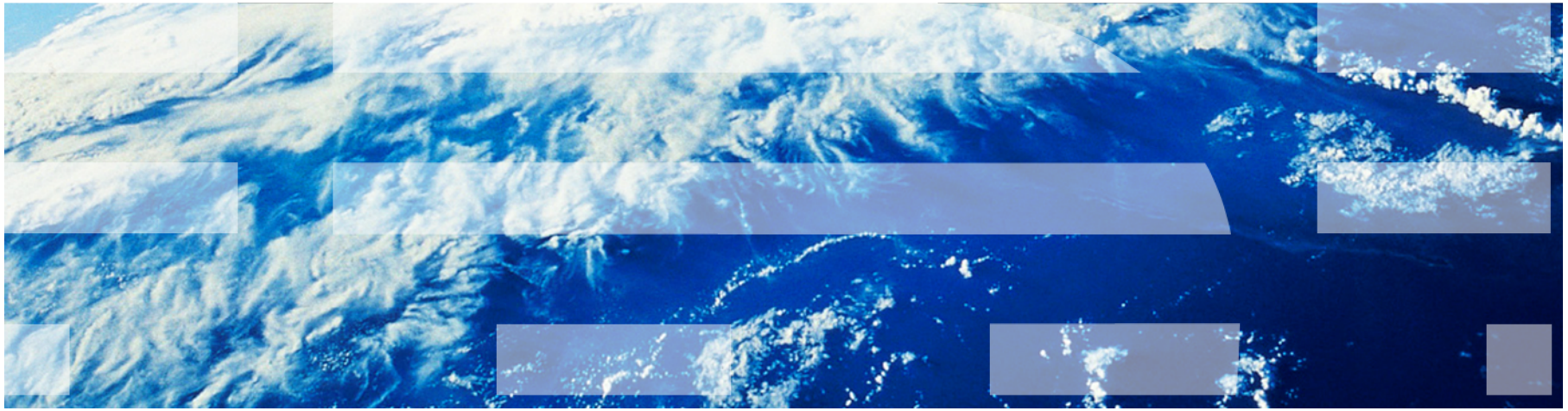
Columbia University

E6893 Big Data Analytics Lecture 4:

Real-Time Stream Analysis

Ching-Yung Lin, Ph.D.

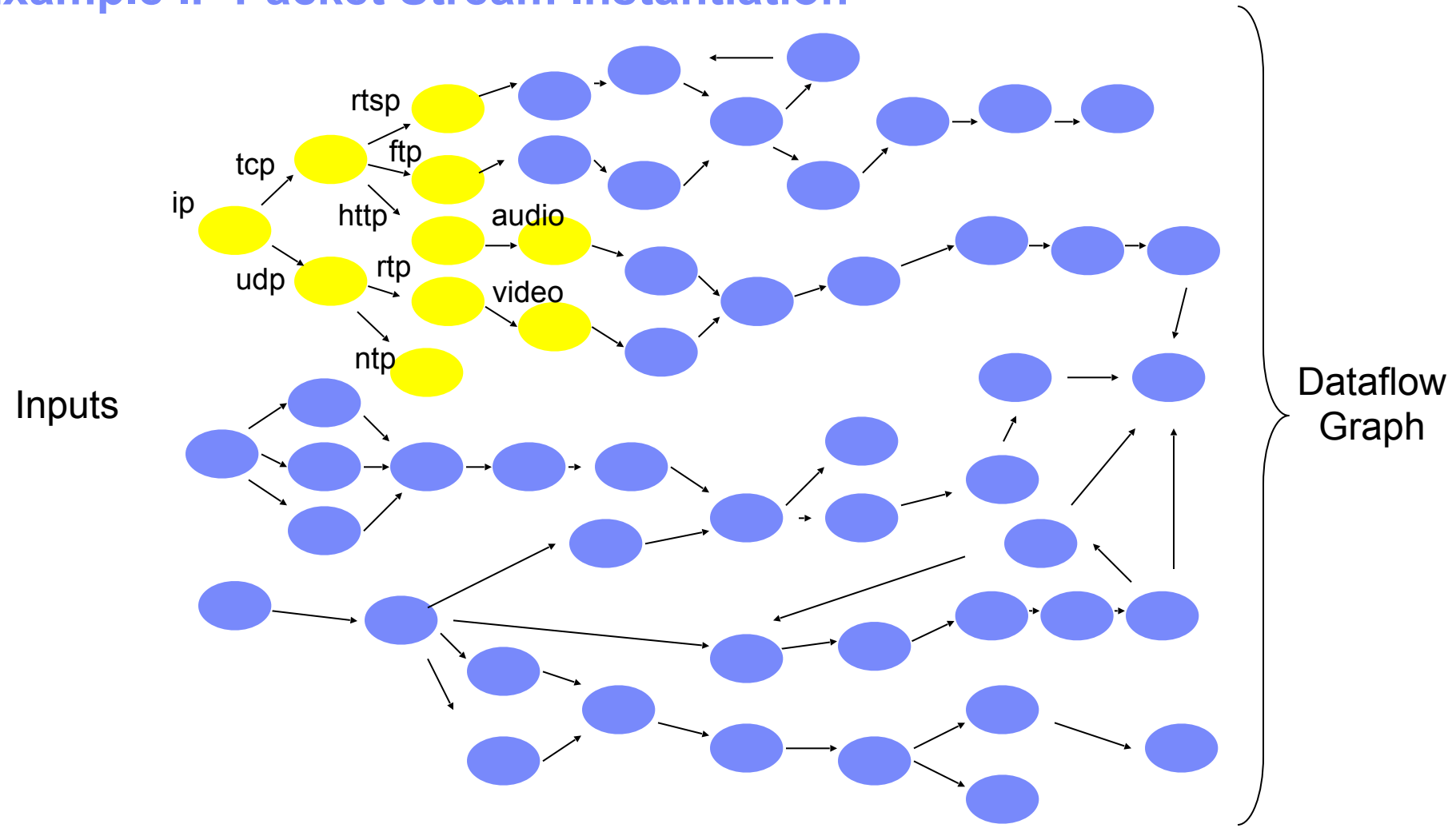
Adjunct Professor, Dept. of Electrical Engineering and Computer Science



September 29, 2023

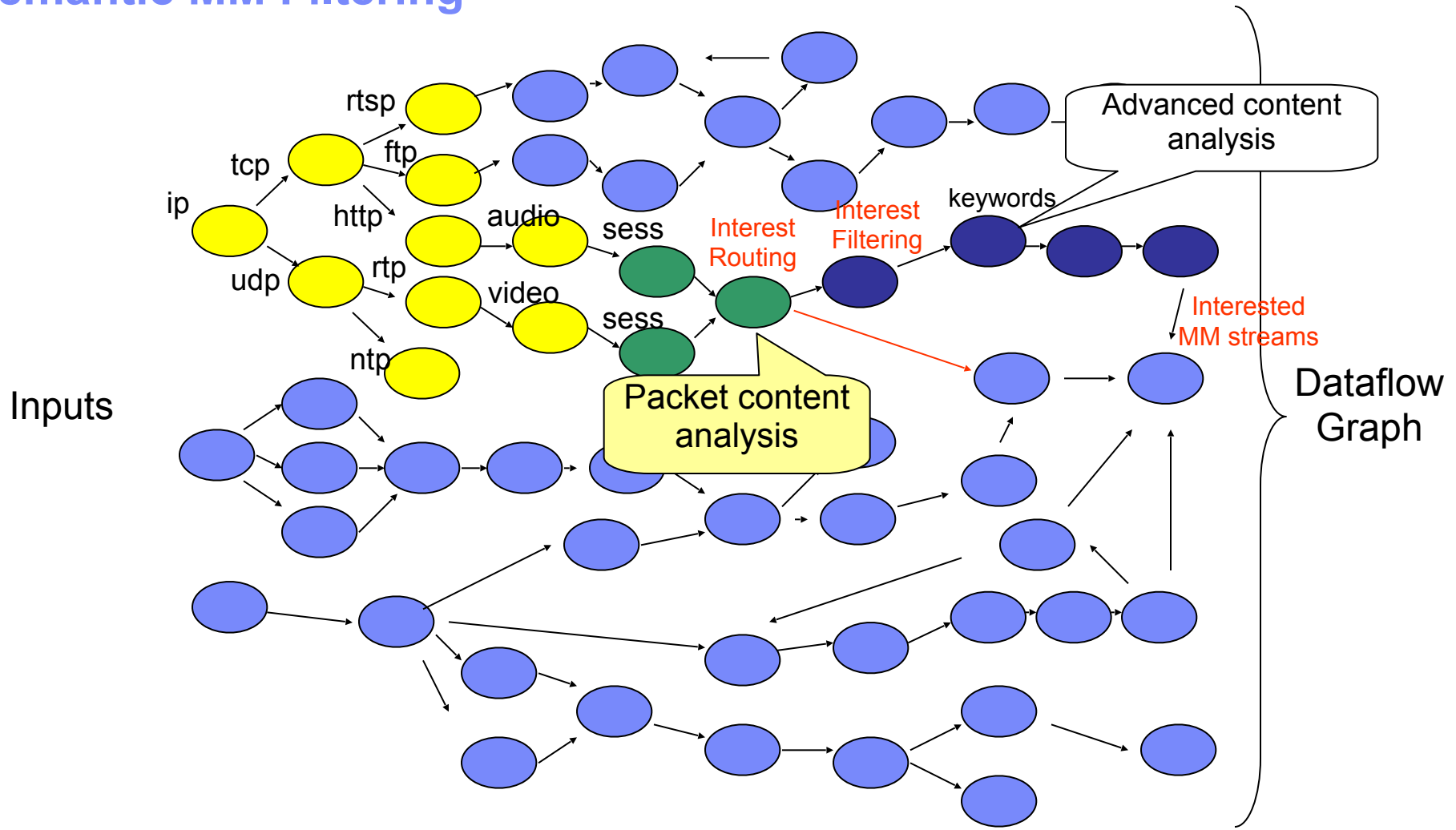
Stream Analyses Technical Challenges

Example IP Packet Stream Instantiation



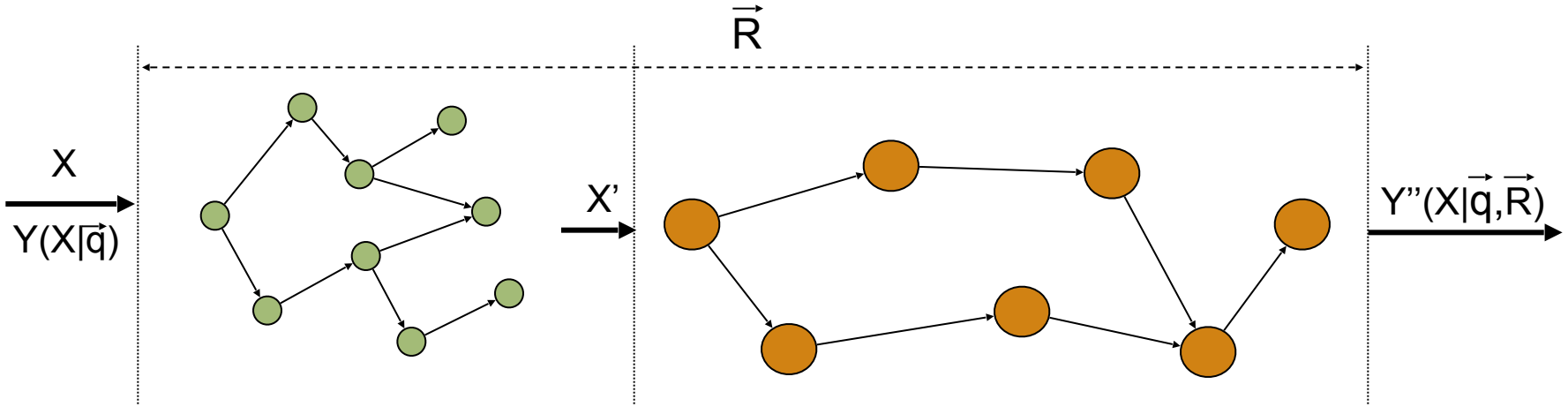
By IBM Dense Information Gliding Team

Semantic MM Filtering



per PE rates	200-500MB/s	~100MB/s	10 MB/s

Resource-Accuracy Trade-Offs



Configurable Parameters of Processing Elements to maximize relevant information:

Y''(X | q, R) > Y'(X | q, R),

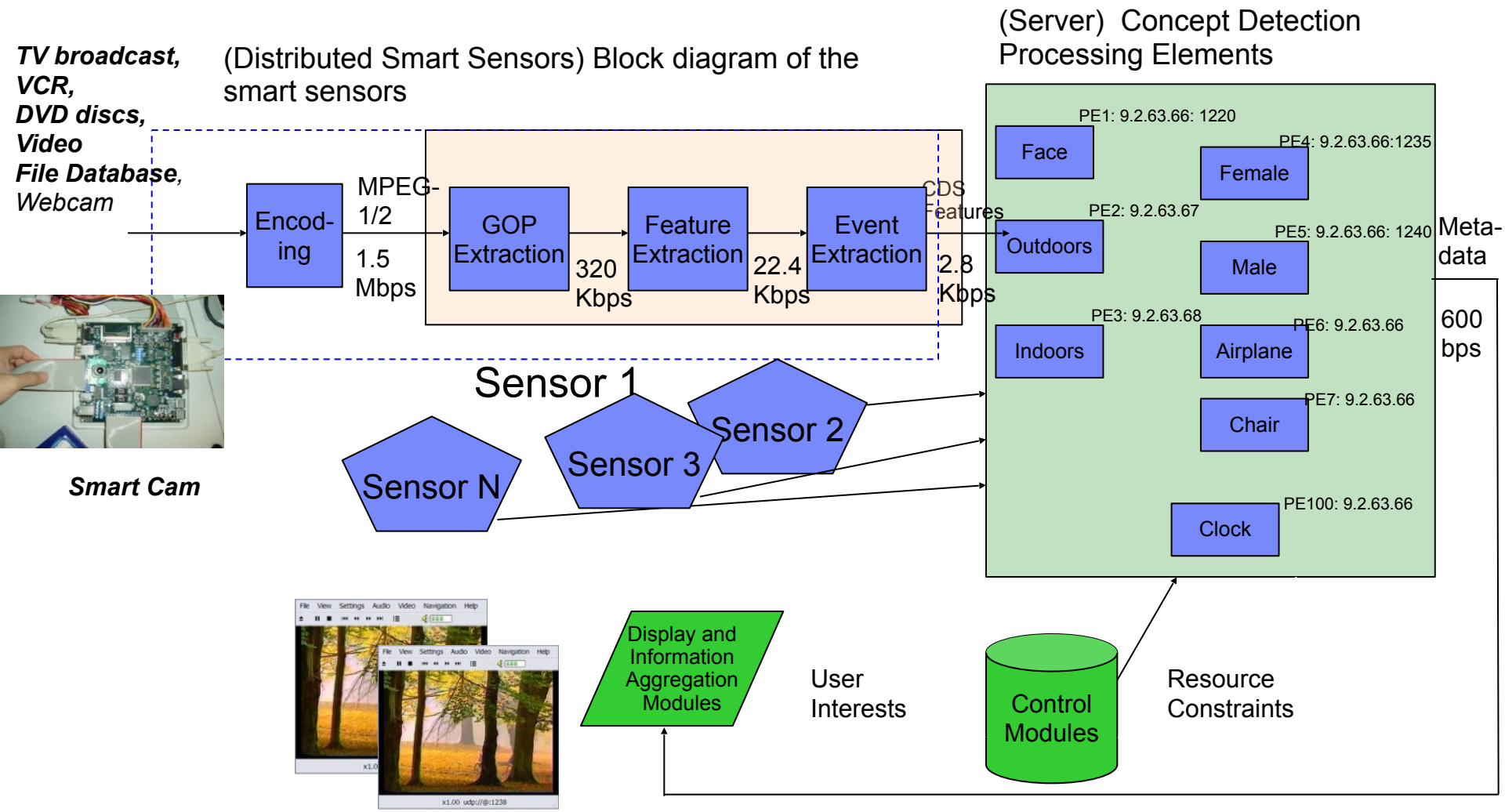
with resource constraint.

Required resource-efficient algorithms for:

Classification, routing and filtering of signal-oriented data: (audio, video and, possibly, sensor data)

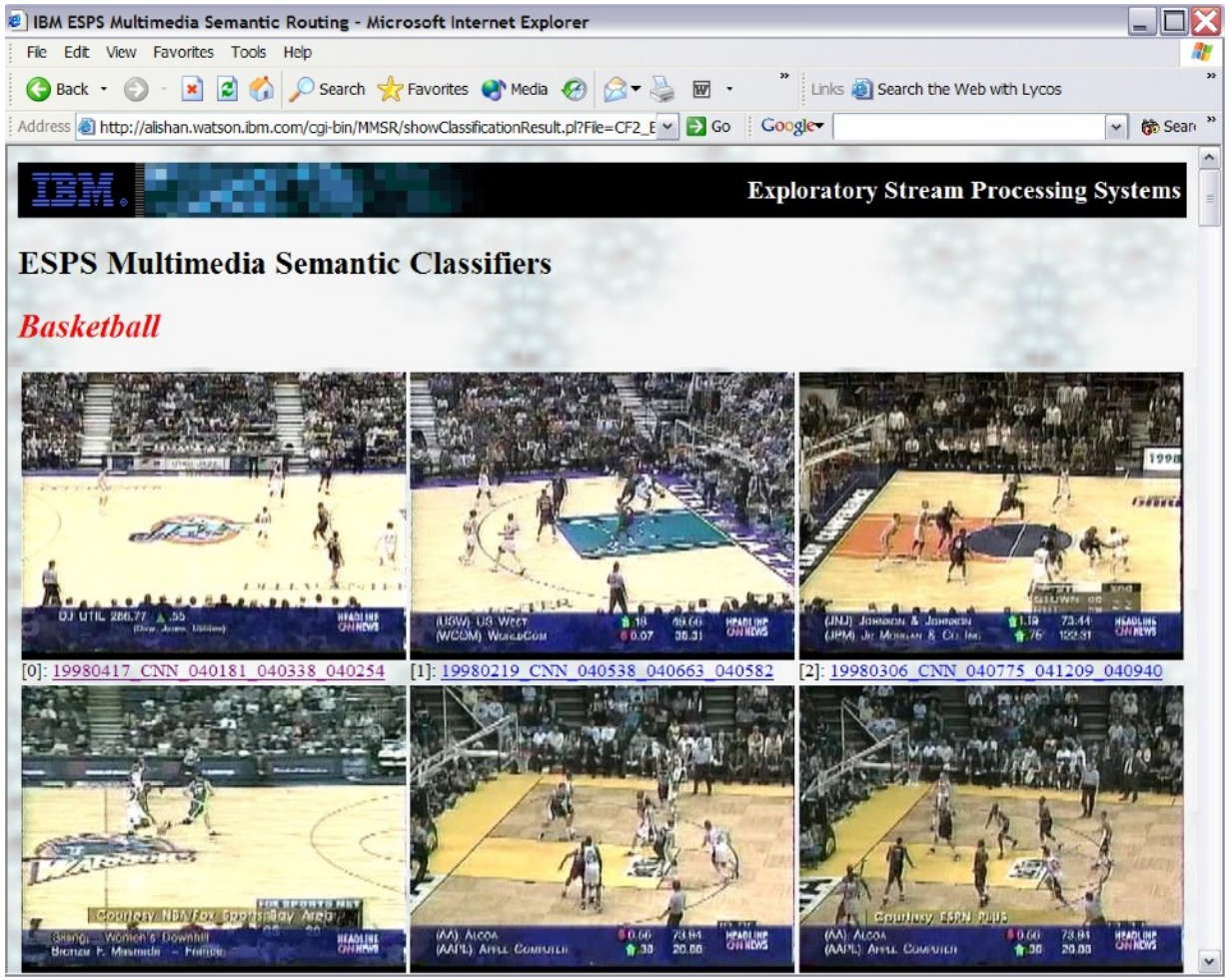
- **Input data X – Queries q – Resource R**
  - Y(X | q): Relevant information
  - Y'(X | q, R) ` Y(X | q): Achievable subset given R

Example: Distributed Video Signal Understanding (Lin et al.)



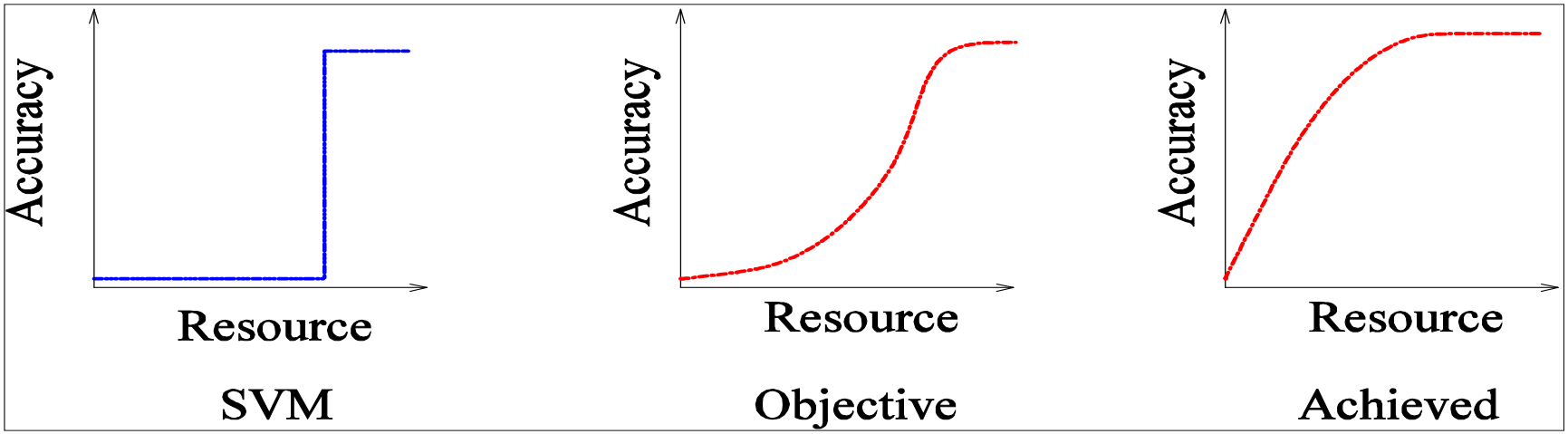
Semantic Concept Filters

E.g.:



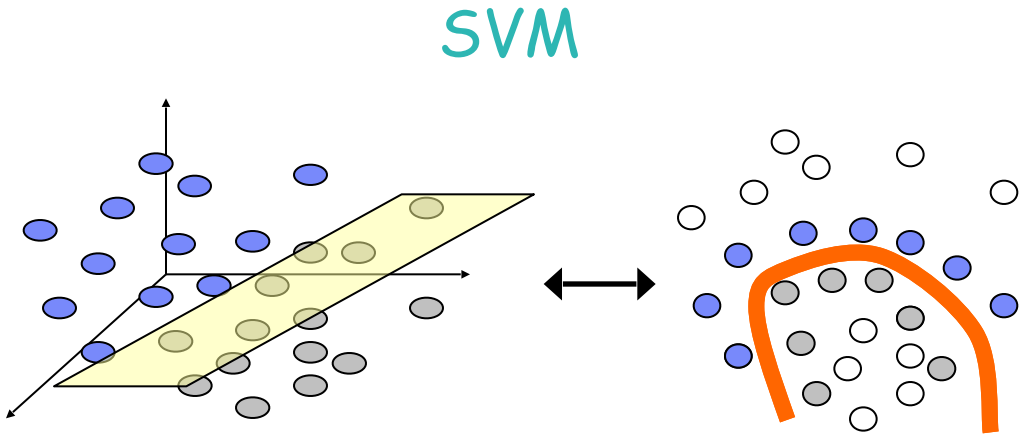
Complexity Reduction Introduction

- Objective: Real-time classification of instances using Support Vector Machines (SVMs)
- Computationally efficient and reasonably accurate solutions
- Techniques capable of adjusting tradeoff between accuracy and speed based on available computational resources



SVM formulation

- **Given :**
  - Training instances $\{\mathbf{x}_i\}$ with labels y_i
- **Objective :**
  - Find maximum margin hyperplane separating positive and negative training instances



Decision

- **Score of unseen instance** $\mathrm{u}_j : \mathrm{w} \cdot \phi(u_j)$
- **In terms of Lagrangian multipliers**

$$\sum_i \alpha_i y_i k(x_i, u_j)$$

- **Computational Cost : O($\mathrm{n}_{sv} d$)**
  - n_{sv}: Number of support vectors
  - d : Dimensionality of each data instance

