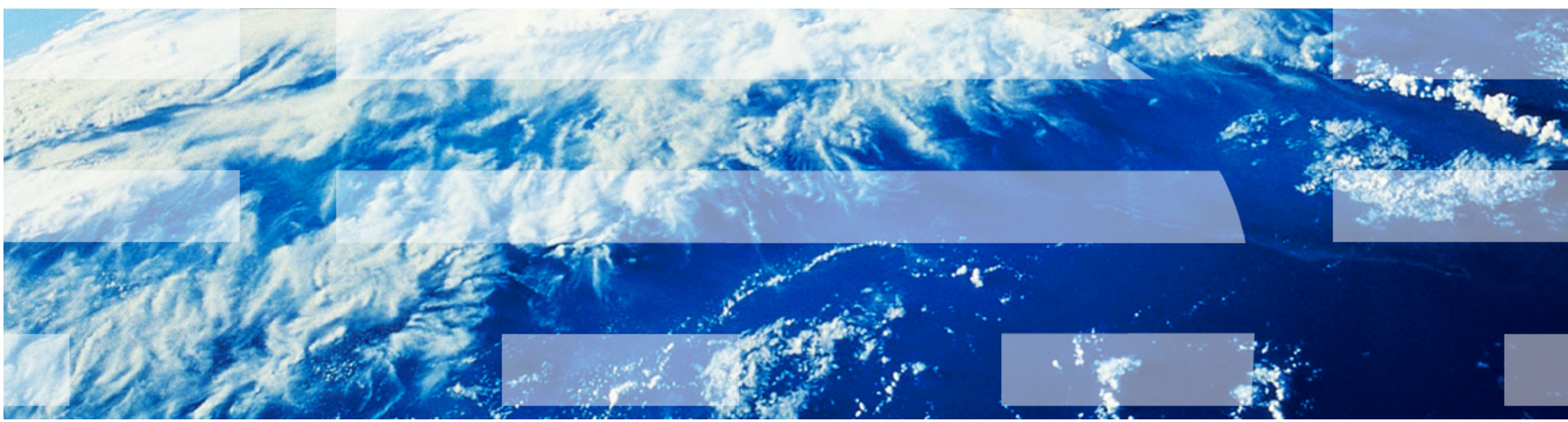
E6893 Big Data Analytics Lecture 3:

Big Data Analytics Algorithms and Stream Processing

Ching-Yung Lin, Ph.D.

Adjunct Professor, Dept. of Electrical Engineering and Computer Science



September 20, 2024

Key Components of Spark MLlib

MLlib: Main Guide

- Basic statistics
- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Frequent Pattern Mining
- Model selection and tuning
- Advanced topics

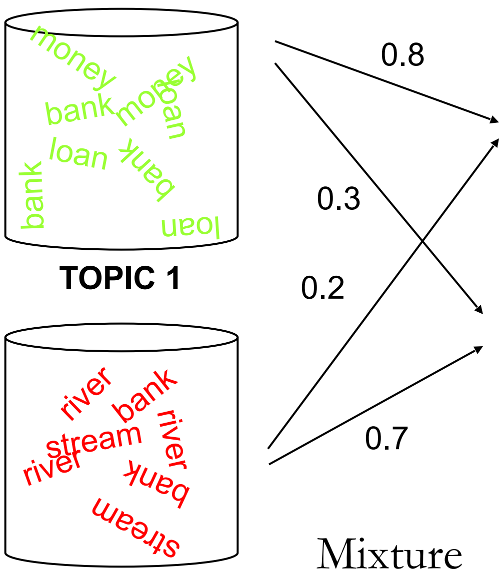
Spark Clustering

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

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: money¹ bank¹ bank¹ loan¹ river² stream² bank¹ money¹ river² bank¹ money¹ bank¹ loan¹ money¹ stream² bank¹ money¹ loan¹ river² stream² bank¹ money¹

DOCUMENT 2: loan¹ river² stream² loan¹ bank² river² bank² bank¹ stream² river² loan¹ bank² stream² bank² money¹ loan¹ river² stream² bank² stream² bank² money¹ river²

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

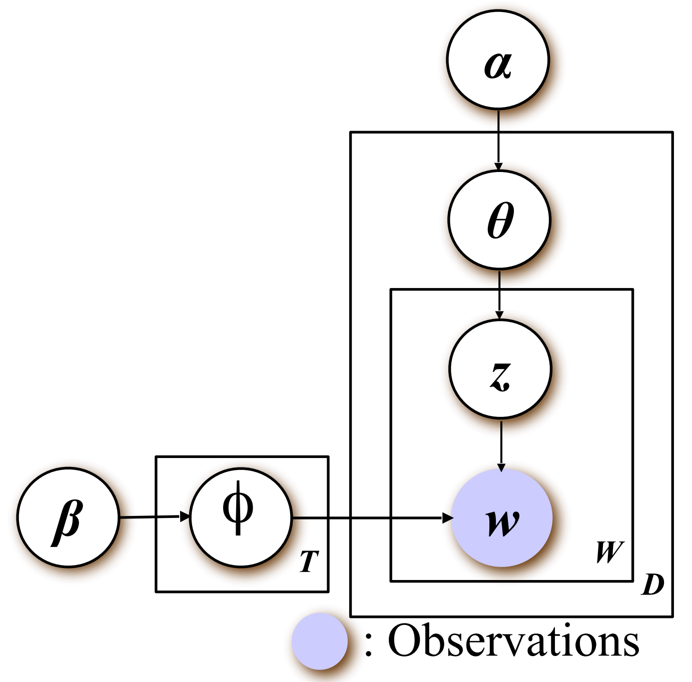
OUTPUT:

- likely topics for a document

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

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

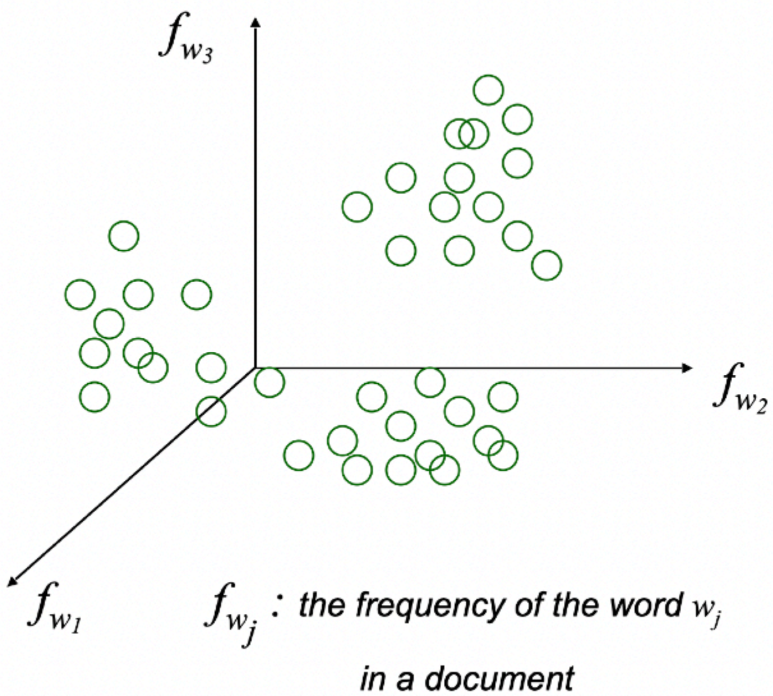
Bayesian approach: use priors
Mixture weights ~ Dirichlet(α)
Mixture components ~ Dirichlet(β)



T: number of topics

MLlib: Main Guide

Traditional Content Clustering



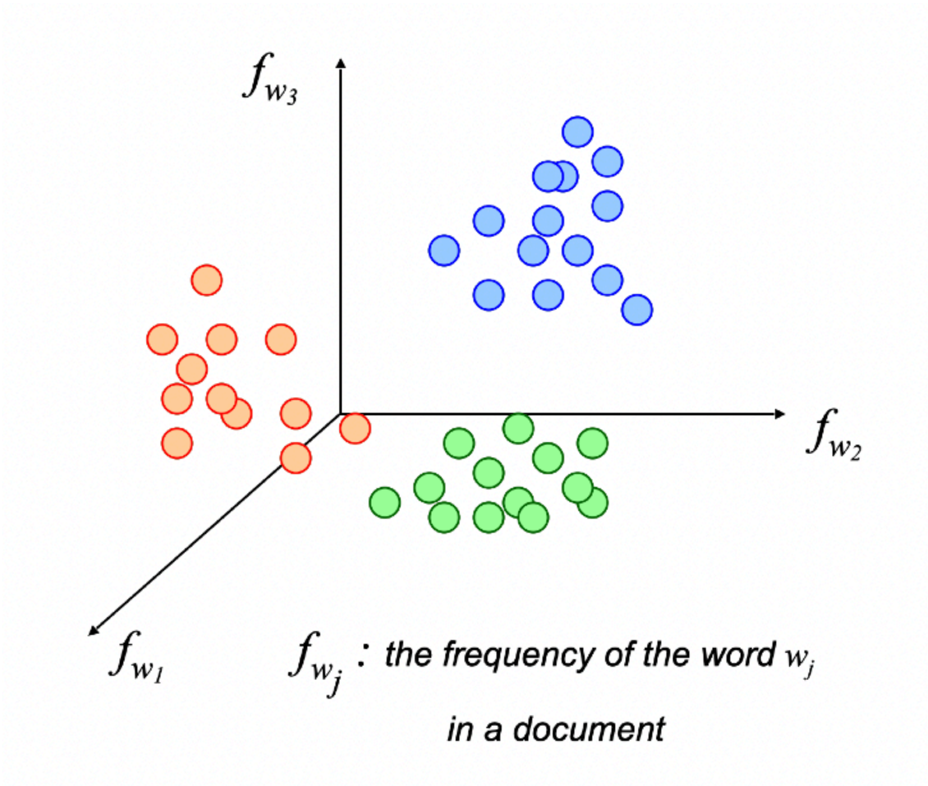
Clustering:

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



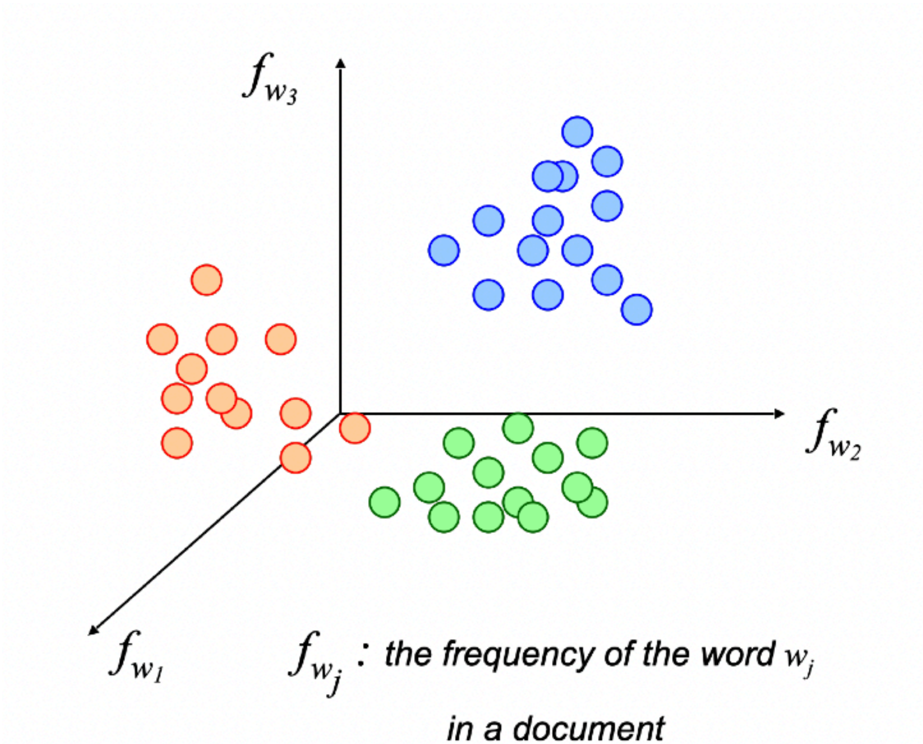
Clustering:

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

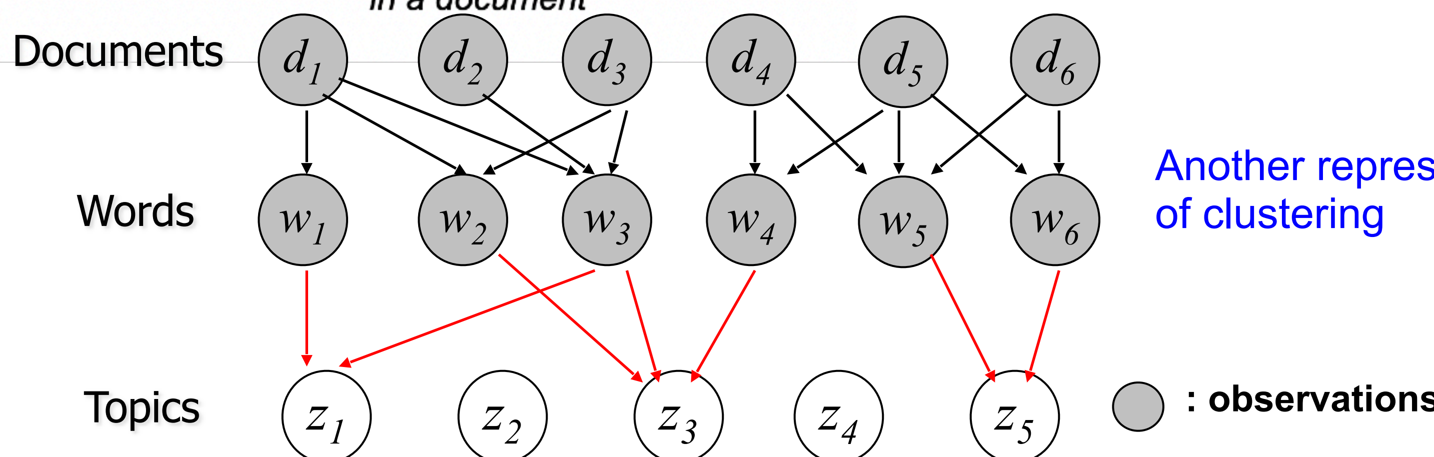


Clustering:

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

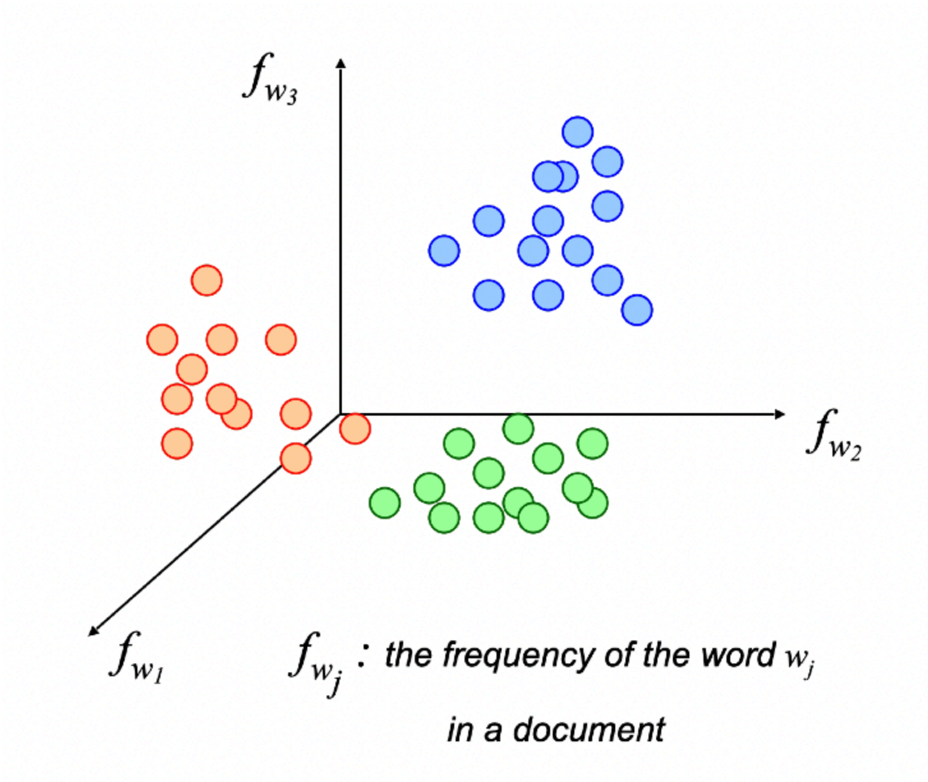
Hard clustering: e.g., K-mean clustering

