2854894/application-development/spark-and-storm-for-real-time-computation.html
7. Ma, Shuangmei, and Zhengli Liang. "Design and Implementation of Smart City Big Data Processing Platform Based on Distributed Architecture."*Intelligent Systems and Knowledge Engineering (ISKE), 2015 10th International Conference on*. IEEE, 2015.
8. Morshed, Sarwar Jahan, Juwel Rana, and Marcelo Milrad. "Open source initiatives and frameworks addressing distributed real-time data analytics."*2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2016.
9. Kaski, Samuel, et al. ”WEBSOMself-organizing maps of document collections.” Neurocomputing 21.1 (1998): 101-117.
10. Honkela, Timo. ”Self-organizing maps of words for natural language processing applications.” Proceedings of the International ICSC Symposium on Soft Computing. 1997.
11. https://en.wikipedia.org/wiki/Self-organizing map
12. https://github.com/TugdualSarazin/spark-clustering
13. https://en.wikipedia.org/wiki/Decision_tree
14. https://en.wikipedia.org/wiki/Random_forest