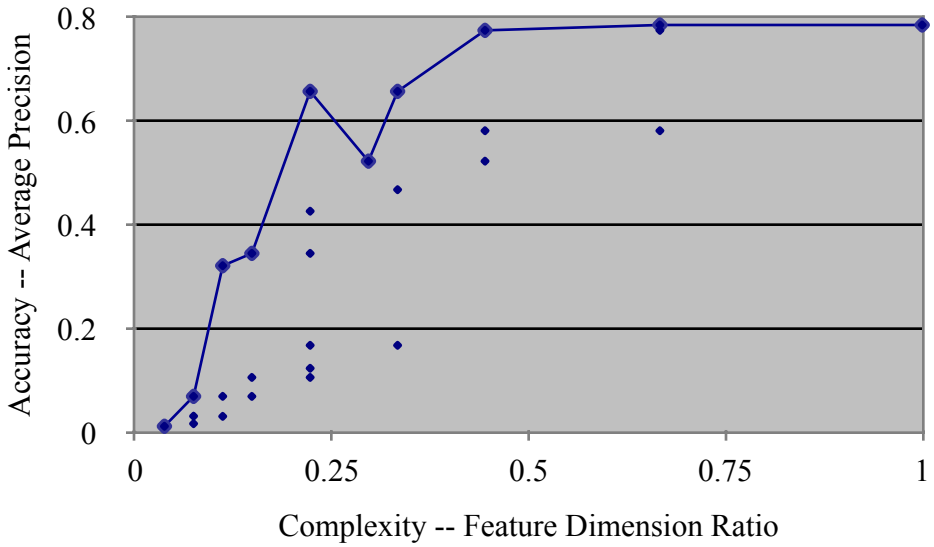
Problems

- **Number of support vectors grows quasi-linearly with size of training set [Tipping 2000]**
- **Inner product with each support vector of dimensionality d expensive**
  - Example TREC2003
    - Human : 19745 support vectors
    - Face : 18090
- **High data rates(10Gbits/sec) means large number of abandoned data**

Example

- Processing Power 1 Ghz
- 10000 support vectors
- 1000 / 2 features per instance
- Order of at least 10^7 operations required per stream per sec
- Translates to less than 100 instances evaluated per sec with only one classifier

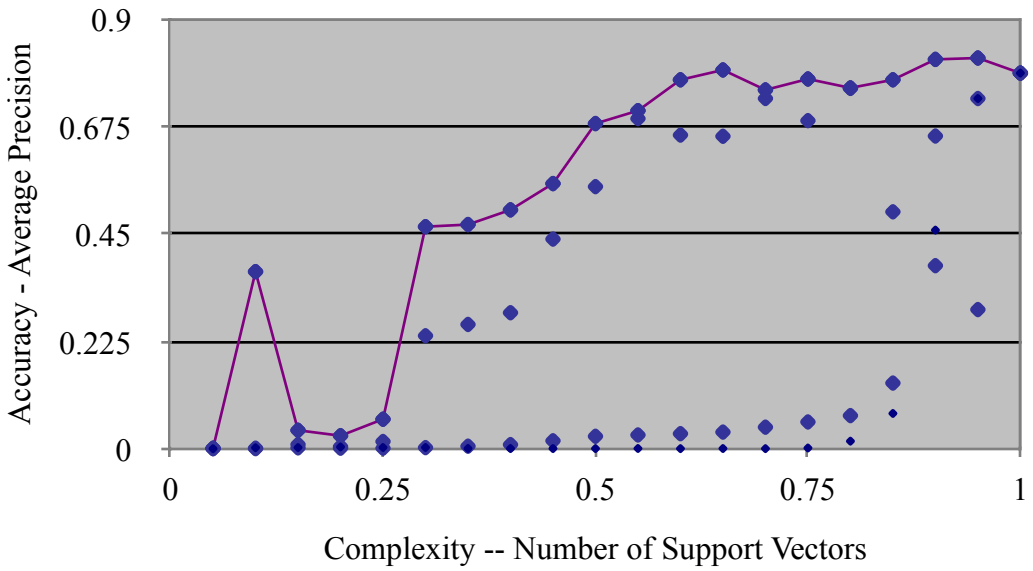
Naïve Approach I – Feature Dimension Reduction



- Experimental Results for Weather_News Detector
- Model Selection based on the Model Validation Set
- E.g., for Feature Dimension Ratio 0.22, (the best selection of features are: 3 slices, 1 color, 2 texture selections), the accuracy is decreased by 17%.

Slice	Color	Texture	Feature Dimension Ratio	AP
3	3	3	1	0.7861
3	3	2	0.666666667	0.7861
3	2	3	0.666666667	0.7757
2	3	3	0.666666667	0.5822
3	2	2	0.444444444	0.7757
2	3	2	0.444444444	0.5822
2	2	3	0.444444444	0.5235
3	3	1	0.333333333	0.4685
3	1	3	0.333333333	0.6581
1	3	3	0.333333333	0.1684
2	2	2	0.296296296	0.5235
3	2	1	0.222222222	0.427
3	1	2	0.222222222	0.6581
2	3	1	0.222222222	0.1241
2	1	3	0.222222222	0.3457
1	3	2	0.222222222	0.1684
1	2	3	0.222222222	0.1065
2	2	1	0.148148148	0.0699
2	1	2	0.148148148	0.3457
1	2	2	0.148148148	0.1065
3	1	1	0.111111111	0.3219
1	3	1	0.111111111	0.0314
1	1	3	0.111111111	0.07
2	1	1	0.074074074	0.0318

Naïve Approach II – Reduction on the Number of Support



- Proposed Novel Reduction Methods:
  - **Ranked Weighting**
  - **P/N Cost Reduction**
  - **Random Selection**
  - **Support Vector Clustering and Centralization**
- Experimental Results on Weather_News Detectors show that complexity can be at 50% for the cost of 14% decrease on accuracy

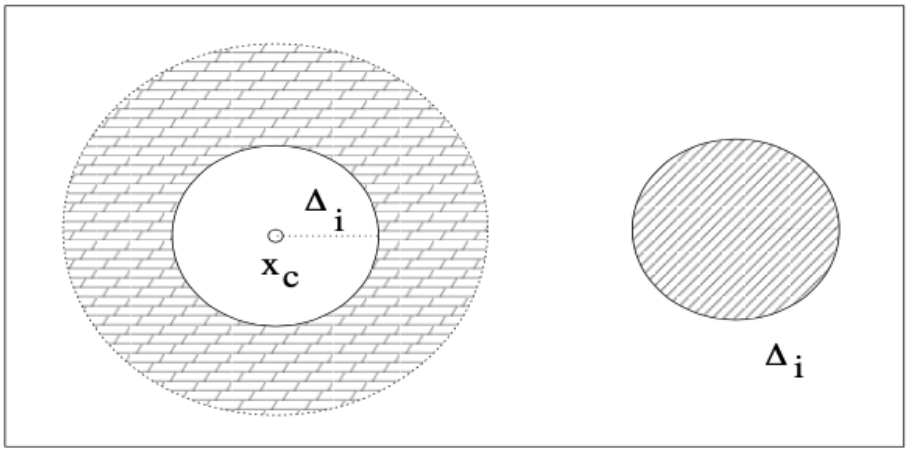
Weighted Clustering Approach

- **Basic steps**
  - Cluster support vectors
  - Use cluster center as representative for all support vectors in cluster
  - Determine scalar weight associated with each cluster center
  - Use only cluster centers to score new instances

Cluster center weight (contd.)

- Choose γ_i minimizing square of difference in scores over all $\pm_i$ and d
- Sub-cases :

$$d \geq \Delta_i \qquad\qquad d < \Delta_i$$



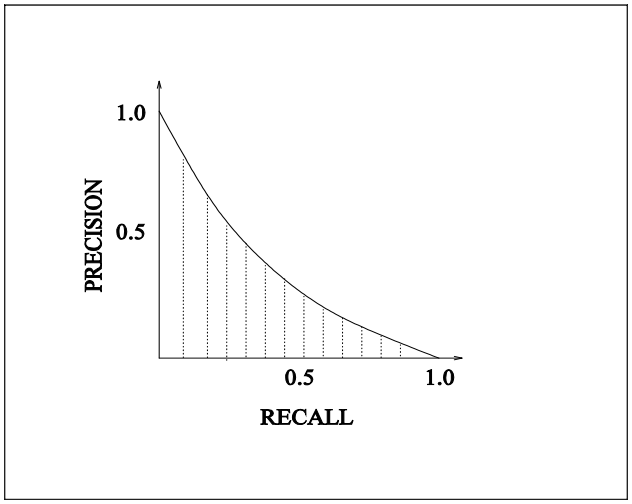
Using the weights

- **For every support vector in cluster**
  - Distance Δ_i known
  - Two weights computed
- **Cumulative effect of all support vectors in clusters additive**
  - Δ_i because of various support vectors added up at center to simulate effect of all support vectors
- Δ_i **sorted, weight arrays rearranged**

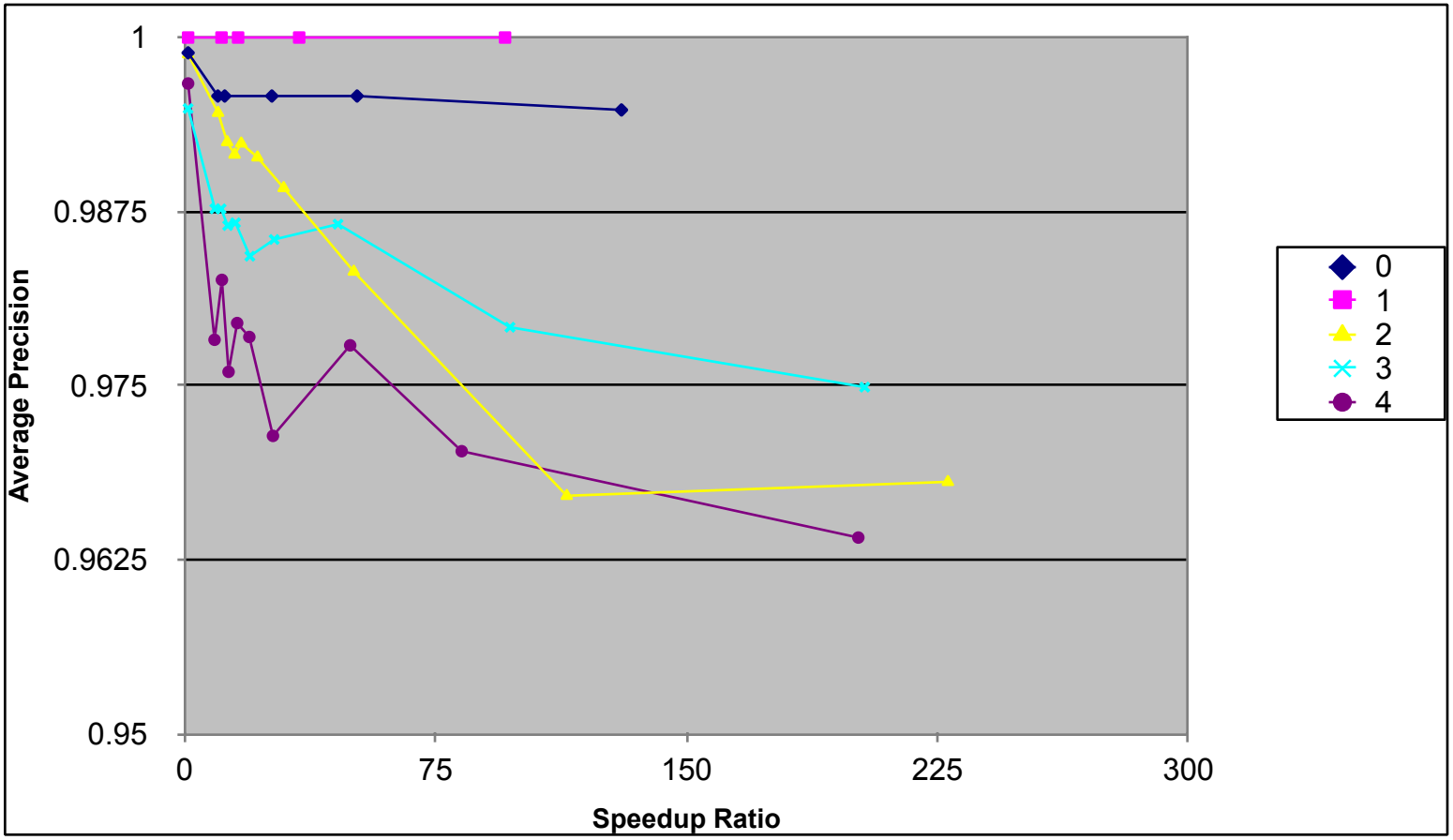
Experiments

- Datasets
  - TREC video datasets (2003 and 2005)
    - 576 features per instance
    - > 20000 test instances overall
  - MNist handwritten digit dataset (RBF kernel)
    - 576 features
    - 60000 training instances, 10000 test instances

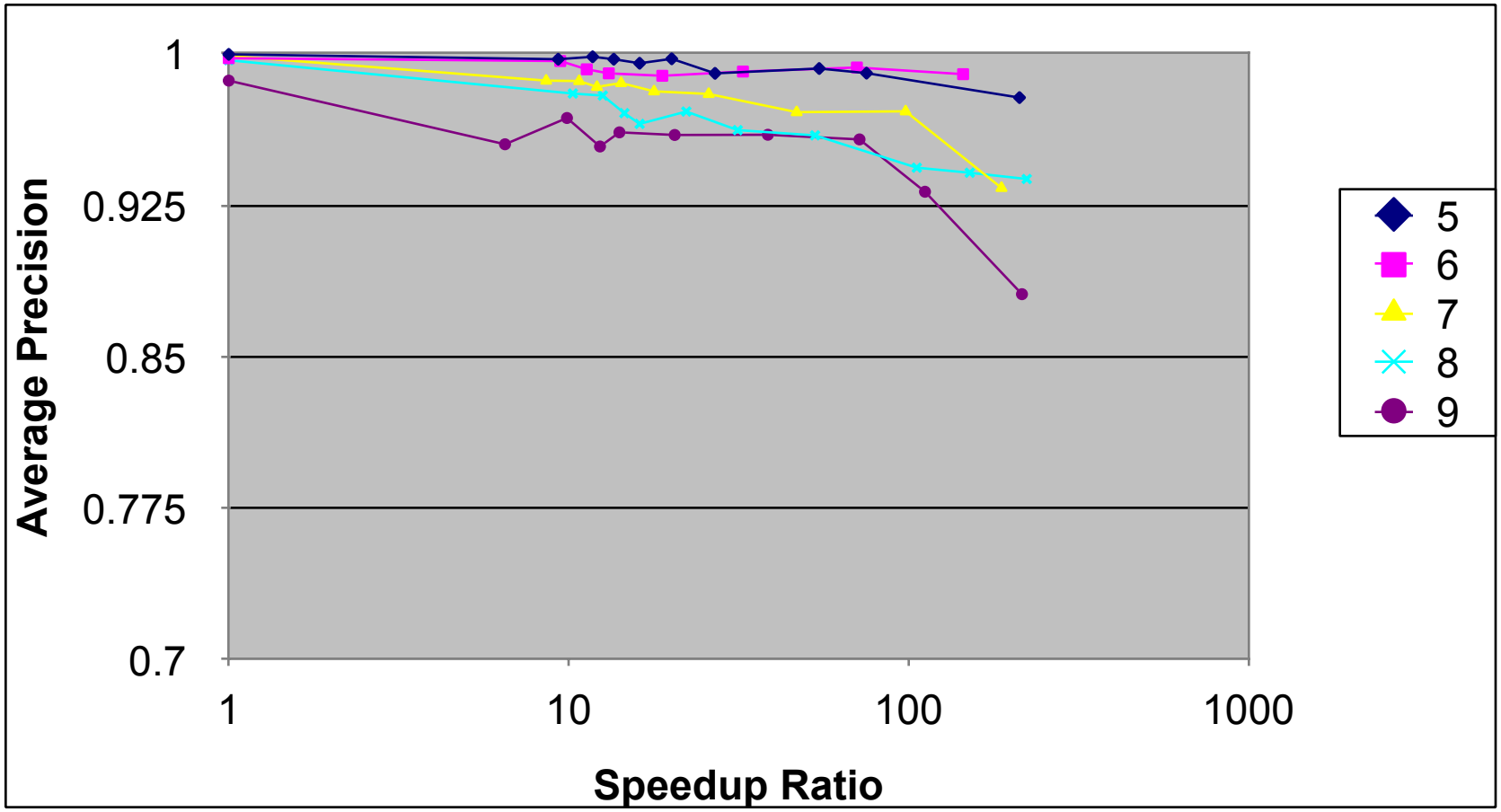
- Performance metrics
  - Speedup achieved over evaluation with all support vectors
  - Average precision achieved



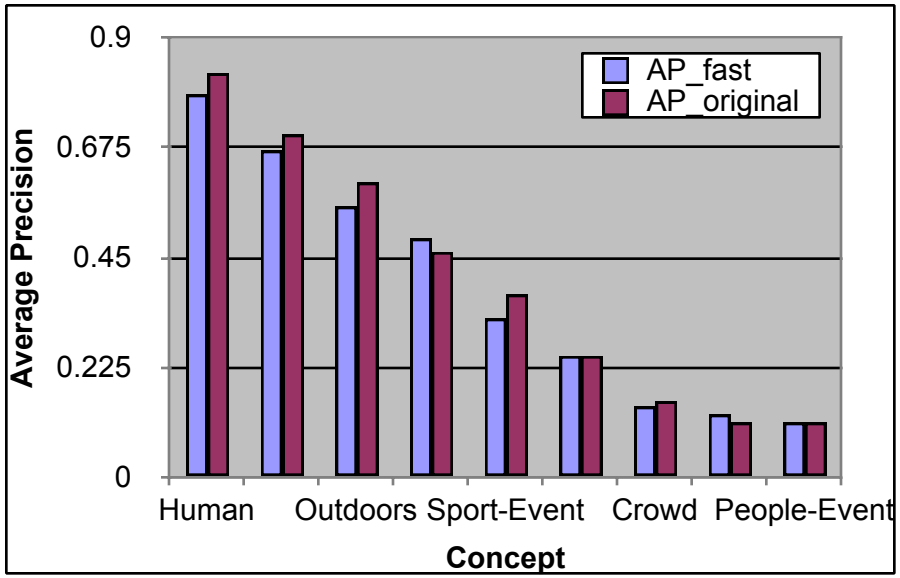
Results (Mnist 0-4)

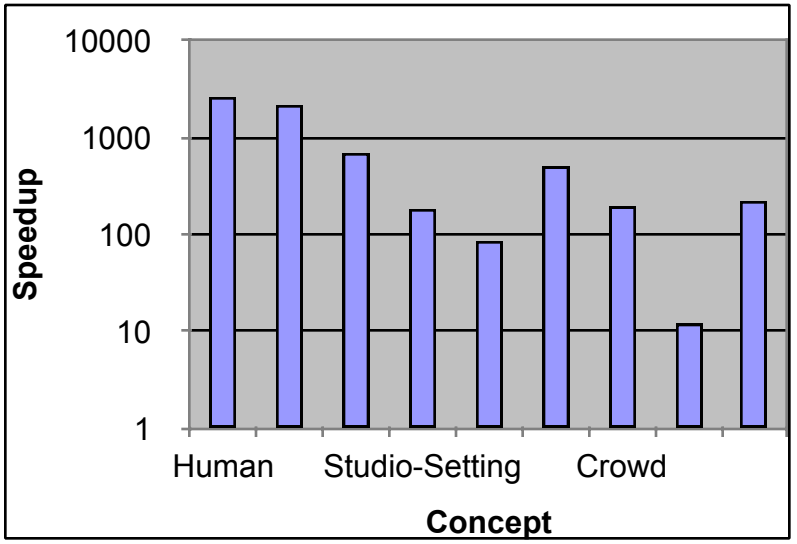


Results (Mnist 5-9)



Results (TREC 2003)





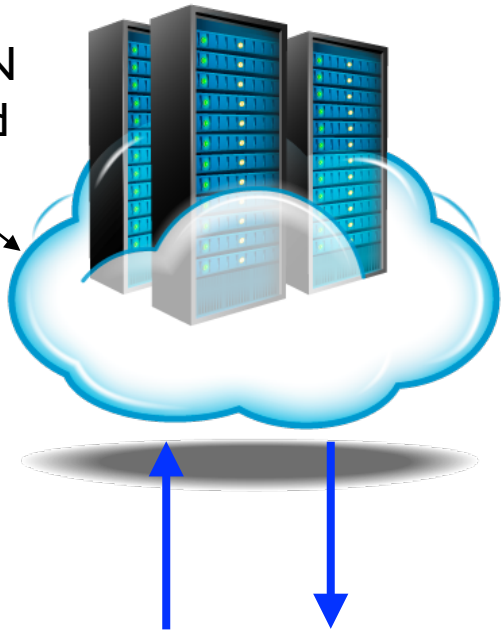
Summary of Complexity Reduction

- Techniques presented demonstrate reasonable performance in terms of both speedup and average precision over multiple concepts in datasets
- Speedups
  - MNist : All concepts at least 50 times faster with AP within 0.04 of original
  - TREC 2003: Eight out of nine concepts speedup greater than 80 times with AP within 0.05 of original
  - TREC 2005: APs in some cases along with speedup respectable
- APs of most concepts close to original APs

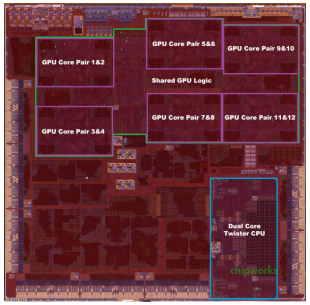
Acceleration of Neural Network for Streams

Methods for Running CNNs on Mobile Devices

Sending CNN jobs to cloud



Acceleration CNN on Local Device



Apple A9X SoC, 12-core GPU

- How to trade off between algorithmic complexity and performance?
- How to utilize hardware effectively

Summary of Acceleration of Neural Network on Mobile Devices

- Porting Deep Convolution Neural Network on iOS Device with near real-time computation
  - Reduce algorithmic complexity with ignorable performance degradation.
  - Utilize computational hardware to achieve better performance.

	iPhone 7	iPhone 7+	iPhone 6s	iPad Pro 12.9"
Alex Net	70 ms	70 ms	130 ms	69 ms
p-Alex Net	N/A	35 ms	N/A	28 ms
GoogLeNet	130 ms	128 ms	195 ms	110 ms
p-GoogLeNet	N/A	80 ms	N/A	70 ms
VGG16 Net	880 ms	883 ms	1450 ms	725 ms

Challenges for Running CNN on Mobile Devices

	Storage	Memory	Speed
	Model Size	Weights	Mult.s
AlexNet	243MB	61M	725M
VGG-S	393MB	103M	2640M
VGG-16	552MB	138M	15484M
GoogLeNet	51MB	6.9M	1566M