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



Another representation of clustering

Traditional Content Clustering

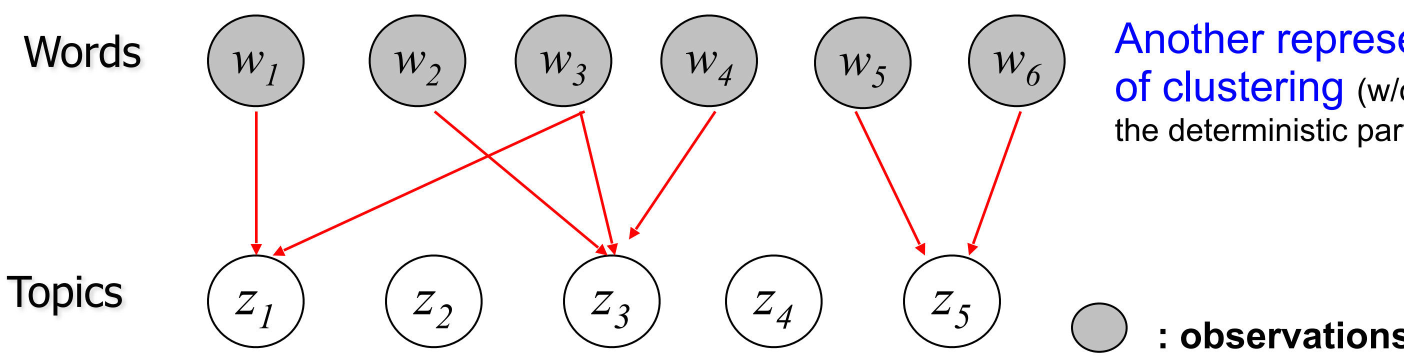


Clustering:

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

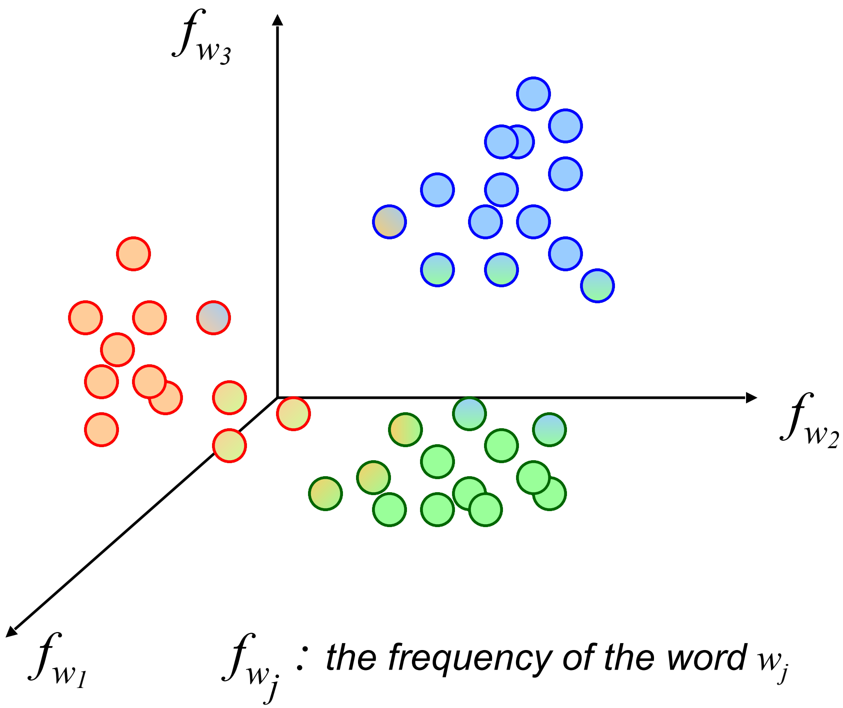
Hard clustering: e.g., K-mean clustering

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



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

Traditional Content Clustering — soft clustering



f_{w_j} : *the frequency of the word* w_j

in a document

Clustering:

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

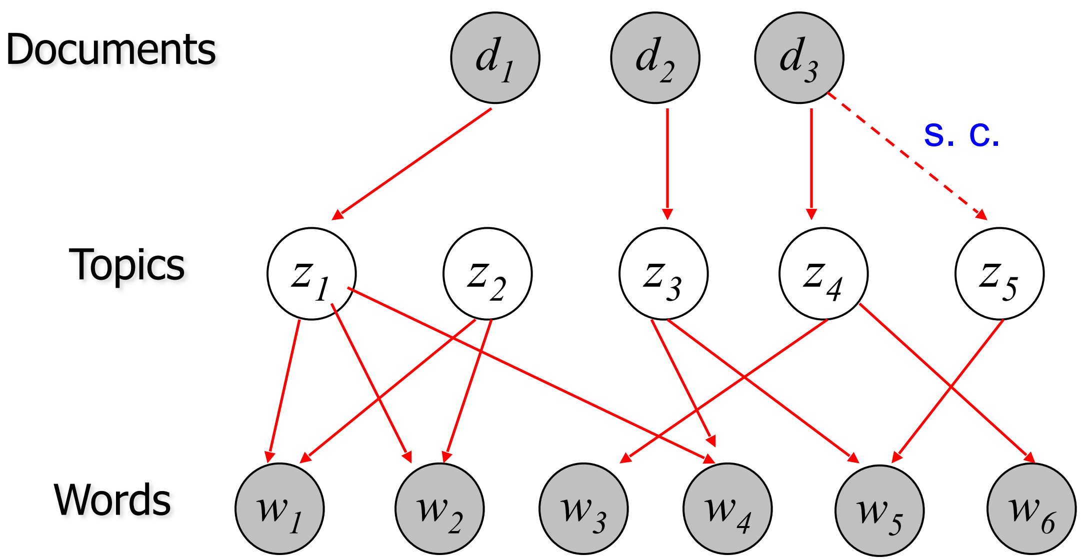
Hard clustering: e.g., K-mean clustering

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

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

$$P(Z \mid \mathbf{W} = \mathbf{f_w})$$

Content Clustering based on Bayesian Network



$h(D)$ hard clustering

$P(Z \mid D)$ soft clustering

$P(W \mid Z)$

: observations

Bayesian Network:

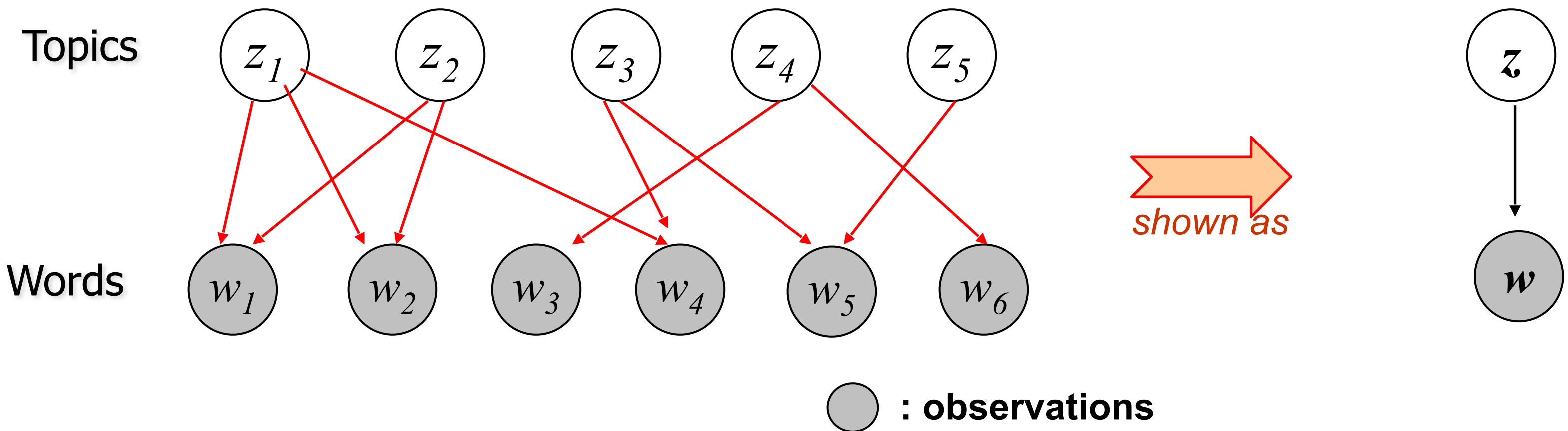
- Causality Network – models the causal relationship of attributes / nodes
- Allows hidden / latent nodes

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



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

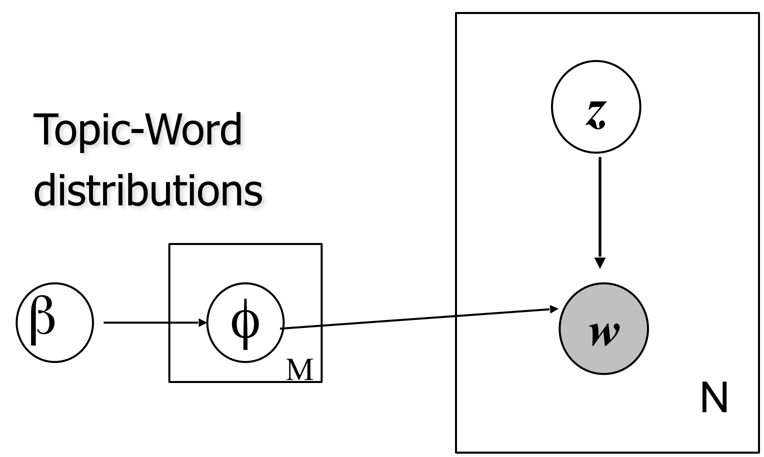
N: the number of words
(The number of topics (M) are pre-determined)

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*)

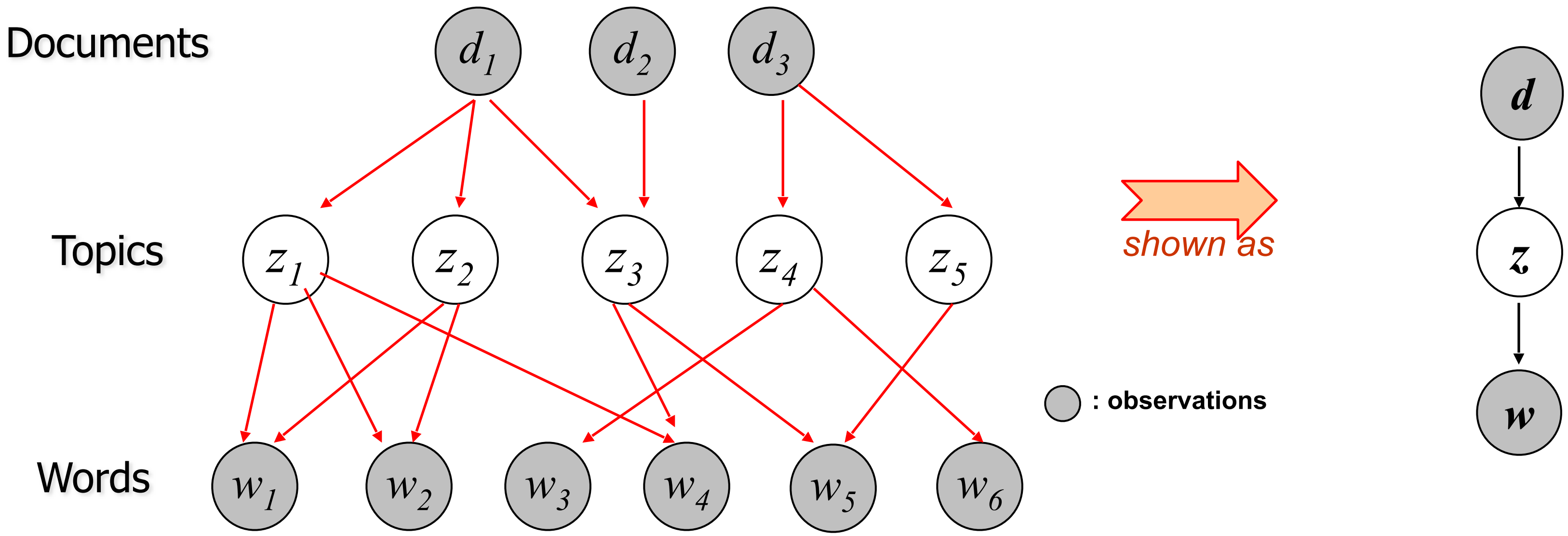
Major Solution 2 -- Gibbs Sampling:

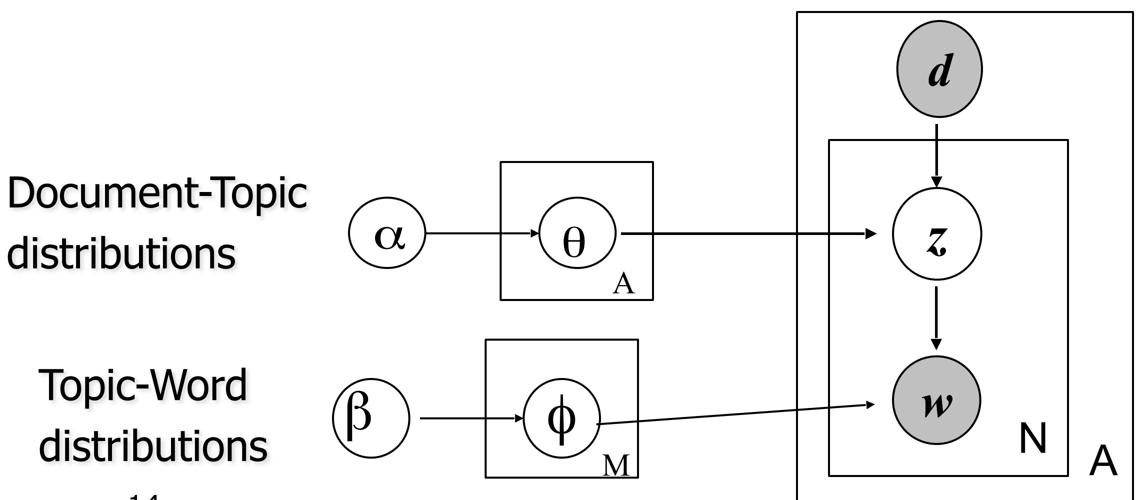
- A Markov chain Monte Carlo (MCMC) method for integration of large samples → calculate P(Z)



Latent Dirichlet Allocation (LDA) (Blei 2003)

Content Clustering based on Bayesian Network – Soft Clustering





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

LDA (Blei 2003)

Some Insight on BN-based Content Clustering

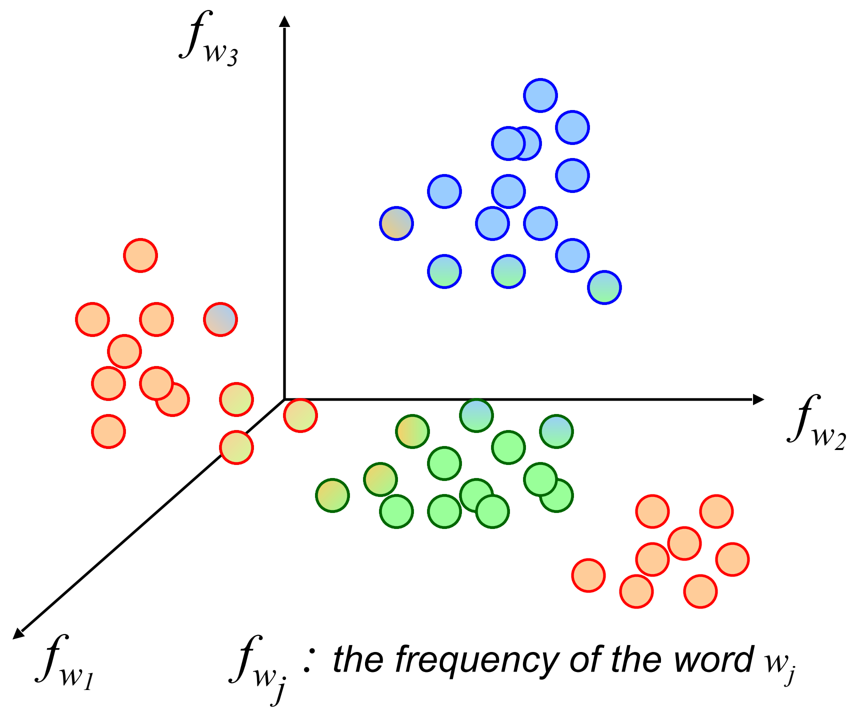
Bayesian Network:

• 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.