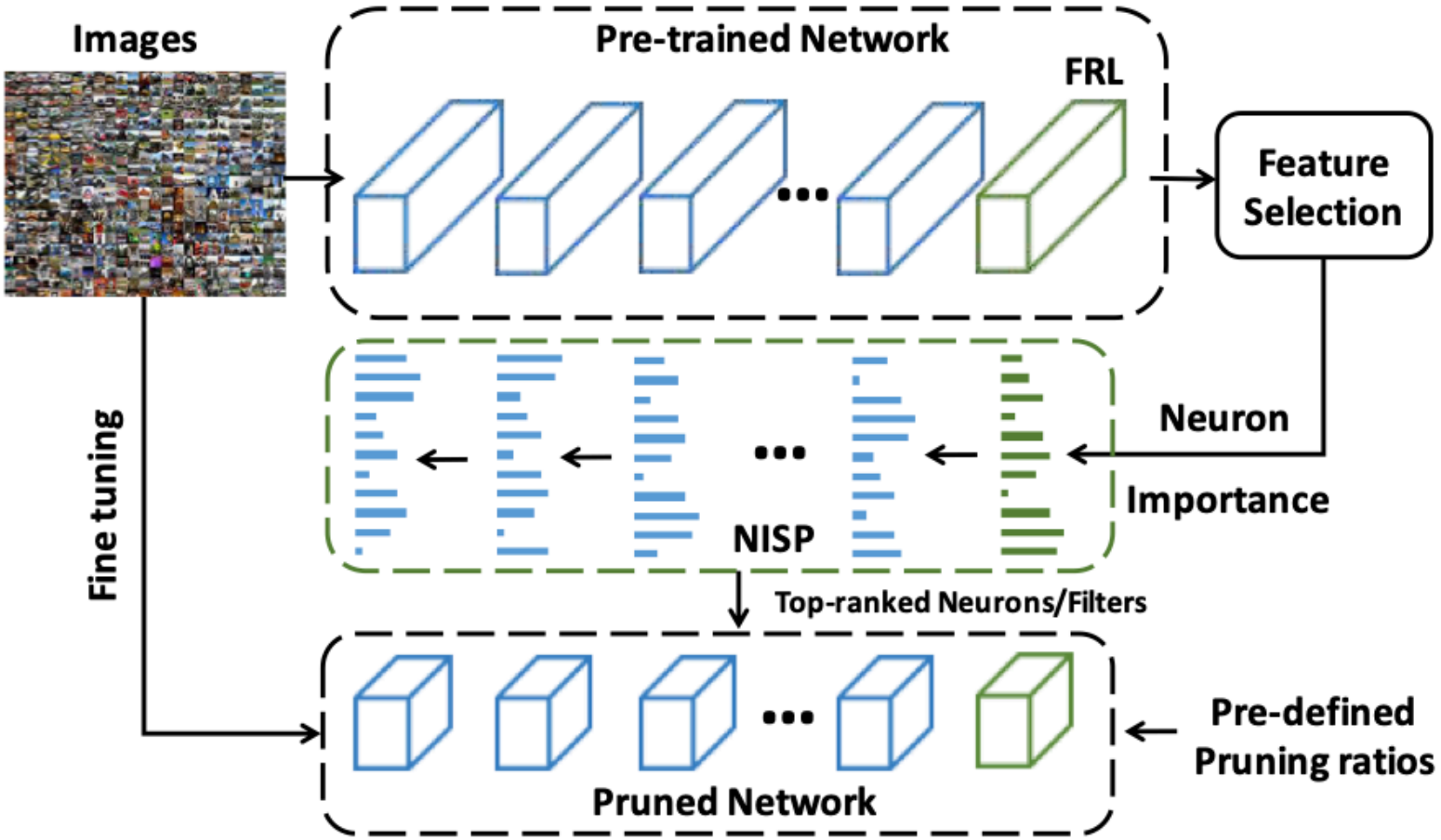
Statistics of some popular CNNS

Reference:
Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications

Computational Resource on iPhone and iPad

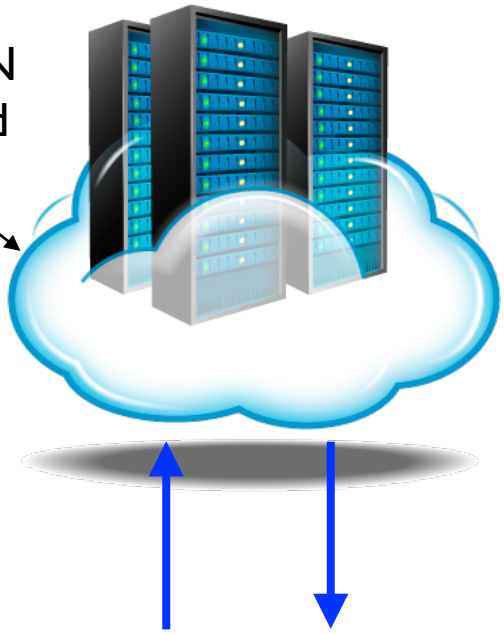
	iPhone 6S (Plus)	iPad Air 2	iPad Pro (12.9/9.7)	iPhone 7 (Plus)
SoC	A9	A8X	A9X	A10 Fusion
CPU	2x Twister @ 1.85 GHz	3x Typhone @ 1.5 GHz	2x Twister @ 2.26 GHz	4-core
GPU	PVR GT7600 (6 cluster)	PVR GXA6850 (8 cluster)	PVR 12 Cluster Series 7	6 cluster GPU?
RAM (shared memory)	2GB LDDR4	2GB LDDR3	4GB LDDR4	3GB on Plus?
Memory bus width	64-bit	128-bit	128-bit	?
Max # of threads per group	512	512	512	?

Neuron Importance Score Propagation (NISP, Yu et al 2018)



Methods for Running CNNs on Mobile Devices

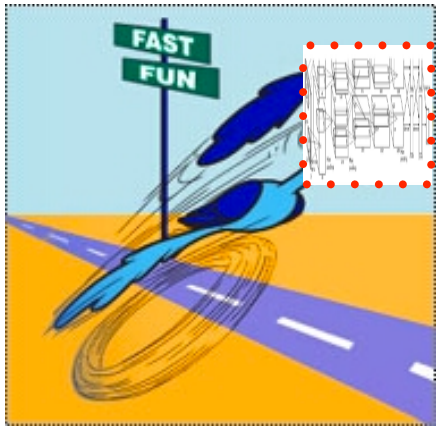
Sending CNN jobs to cloud



Compression (pruning) of CNN

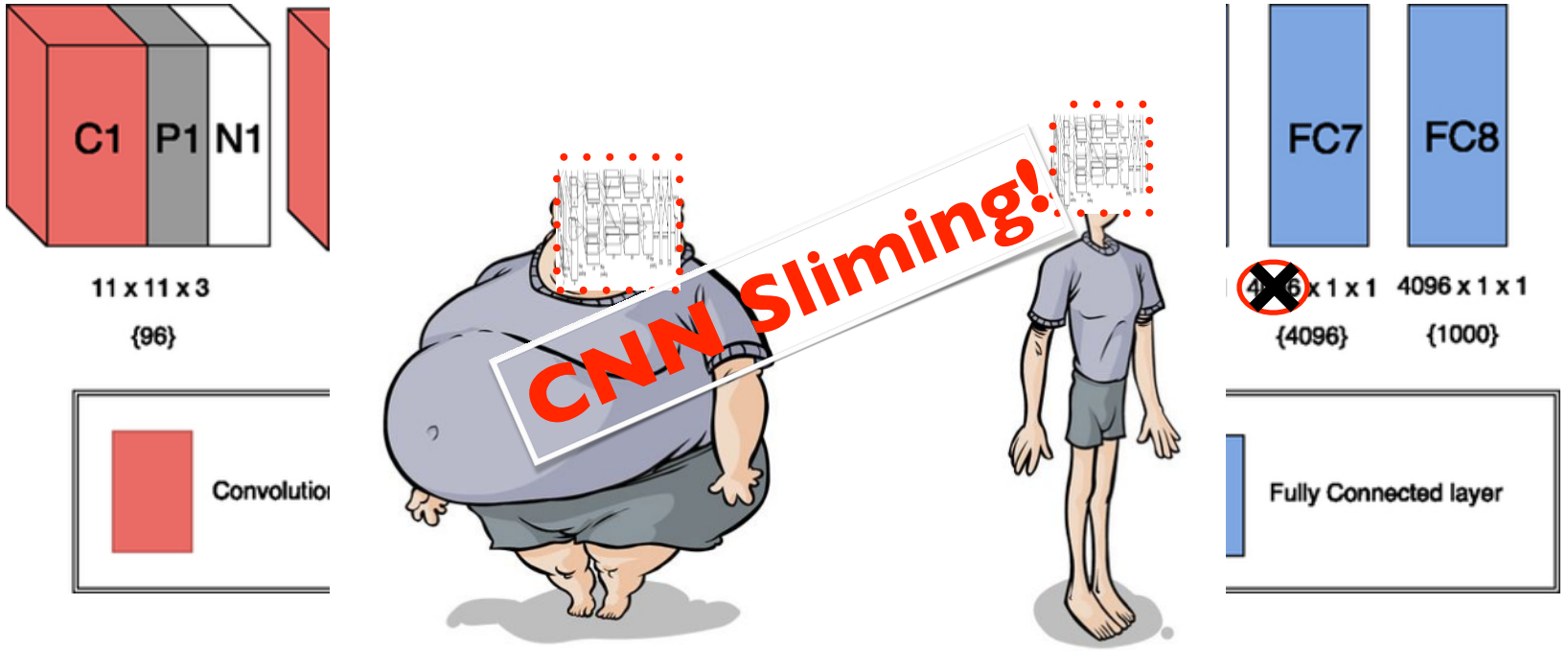
Speeding up CNN

+



Thinking Differently

- All existing methods can be viewed as approximations of an overly-redundant CNN, but do we really need such a CNN as the starting

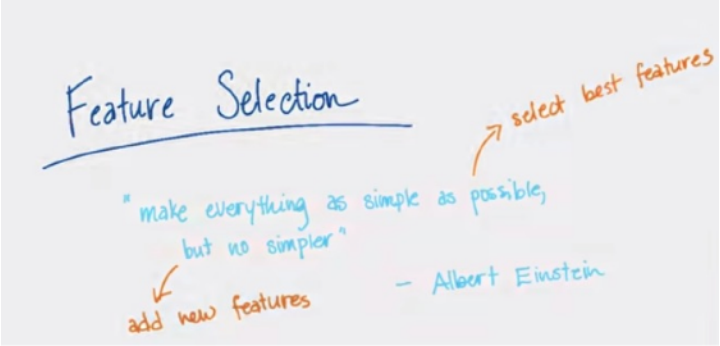


Slim CNN

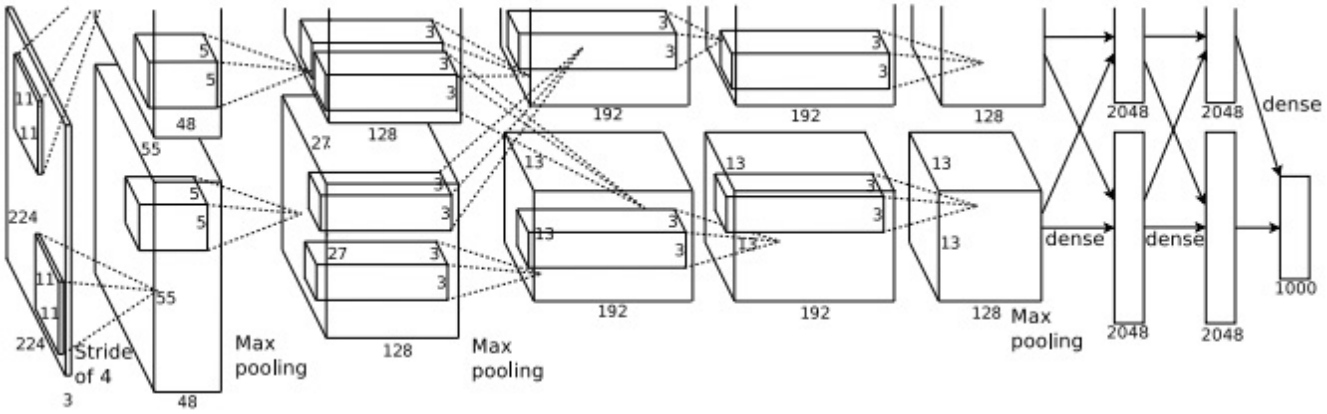
- Slim CNN leads to:
  - less storage space
  - less memory usage
  - less computation
  - less power consumption

Feature Selection on CNN

- CNNs can be viewed as a set of "overly-redundant" feature

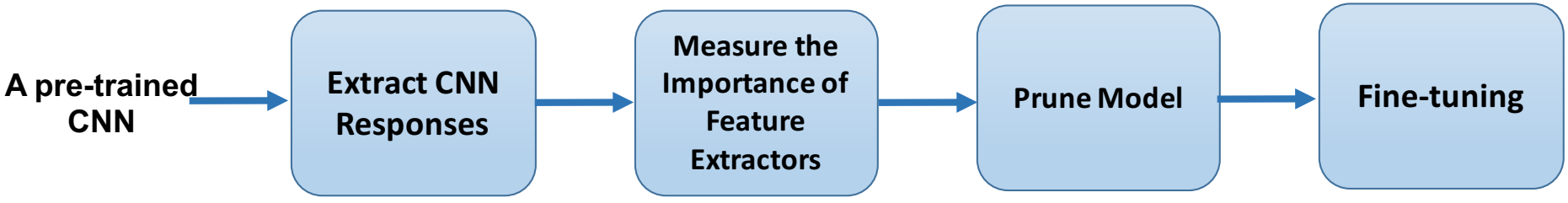


features



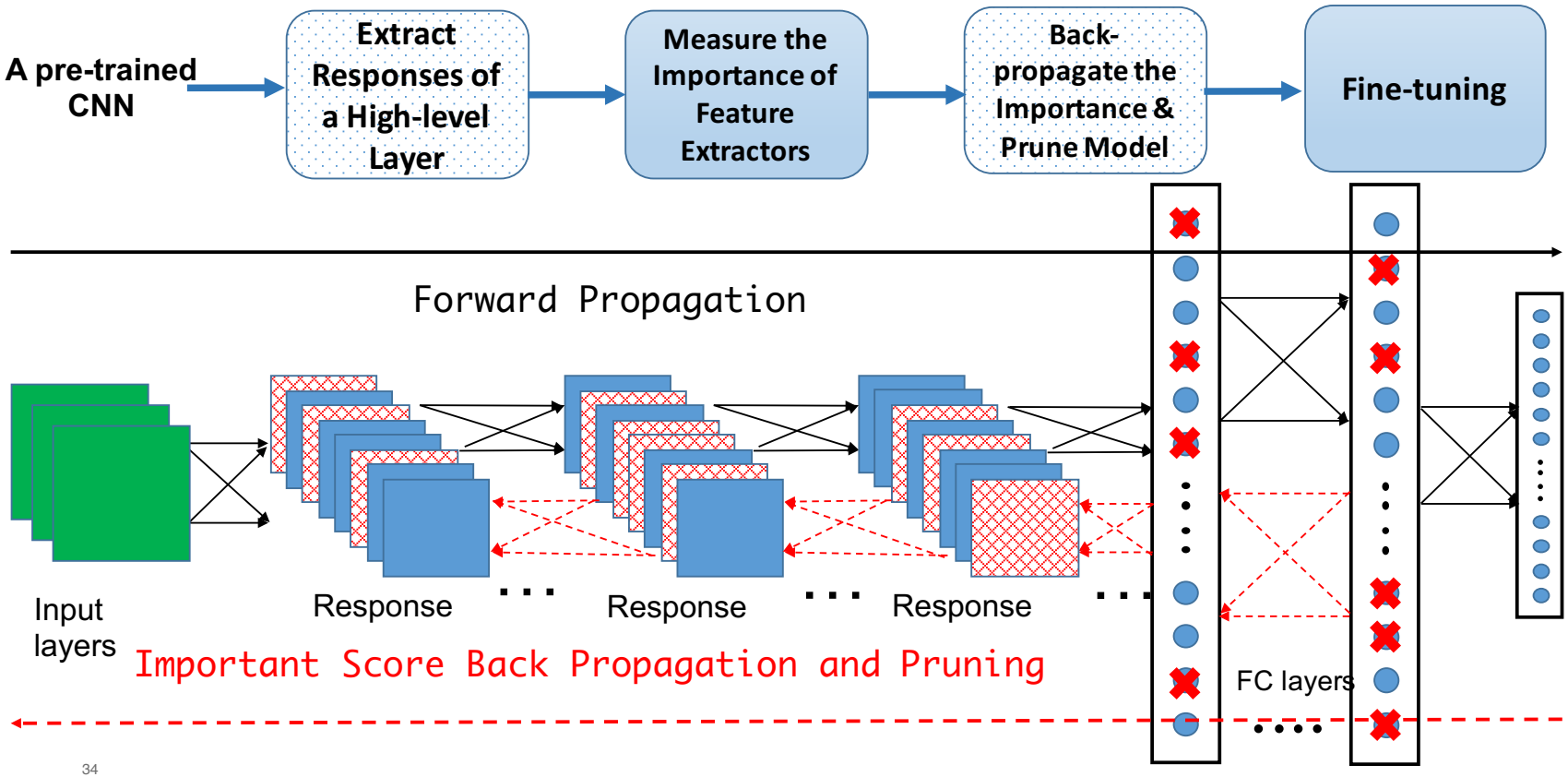
A method for Pruning Redundant Neurons and Kernels of

Apply thermal



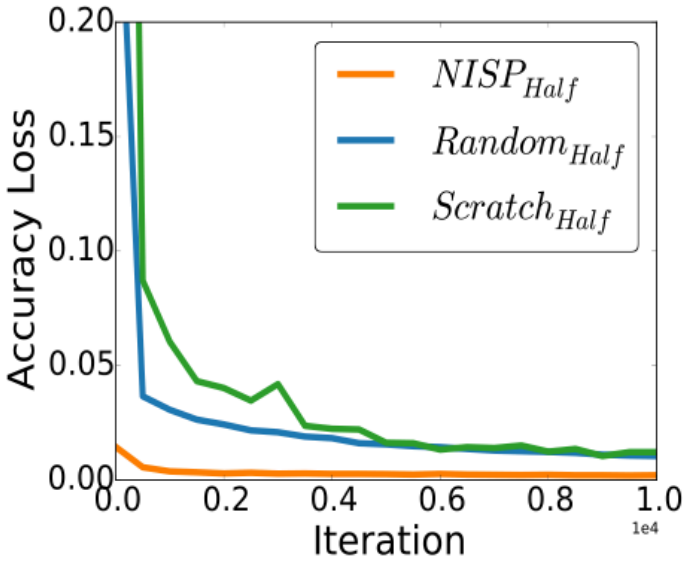
A method for Pruning Redundant Neurons and Kernels of Deep Convolutional Neural Networks (NISP)

- Intractable ⟶ tractable
- Inconsistent ⟶ consistent



Fine-tuning the Pruned Model

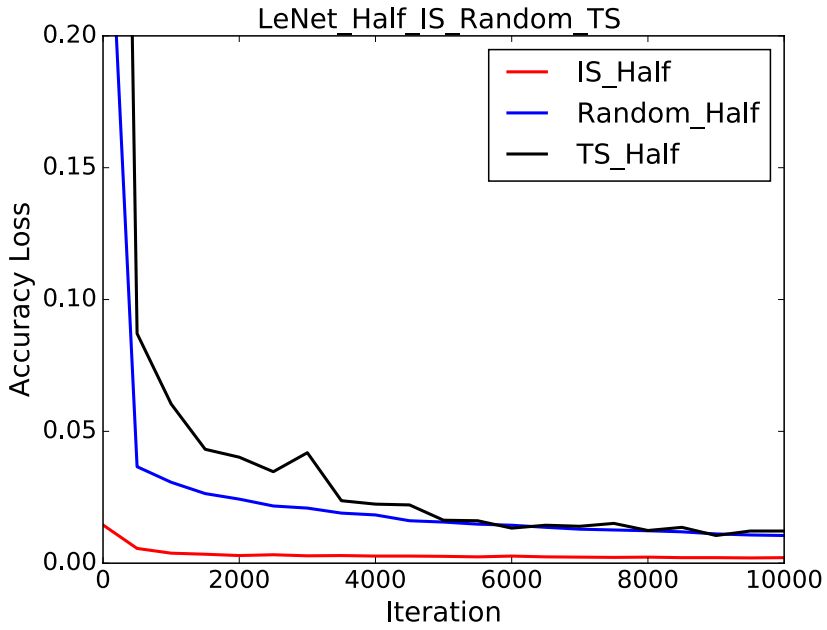
- Our method outperforms the baselines in three aspects
  - Very small accuracy loss at the beginning ==> retains the most important neurons
  - Converges much faster than baselines
  - For LeNet on MIST, our method only decreases 0.02% top-1 accuracy with a running ratio of 50% as compared to the pre-pruned network.

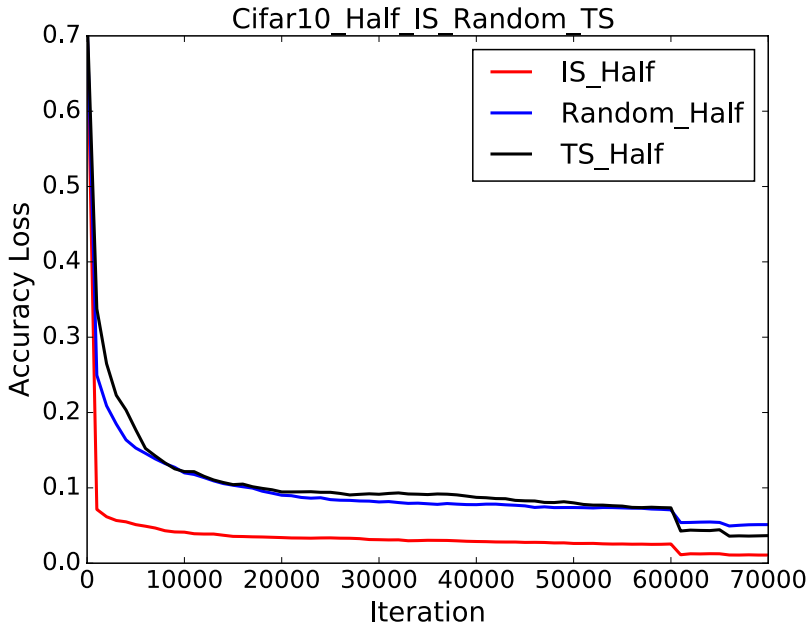


(a) MNIST

Fine-tuning the Pruned Model

- The pruned model consists of important feature extractors, but will suffer loss of accuracy due to loss of redundant features
  - Good starting point on the learning curve due to feature selection
  - Fine-tuning the pruned model with a lower learning rate to recover the performance





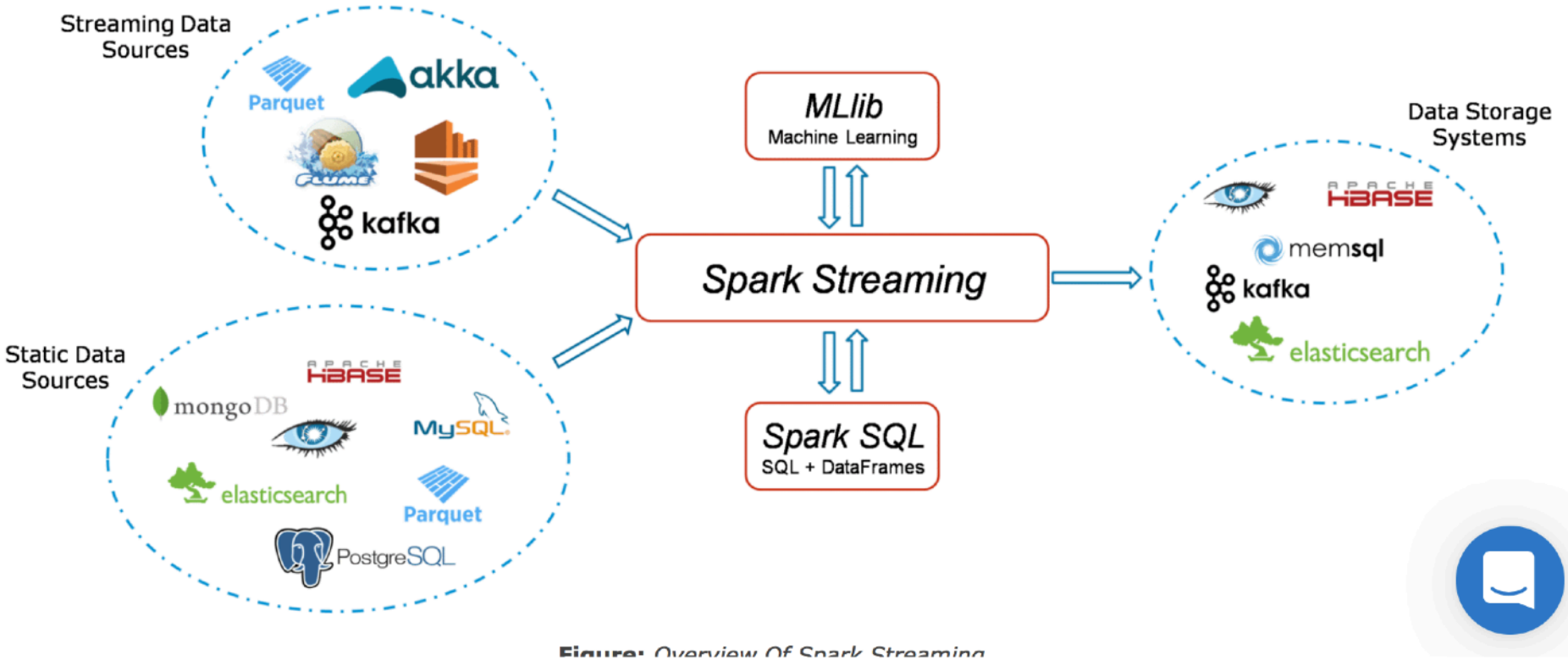
Spark Streaming

- Basic Concepts
  - Linking
  - Initializing StreamingContext
  - Discretized Streams (DStreams)
  - Input DStreams and Receivers
  - Transformations on DStreams
  - Output Operations on DStreams
  - DataFrame and SQL Operations
  - MLlib Operations
  - Caching / Persistence
  - Checkpointing
  - Accumulators, Broadcast Variables, and Checkpoints
  - Deploying Applications
  - Monitoring Applications
- Performance Tuning
  - Reducing the Batch Processing Times
  - Setting the Right Batch Interval
  - Memory Tuning
- Fault-tolerance Semantics

Spark Streaming

Spark Streaming

Spark Streaming



https://www.edureka.co/blog/spark-streaming/

Spark Streaming Example

COLUMBIA UNIVERSITY

First, we import StreamingContext, which is the main entry point for all streaming functionality. We create a local StreamingContext with two execution threads, and batch interval of 1 second.

```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

# Create a local StreamingContext with two working thread and batch interval of 1 second
sc = SparkContext("local[2]", "NetworkWordCount")
ssc = StreamingContext(sc, 1)
```

Using this context, we can create a DStream that represents streaming data from a TCP source, specified as hostname (e.g. `localhost`) and port (e.g. `9999`).

```
# Create a DStream that will connect to hostname:port, like localhost:9999
lines = ssc.socketTextStream("localhost", 9999)
```

This `lines` DStream represents the stream of data that will be received from the data server. Each record in this DStream is a line of text. Next, we want to split the lines by space into words.

```
# Split each line into words
words = lines.flatMap(lambda line: line.split(" "))
```

```
# Count each word in each batch
pairs = words.map(lambda word: (word, 1))
wordCounts = pairs.reduceByKey(lambda x, y: x + y)

# Print the first ten elements of each RDD generated in this DStream to the console
wordCounts.pprint()
```

Spark Streaming Example

```
$ ./bin/spark-submit examples/src/main/python/streaming/network_wordcount.py localhost 9999
```

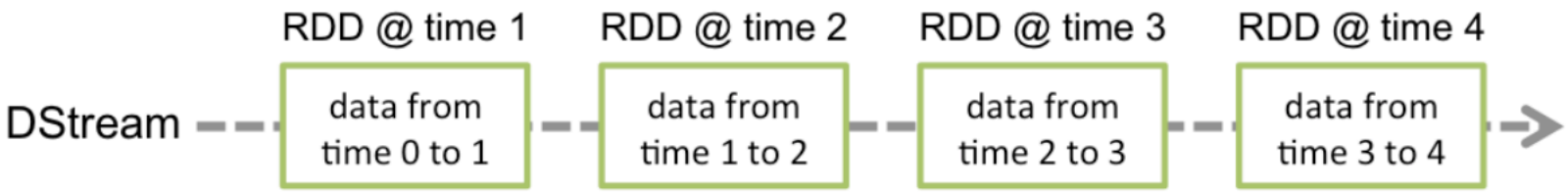
Then, any lines typed in the terminal running the netcat server will be counted and printed on screen every second. It will look something like the following.

Scala Java **Python**

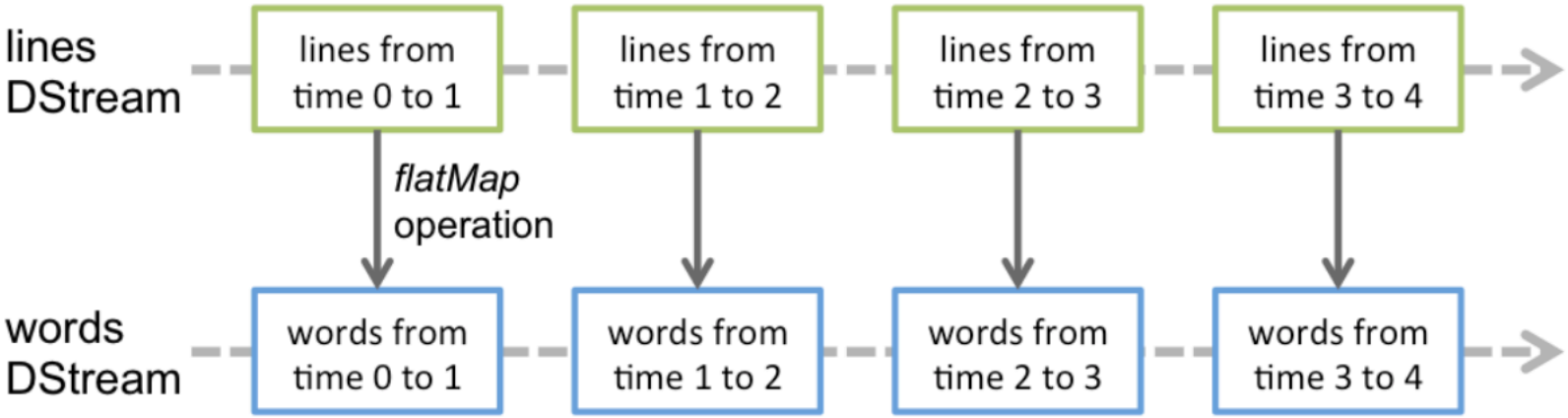
```
# TERMINAL 1:
# Running Net
cat

$ nc -lk 9999

hello world



...
```

```
# TERMINAL 2: RUNNING network_wordcount.py

$ ./bin/spark-submit examples/src/main/python/streaming/network_wordcount.py local
host 9999
...
-------------------------------------------
Time: 2014-10-14 15:25:21
-------------------------------------------
(hello,1)
(world,1)
...
```

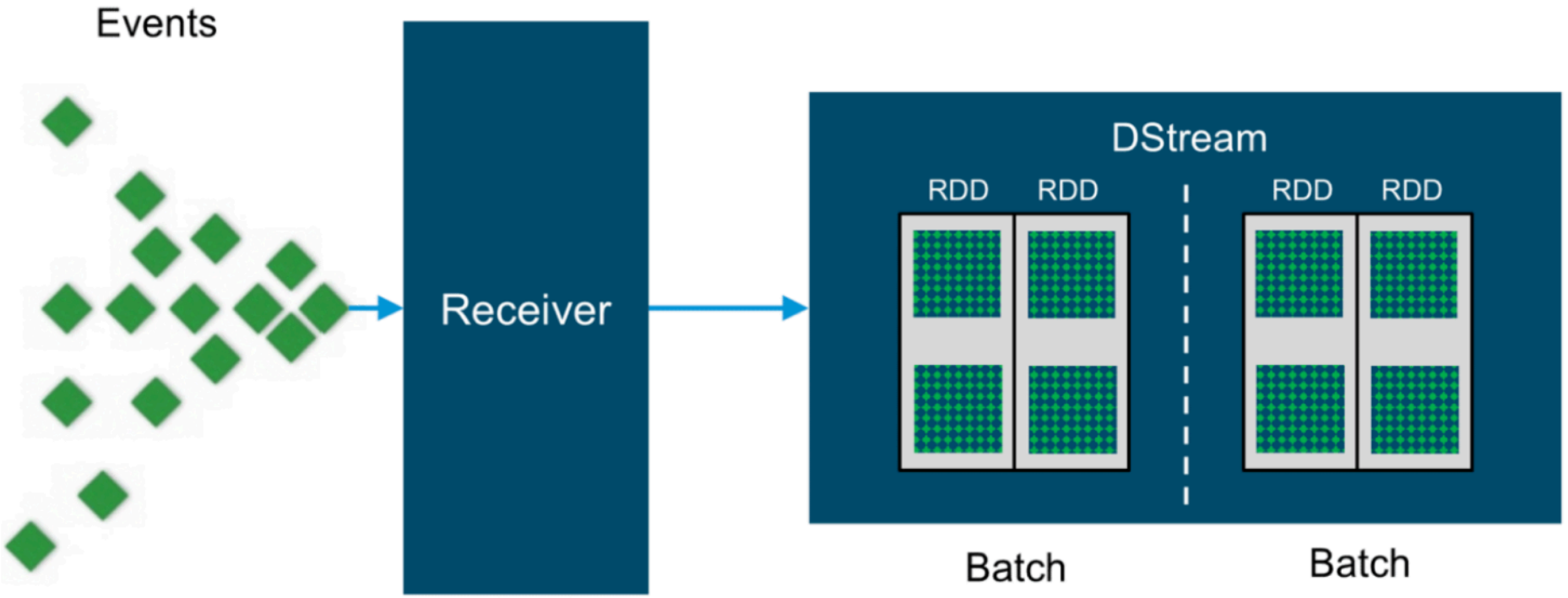
Discretized Streams



Discretized Streams

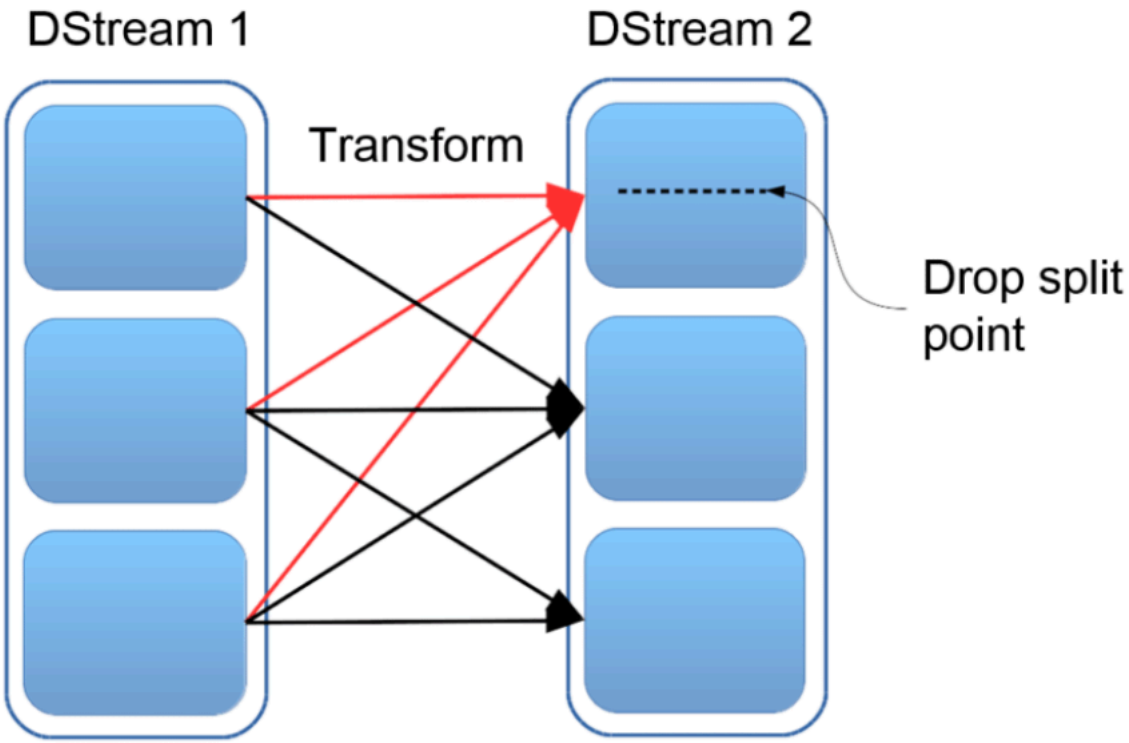


Discretized Streams



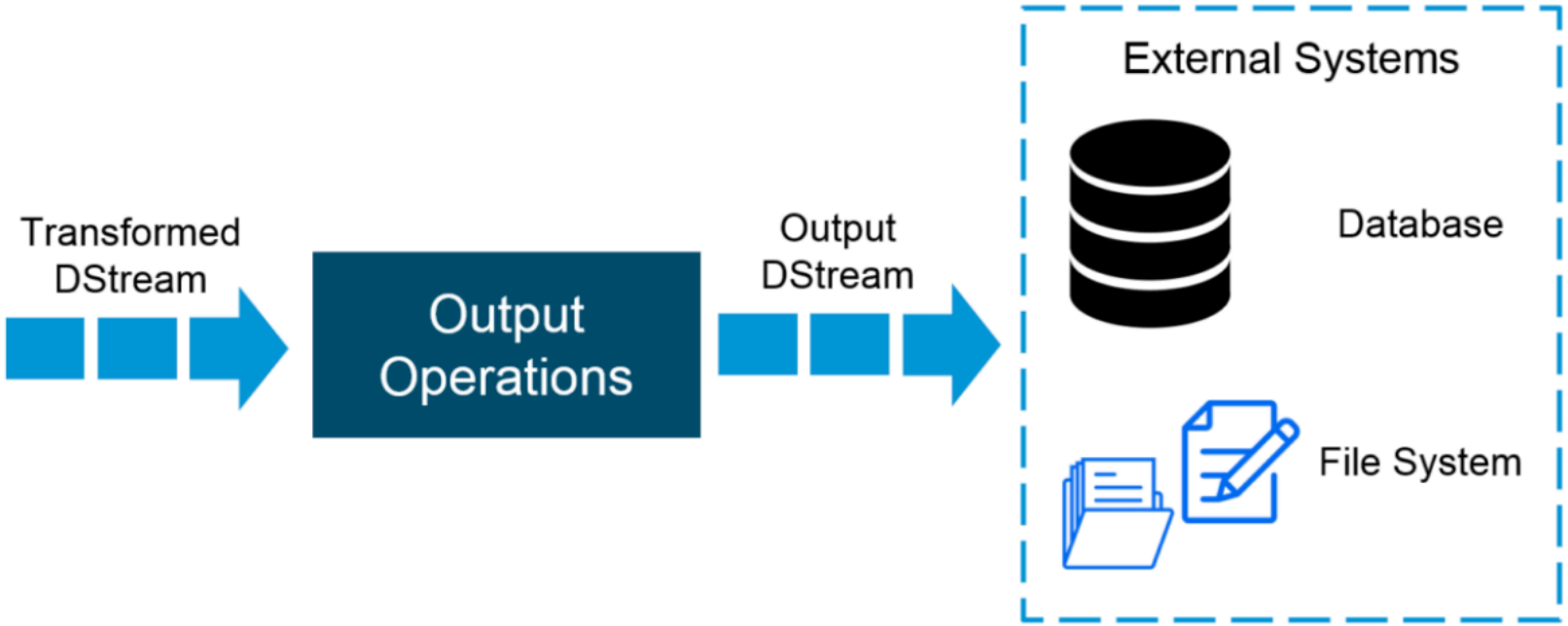
https://www.edureka.co/blog/spark-streaming/

DStream Transforms



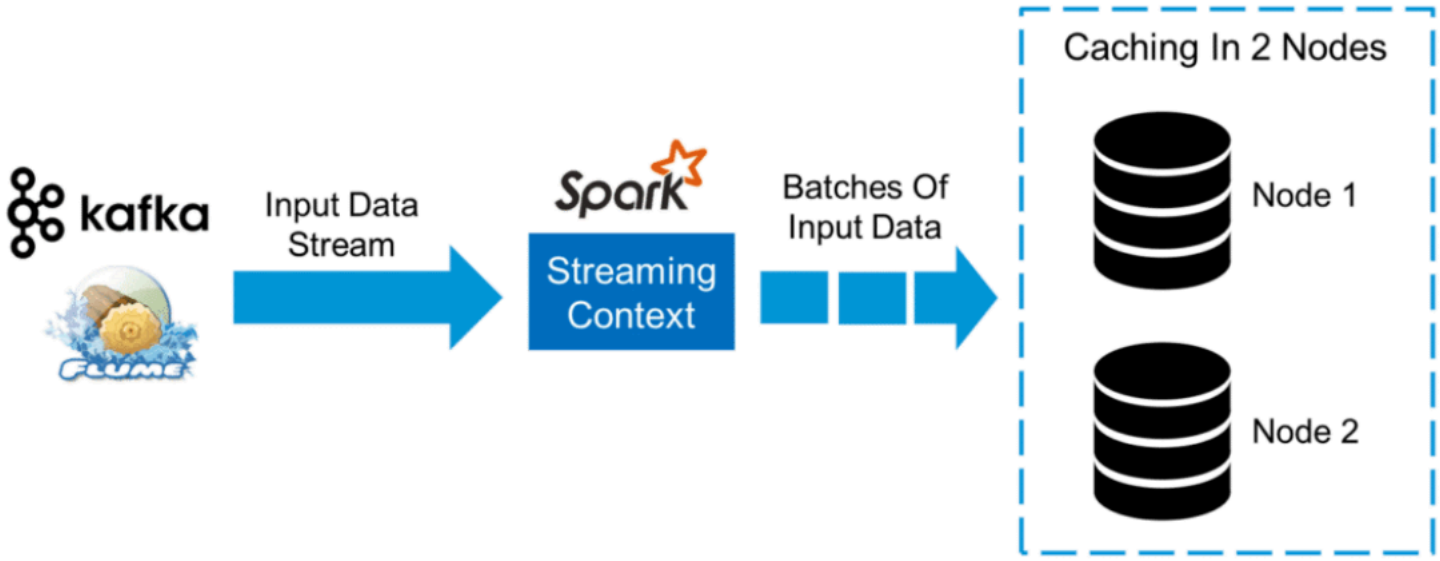
https://www.edureka.co/blog/spark-streaming/

Output DStreams



https://www.edureka.co/blog/spark-streaming/

DStreams Caching



https://www.edureka.co/blog/spark-streaming/

DStreams Example — Twitter Sentiment Analysis

```
//Import the necessary packages into the Spark Program
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.SparkContext._
...
import java.io.File

object twitterSentiment {

def main(args: Array[String]) {
if (args.length < 4) {
System.err.println("Usage: TwitterPopularTags <consumer key> <consumer secret> " + "<access token> <access token secret>
System.exit(1)
}

StreamingExamples.setStreamingLogLevels()
//Passing our Twitter keys and tokens as arguments for authorization
val Array(consumerKey, consumerSecret, accessToken, accessTokenSecret) = args.take(4)
val filters = args.takeRight(args.length - 4)

// Set the system properties so that Twitter4j library used by twitter stream
// Use them to generate OAuth credentials
System.setProperty("twitter4j.oauth.consumerKey", consumerKey)
...
System.setProperty("twitter4j.oauth.accessTokenSecret", accessTokenSecret)

val sparkConf = new SparkConf().setAppName("twitterSentiment").setMaster("local[2]")
val ssc = new Streaming Context
val stream = TwitterUtils.createStream(ssc, None, filters)
```

https://www.edureka.co/blog/spark-streaming/

DStreams Example — Twitter Sentiment Analysis

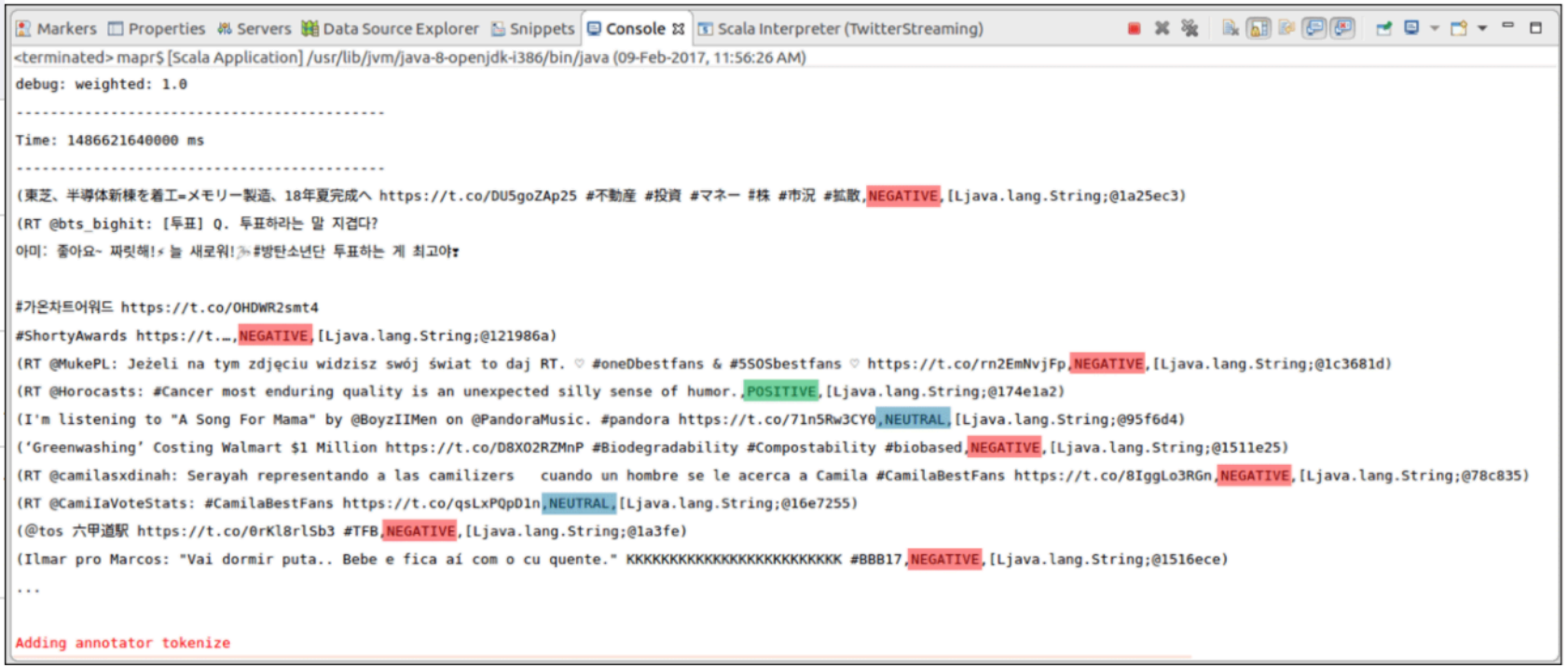
```
//Input DStream transformation using flatMap
val tags = stream.flatMap { status => Get Text From The Hashtags }

//RDD transformation using sortBy and then map function
tags.countByValue()
.foreachRDD { rdd =>
val now = Get current time of each Tweet
rdd
.sortBy(_._2)
.map(x => (x, now))
//Saving our output at ~/twitter/ directory
.saveAsTextFile(s"~/twitter/$now")
}

//DStream transformation using filter and map functions
val tweets = stream.filter {t =>
val tags = t. Split On Spaces .filter(_.startsWith("#")). Convert To Lower Case
tags.exists { x => true }
}

val data = tweets.map { status =>
val sentiment = SentimentAnalysisUtils.detectSentiment(status.getText)
val tagss = status.getHashtagEntities.map(_.getText.toLowerCase)
(status.getText, sentiment.toString, tagss.toString())
}

data.print()
//Saving our output at ~/ with filenames starting like twitters
data.saveAsTextFiles("~/twitters","20000")

ssc.start()
ssc.awaitTermination()
  }
}
```

https://www.edureka.co/blog/spark-streaming/

DStreams Example — Twitter Sentiment Analysis

Results:

The following are the results that are displayed in the Eclipse IDE while running the Twitter Sentiment Streaming program.