f_{w_j} : *the frequency of the word* w_j *in a document*

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

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)
```

MLlib: Main Guide

- Basic statistics
- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Frequent Pattern Mining
- Model selection and tuning
- Advanced topics

- Linear methods
- Decision trees
  - Inputs and Outputs
    - Input Columns
    - Output Columns
- Tree Ensembles
  - Random Forests
    - Inputs and Outputs
      - Input Columns
      - Output Columns (Predictions)
  - Gradient-Boosted Trees (GBTs)
    - Inputs and Outputs
      - Input Columns
      - Output Columns (Predictions)

Classification — definition

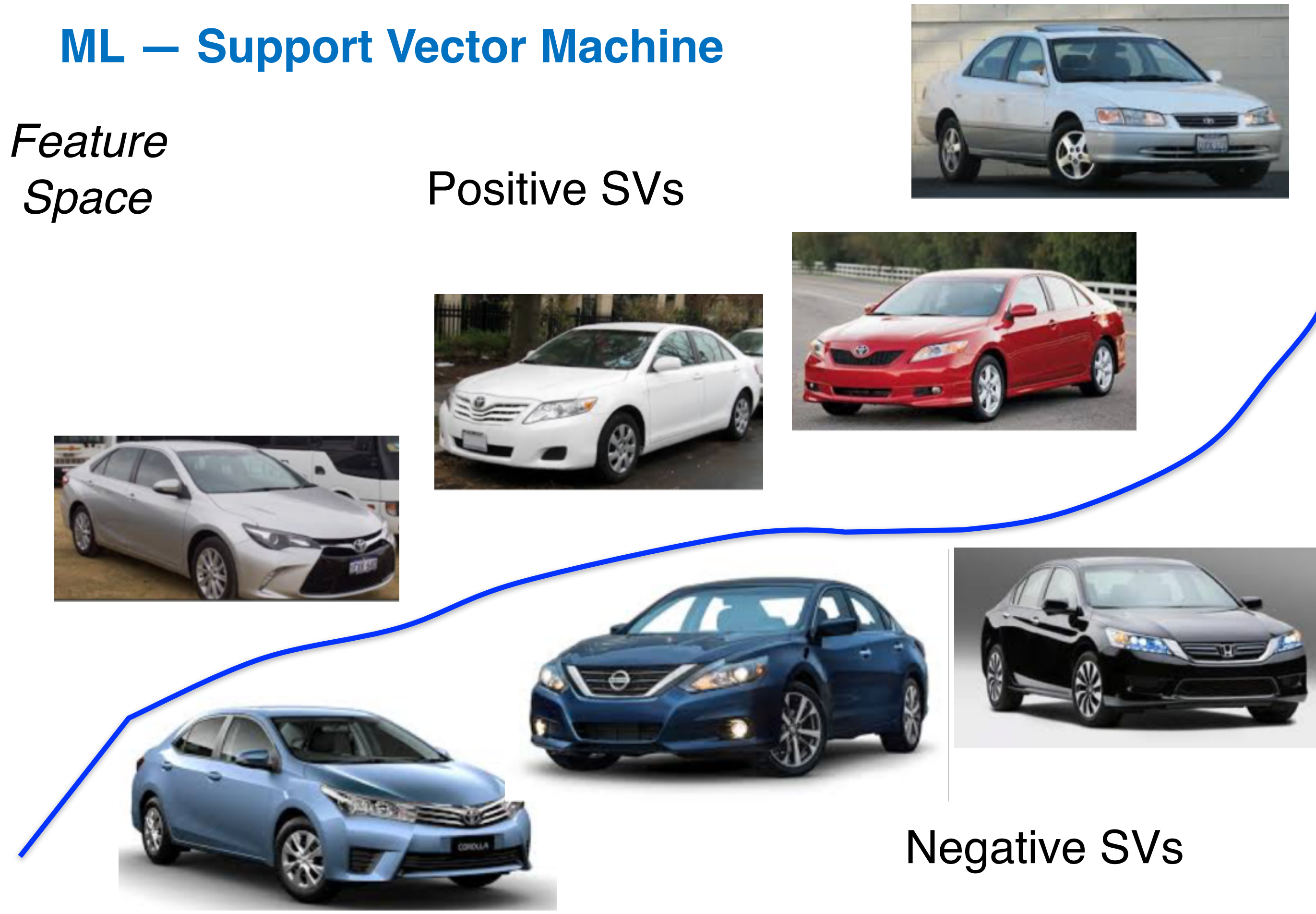
DEFINITION Computer classification systems are a form of machine learning that use learning algorithms to provide a way for computers to make decisions based on experience and, in the process, emulate certain forms of human decision making.

Classification example: using SVM to recognize a Toyota Camry

Non-ML

Rule 1.Symbol has something like bull's head
Rule 2.Big black portion in front of car.
Rule 3. …..????

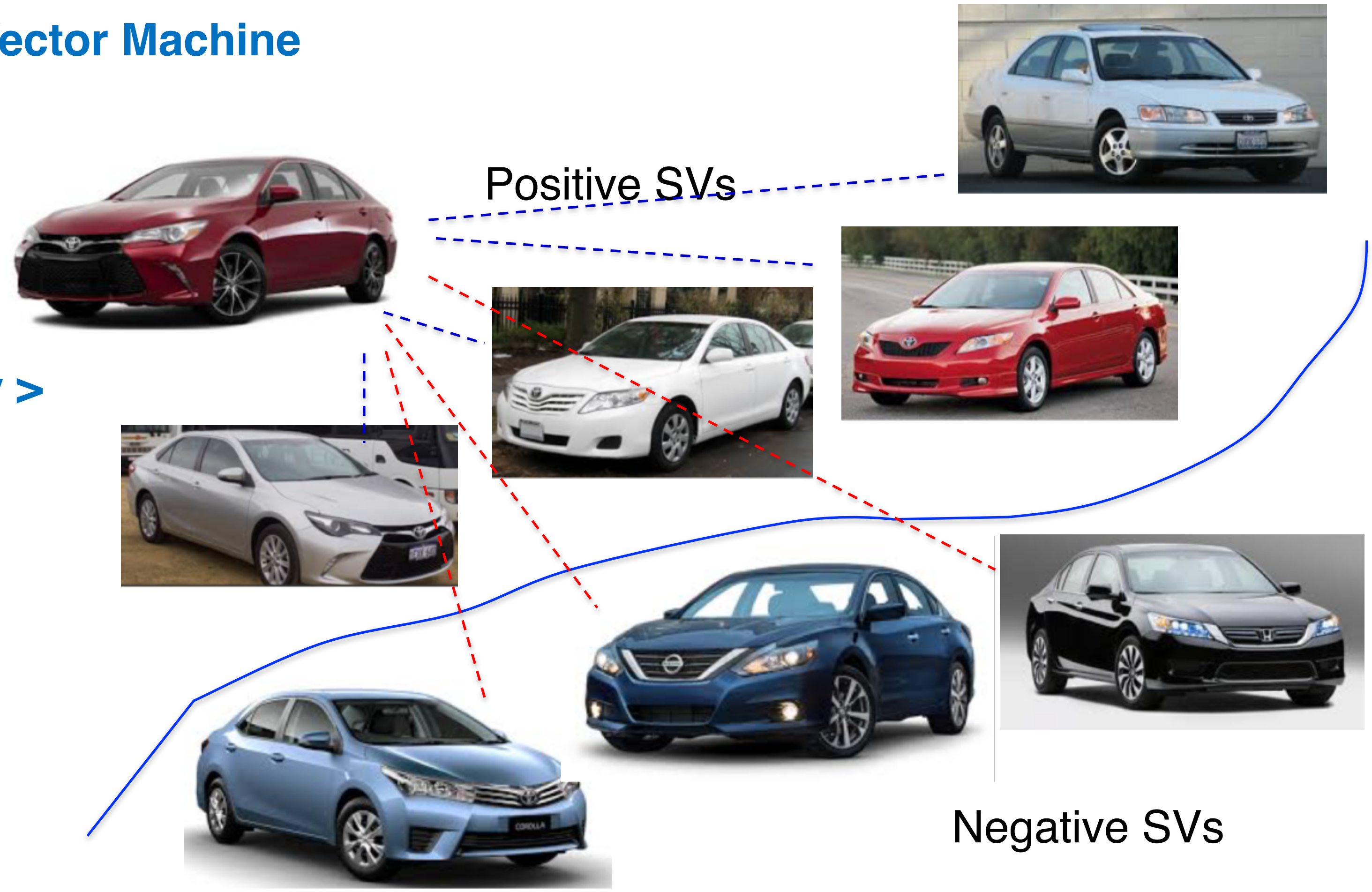
ML — Support Vector Machine



Classification example: using SVM to recognize a Toyota Camry

ML — Support Vector Machine

$\mathbf{P}_{\text{Camry}} > 0.95$

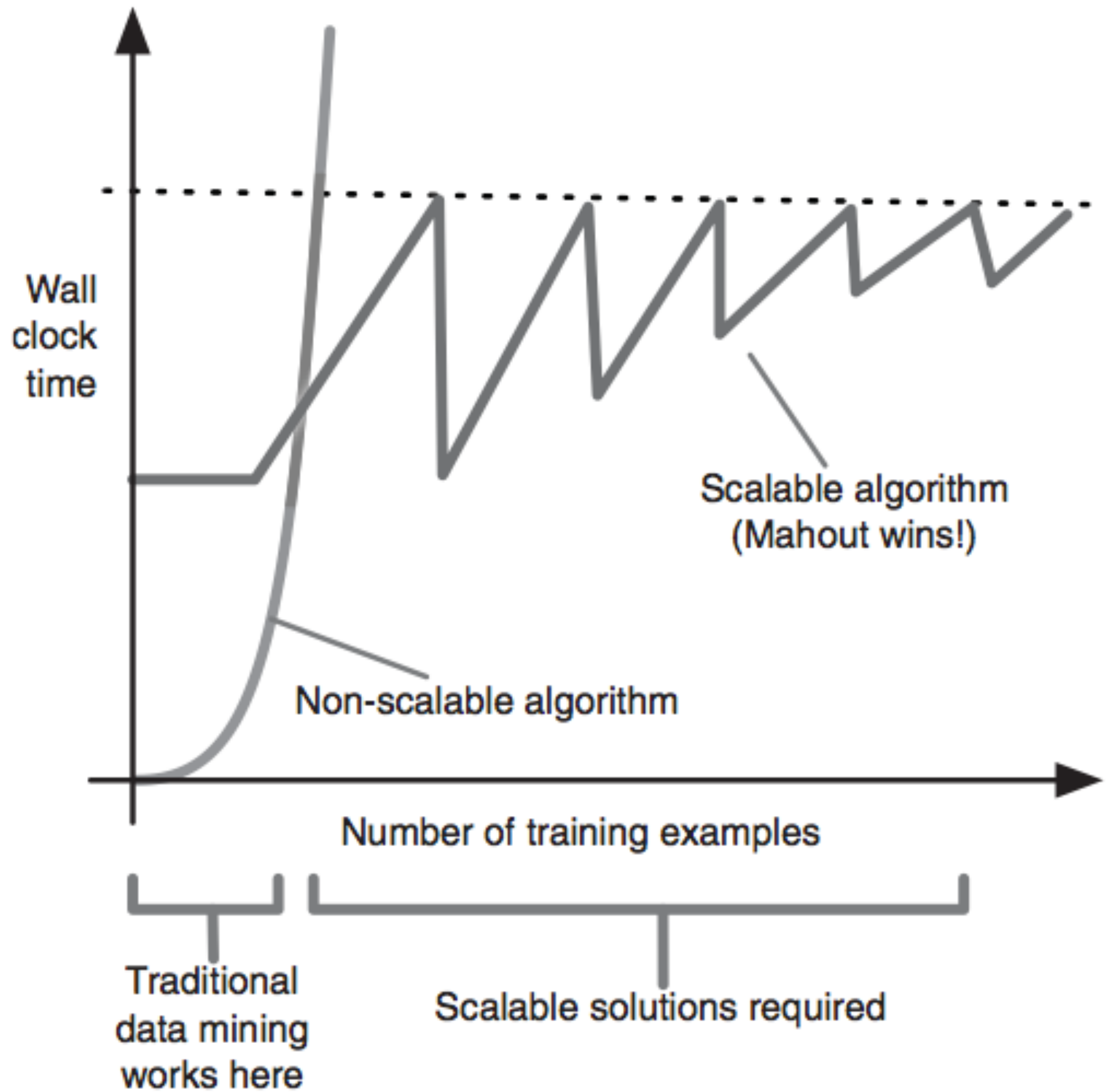


Feature Space

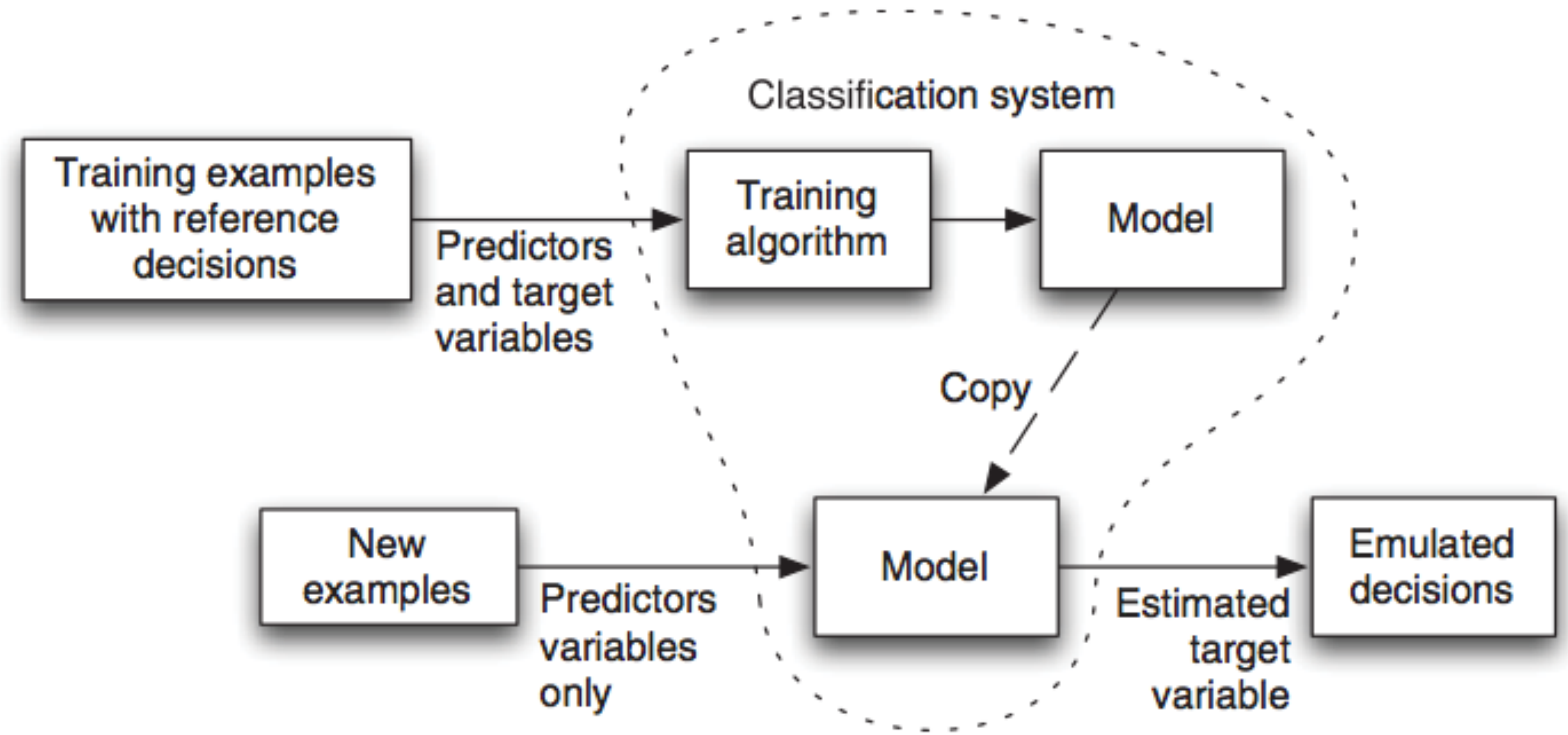
When to use Big Data System for classification?

System size in number of examples	Choice of classification approach
< 100,000	Traditional, non-Mahout approaches should work very well. Mahout may even be slower for training.
100,000 to 1 million	Mahout begins to be a good choice. The flexible API may make Mahout a preferred choice, even though there is no performance advantage.
1 million to 10 million	Mahout is an excellent choice in this range.
> 10 million	Mahout excels where others fail.

The advantage of using Big Data System for classification



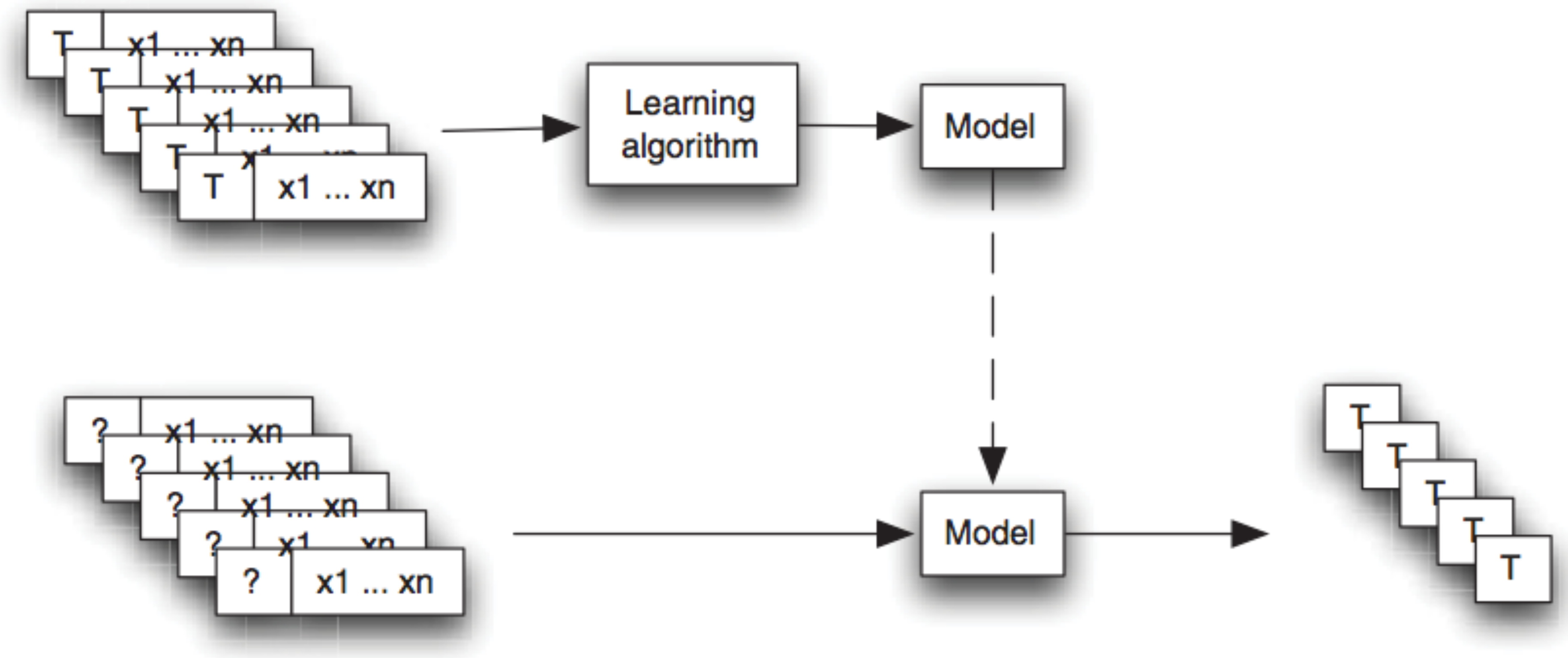
How does a classification system work?



Key terminology for classification

Key Idea	Description
Model	A computer program that makes decisions; in classification, the output of the training algorithm is a model.
Training data	A subset of training examples labeled with the value of the target variable and used as input to the learning algorithm to produce the model.
Test data	A withheld portion of the training data with the value of the target variable hidden so that it can be used to evaluate the model.
Training	The learning process that uses training data to produce a model. That model can then compute estimates of the target variable given the predictor variables as inputs.
Training example	An entity with features that will be used as input for learning algorithm.
Feature	A known characteristic of a training or a new example; a feature is equivalent to a characteristic.
Variable	In this context, the value of a feature or a function of several features. This usage is somewhat different from the use of *variable* in a computer program.
Record	A container where an example is stored; such a record is composed of fields.
Field	Part of a record that contains the value of a feature (a variable).
Predictor variable	A feature selected for use as input to a classification model. Not all features need be used. Some features may be algorithmic combinations of other features.
Target variable	A feature that the classification model is attempting to estimate: the target variable is categorical, and its determination is the aim of the classification system.

Input and Output of a classification model



Four types of values for predictor variables

Type of value	Description
Continuous	This is a floating-point value. This type of value might be a price, a weight, a time, or anything else that has a numerical magnitude and where this magnitude is the key property of the value.
Categorical	A categorical value can have one of a set of prespecified values. Typically the set of categorical values is relatively small and may be as small as two, although the set can be quite large. Boolean values are generally treated as categorical values. Another example might be a vendor ID.
Word-like	A word-like value is like a categorical value, but it has an open-ended set of possible values.
Text-like	A text-like value is a sequence of word-like values, all of the same kind. Text is the classic example of a text-like value, but a list of email addresses or URLs is also text-like.

Sample data that illustrates all four value types

Name	Type	Value
from-address	Word-like	George <george@fumble-tech.com>
in-address-book?	Categorical (TRUE, FALSE)	TRUE
non-spam-words	Text-like	Ted, Mahout, User, lunch
spam-words	Text-like	available
unknown-words	Continuous	0
message-length	Continuous	31

Supervised vs. Unsupervised Learning

Classification algorithms are related to, but still quite different from, clustering algorithms such as the k-means algorithm described in previous chapters. Classification algorithms are a form of supervised learning, as opposed to unsupervised learning, which happens with clustering algorithms. A supervised learning algorithm is one that's given examples that contain the desired value of a target variable. Unsupervised algorithms aren't given the desired answer, but instead must find something plausible on their own.