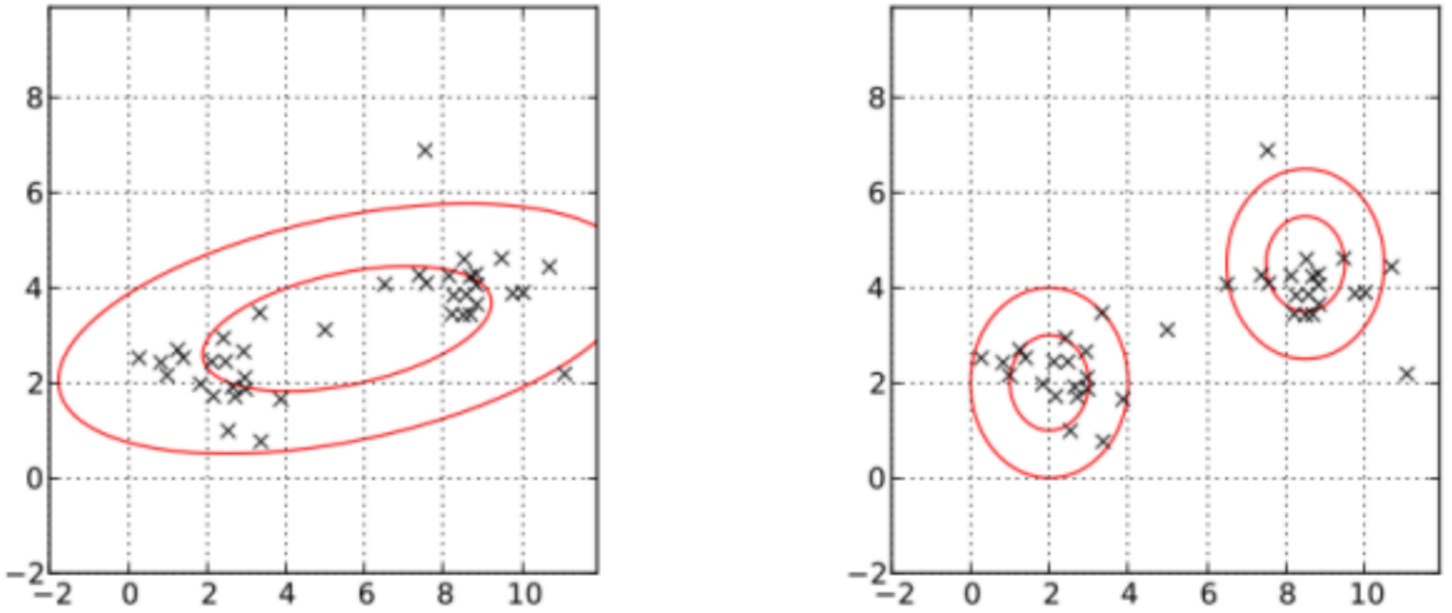
All the tweets are categorized into Positive, Neutral and Negative according to the sentiment of the contents of the tweets

https://www.edureka.co/blog/spark-streaming/

Spark Clustering

- K-means
  - Input Columns
  - Output Columns
- Latent Dirichlet allocation (LDA)
- Bisecting k-means
- Gaussian Mixture Model (GMM)
  - Input Columns
  - Output Columns

Clustering — Gaussian Mixture Models



(Left) Fit with one Gaussian distribution (Right) Fit with Gaussian mixture model with two components

Gaussian Mixture Model

Columbia University

One-dimensional Model

$$p(x) = \sum_{i=1}^{K} \phi_i \mathcal{N}(x \mid \mu_i, \sigma_i)$$

$$\mathcal{N}(x \mid \mu_i, \sigma_i) = \frac{1}{\sigma_i \sqrt{2\pi}} \exp\left(- \frac{(x - \mu_i)^2}{2\sigma_i^2} \right)$$

$$\sum_{i=1}^{K} \phi_i = 1$$

Gaussian Mixture Model

Columbia University

Multi-dimensional Model

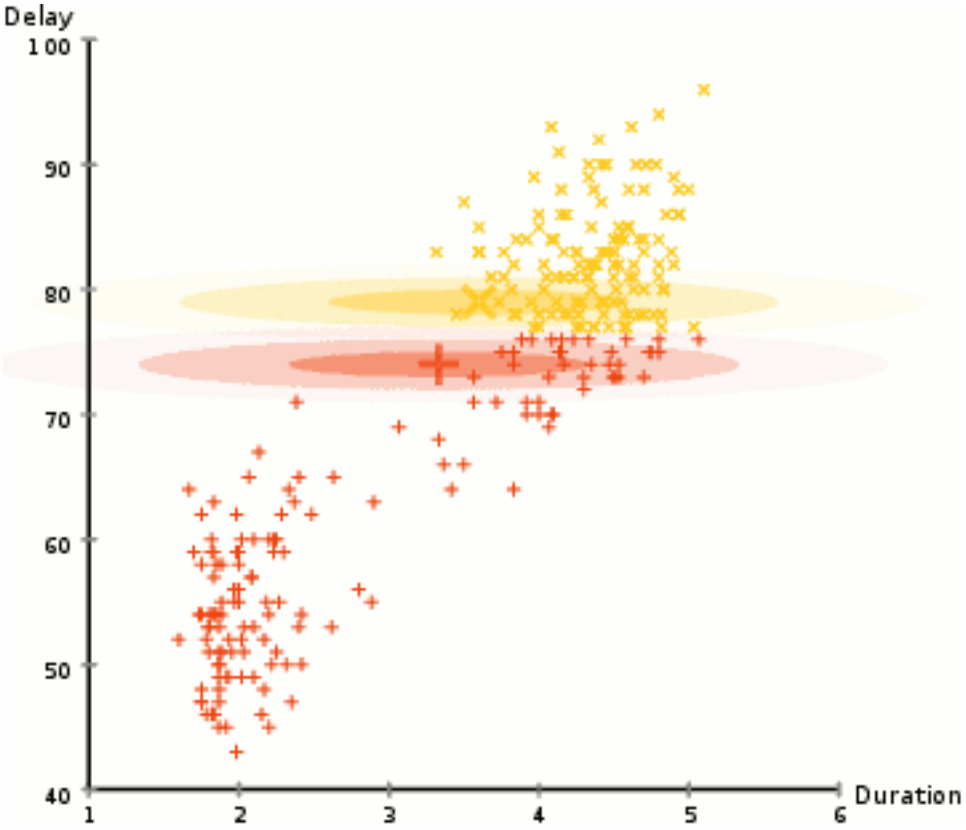
$$p(\vec{x}) = \sum_{i=1}^{K} \phi_i \mathcal{N}(\vec{x} \mid \vec{\mu}_i, \Sigma_i)$$

$$\mathcal{N}(\vec{x} \mid \vec{\mu}_i, \Sigma_i) = \frac{1}{\sqrt{(2\pi)^K |\Sigma_i|}} \exp\left(-\frac{1}{2} (\vec{x} - \vec{\mu}_i)^{\mathrm{T}} {\Sigma_i}^{-1} (\vec{x} - \vec{\mu}_i) \right)$$

$$\sum_{i=1}^{K} \phi_i = 1$$

Gaussian Mixture Model — EM

Expectation maximization (**EM**) is a numerical technique for maximum likelihood estimation, and is usually used when closed form expressions for updating the model parameters can be calculated (which will be shown below). Expectation maximization is an iterative algorithm and has the convenient property that the maximum likelihood of the data strictly increases with each subsequent iteration, meaning it is guaranteed to approach a local maximum or saddle point.



https://brilliant.org/wiki/gaussian-mixture-model/

Gaussian Mixture Model spark code

Input Columns

Param name	Type(s)	Default	Description
featuresCol	Vector	"features"	Feature vector

Output Columns

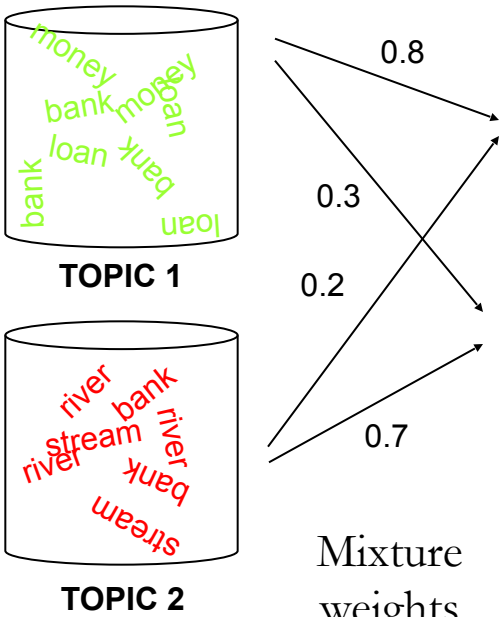
Param name	Type(s)	Default	Description
predictionCol	Int	"prediction"	Predicted cluster center
probabilityCol	Vector	"probability"	Probability of each cluster

```
from pyspark.ml.clustering import GaussianMixture

# loads data
dataset = spark.read.format("libsvm").load("data/mllib/sample_kmeans_data.txt")

gmm = GaussianMixture().setK(2).setSeed(538009335)
model = gmm.fit(dataset)

print("Gaussians shown as a DataFrame: ")
model.gaussiansDF.show(truncate=False)
```

Content Analysis - Latent Dirichlet Allocation (LDA) [Blei *et al.* 2003]

Goal – categorize the documents into topics

- Each document is a probability distribution over topics
- Each topic is a probability distribution over words



Mixture components

Mixture weights

DOCUMENT 1: money[1] bank[1] bank[1] loan[1] river[2] stream[2] bank[1] money[1] river[2] bank[1] money[1] bank[1] loan[1] money[1] stream[2] bank[1] money[1] loan[1] river[2] stream[2] bank[1] money[1]

DOCUMENT 2: loan[1] river[2] stream[2] loan[1] bank[2] river[2] bank[2] bank[1] stream[2] river[2] loan[1] bank[2] stream[2] bank[2] money[1] loan[1] river[2] stream[2] bank[2] stream[2] bank[2] money[1] river[2]

The probability of ith word in a given document

$$P(w_i) = \sum_{j=1}^{T} P(w_i \mid z_i = j) P(z_i = j)$$

The probability of the word w_i under the jth topic $\theta_j^{(d)}$

The probability of choosing a word from the jth topic in the current document $\phi_w^{(j)}$

LDA (cont.)

INPUT:

- document-word counts
  - D documents, W words

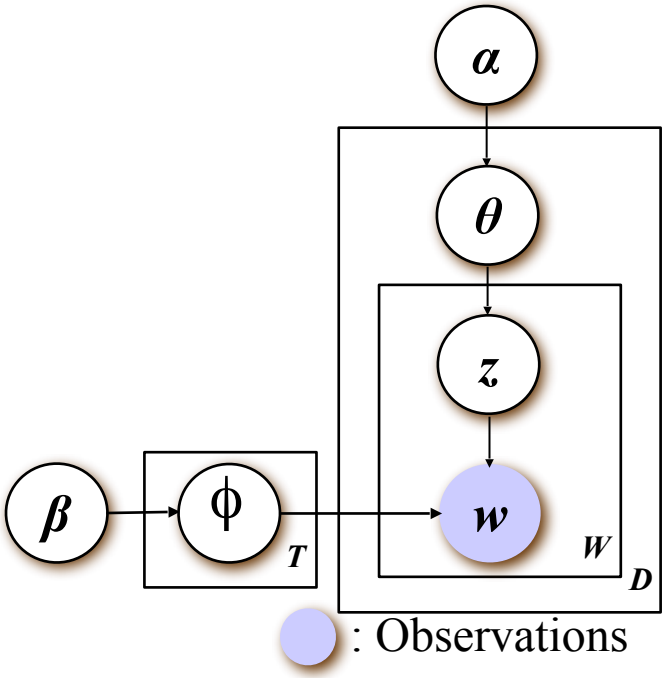
OUTPUT:

- likely topics for a document

$$P(z \mid w) \propto P(w \mid z)P(z)$$

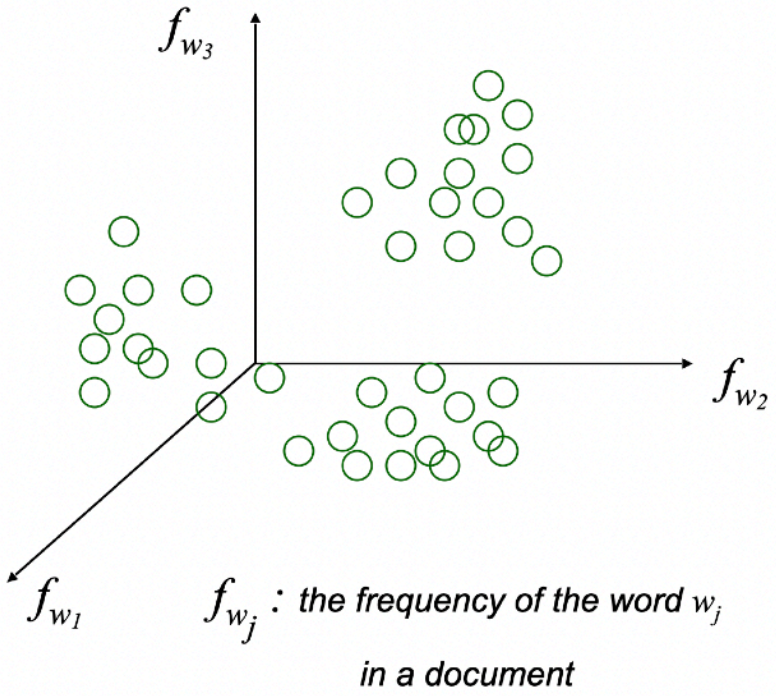
- Parameters can be estimated by Gibbs Sampling
- Outperform Latent Semantic Analysis (LSA) and Probabilistic LSA in various experiments [Blei *et al.* 2003]

Bayesian approach: use priors
Mixture weights ~ Dirichlet(α)
Mixture components ~ Dirichlet(β)



T: number of topics

Traditional Content Clustering



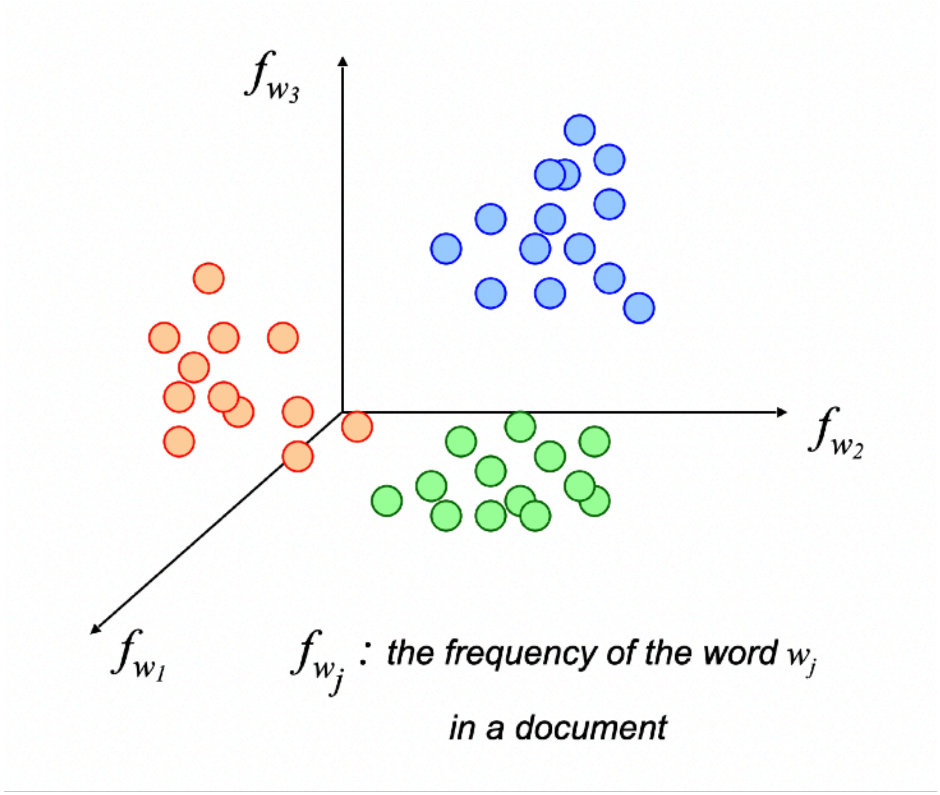
Clustering:

Partition the feature space into *segments* based on training documents. Each segment represents a topic / category. (🡐 Topic Detection)

Hard clustering: e.g., K-mean clustering

$$d = \{f_{w_1}, f_{w_2}, \ldots, f_{w_N}\} \rightarrow z$$

Traditional Content Clustering



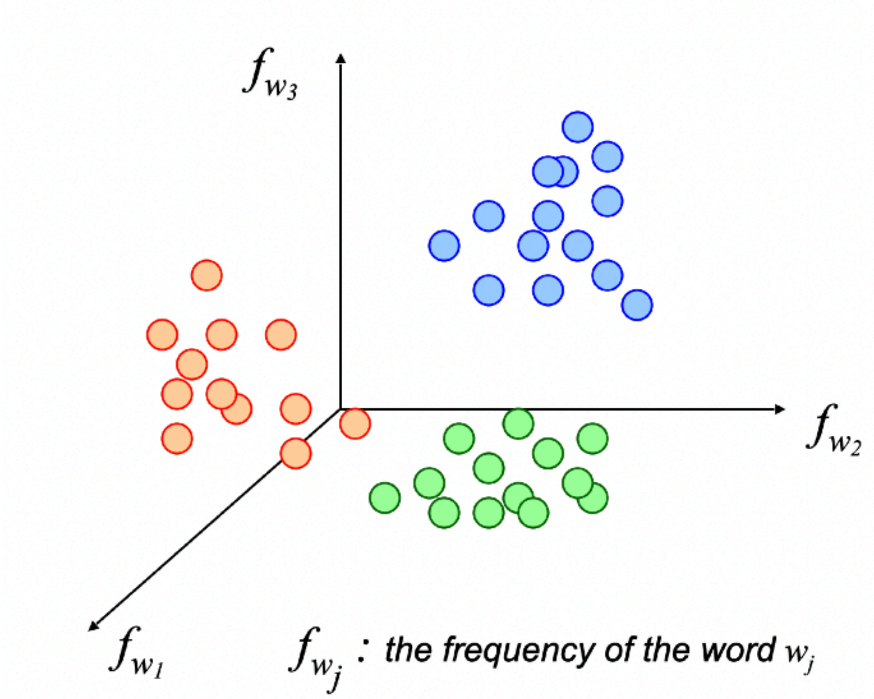
Clustering:

Partition the feature space into *segments* based on training documents. Each segment represents a topic / category. (🡐 Topic Detection)

Hard clustering: e.g., K-mean clustering

$$d = \{f_{w_1}, f_{w_2}, \ldots, f_{w_N}\} \rightarrow z$$

Traditional Content Clustering

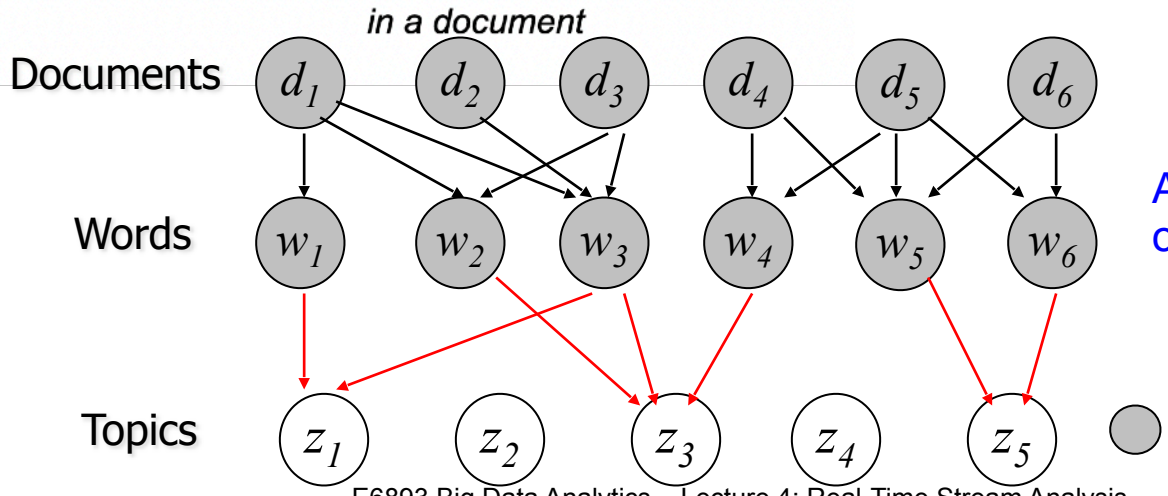


Clustering:

Partition the feature space into *segments* based on training documents. Each segment represents a topic / category. (🡐 Topic Detection)

Hard clustering: e.g., K-mean clustering

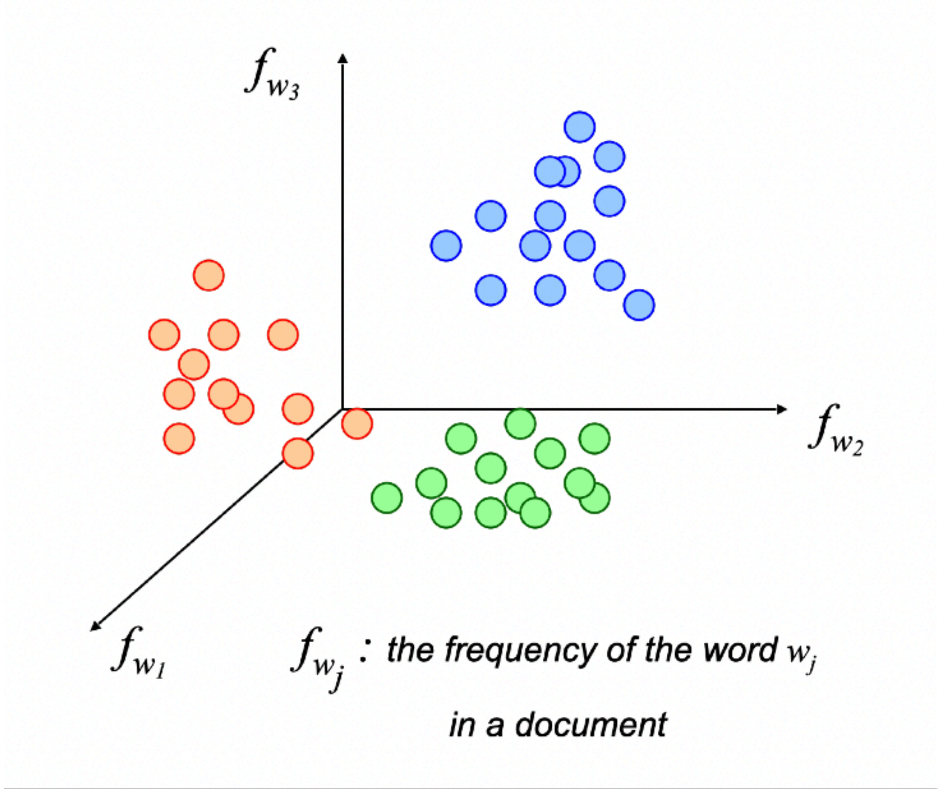
$$d = \{f_{w_1}, f_{w_2}, \ldots, f_{w_N}\} \rightarrow z$$



Another representation of clustering

: observations

Traditional Content Clustering

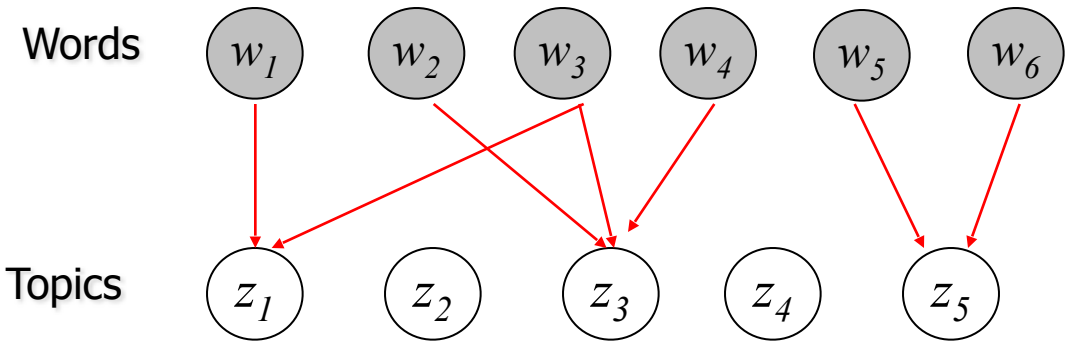


Clustering:

Partition the feature space into *segments* based on training documents. Each segment represents a topic / category. (🡐 Topic Detection)

Hard clustering: e.g., K-mean clustering

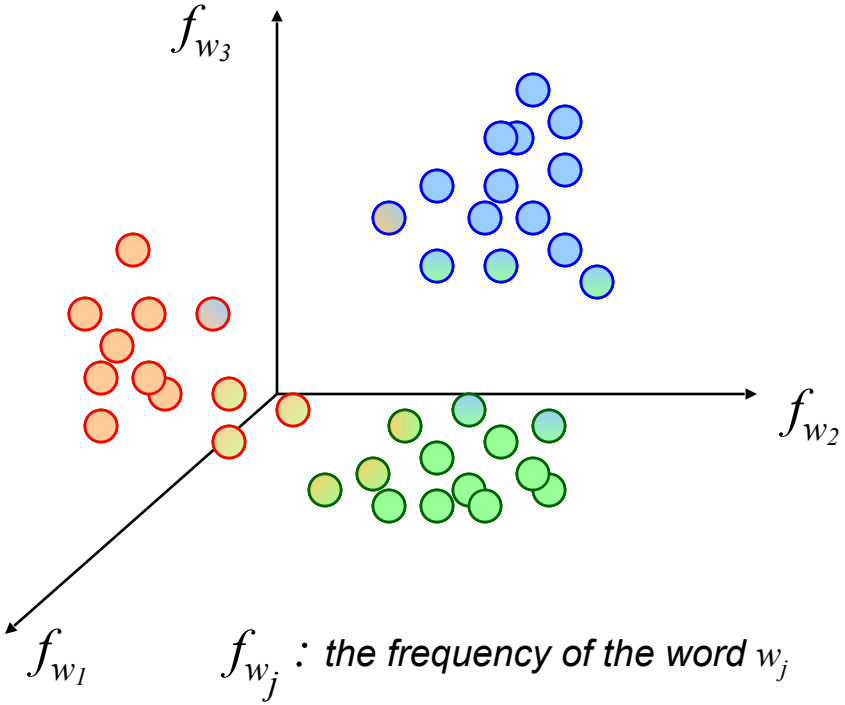
$$d = \{f_{w_1}, f_{w_2}, \ldots, f_{w_N}\} \rightarrow z$$



Another representation of clustering (w/o showing the deterministic part)

: observations

Traditional Content Clustering — soft clustering



Clustering:

Partition the feature space into *segments* based on training documents. Each segment represents a topic / category. (🡐 Topic Detection)

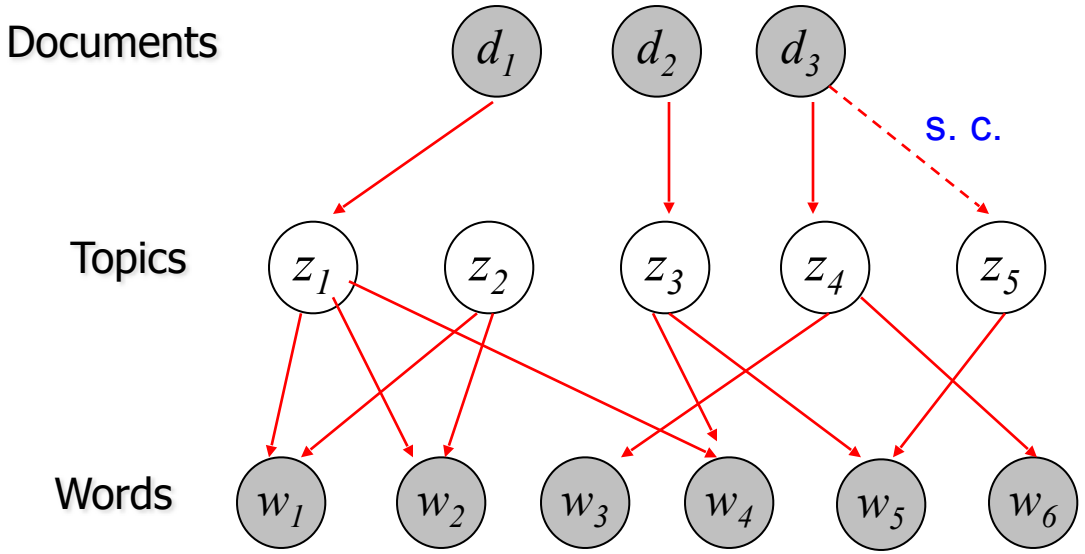
Hard clustering: e.g., K-mean clustering

$$d = \{f_{w_1}, f_{w_2}, \ldots, f_{w_N}\} \rightarrow z$$

Soft clustering: e.g., Fuzzy C-mean clustering

$$P(Z \mid \mathbf{W} = \mathbf{f}_{\mathbf{w}})$$

Content Clustering based on Bayesian Network



$h(D)$ hard clustering

$P(Z \mid D)$ soft clustering

$P(W \mid Z)$

: observations

Bayesian Network:

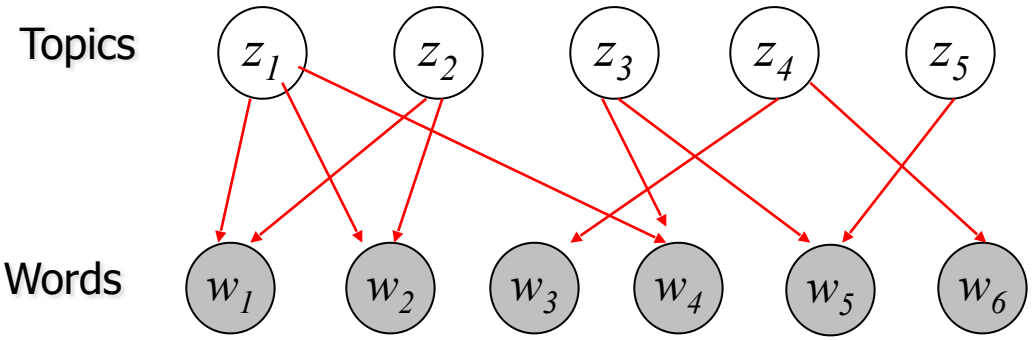
- Causality Network – models the causal relationship of attributes / nodes
- Allows hidden / latent nodes

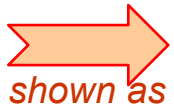
Hard clustering:

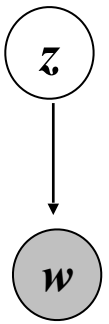
$$h(D = d) = \arg\max_{z} P(\mathbf{W} = \mathbf{f}_{\mathbf{w}} \mid Z)$$ <= MLE

$$P(W \mid Z) = \frac{P(Z \mid W)P(W)}{P(Z)}$$ <= Bayes Theorem

Content Clustering based on Bayesian Network – Hard Clustering







: observations

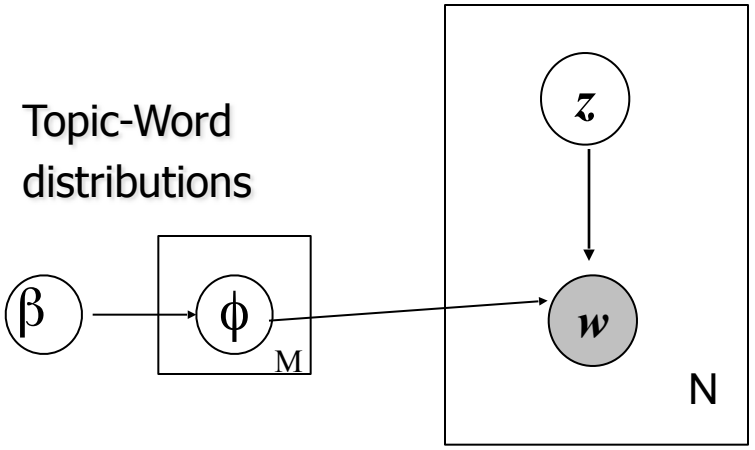
$$P(W \mid Z) = \frac{P(Z \mid W)P(W)}{P(Z)} = \frac{P(Z \mid W)P(W)}{\int P(Z \mid W)dW}$$

N: the number of words
(The number of topics (M) are pre-determined)

Major Solution 1 -- Dirichlet Process:

- Models P(W | Z) as mixtures of Dirichlet probabilities
- Before training, the prior of P(W|Z) can be a easy Dirichlet (uniform distribution). After training, P(W|Z) will still be Dirichlet. (← *The reason of using Dirichlet*)

Topic-Word distributions

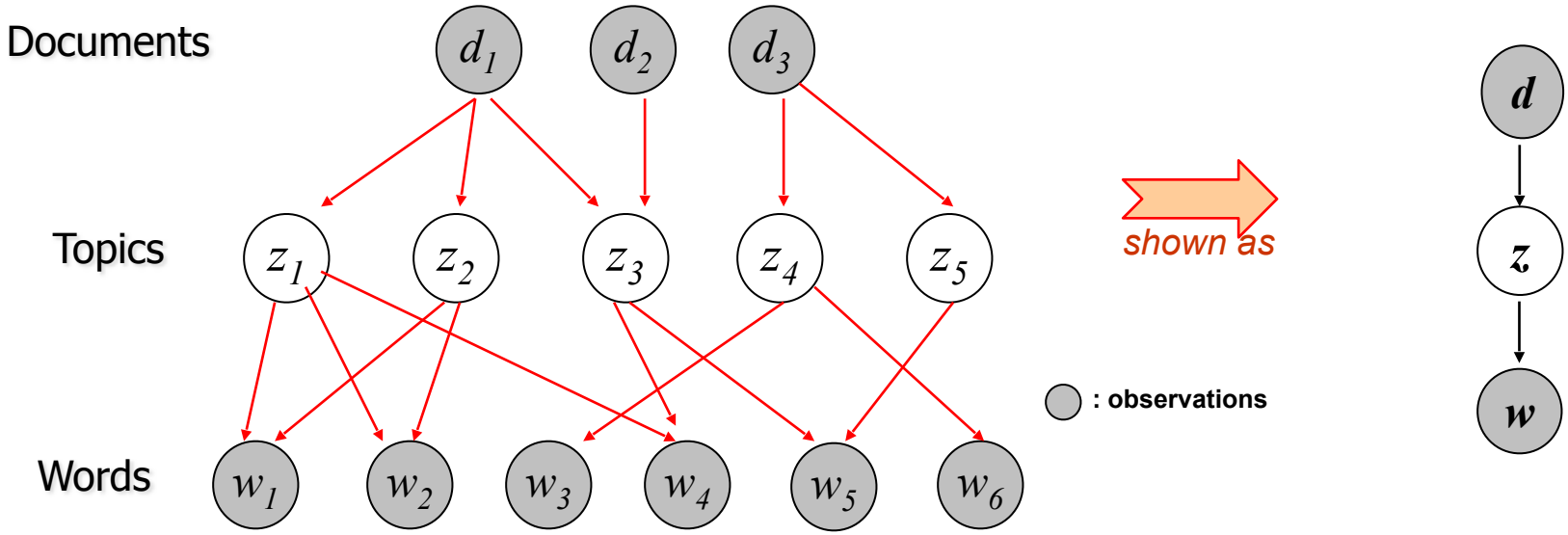


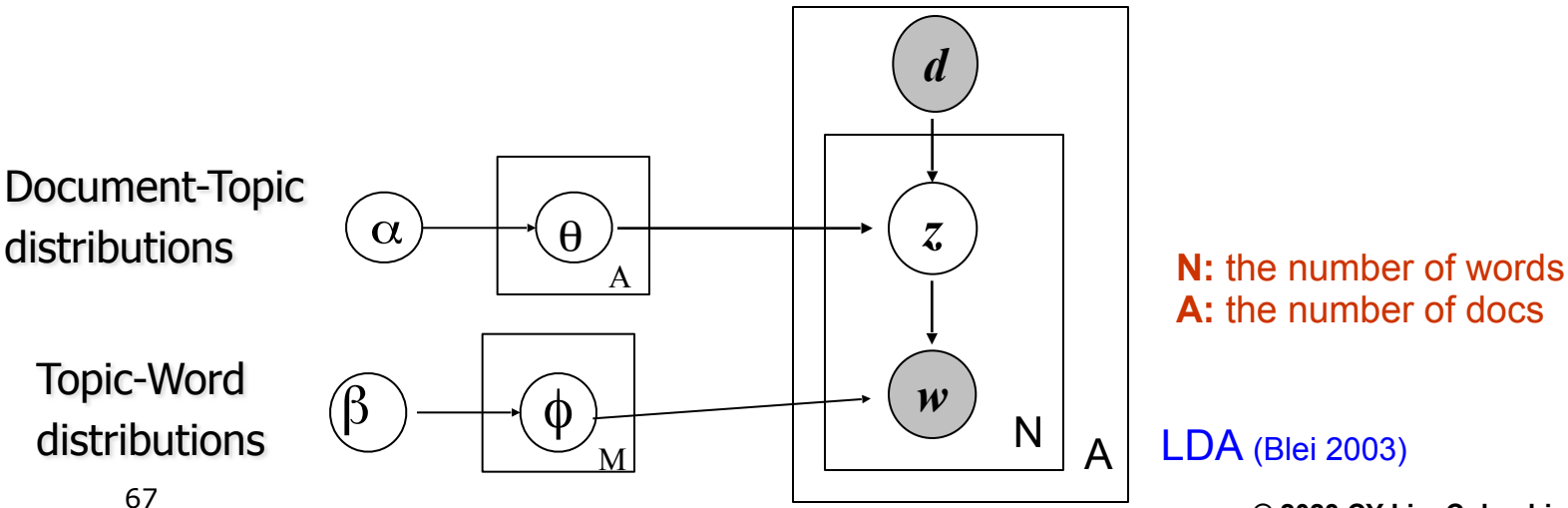
Latent Dirichlet Allocation (LDA) (Blei 2003)

Major Solution 2 -- Gibbs Sampling:

- A Markov chain Monte Carlo (MCMC) method for integration of large samples ➔ calculate P(Z)

Content Clustering based on Bayesian Network – Soft Clustering





N: the number of words
A: the number of docs

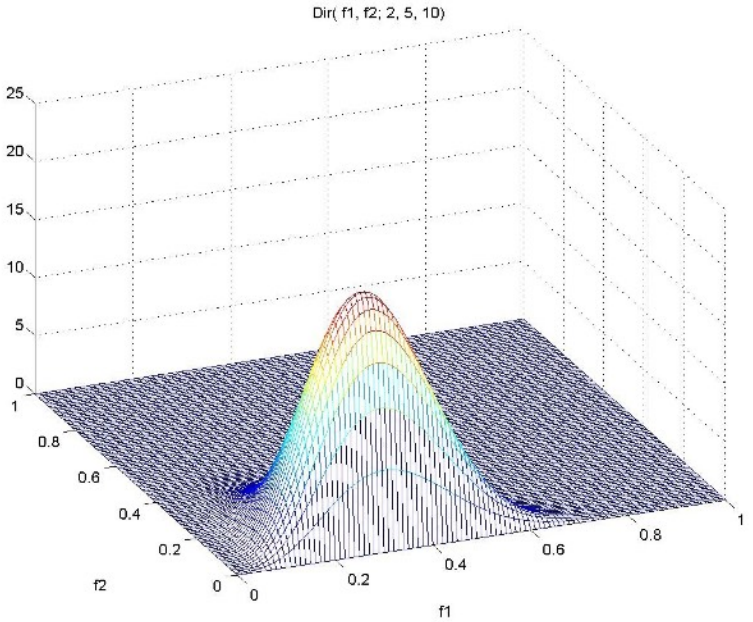
LDA (Blei 2003)

Comparison of Dirichlet Distribution with Gaussian Mixture Models (1)

Dirichlet Distribution:

$$\rho(f_1, f_2, ..., f_{r-1}; a_1, a_2, ..., a_r) = \frac{\Gamma(N)}{\prod_{k=1}^{r} \Gamma(a_k)} f_1^{a_1-1} f_2^{a_2-1} ... f_r^{a_r-1}$$

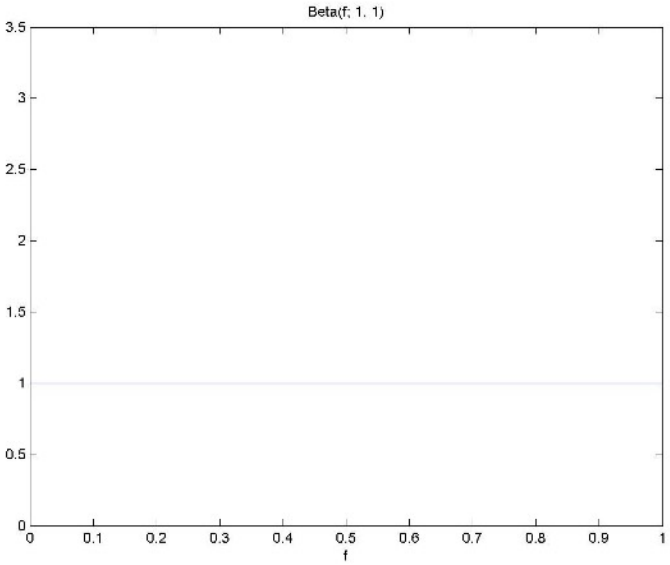
$$0 \le f_k \le 1 \qquad \sum_{k=1}^{r} f_k = 1$$

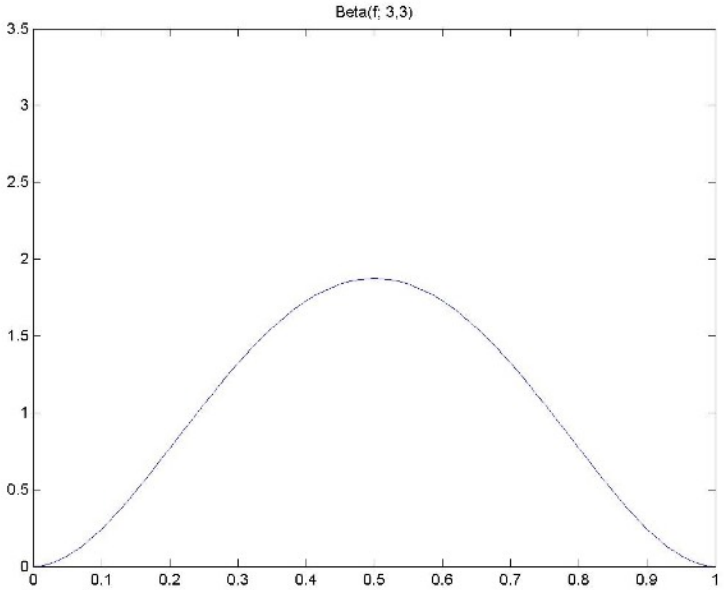


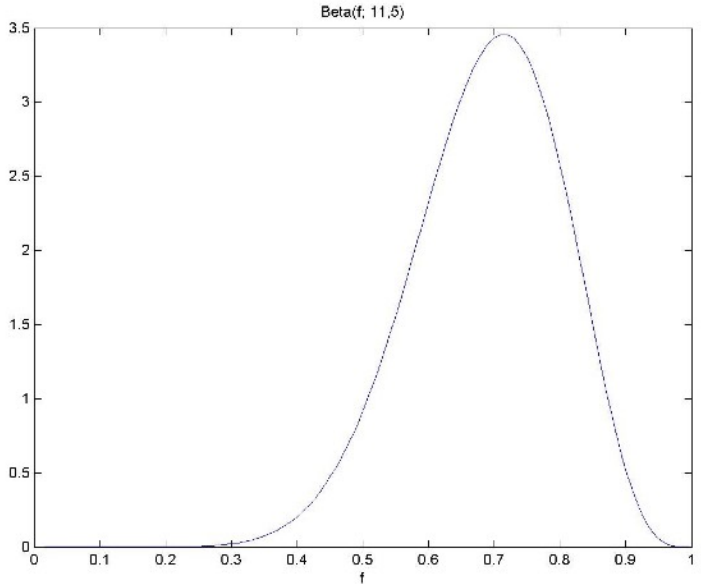
- Multivariate Gaussian:

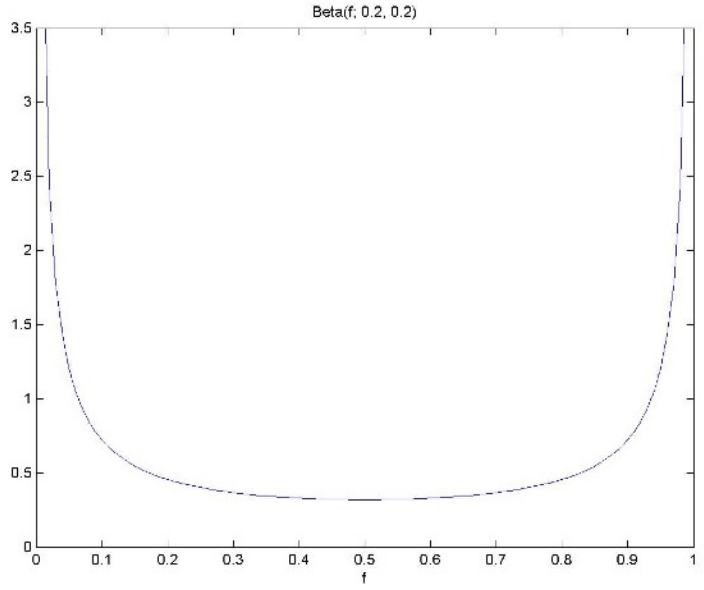
$$\rho(f_1, f_2, ..., f_{r-1}; \mu_1, \sigma_1, \mu_2, \sigma_2, ..., \mu_{r-1}, \sigma_{r-1}) = \frac{1}{(2\pi)^{r-1} \prod_{k=1}^{r-1} \sigma_k} e^{-\frac{(f_1-\mu_1)^2}{\sigma_1^2}} e^{-\frac{(f_2-\mu_2)^2}{\sigma_2^2}} ... e^{-\frac{(f_{r-1}-\mu_{r-1})^2}{\sigma_{r-1}^2}}$$

Beyond Gaussian: Examples of Dirichlet Distribution









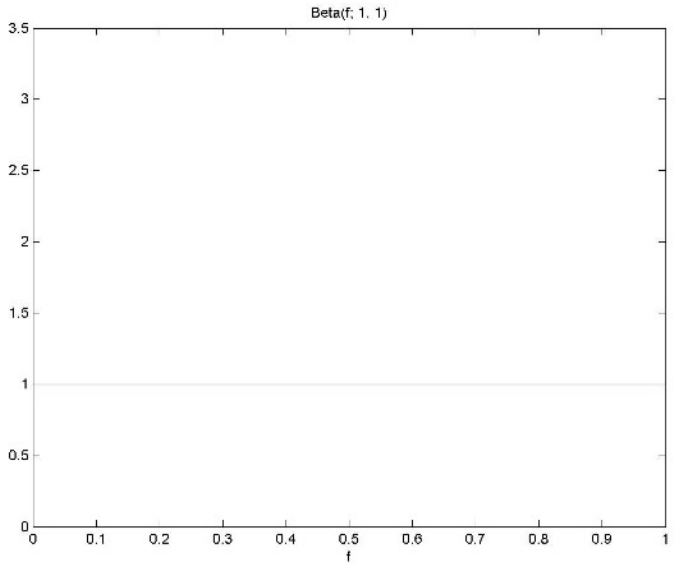
Use Dirichlet Distribution to model prior and posterior beliefs

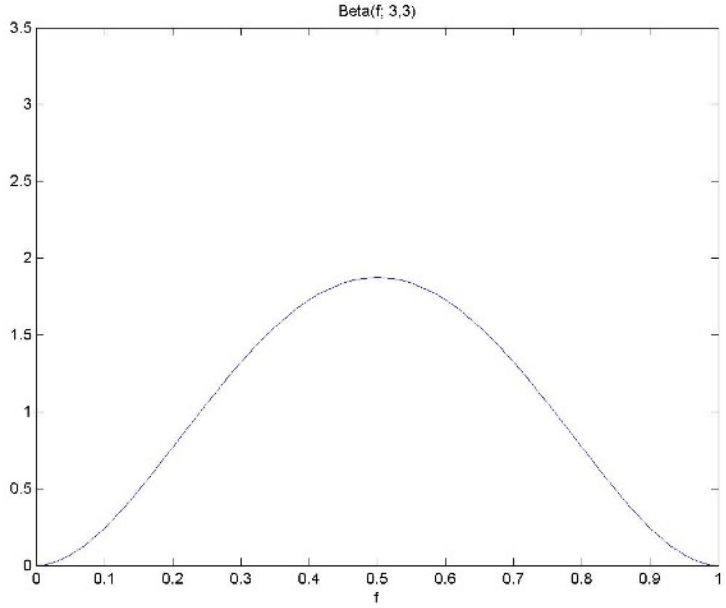
Prior beliefs:

E.g.: *fair* coin?

$$\rho(f) = beta(f; a, b)$$

Flipping a coin, what's the probability of getting 'head'.