Supervised and unsupervised learning algorithms can often be usefully combined. A clustering algorithm can be used to create features that can then be used by a learning algorithm, or the output of several classifiers can be used as features by a clustering algorithm. Moreover, clustering systems often build a model that can be used to categorize new data. This clustering system model works much like the model produced by a classification system. The difference lies in what data was used to produce the model. For classification, the training data includes the target variables; for clustering, the training data doesn't include target variables.

Work flow in a typical classification project

Stage	Step
1. Training the model	Define target variable. Collect historical data. Define predictor variables. Select a learning algorithm. Use the learning algorithm to train the model.
2. Evaluating the model	Run test data. Adjust the input (use different predictor variables, different algorithms, or both).
3. Using the model in production	Input new examples to estimate unknown target values. Retrain the model as needed.

Fundamental classification algorithms

Example of fundamental classification algorithms:

- Naive Bayesian
- Complementary Naive Bayesian
- Stochastic Gradient Descent (SDG)
- Random Forest
- Support Vector Machines

Choose algorithm

Size of data set	Mahout algorithm	Execution model	Characteristics
Small to medium (less than tens of millions of training examples)	Stochastic gradient descent (SGD) family: `OnlineLogisticRegression`, `CrossFoldLearner`, `AdaptiveLogisticRegression`	Sequential, online, incremental	Uses all types of predictor variables; sleek and efficient over the appropriate data range (up to millions of training examples)
	Support vector machine (SVM)	Sequential	Experimental still; sleek and efficient over the appropriate data range
Medium to large (millions to hundreds of millions of training examples)	Naive Bayes	Parallel	Strongly prefers text-like data; medium to high overhead for training; effective and useful for data sets too large for SGD or SVM
	Complementary naive Bayes	Parallel	Somewhat more expensive to train than naive Bayes; effective and useful for data sets too large for SGD, but has similar limitations to naive Bayes
Small to medium (less than tens of millions of training examples)	Random forests	Parallel	Uses all types of predictor variables; high overhead for training; not widely used (yet); costly but offers complex and interesting classifications, handles nonlinear and conditional relationships in data better than other techniques

Stochastic Gradient Descent (SGD)

Both statistical estimation and machine learning consider the problem of minimizing an objective function that has the form of a sum:

$$Q(w) = \sum_{i=1}^{n} Q_i(w),$$

where the parameter w is to be estimated and where typically each summand function $Q_i()$ is associated with the i-th observation in the data set (used for training).

- Choose an initial vector of parameters w and learning rate α.
- Randomly shuffle examples in the training set.
- Repeat until an approximate minimum is obtained:
  - For $i = 1, 2, ..., n$, do:
    - $w := w - \alpha \nabla Q_i(w).$

Let's suppose we want to fit a straight line $y = w_1 + w_2 x$ to a training set of two-dimensional points $(x_1, y_1), \ldots, (x_n, y_n)$ using least squares. The objective function to be minimized is:

$$Q(w) = \sum_{i=1}^{n} Q_i(w) = \sum_{i=1}^{n} (w_1 + w_2 x_i - y_i)^2 .$$

The last line in the above pseudocode for this specific problem will become:

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} := \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \alpha \begin{bmatrix} \sum_{i=1}^{n} 2(w_1 + w_2 x_i - y_i) \\ \sum_{i=1}^{n} 2x_i(w_1 + w_2 x_i - y_i) \end{bmatrix}.$$

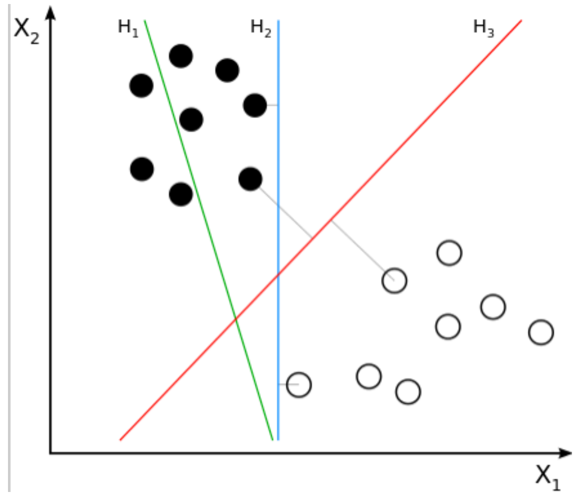
Characteristic of SGD

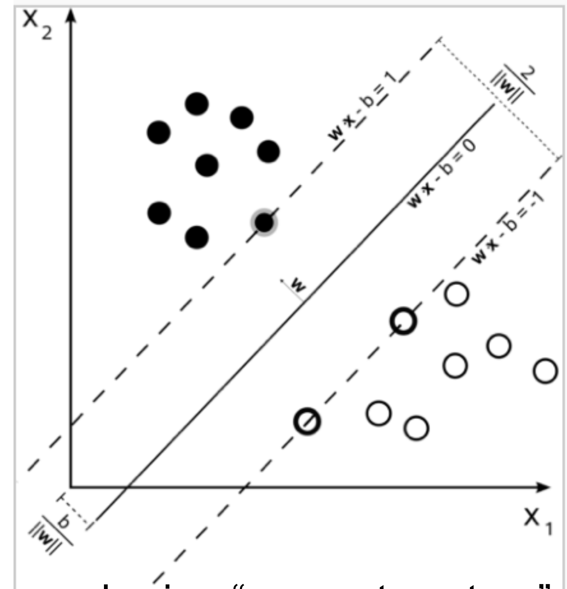
THE SGD ALGORITHM

Stochastic gradient descent (SGD) is a widely used learning algorithm in which each training example is used to tweak the model slightly to give a more correct answer for that one example. This incremental approach is repeated over many training examples. With some special tricks to decide how much to nudge the model, the model accurately classifies new data after seeing only a modest number of examples. Although SGD algorithms are difficult to parallelize effectively, they're often so fast that for a wide variety of applications, parallel execution isn't necessary.

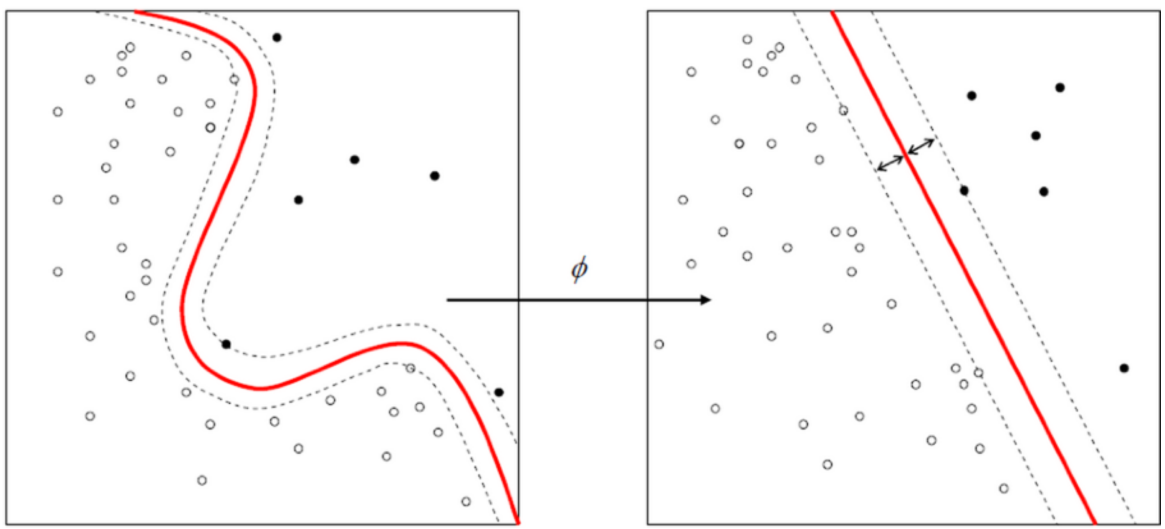
Importantly, because these algorithms do the same simple operation for each training example, they require a constant amount of memory. For this reason, each training example requires roughly the same amount of work. These properties make SGD-based algorithms scalable in the sense that twice as much data takes only twice as long to process.

Support Vector Machine (SVM)





maximize boundary distances; remembering "support vectors"



nonlinear kernels

Example SVM code in Spark