beta(1,1): No prior knowledge

beta(3,3): prior knowledge -- this coin may be fair

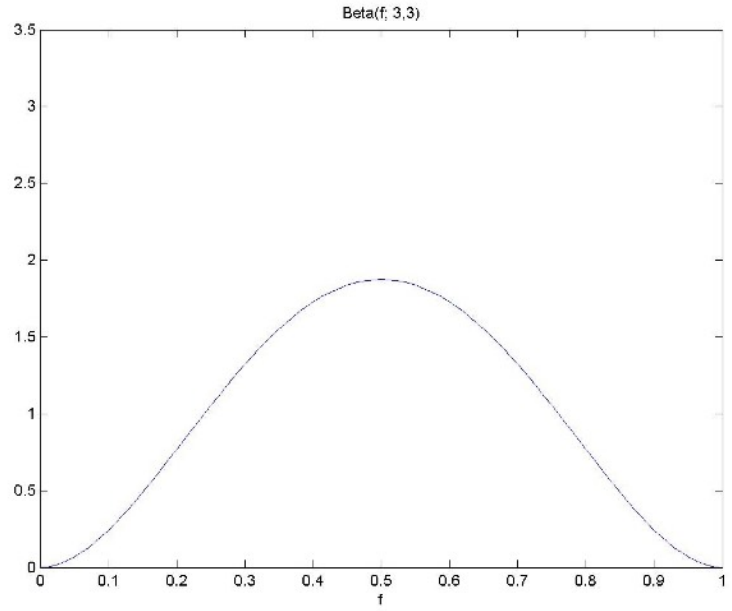
Use Dirichlet Distribution to model prior and posterior beliefs

Prior beliefs:

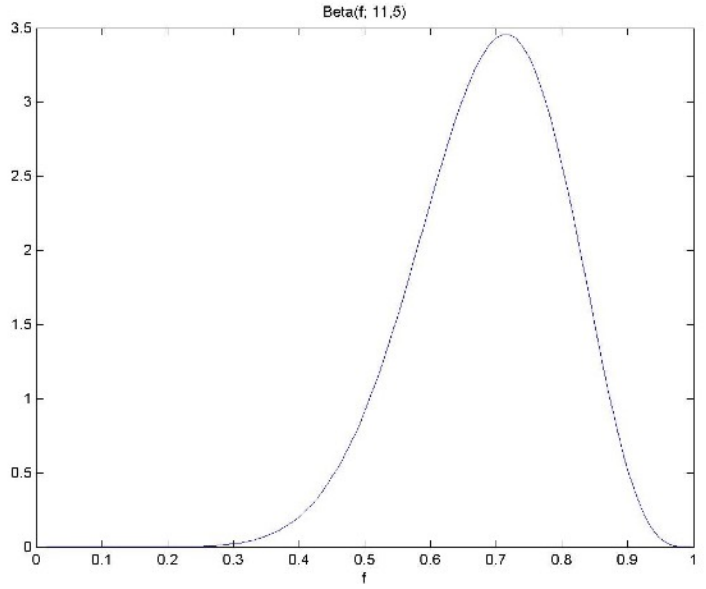
Posterior beliefs:

$$\rho(f) = beta(f; a, b)$$

$$\rho(f \mid d) = beta(f; a+s, b+t)$$



d: 8, 2



beta(3,3): prior knowledge -- this coin may be fair

beta(11,5): posterior belief -- the coin may not be fair after tossing 8 heads and 2 tails

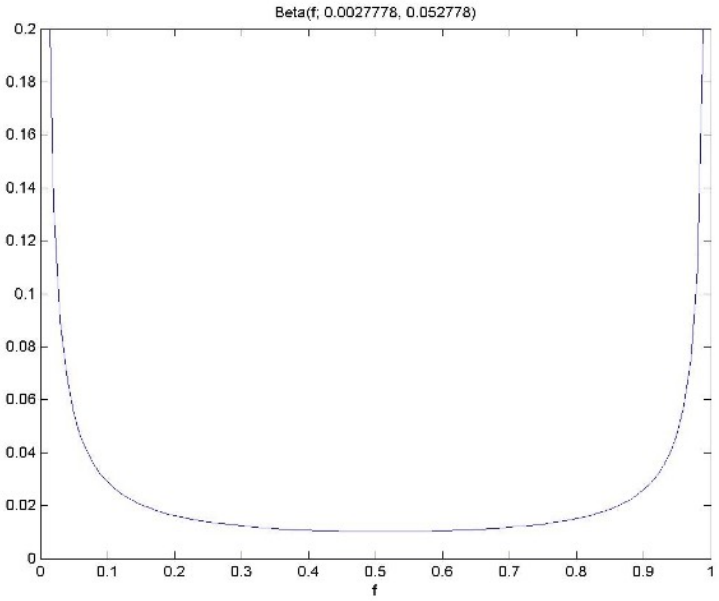
Use Dirichlet Distribution to model prior and posterior beliefs

Prior beliefs:

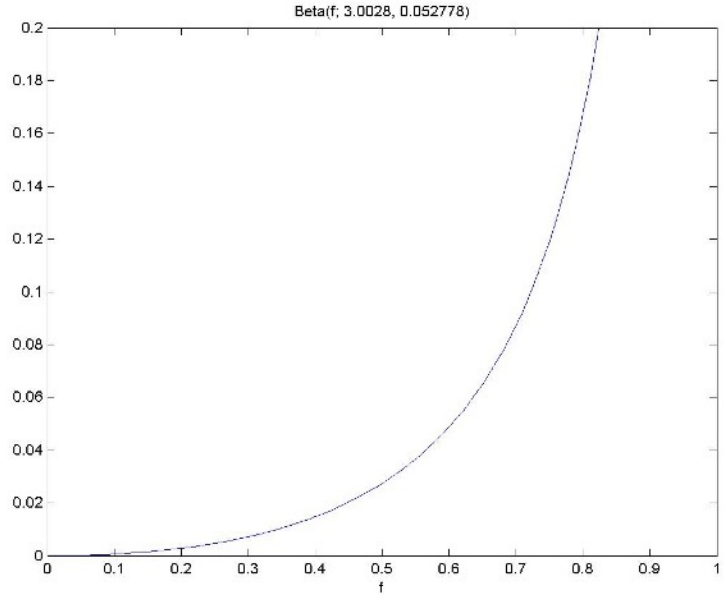
Posterior beliefs:

$$\rho(f) = beta(f; a, b)$$

$$\rho(f \mid d) = beta(f; a+s, b+t)$$



d: 3,0



beta(1/360,19/360): prior knowledge that an 5% chance-event may be true

beta(1/360+3,19/360): posterior belief that an 5% chance-event may be true

Importance of Dirichlet Distribution

In 1982, Sandy Zabell proved that, if we make certain assumptions about an individual's beliefs, then that individual must use the Dirichlet density function to quantify any prior beliefs about a relative frequency.

Some Insight on BN-based Content Clustering

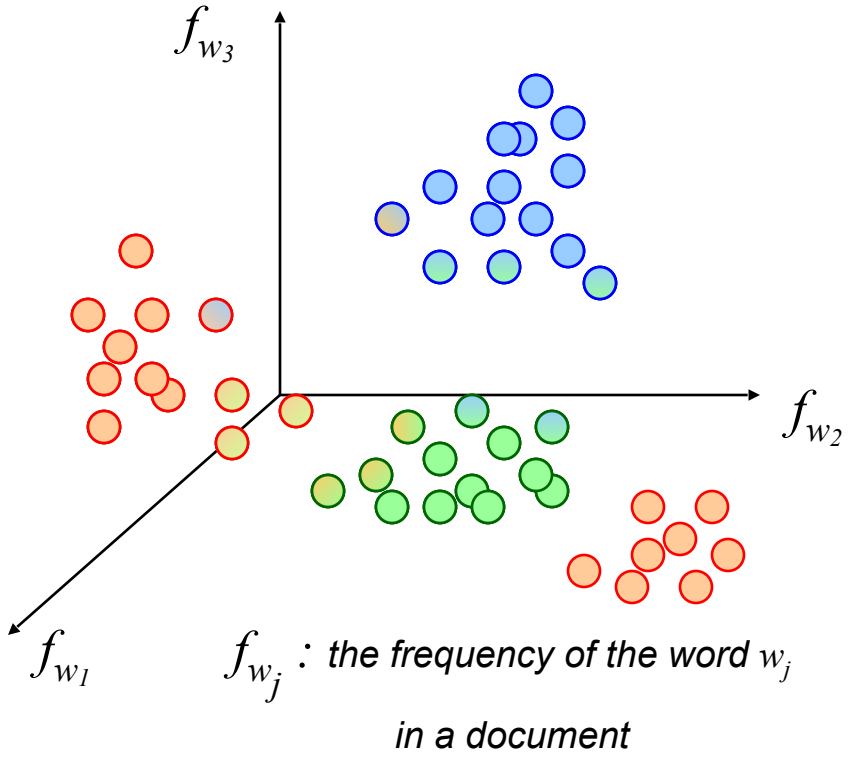
Bayesian Network:

• Models the *practical* causal relationships..

Content Clustering:

• Because documents and words are dependent,

➔ only close documents in the feature space can be clustered together as one topic.



⇒Incorporating human factors can possibly *link* multiple clusters together.

Gibbs Sampling

- Suppose that it is hard to sample p(x) but that it is possible to "walk around" in X using local state transitions
- Insight: we can use a "random walk" to help us draw random samples from p(x)
- At each transition change the state of just on X_i
- We can describe the transition probability as a stochastic procedure:

  Input: a state $x_1,\ldots,x_n$

  Choose i at random (using uniform probability)

  Sample x'_i from

  $$P(X_i | x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n, \boldsymbol{e})$$

  let $x'_j = x_j$ for all $j \neq i$

  return $x'_1,\ldots,x'_n$

Spark ML LDA code example

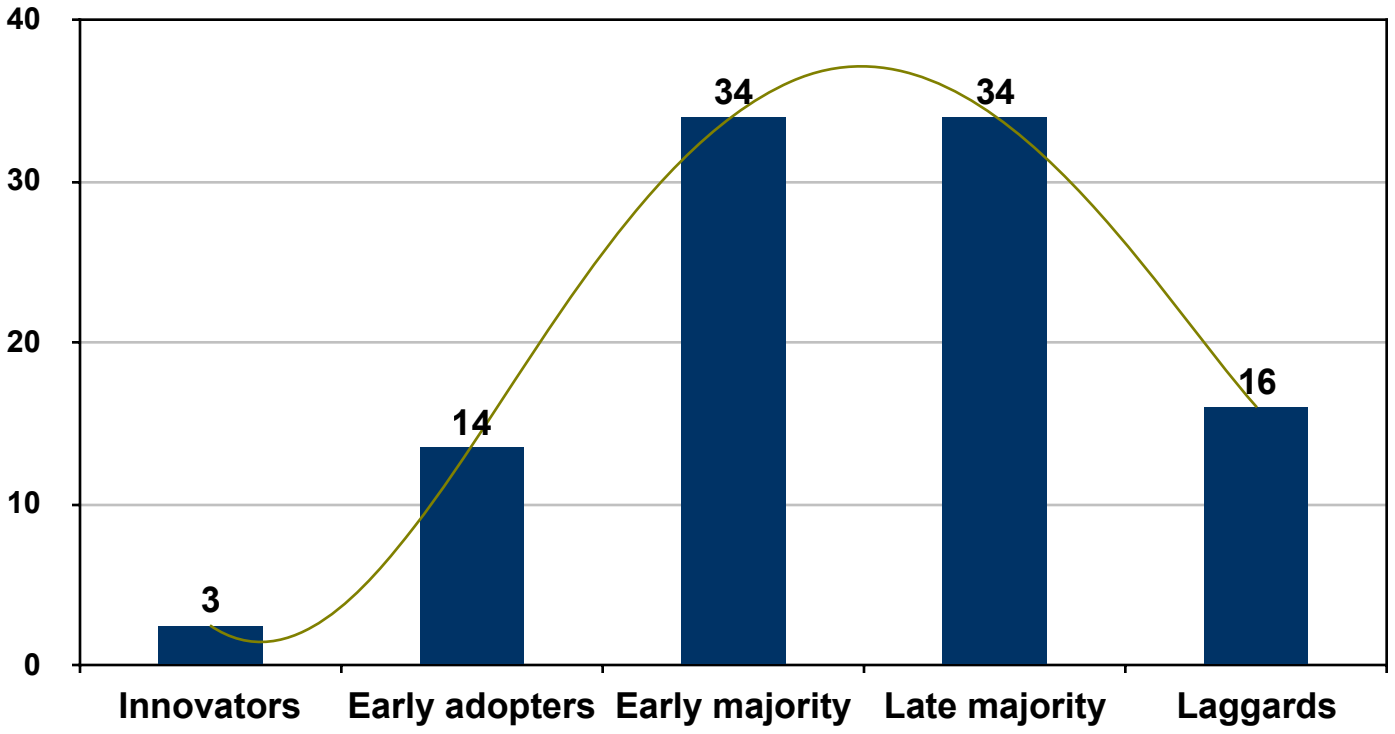
```scala
import org.apache.spark.ml.clustering.LDA

// Loads data.
val dataset = spark.read.format("libsvm")
  .load("data/mllib/sample_lda_libsvm_data.txt")

// Trains a LDA model.
val lda = new LDA().setK(10).setMaxIter(10)
val model = lda.fit(dataset)

val ll = model.logLikelihood(dataset)
val lp = model.logPerplexity(dataset)
println(s"The lower bound on the log likelihood of the entire corpus: $ll")
println(s"The upper bound on perplexity: $lp")

// Describe topics.
val topics = model.describeTopics(3)
println("The topics described by their top-weighted terms:")
topics.show(false)

// Shows the result.
val transformed = model.transform(dataset)
transformed.show(false)
```

Use Cases Example: Recommendation with Community Clustering or Information Flow

Exploiting Dynamic Patterns for Recommendation Systems:

- Community based Dynamic Recommendation [Song *et al.* SDM'06]
- Personalized Recommendation Driven by Information Flow [Song *et al.* SIGIR'06]

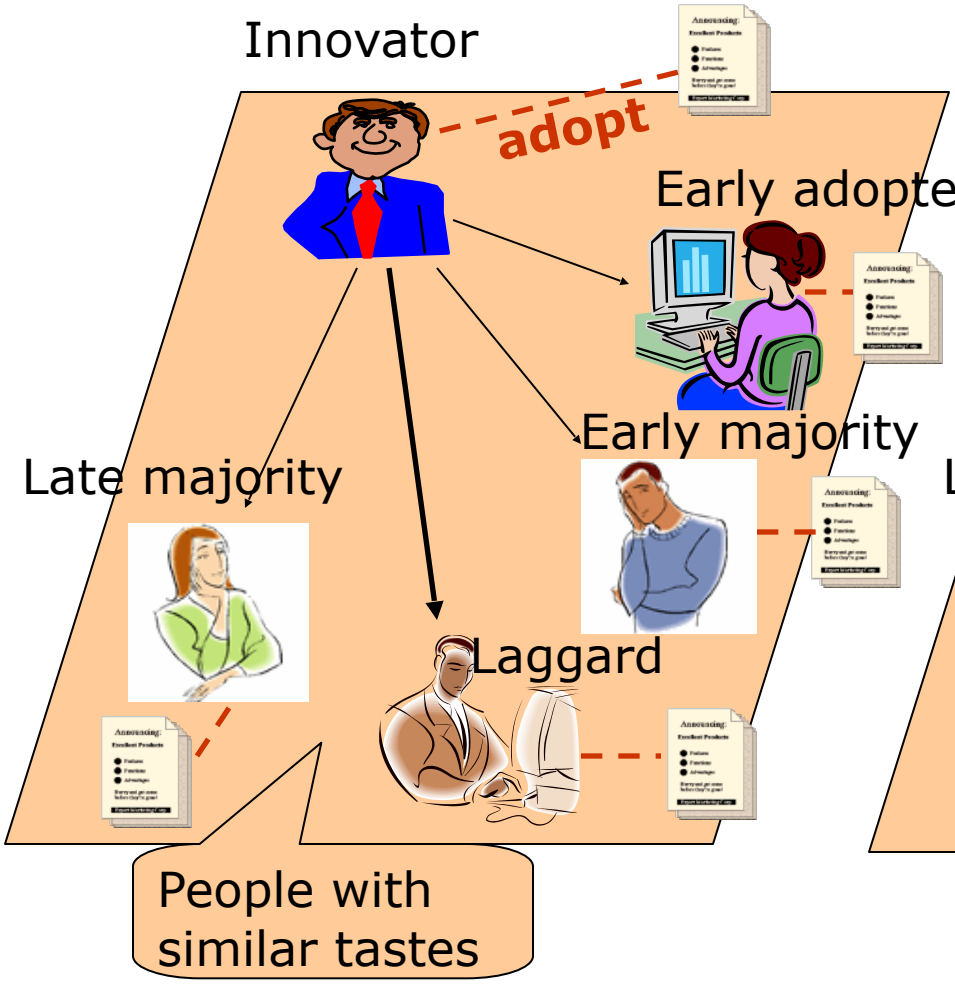
In E-Commerce: Rogers' Diffusion of Innovations Theory

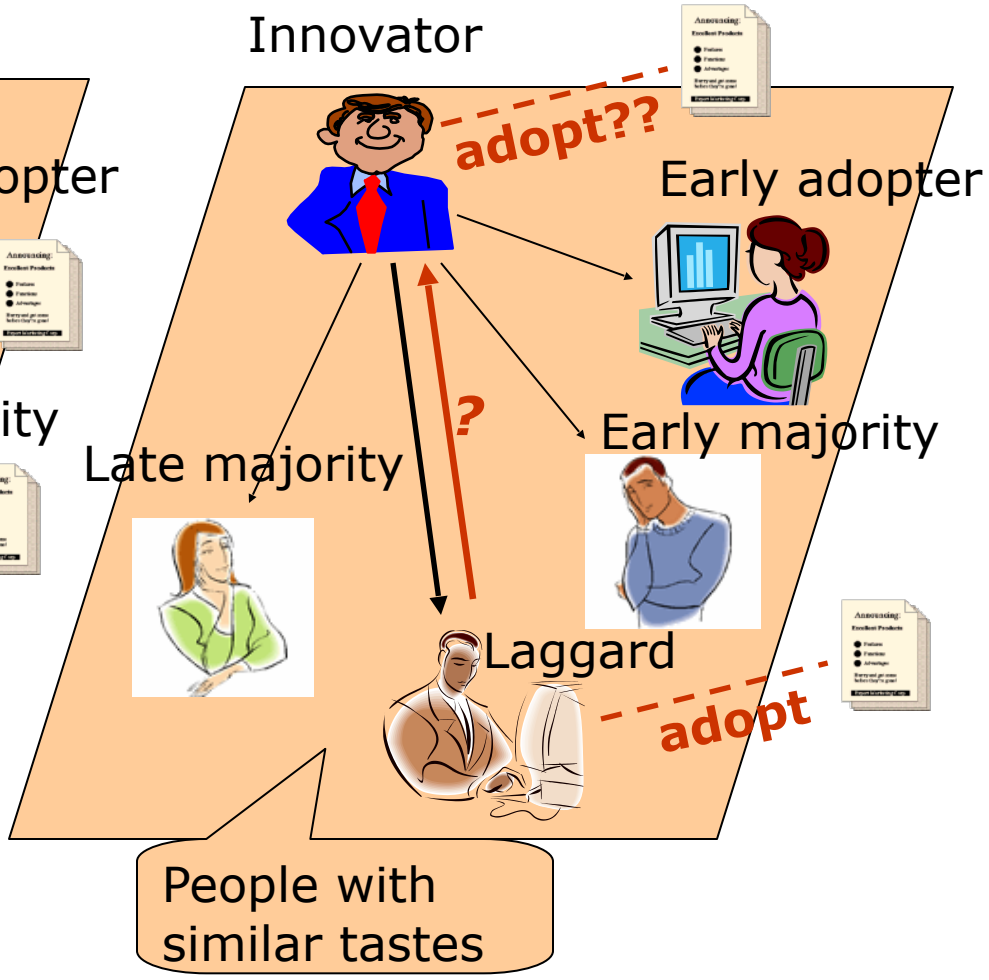


Users' adoption patterns: Some users tend to adopt innovations earlier than others

→ Information virtually flows from early adopters to late adopters

Recommendation Driven by Information Flow



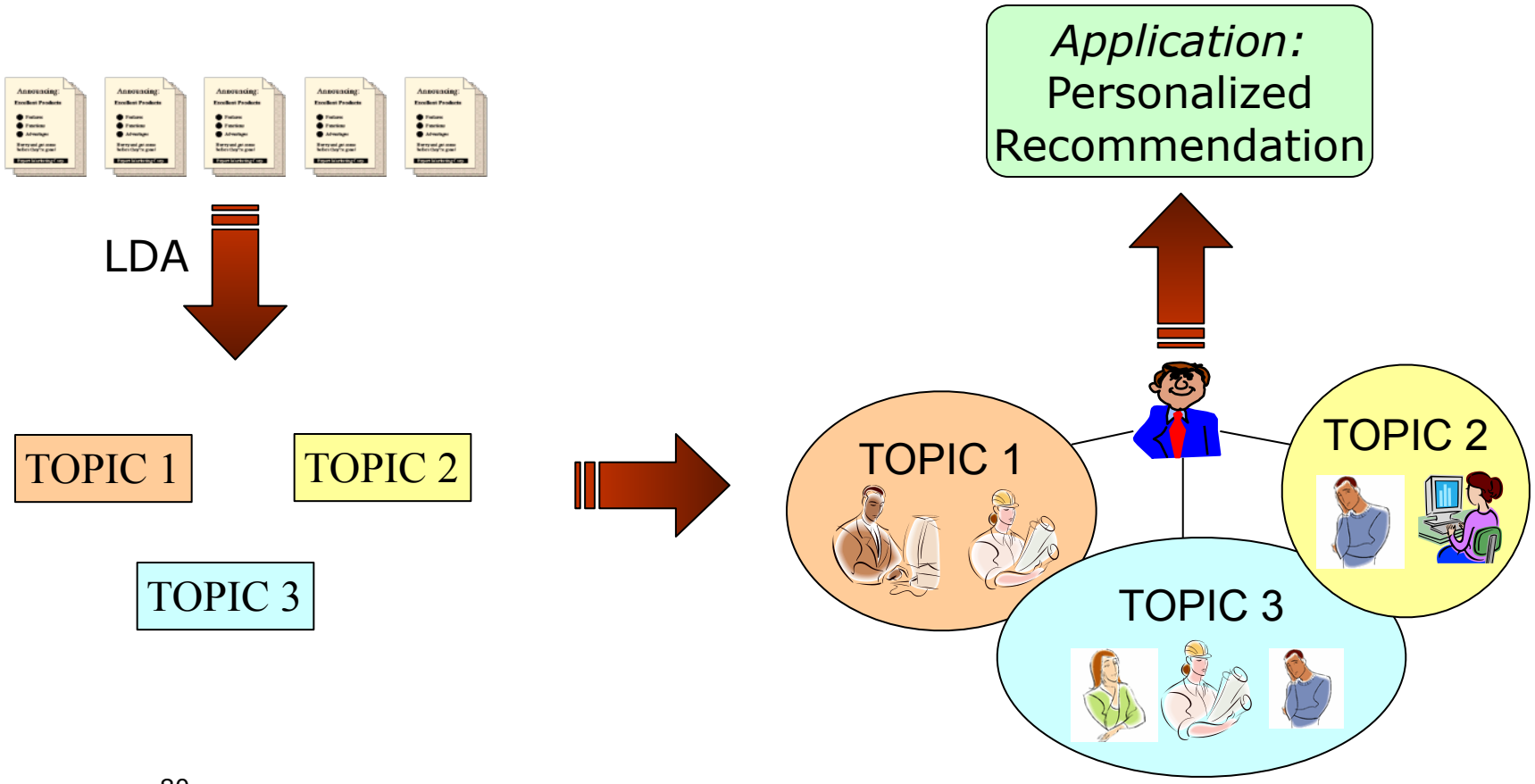


Influence is not symmetric!