```
from pyspark.ml.classification import LinearSVC

# Load training data
training = spark.read.format("libsvm").load("data/mllib/sample_libsvm_data.txt")

lsvc = LinearSVC(maxIter=10, regParam=0.1)

# Fit the model
lsvcModel = lsvc.fit(training)

# Print the coefficients and intercept for linear SVC
print("Coefficients: " + str(lsvcModel.coefficients))
print("Intercept: " + str(lsvcModel.intercept))
```

Naive Bayes

Training set:

sex	height (feet)	weight (lbs)	foot size(inches)
male	6	180	12
male	5.92 (5'11")	190	11
male	5.58 (5'7")	170	12
male	5.92 (5'11")	165	10
female	5	100	6
female	5.5 (5'6")	150	8
female	5.42 (5'5")	130	7
female	5.75 (5'9")	150	9

Classifier using Gaussian distribution assumptions:

sex	mean (height)	variance (height)	mean (weight)	variance (weight)	mean (foot size)	variance (foot size)
male	5.855	3.5033e-02	176.25	1.2292e+02	11.25	9.1667e-01
female	5.4175	9.7225e-02	132.5	5.5833e+02	7.5	1.6667e+00

$$posterior(male) = \frac{P(male)\,p(height|male)\,p(weight|male)\,p(footsize|male)}{evidence}$$

Test Set:

sex	height (feet)	weight (lbs)	foot size(inches)
sample	6	130	8

$$evidence = P(male)\,p(height|male)\,p(weight|male)\,p(footsize|male) \\ +P(female)\,p(height|female)\,p(weight|female)\,p(footsize|female)$$

$$P(male) = 0.5$$
$$p(\text{height}|\text{male}) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(\frac{-(6-\mu)^2}{2\sigma^2}\right) \approx 1.5789,$$
$$p(\text{weight}|\text{male}) = 5.9881 \cdot 10^{-6}$$
$$p(\text{foot size}|\text{male}) = 1.3112 \cdot 10^{-3}$$
$$\text{posterior numerator (male)} = \text{their product} = 6.1984 \cdot 10^{-9}$$

$$P(\text{female}) = 0.5$$
$$p(\text{height}|\text{female}) = 2.2346 \cdot 10^{-1}$$
$$p(\text{weight}|\text{female}) = 1.6789 \cdot 10^{-2}$$
$$p(\text{foot size}|\text{female}) = 2.8669 \cdot 10^{-1}$$
$$\text{posterior numerator (female)} = \text{their product} = 5.3778 \cdot 10^{-4}$$

==> female

Random Forest

Random forests are an ensemble learning method for classification (and regression) that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes output by individual trees.

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x_1, \ldots, x_n$ with responses $Y = y_1$ through y_n, bagging repeatedly selects a bootstrap sample of the training set and fits trees to these samples:

For $b = 1$ through B:

1. Sample, with replacement, n training examples from X, Y; call these X_b, Y_b.
2. Train a decision or regression tree f_b on X_b, Y_b.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

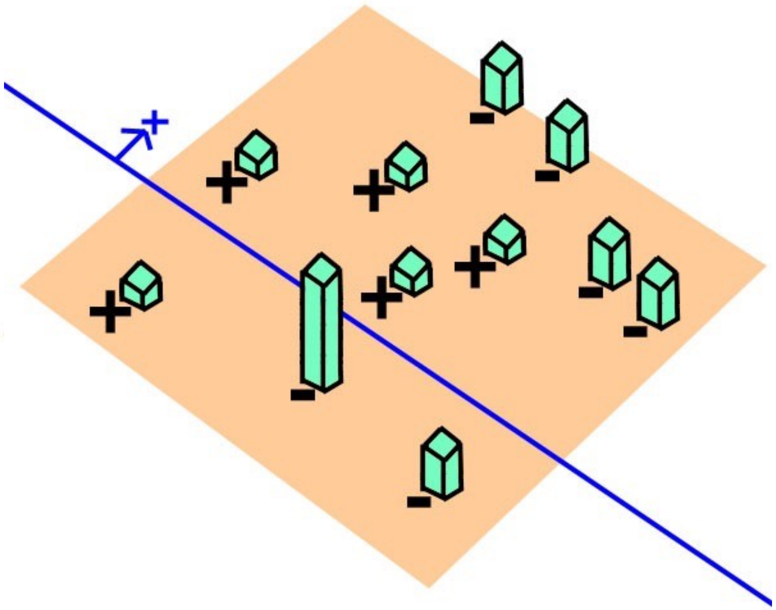
$$\hat{f} = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(x')$$

or by taking the majority vote in the case of decision trees.

Random forest uses a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features.

Adaboost Example

- 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

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

Example — User Modeling using Time-Sensitive Adaboost

- Obtain simple classifier on each feature, e.g., setting threshold on parameters, or binary inference on input parameters.

- The system classify whether a new document is interested by a person via Adaptive Boosting (Adaboost):
  - The final classifier is a linear weighted combination of single-feature classifiers.
  - Given the single-feature simple classifiers, assigning weights on the training samples based on whether a sample is correctly or mistakenly classified. <== Boosting.
  - Classifiers are considered sequentially. The selected weights in previous considered classifiers will affect the weights to be selected in the remaining classifiers. <== Adaptive.
  - According to the summed errors of each simple classifier, assign a weight to it. The final classifier is then the weighted linear combination of these simple classifiers.

- Our new Time-Sensitive Adaboost algorithm:
  - In the AdaBoost algorithm, all samples are regarded equally important at the beginning of the learning process
  - We propose a time-adaptive AdaBoost algorithm that assigns larger weights to the latest training samples

People select apples according to their shapes, sizes, other people's interest, etc.

Each attribute is a simple classifier used in Adaboost.

Time-Sensitive Adaboost [Song, Lin, et al. 2005]

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

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

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

Algorithm: ***Time-Sensitive Adaboost***

Given: $(x_1, c_1, t_1), \ldots, (x_N, c_N, t_N)$ where $x_i \in X$, $c_i \subseteq \{-1, 1\}$, N is the size of samples in the training set; current time t, and τ

For $s = 1, \ldots, S$

Initialize $D_1(i) = 1 / (N \cdot \exp(\tau \cdot (t - t_i)))$.

Set the weight α_s of the current weak hypothesis h_s according to its weighted error rate ε_s

$$\alpha_s = \frac{1}{2} \ln\left(\frac{1 - \varepsilon_s}{\varepsilon_s}\right)$$

where $\varepsilon_s = \sum_{i=1}^{N} D_s(i) h_s(x_i) c_i$.

Update $D_{s+1}(i) = \dfrac{D_s(i) \exp(-\alpha_s c_i \exp(-\tau \cdot (t - t_i)) h_s(x_i))}{Z_s}$

where Z_s is a normalization term.

End

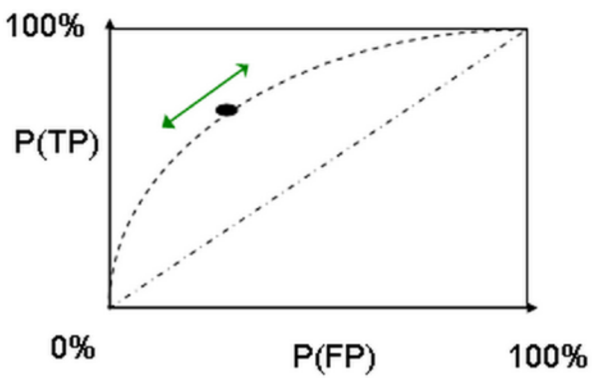
Find weak hypothesis by: $h_s = \arg\min_{h_j \in H} \varepsilon_j$.

Output: the final hypothesis: $H(x) = \text{sign}(F(x))$

where $F(x) = \sum_{s=1}^{S} \alpha_s h_s(x)$.

Evaluate the model

```
$  bin/mahout runlogistic --input donut.csv --model ./model \
        --auc --confusion
AUC = 0.57
confusion: [[27.0, 13.0], [0.0, 0.0]]
...
```



AUC (0 ~ 1):
- 1 — perfect
- 0 — perfectly wrong
- 0.5 — random

True positive	**False positive** (Type I error)
False negative (Type II error)	**True negative**

confusion matrix

Option	What it does
`--quiet`	Produces less status and progress output.
`--auc`	Prints AUC score for model versus input data after reading data.
`--scores`	Prints target variable value and scores for each input example.
`--confusion`	Prints confusion matrix for a particular threshold (see `--threshold`).
`--input <input>`	Reads data records from specified file or resource.
`--model <model>`	Reads model from specified file.

Confusion Matrix

```
=======================================================
Confusion Matrix
-------------------------------------------------------
a    b    c    d    e    f    <--Classified as
9    0    1    0    0    0     |  10       a      = one
0    9    0    0    1    0     |  10       b      = two
0    0    10   0    0    0     |  10       c      = three
0    0    1    8    1    0     |  10       d      = four
1    1    0    0    7    1     |  10       e      = five
0    0    0    0    1    9     |  10       f      = six
Default Category: one: 6
```

```
BufferedReader in = new
     BufferedReader(new FileReader(inputFile));
List<String> symbols = new ArrayList<String>();
String line = in.readLine();
while (line != null) {
  String[] pieces = line.split(",");
  if (!symbols.contains(pieces[0])) {
     symbols.add(pieces[0]);
  }
  line = in.readLine();
}

ConfusionMatrix x2 = new ConfusionMatrix(symbols, "unknown");

in = new BufferedReader(new FileReader(inputFile));
line = in.readLine();
while (line != null) {
  String[] pieces = line.split(",");
  String trueValue = pieces[0];
  String estimatedValue = pieces[1];
  x2.addInstance(trueValue, estimatedValue);
  line = in.readLine();
}
System.out.printf("%s\n\n", x2.toString());
```

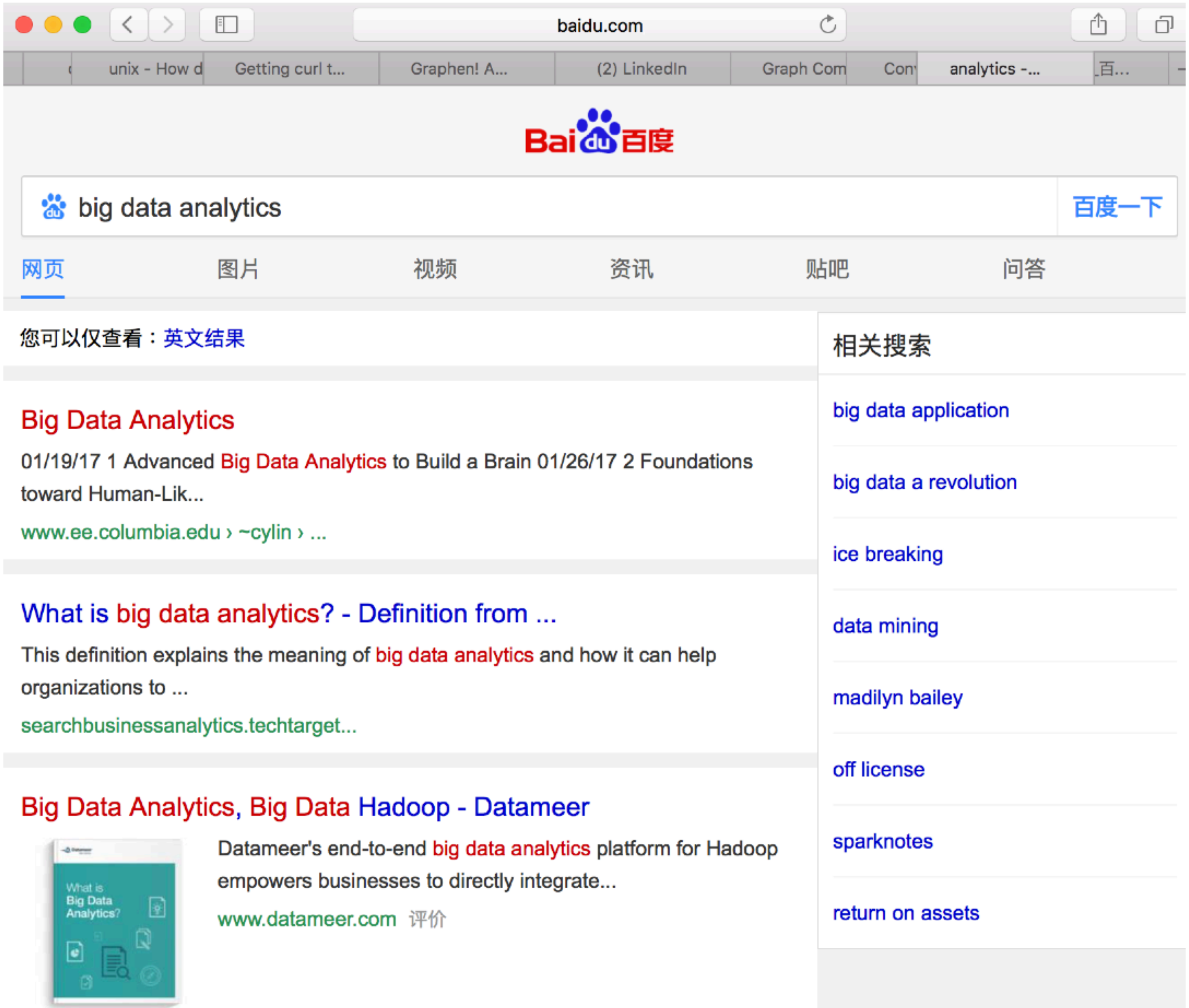
Reads and remembers values

Counts the pairs

Input contains target and model output

x2 gets true answer and score

Average Precision — commonly used in sorted results



'Average Precision' is the metric that is used for evaluating 'sorted' results.

— commonly used for search & retrieval, anomaly detection, etc.

Average Precision = average of the precision values of all correct answers up to them,
==> i.e., calculating the precision value up to the Top n 'correct' answers.
Average all Pn.

Classifiers that go bad

When working with real data and real classifiers it's almost a rule that the first attempts to build models will fail, occasionally spectacularly. Unlike normal software engineering, the failures of models aren't usually as dramatic as a null pointer dereference or out-of-memory exception. Instead, a failing model can appear to produce miraculously accurate results. Such a model can also produce results so wrong that it seems the model is trying to be incorrect. It's important to be somewhat dubious of extremely good or bad results, especially if they occur early in a model's development.

Target leak

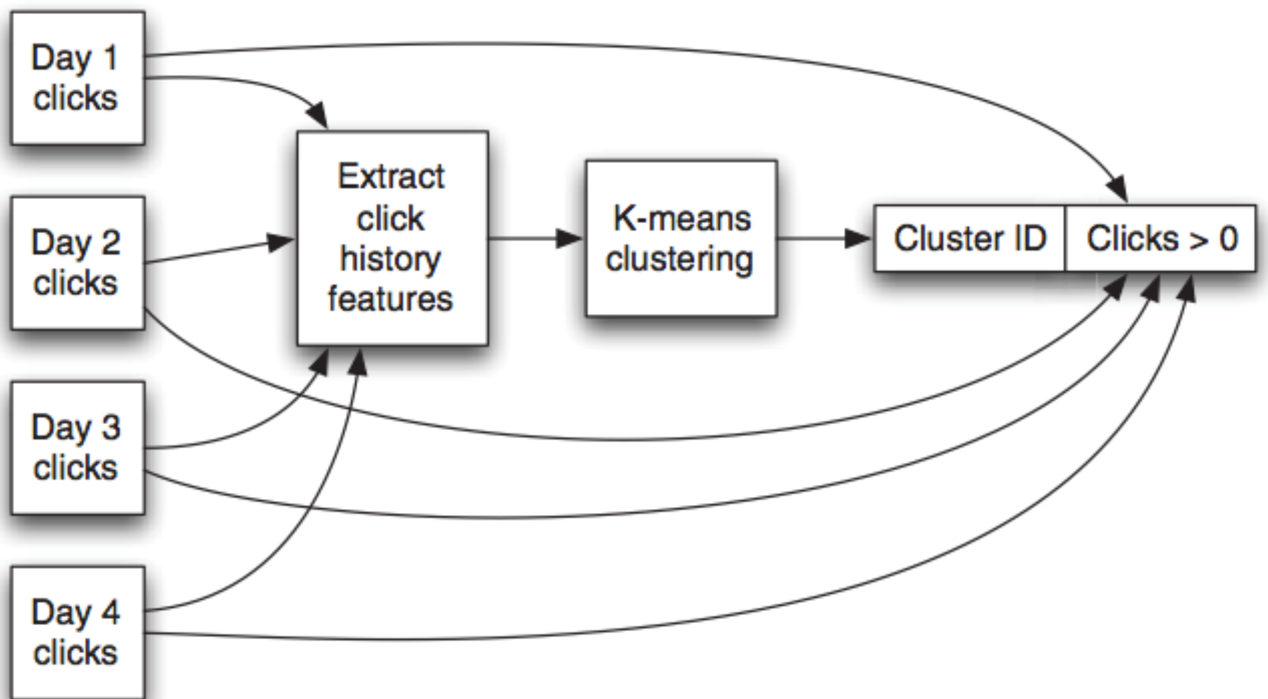
- A target leak is a bug that involves unintentionally providing data about the target variable in the section of the predictor variables.
- Don't confused with intentionally including the target variable in the record of a training example.
- Target leaks can seriously affect the accuracy of the classification system.

Example: Target Leak

```
private static final SimpleDateFormat("MMM-yyyy");
// 1997-01-15 00:01:00 GMT
private static final long DATE_REFERENCE = 853286460;