Topic-Sensitive Early Adoption Based Information Flow (TEABIF) Network

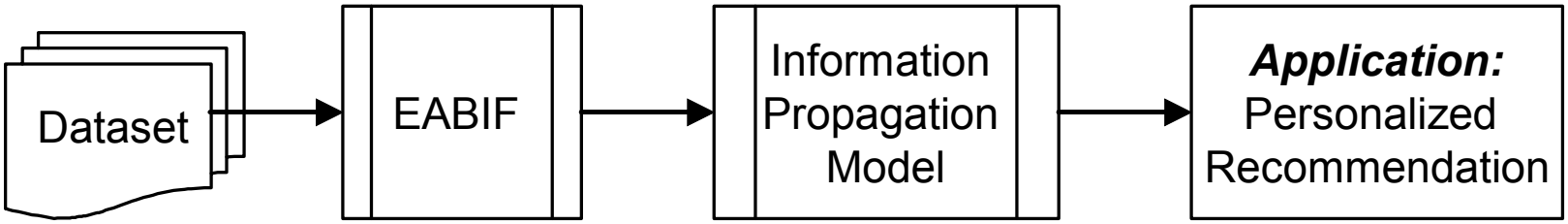
Adoption is typically category specific

An early adopter of fashion may not be an early adopter of technology



Scheme Overview

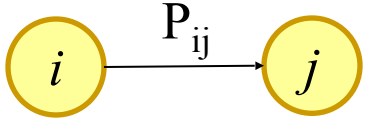
Leverage the uneven influence



EABIF: Early Adoption based Information Flow Network

EABIF (1) -- Markov Chain

A Markov chain has two components



- A network structure where each node is called a state
- A transition probability of traversing a link given that the chain is in a state
  (P: transition probability matrix)

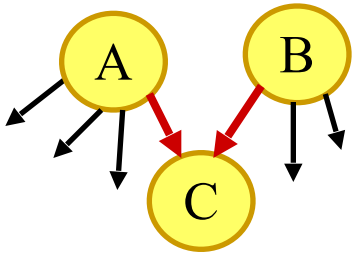
A stationary distribution is a probability distribution q such that q = qP → how likely you will stay at one node

Application -- PageRank [Brin and Page '98]

- Assumption: A link from page A to page B is a recommendation of page B by the author of A

→ Quality of a page is related to

- Number of pages linking to it
- The quality of pages linking to it



Assume the web is a Markov chain

- PageRank = stationary distribution of this Markov chain

EABIF (2)

Early Adoption Matrix (EAB)

- Count how many items one user accesses earlier than the other – pairwise comparison

Markov Chain Model

- Normalize EAB to a transition matrix F of a Markov chain
- Adjustment F to guarantee the existence of stationary distribution of the Markov chain
  - Make the matrix stochastic
  - Make the Markov chain irreducible

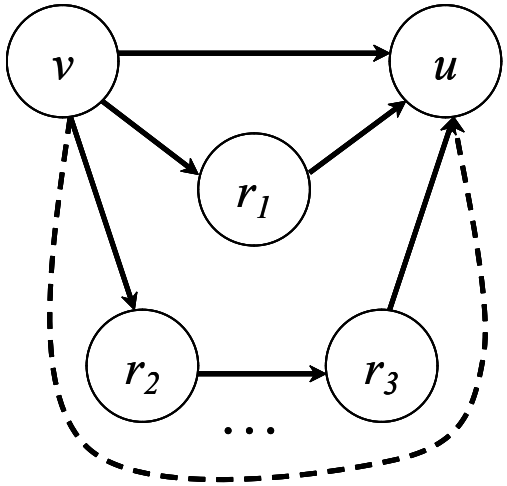
$$\sum_j \bar{F}_{ij} = 1$$

$$\bar{\bar{F}}_{ij} \neq 0, \quad 1 \leq i, j \leq N$$

Information Propagation Models

1. Summation of various propagation steps

$$\mathbf{F}_{if(m)} = \left(\bar{\bar{\mathbf{F}}} + \bar{\bar{\mathbf{F}}}^{(2)} + \cdots + \bar{\bar{\mathbf{F}}}^{(m)}\right) \Big/ m$$



2. Direct summation

$$\mathbf{F}_{if(d)} = \frac{\bar{\bar{\mathbf{F}}} \cdot \left(\mathbf{I} - \bar{\bar{\mathbf{F}}}^{(N-1)}\right)}{\mathbf{I} - \bar{\bar{\mathbf{F}}}}$$

3. Exponential weighted summation N: number of the nodes

$$\mathbf{F}_{if(\exp)} = \left(\left(\beta \bar{\bar{\mathbf{F}}}\right) + \frac{1}{2!}\left(\beta \bar{\bar{\mathbf{F}}}\right)^2 + \cdots + \frac{1}{(N-1)!}\left(\beta \bar{\bar{\mathbf{F}}}\right)^{(N-1)} + \cdots \right) \exp(-\beta)$$

Experimental Setup

ER dataset

- 2004 Apr. to 2005 Apr. as training data
- 2005 May to 2005 Jul. as test data

  1033 users, 586 documents

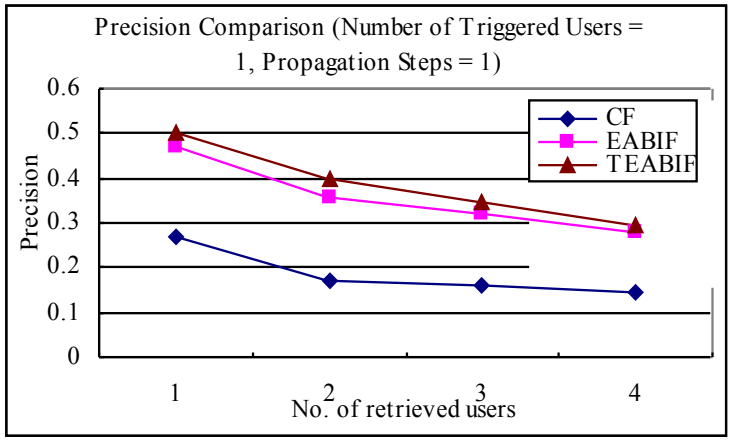
Process

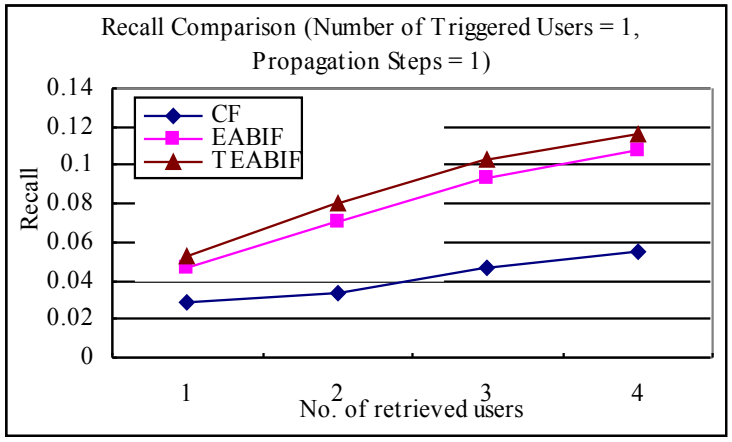
- Construct information flow network based on the training data
- Trigger earliest users to start the process
- Predict who will be also interested in these documents

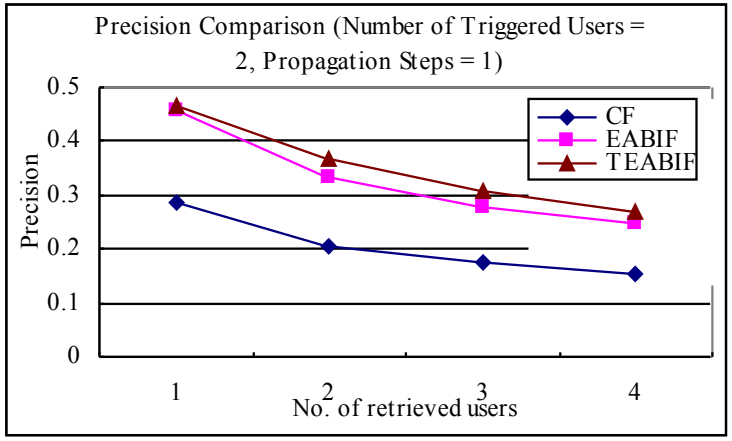
Evaluation

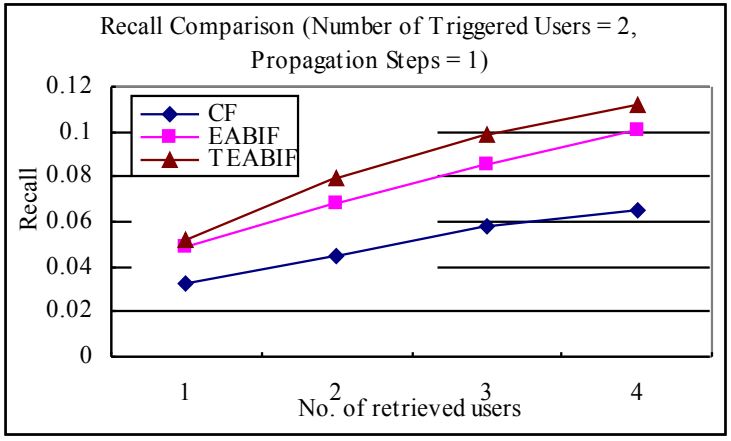
- Precision & Recall

Experimental Results --Recommendation Quality





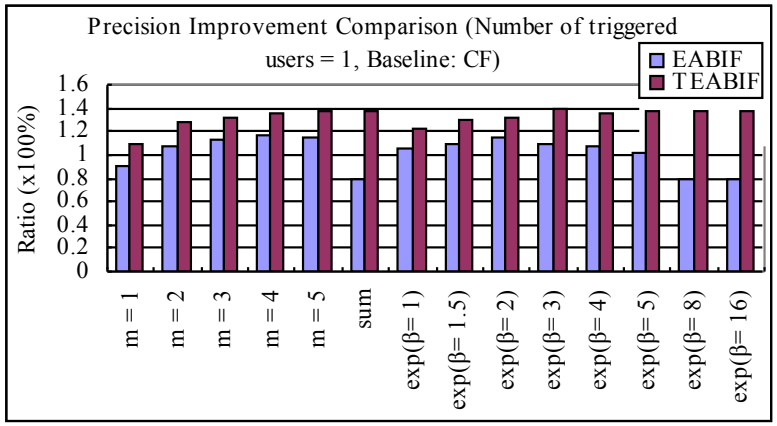


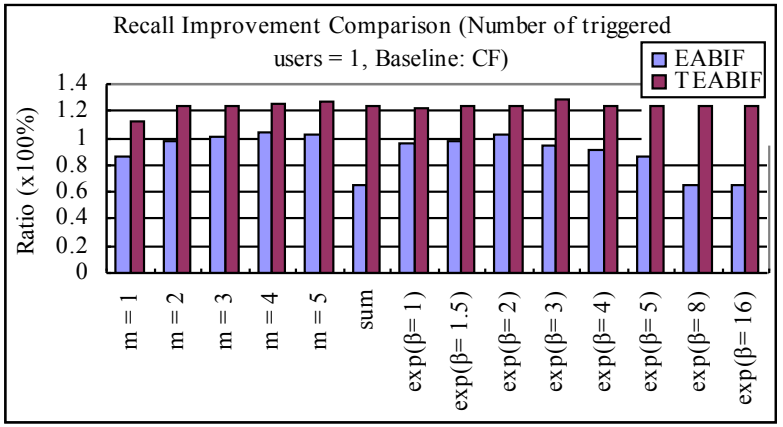


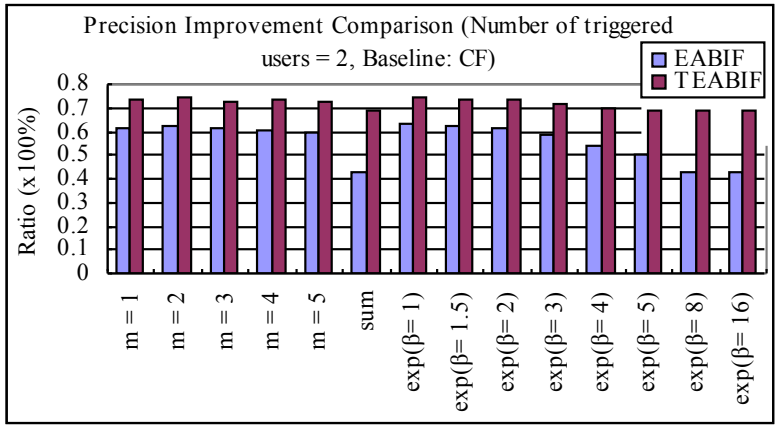
→ Comparing to Collaborative Filtering (CF),
precision: EABIF is 91.0% better, TEABIF is 108.5% better
Recall: EABIF is 87.1% better, TEABIF is 112.8% better

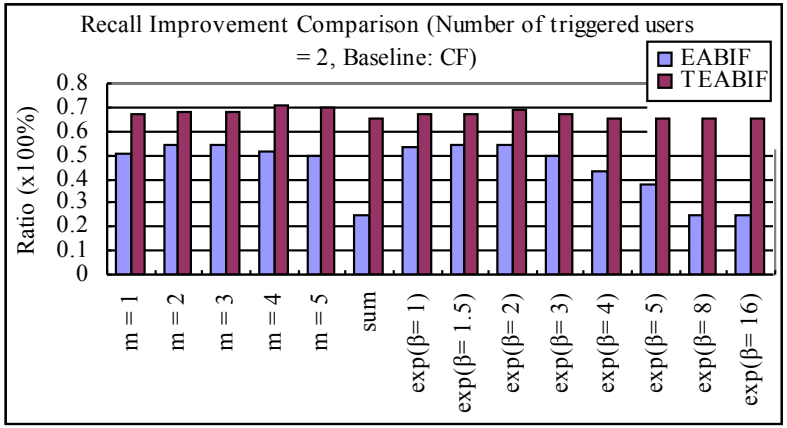
Experimental Results --

Propagation Performance









→ TEABIF with exponential weighted summation ($\beta = 3$)
achieves the best performance:
improves 108.5% on precision and 116.9% on recall comparing to CF

Summary

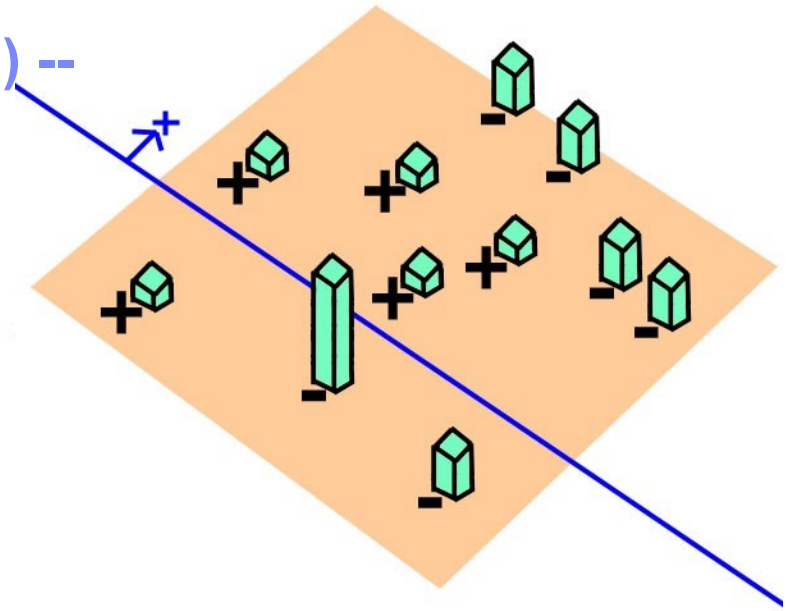
Exploit dynamic patterns including

- Leverage dynamic patterns from both documents and users' perspective
  - Analyzing documents accessing types
  - Predicting documents' expiration date
  - Detecting users' intentions
  - Identifying users interests evolving over time – CTC model
  - Ranking the documents adaptively – Time-sensitive Adaboost
- Utilize users' adoption patterns
  - Information virtually flows from early adopters to late adopters

Experimental results demonstrate

- Dynamic factors are important for recommendations

Community based recommendation (1) -- AdaBoost Modeling



- Adaboost [Freund and Schapire 1996]
  - Constructing a "strong" learner as a linear combination of weak learners

- Start with a uniform distribution ("weights") over training examples (The weights tell the weak learning algorithm which examples are important)

- Obtain a weak classifier from the weak learning algorithm, $h_{j_t}: X \rightarrow \{-1, 1\}$

- Increase the weights on the training examples that were misclassified

- (Repeat)

The final classifier is a linear combination of the weak classifiers obtained at all iterations

$$f_{\text{final}}(\mathbf{x}) = sign\left(\sum_{s=1}^{S} \alpha_s h_s(x)\right)$$

Community based recommendation (2) -- Time-Sensitive AdaBoost Modeling

In AdaBoost, the goal is to minimize the energy function:

$$\sum_{i=1}^{N} \exp\left(-c_i \sum_{s=1}^{S} \alpha_s h_s(x_i)\right)$$

All samples are regarded equally important at the beginning of the learning process

We proposed a time-adaptive AdaBoost algorithm that assigns larger weights to the latest documents to indicate their importance

- Weak learners
  - linear classifiers corresponding to the content, community and dynamic patterns

$$\sum_{i=1}^{N} \exp\left(-c_i \sum_{s=1}^{S} \alpha_s \exp\left(-\tau \cdot (t - t_i)\right) h_s(x_i, t)\right)$$

Experiments - Methodology

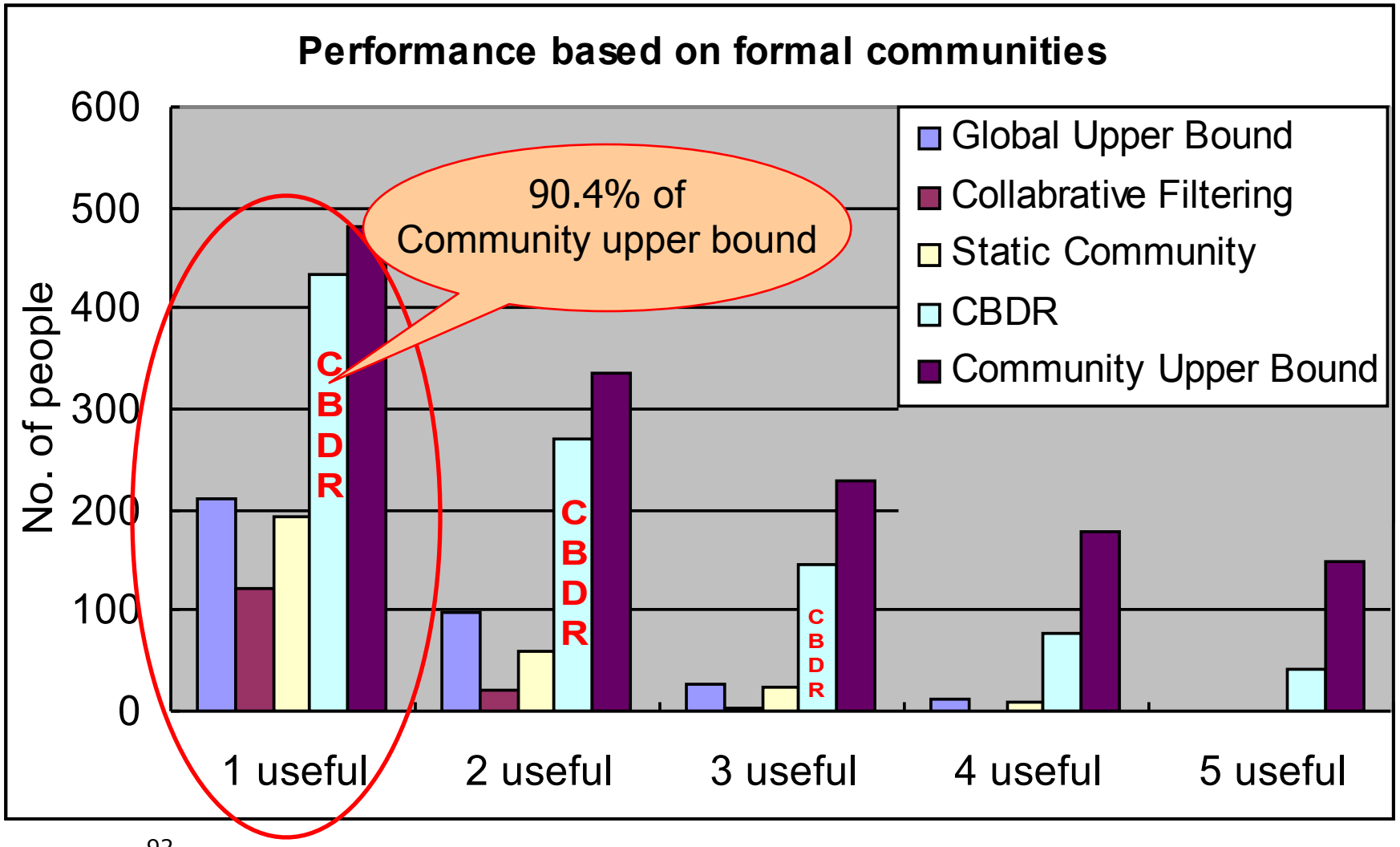
Dataset

- "EigyoRyoku"(Sales-Force) system (31,927 users, 26,631 documents)
- Log files
  - Apr. 2004 to Mar. 2005
    - Training data - Apr. 2004 to Feb. 2005
    - Test data – Mar. 2005
  - Nine user actions
    - "Login", "Register_Feedback", "Preview", "Abstract", "Document Download", "Search", "Register", "Update", "Delete"

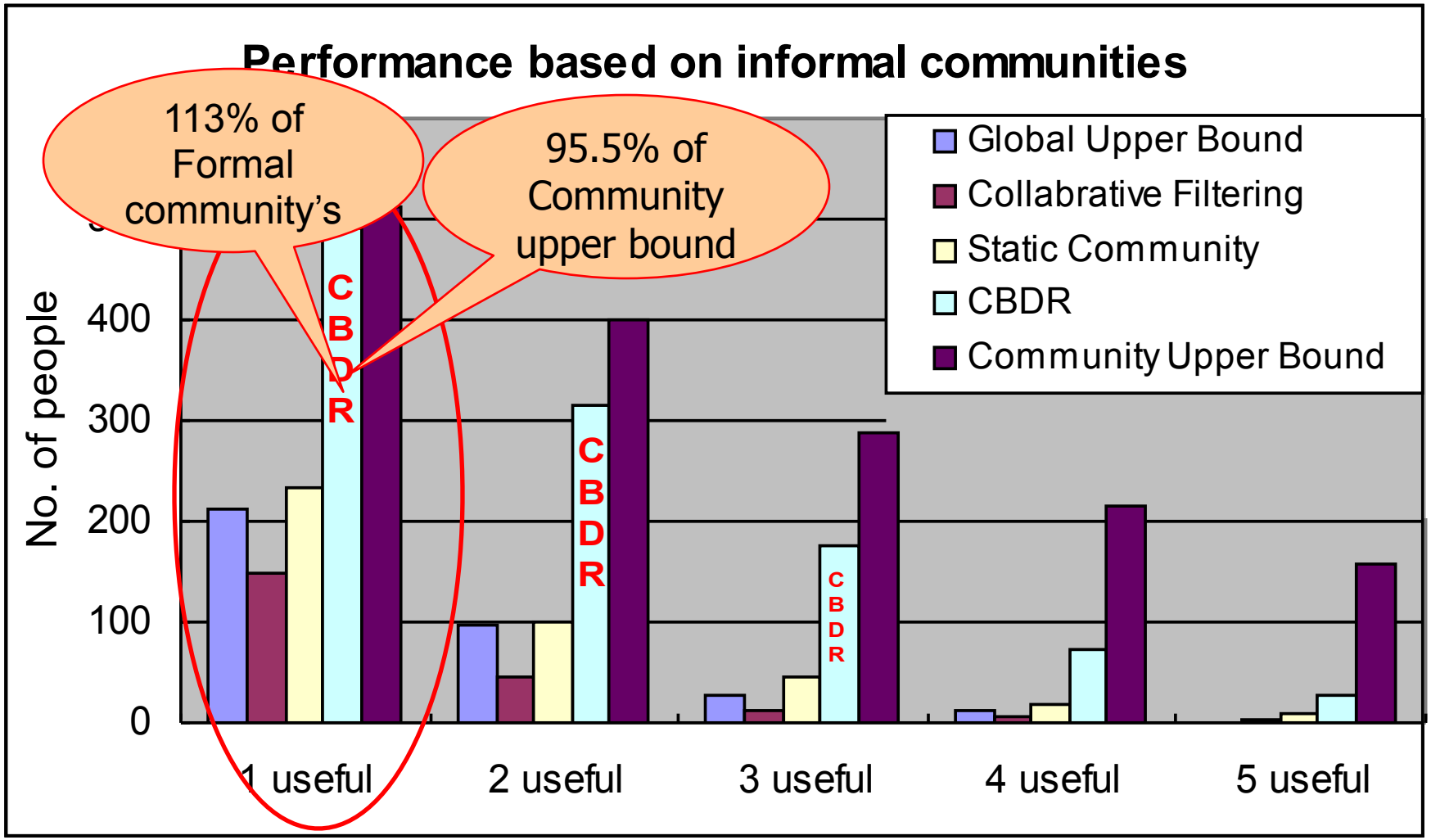
Evaluation -- user satisfaction

- How many people really downloaded the documents among these five recommendations during the testing period

Formal Community Recommendations



Informal Community Recommendations



Questions?