...
  long date = (long) (1000 *
      (DATE_REFERENCE + target * MONTH + 1 * WEEK * rand.nextDouble()));
  Reader dateString = new StringReader(df.format(new Date(date)));
  countWords(analyzer, words, dateString);
```

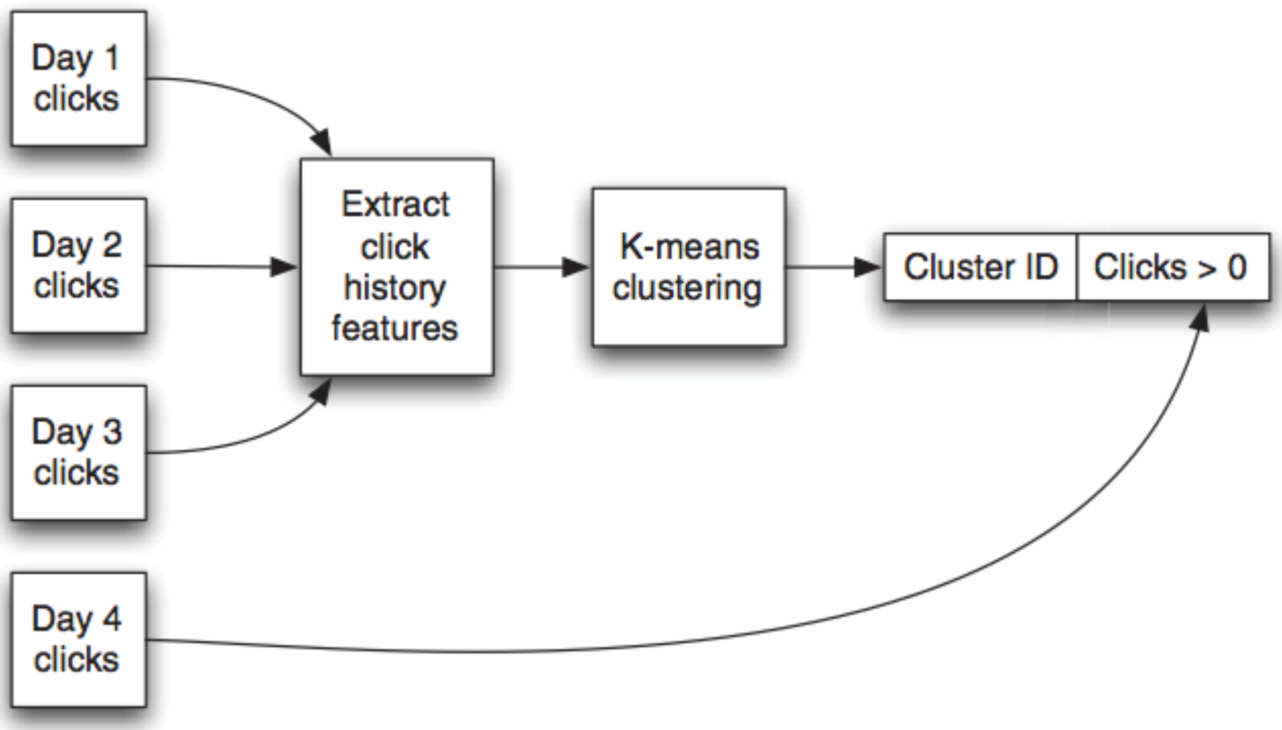
This date field is chosen so that all the documents from the same newsgroup appear to have come from the same month, but documents from different newsgroups come from different months.

Avoid Target Leaks



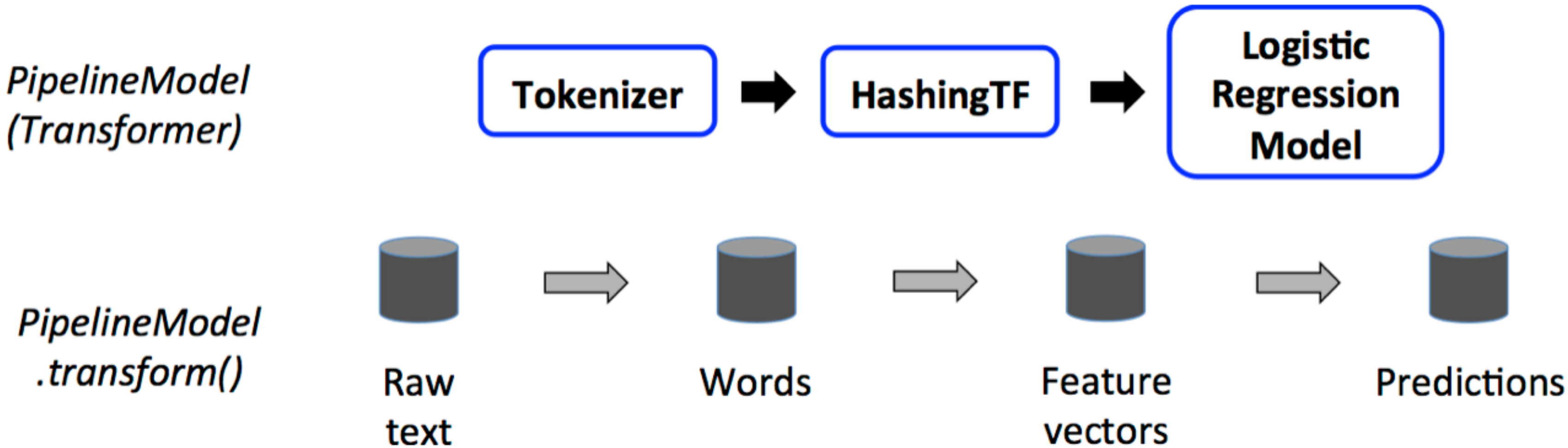
Don't do this: click-history clustering can introduce a target leak in the training data because the target variable (Clicks > 0) is based on the same data as the cluster ID.

Avoid Target Leaks — II



A good way to avoid a target leak: compute click history clusters based on days 1, 2, and 3, and derive the target variable (Clicks > 0) from day 4.

Spark ML Pipeline Example — classifier



Spark Tokenizer

```
import org.apache.spark.ml.feature.Tokenizer
val tok = new Tokenizer()

// dataset to transform
val df = Seq(
  (1, "Hello world!"),
  (2, "Here is yet another sentence.")).toDF("id", "sentence")

val tokenized = tok.setInputCol("sentence").setOutputCol("tokens").transform(df)

scala> tokenized.show(truncate = false)
+---+-----------------------------+-----------------------------------+
|id |sentence                     |tokens                             |
+---+-----------------------------+-----------------------------------+
|1  |Hello world!                 |[hello, world!]                    |
|2  |Here is yet another sentence.|[here, is, yet, another, sentence.]|
+---+-----------------------------+-----------------------------------+
```

Spark Tokenizer

```python
from pyspark.ml.feature import Tokenizer, RegexTokenizer
from pyspark.sql.functions import col, udf
from pyspark.sql.types import IntegerType

sentenceDataFrame = spark.createDataFrame([
    (0, "Hi I heard about Spark"),
    (1, "I wish Java could use case classes"),
    (2, "Logistic,regression,models,are,neat")
], ["id", "sentence"])

tokenizer = Tokenizer(inputCol="sentence", outputCol="words")

regexTokenizer = RegexTokenizer(inputCol="sentence", outputCol="words", pattern="\\W")
# alternatively, pattern="\\w+", gaps(False)

countTokens = udf(lambda words: len(words), IntegerType())

tokenized = tokenizer.transform(sentenceDataFrame)
tokenized.select("sentence", "words")\
    .withColumn("tokens", countTokens(col("words"))).show(truncate=False)

regexTokenized = regexTokenizer.transform(sentenceDataFrame)
regexTokenized.select("sentence", "words") \
    .withColumn("tokens", countTokens(col("words"))).show(truncate=False)
```

Vectorization of text

Vector Space Model: Term Frequency (TF)

For example, if the word *horse* is assigned to the 39,905th index of the vector, the word *horse* will correspond to the 39,905th dimension of document vectors. A document's vectorized form merely consists, then, of the number of times each word occurs in the document, and that value is stored in the vector along that word's dimension. The dimension of these document vectors can be very large.

Stop Words: *a*, *an*, *the*, *who*, *what*, *are*, *is*, *was*, and so on.

Stemming:

A stemmer for English, for example, should identify the string "cats" (and possibly "catlike", "catty" etc.) as based on the root "cat", and "stemmer", "stemming", "stemmed" as based on "stem". A stemming algorithm reduces the words "fishing", "fished", and "fisher" to the root word, "fish". On the other hand, "argue", "argued", "argues", "arguing", and "argus" reduce to the stem "argu" (illustrating the case where the stem is not itself a word or root) but "argument" and "arguments" reduce to the stem "argument".

Spark StopWordsRemover

Examples

Assume that we have the following DataFrame with columns `id` and `raw`:

```
 id | raw
----|----------
 0  | [I, saw, the, red, baloon]
 1  | [Mary, had, a, little, lamb]
```

Applying `StopWordsRemover` with `raw` as the input column and `filtered` as the output column, we should get the following:

```
 id | raw                         | filtered
----|-----------------------------|--------------------
 0  | [I, saw, the, red, baloon]  |  [saw, red, baloon]
 1  | [Mary, had, a, little, lamb]|[Mary, little, lamb]
```

In `filtered`, the stop words “I”, “the”, “had”, and “a” have been filtered out.

Spark StopWordsRemover

```
from pyspark.ml.feature import StopWordsRemover

sentenceData = spark.createDataFrame([
    (0, ["I", "saw", "the", "red", "balloon"]),
    (1, ["Mary", "had", "a", "little", "lamb"])
], ["id", "raw"])

remover = StopWordsRemover(inputCol="raw", outputCol="filtered")
remover.transform(sentenceData).show(truncate=False)
```

Most Popular Stemming algorithms

Lookup algorithms

A simple stemmer looks up the inflected form in a lookup table. The advantages of this approach is that it is simple, fast, and easily handles exceptions. The disadvantages are that all inflected forms must be explicitly listed in the table: new or unfamiliar words are not handled, even if they are perfectly regular (e.g. iPads ~ iPad), and the table may be large. For languages with simple morphology, like English, table sizes are modest, but

Suffix-stripping algorithms

Suffix stripping algorithms do not rely on a lookup table that consists of inflected forms and root form relations. Instead, a typically smaller list of "rules" is stored which provides a path for the algorithm, given an input word form, to find its root form. Some examples of the rules include:

- if the word ends in 'ed', remove the 'ed'
- if the word ends in 'ing', remove the 'ing'
- if the word ends in 'ly', remove the 'ly'

n-gram

It was the best of time. it was the worst of times. ==> bigram

It was
was the
the best
best of
of times
times it
it was
was the
the worst
worst of
of times

Mahout provides a log-likelihood test to reduce the dimensions of n-grams

N-gram code example

```
from pyspark.ml.feature import NGram

wordDataFrame = spark.createDataFrame([
    (0, ["Hi", "I", "heard", "about", "Spark"]),
    (1, ["I", "wish", "Java", "could", "use", "case", "classes"]),
    (2, ["Logistic", "regression", "models", "are", "neat"])
], ["id", "words"])

ngram = NGram(n=2, inputCol="words", outputCol="ngrams")

ngramDataFrame = ngram.transform(wordDataFrame)
ngramDataFrame.select("ngrams").show(truncate=False)
```

Word2Vec

The `Word2VecModel` transforms each document into a vector using the average of all words in the document; this vector can then be used as features for prediction, document similarity calculations, etc.

```
from pyspark.ml.feature import Word2Vec

# Input data: Each row is a bag of words from a sentence or document.
documentDF = spark.createDataFrame([
    ("Hi I heard about Spark".split(" "), ),
    ("I wish Java could use case classes".split(" "), ),
    ("Logistic regression models are neat".split(" "), )
], ["text"])

# Learn a mapping from words to Vectors.
word2Vec = Word2Vec(vectorSize=3, minCount=0, inputCol="text", outputCol="result")
model = word2Vec.fit(documentDF)

result = model.transform(documentDF)
for row in result.collect():
    text, vector = row
    print("Text: [%s] => \nVector: %s\n" % (", ".join(text), str(vector)))
```

CountVectorizer

`CountVectorizer` and `CountVectorizerModel` aim to help convert a collection of text documents to vectors token counts. When an a-priori dictionary is not available, `CountVectorizer` can be used as an `Estimator` to extract the vocabulary, and generates a `CountVectorizerModel`. The model produces sparse representation the documents over the vocabulary, which can then be passed to other algorithms like LDA.

Examples

Assume that we have the following DataFrame with columns `id` and `texts`:

```
 id | texts
----|----------
 0  | Array("a", "b", "c")
 1  | Array("a", "b", "b", "c", "a")
```

each row in `texts` is a document of type Array[String]. Invoking fit of `CountVectorizer` produces a `CountVectorizerModel` with vocabulary (a, b, c). Then the output column "vector" after transformation contains:

```
 id | texts                           | vector
----|---------------------------------|---------------
 0  | Array("a", "b", "c")            | (3,[0,1,2],[1.0,1.0,1.0])
 1  | Array("a", "b", "b", "c", "a")  | (3,[0,1,2],[2.0,2.0,1.0])
```

Each vector represents the token counts of the document over the vocabulary.

CountVectorizer

```
from pyspark.ml.feature import CountVectorizer

# Input data: Each row is a bag of words with a ID.
df = spark.createDataFrame([
    (0, "a b c".split(" ")),
    (1, "a b b c a".split(" "))
], ["id", "words"])

# fit a CountVectorizerModel from the corpus.
cv = CountVectorizer(inputCol="words", outputCol="features", vocabSize=3, minDF=2.0)

model = cv.fit(df)

result = model.transform(df)
result.show(truncate=False)
```

FeatureHasher

Feature hashing projects a set of categorical or numerical features into a feature vector of specified dimension (typically substantially smaller than that of the original feature space). This is done using the hashing trick to map features to indices in the feature vector.

The `FeatureHasher` transformer operates on multiple columns. Each column may contain either numeric or categorical features. Behavior and handling of column data types is as follows:

- **Numeric columns:** For numeric features, the hash value of the column name is used to map the feature value to its index in the feature vector. By default, numeric features are not treated as categorical (even when they are integers). To treat them as categorical, specify the relevant columns using the `categoricalCols` parameter.
- **String columns:** For categorical features, the hash value of the string “column_name=value” is used to map to the vector index, with an indicator value of `1.0`. Thus, categorical features are “one-hot” encoded (similarly to using OneHotEncoder with `dropLast=false`).
- **Boolean columns:** Boolean values are treated in the same way as string columns. That is, boolean features are represented as “column_name=true” or “column_name=false”, with an indicator value of `1.0`.

Null (missing) values are ignored (implicitly zero in the resulting feature vector).

FeatureHasher

Examples

Assume that we have a DataFrame with 4 input columns `real`, `bool`, `stringNum`, and `string`. These different data types as input will illustrate the behavior of the transform to produce a column of feature vectors.

```
real| bool|stringNum|string
----|-----|---------|------
 2.2| true|        1|   foo
 3.3|false|        2|   bar
 4.4|false|        3|   baz
 5.5|false|        4|   foo
```

Then the output of `FeatureHasher.transform` on this DataFrame is:

```
real|bool |stringNum|string|features
----|-----|---------|------|-------------------------------------------------------
2.2 |true |1        |foo   |(262144,[51871, 63643,174475,253195],[1.0,1.0,2.2,1.0])
3.3 |false|2        |bar   |(262144,[6031,  80619,140467,174475],[1.0,1.0,1.0,3.3])
4.4 |false|3        |baz   |(262144,[24279,140467,174475,196810],[1.0,1.0,4.4,1.0])
5.5 |false|4        |foo   |(262144,[63643,140467,168512,174475],[1.0,1.0,1.0,5.5])
```

The resulting feature vectors could then be passed to a learning algorithm.

The value of word is reduced more if it is used frequently across all the documents in the dataset.

To calculate the inverse document frequency, the document frequency (DF) for each word is first calculated. Document frequency is the number of documents the word occurs in. The number of times a word occurs in a document isn't counted in document frequency. Then, the inverse document frequency or IDF_i for a word, w_i, is

$$IDF_i = \frac{1}{DF_i}$$

$$W_i = TF_i \cdot IDF_i = TF_i \cdot \frac{N}{DF_i}$$

or

$$W_i = TF_i \cdot \log \frac{N}{DF_i}$$

TF-IDF example

```
doc <- c( "The sky is blue.", "The sun is bright today.",
          "The sun in the sky is bright.", "We can see the shining sun, the bright sun.")
```

TF

```
##              Docs
## Terms       1 2 3 4
##    blue     1 0 0 0
##    bright   0 1 1 1
##    can      0 0 0 1
##    see      0 0 0 1
##    shining  0 0 0 1
##    sky      1 0 1 0
##    sun      0 1 1 2
##    today    0 1 0 0
```

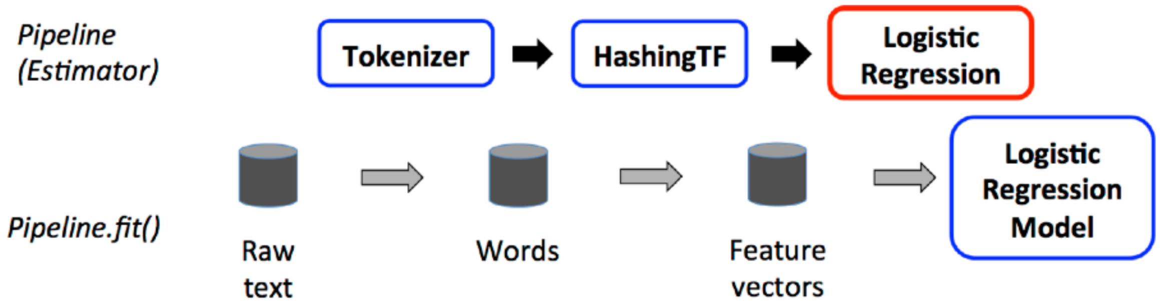
IDF (using the alternative formula)

```
##      blue    bright       can       see   shining       sky       sun
## 0.6931472 0.0000000 0.6931472 0.6931472 0.6931472 0.2876821 0.0000000
##     today
## 0.6931472
```

TF-IDF

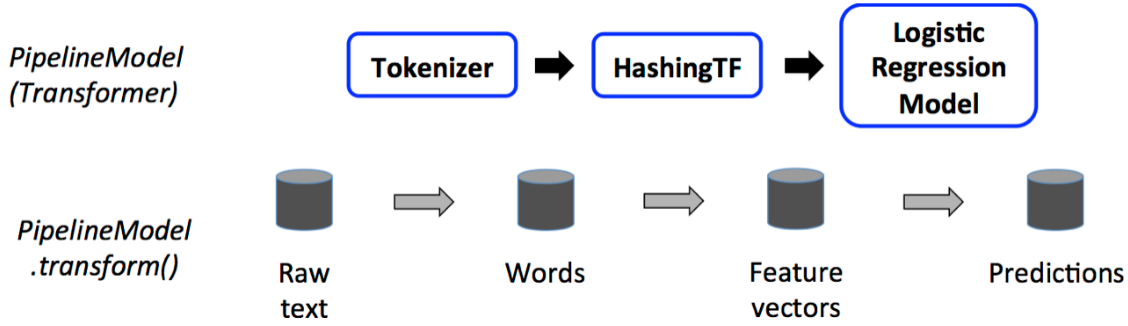
```
from pyspark.ml.feature import HashingTF, IDF, Tokenizer

sentenceData = spark.createDataFrame([
    (0.0, "Hi I heard about Spark"),
    (0.0, "I wish Java could use case classes"),
    (1.0, "Logistic regression models are neat")
], ["label", "sentence"])

tokenizer = Tokenizer(inputCol="sentence", outputCol="words")
wordsData = tokenizer.transform(sentenceData)

hashingTF = HashingTF(inputCol="words", outputCol="rawFeatures", numFeatures=20)
featurizedData = hashingTF.transform(wordsData)
# alternatively, CountVectorizer can also be used to get term frequency vectors

idf = IDF(inputCol="rawFeatures", outputCol="features")
idfModel = idf.fit(featurizedData)
rescaledData = idfModel.transform(featurizedData)

rescaledData.select("label", "features").show()
```

Spark ML Pipeline Example Code I



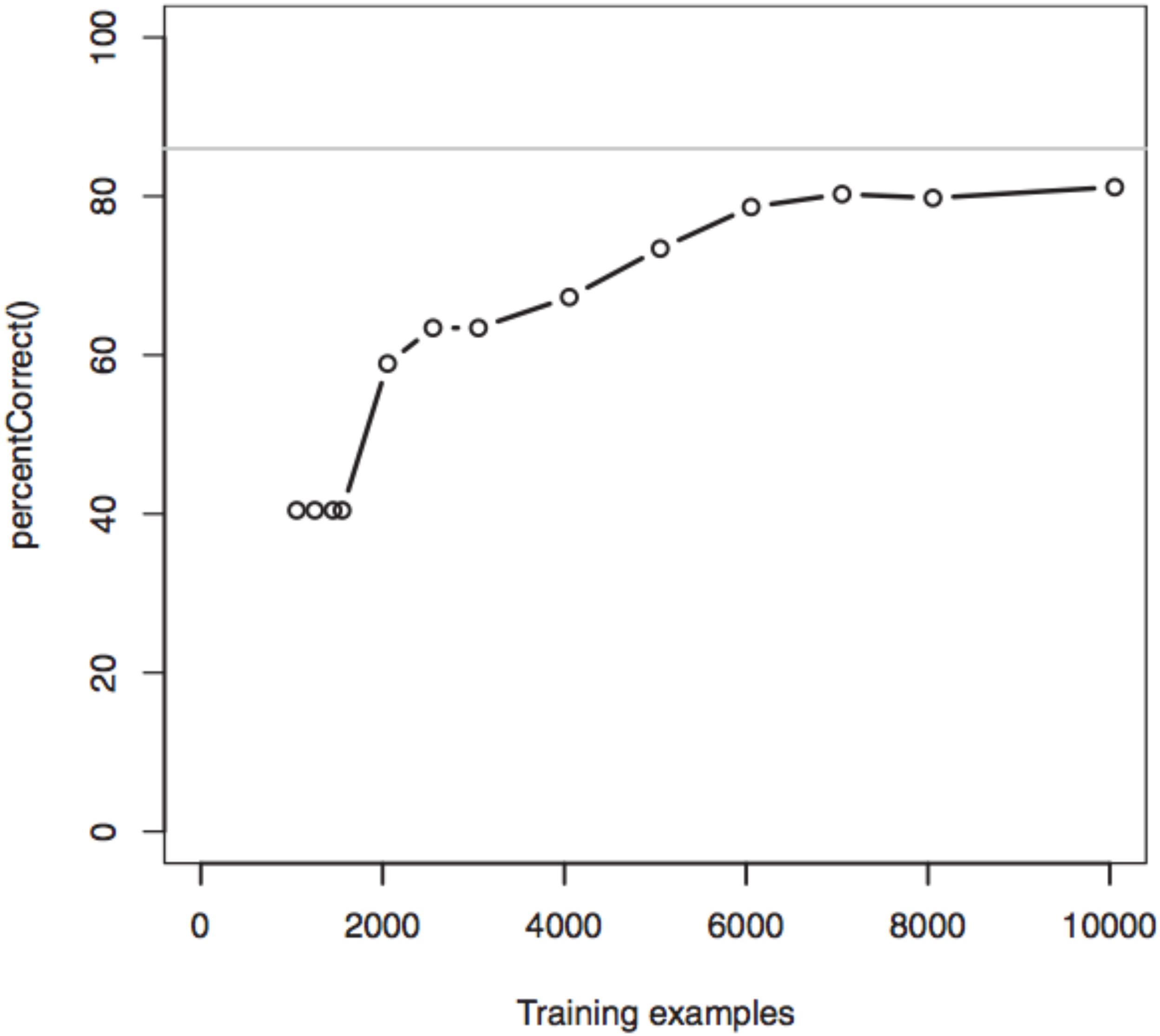

```python
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer

# Prepare training documents from a list of (id, text, label) tuples.
training = spark.createDataFrame([
    (0, "a b c d e spark", 1.0),
    (1, "b d", 0.0),
    (2, "spark f g h", 1.0),
    (3, "hadoop mapreduce", 0.0)
], ["id", "text", "label"])

# Configure an ML pipeline, which consists of three stages: tokenizer, hashingTF, and lr.
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.001)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])
```

Spark ML Pipeline Example Code II




```python
# Prepare test documents, which are unlabeled (id, text) tuples.
test = spark.createDataFrame([
    (4, "spark i j k"),
    (5, "l m n"),
    (6, "spark hadoop spark"),
    (7, "apache hadoop")
], ["id", "text"])

# Make predictions on test documents and print columns of interest.
prediction = model.transform(test)
selected = prediction.select("id", "text", "probability", "prediction")
for row in selected.collect():
    rid, text, prob, prediction = row
    print("(%d, %s) --> prob=%s, prediction=%f" % (rid, text, str(prob), prediction))
```

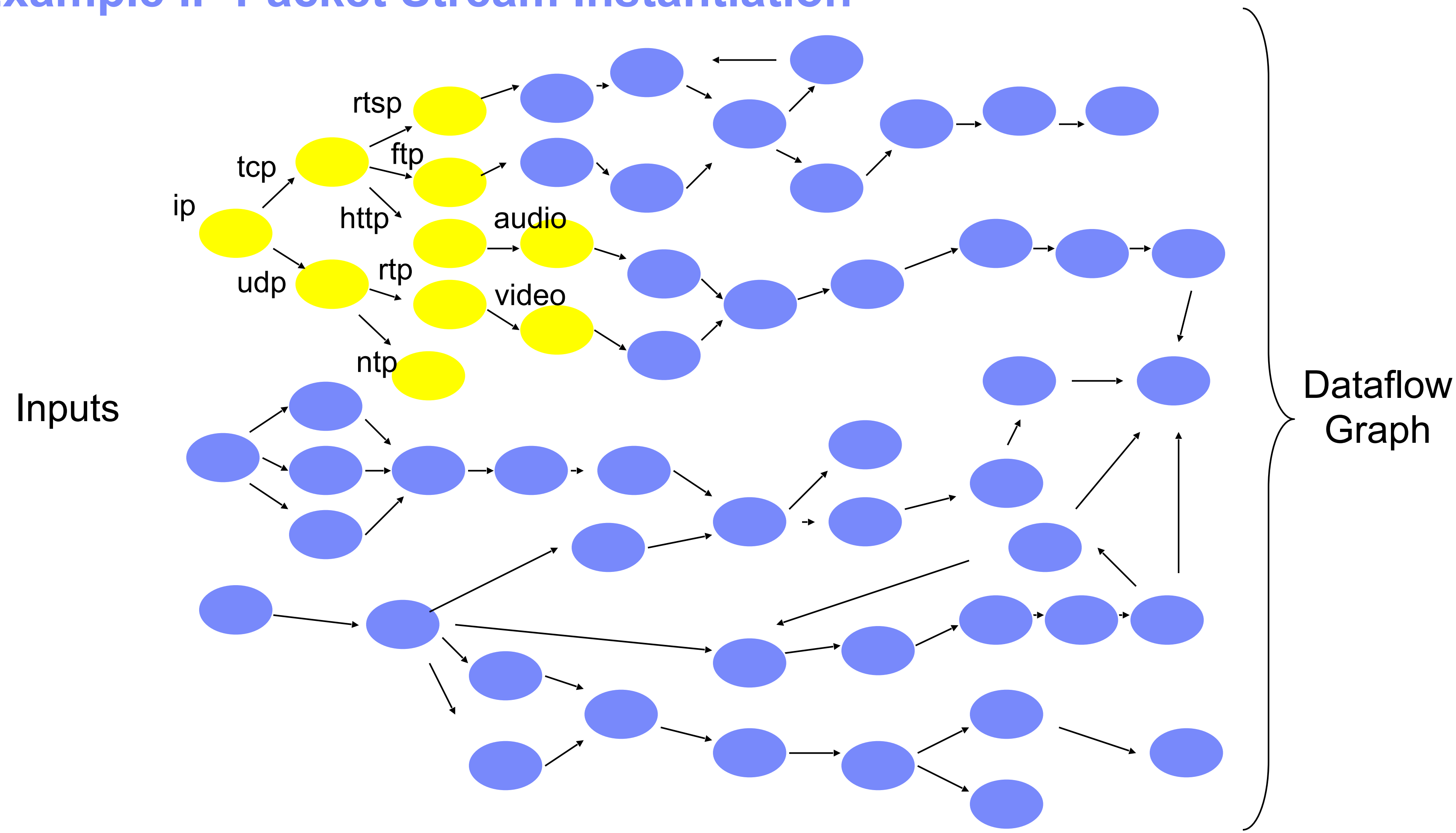
Number of Training Examples vs Accuracy



Increase in average percent correct with increasing number of training examples. The gray bar shows the reasonable maximum performance level that you can expect, based on the best results reported in the research literature.

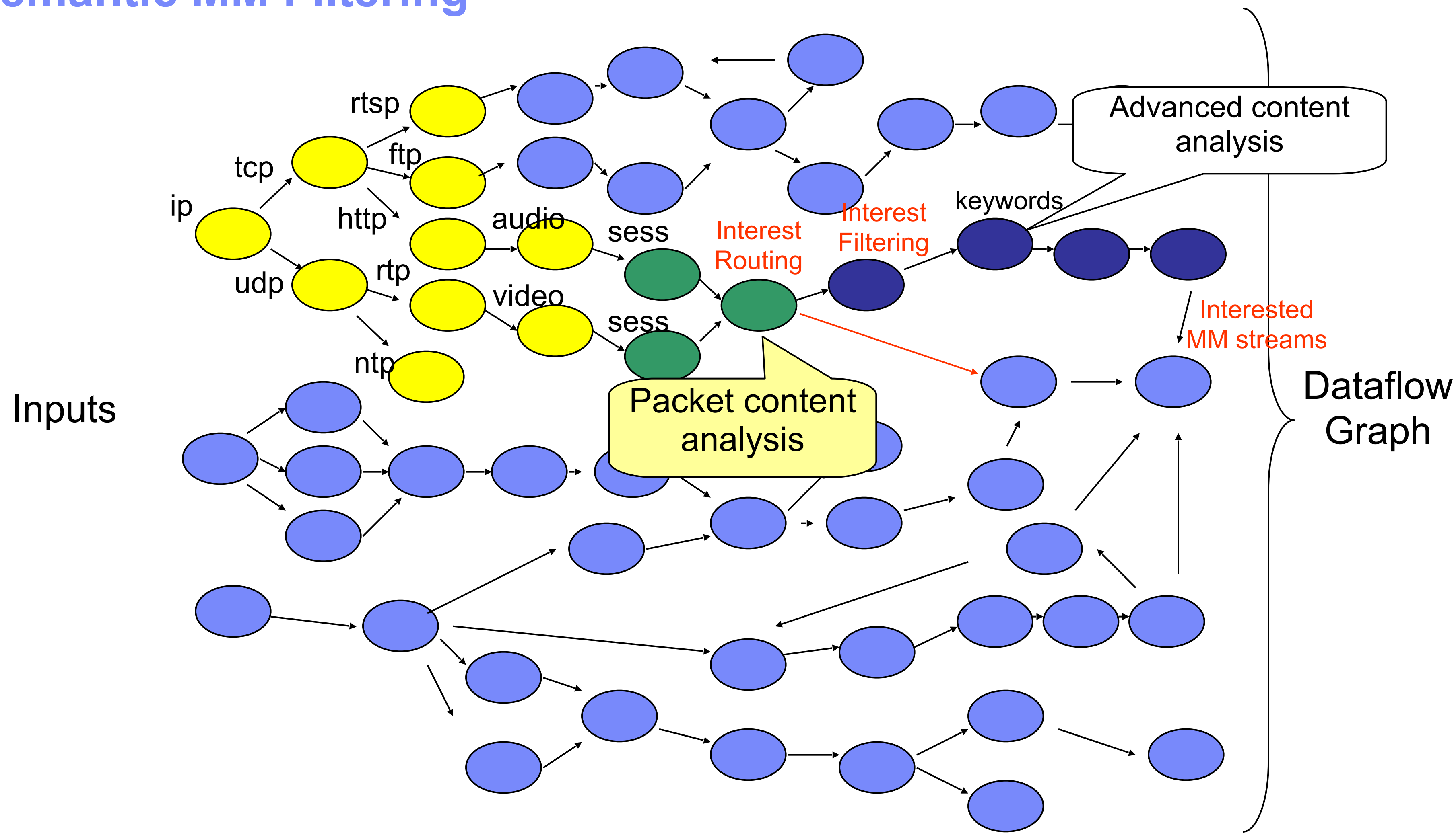
Stream Analyses Technical Challenges

Example IP Packet Stream Instantiation



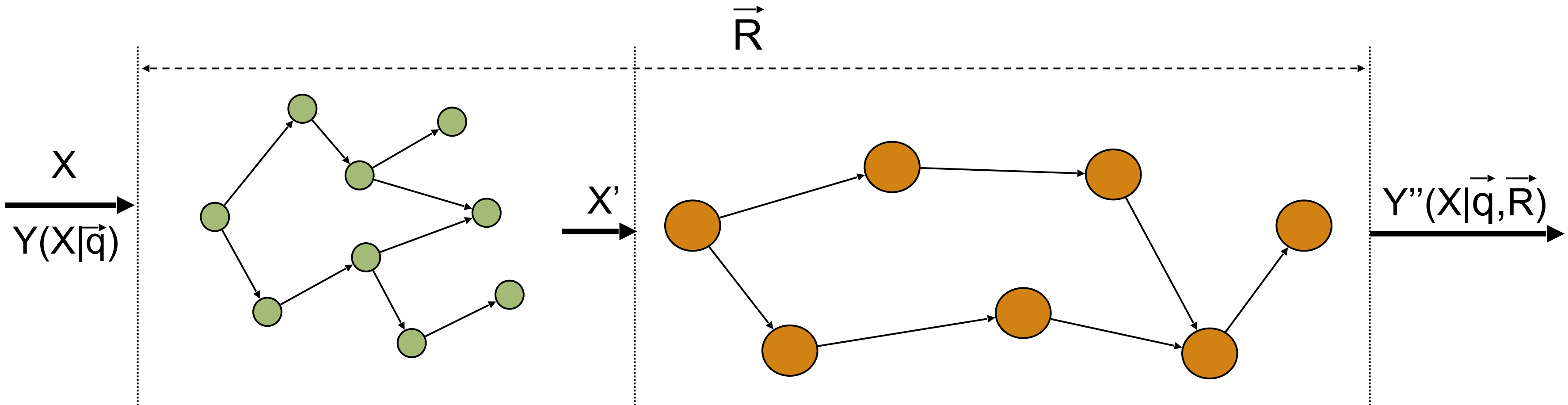
By IBM Dense Information Gliding Team

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),

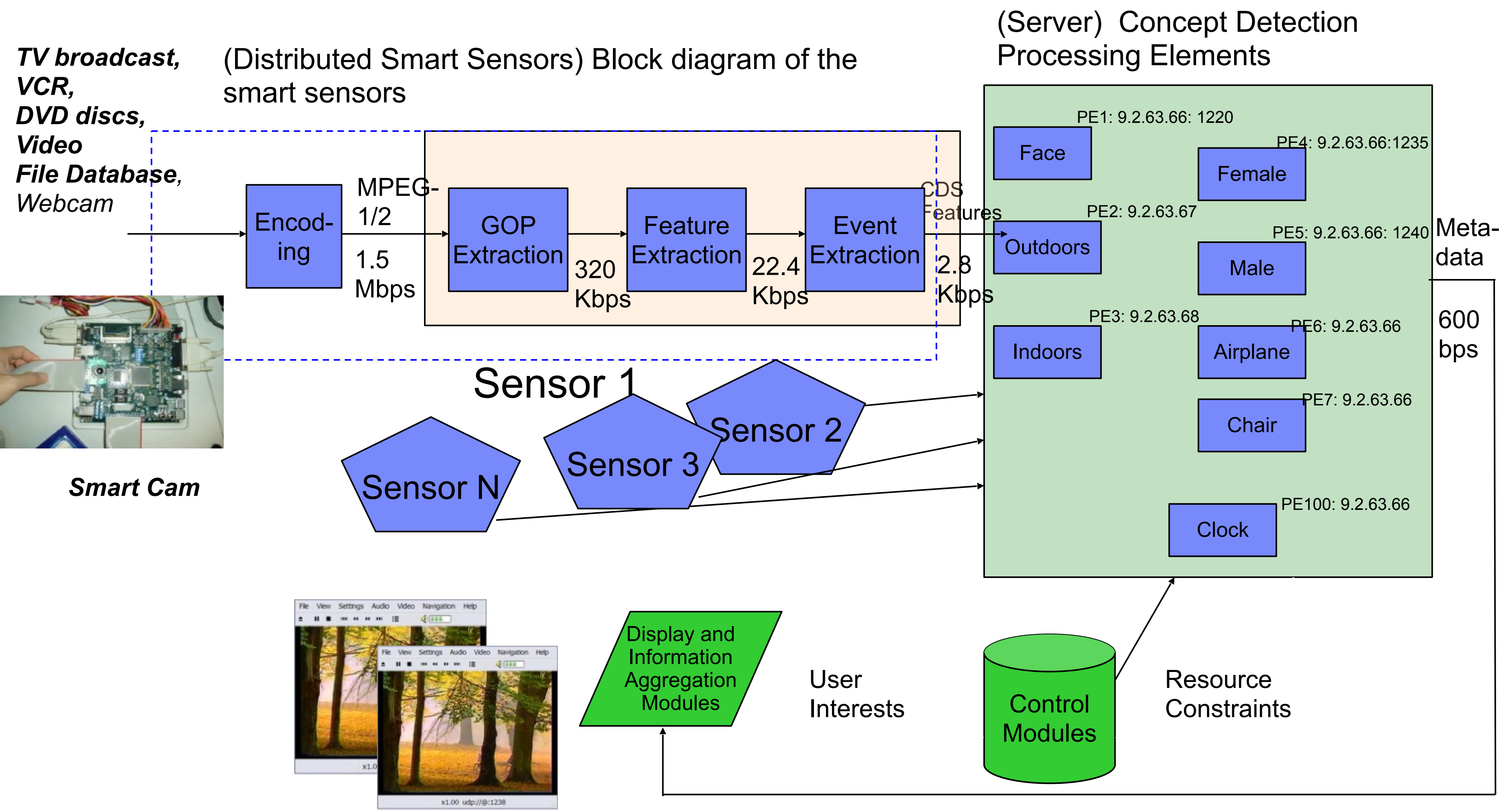
with resource constraint.

Required resource-efficient algorithms for:

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

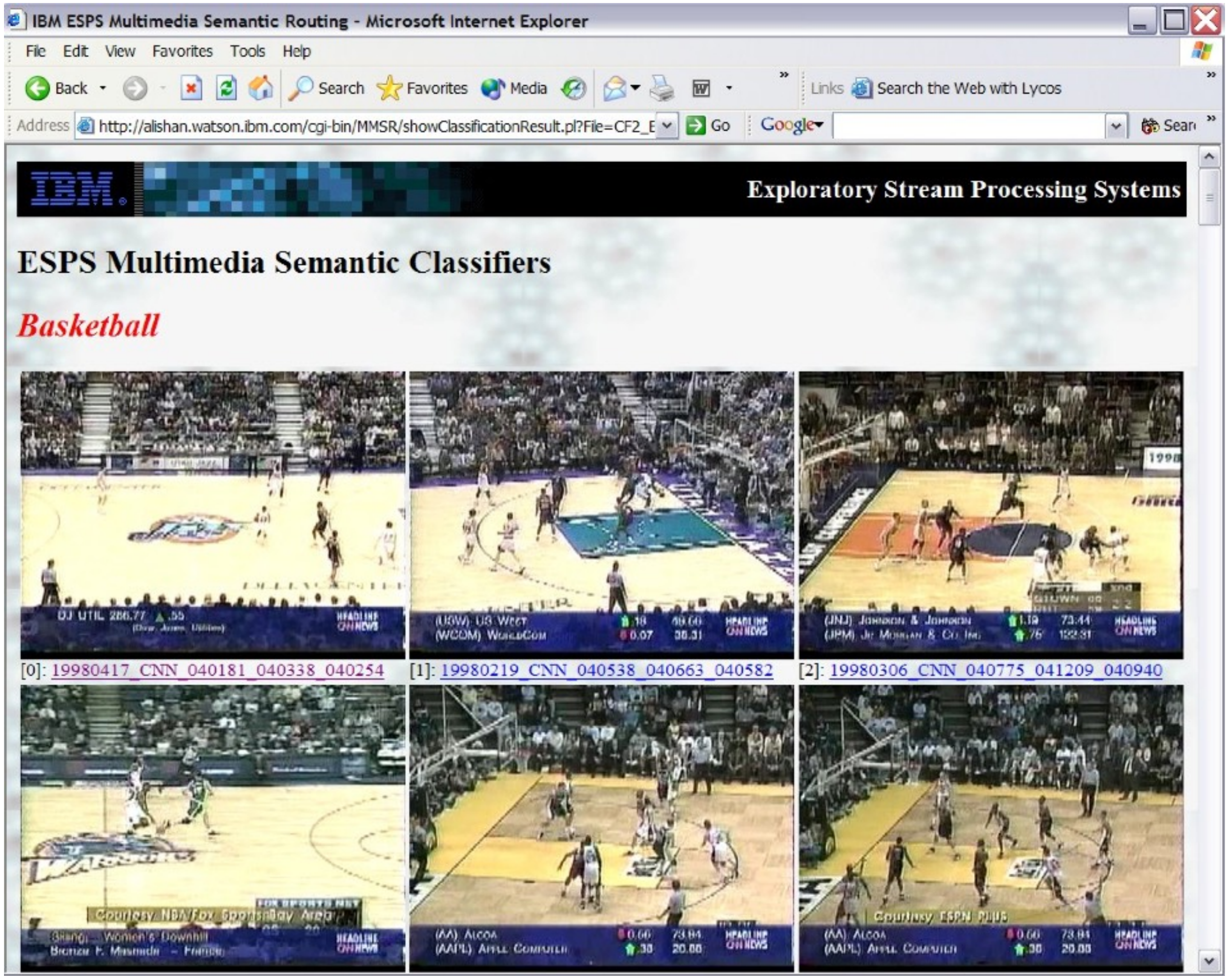
- **Input data X – Queries q – Resource R**
  - Y(X | q): Relevant information
  - Y'(X | q, R) ` Y(X | q): Achievable subset given R

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



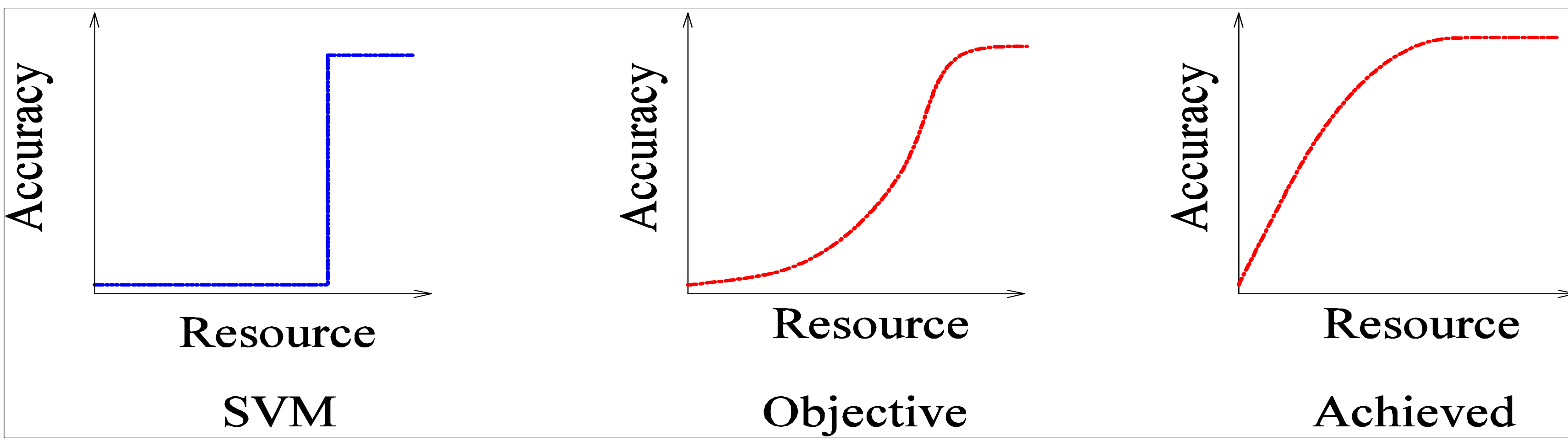
Semantic Concept Filters

E.g.:



Complexity Reduction Introduction

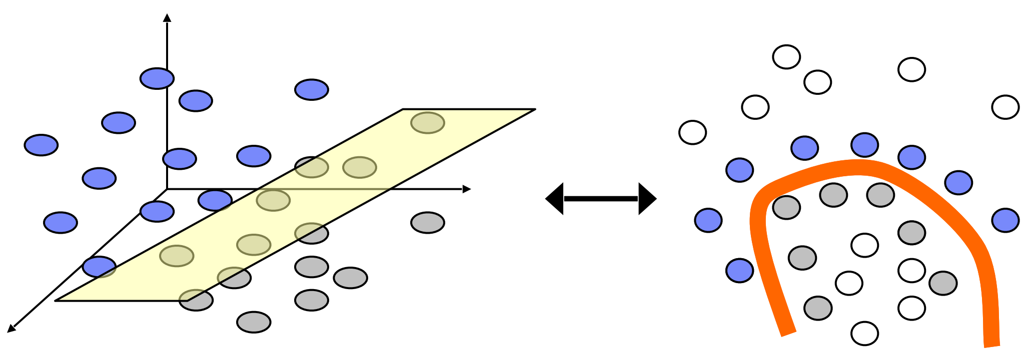
- Objective: Real-time classification of instances using Support Vector Machines (SVMs)
- Computationally efficient and reasonably accurate solutions
- Techniques capable of adjusting tradeoff between accuracy and speed based on available computational resources



SVM formulation

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

SVM



Decision

- **Score of unseen instance** 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

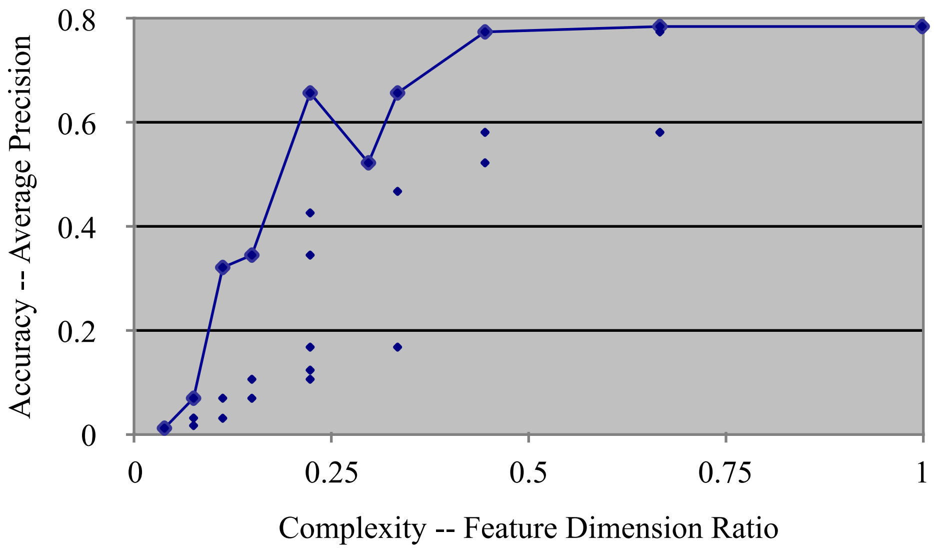
Problems

- **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**

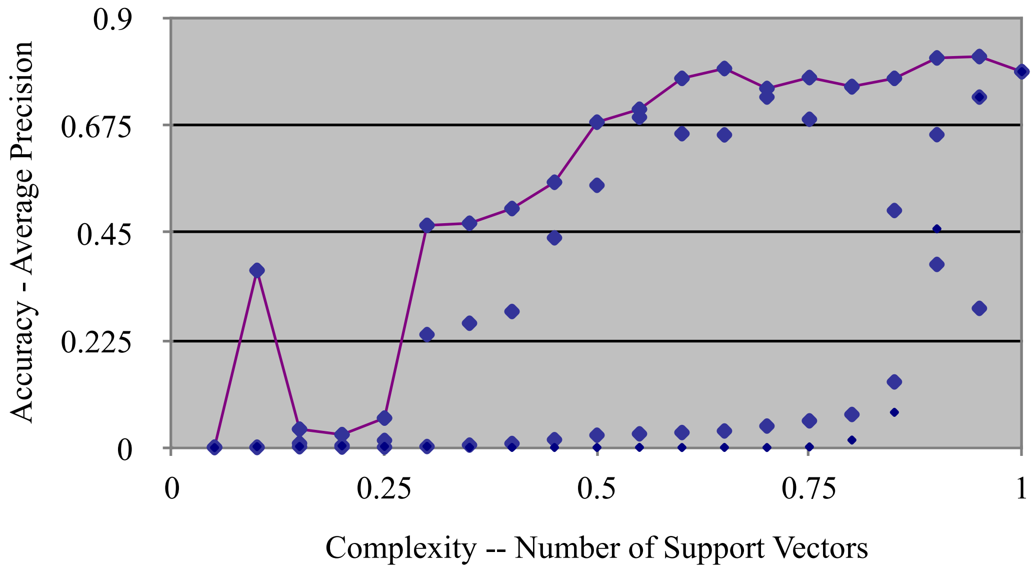
Naïve Approach I – Feature Dimension Reduction



- 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
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

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

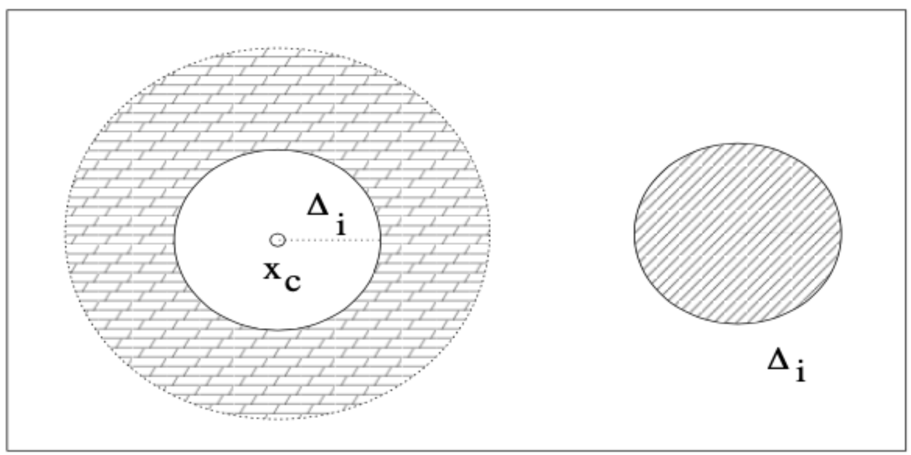
Weighted Clustering Approach

- **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 :

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

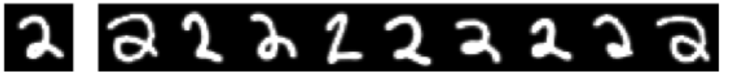


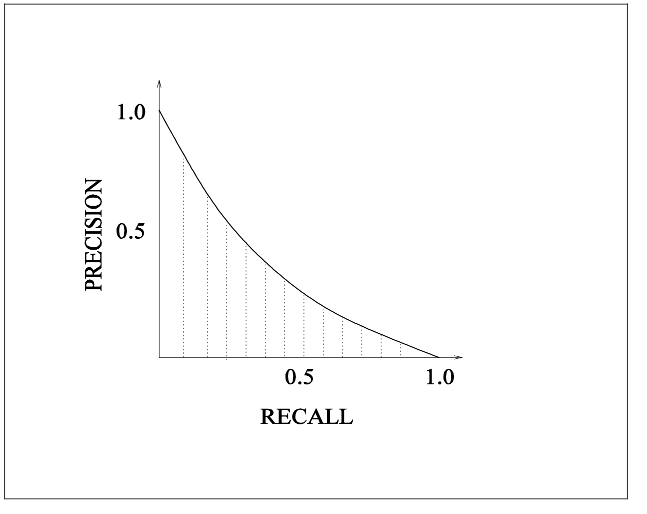
Using the weights

- **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**

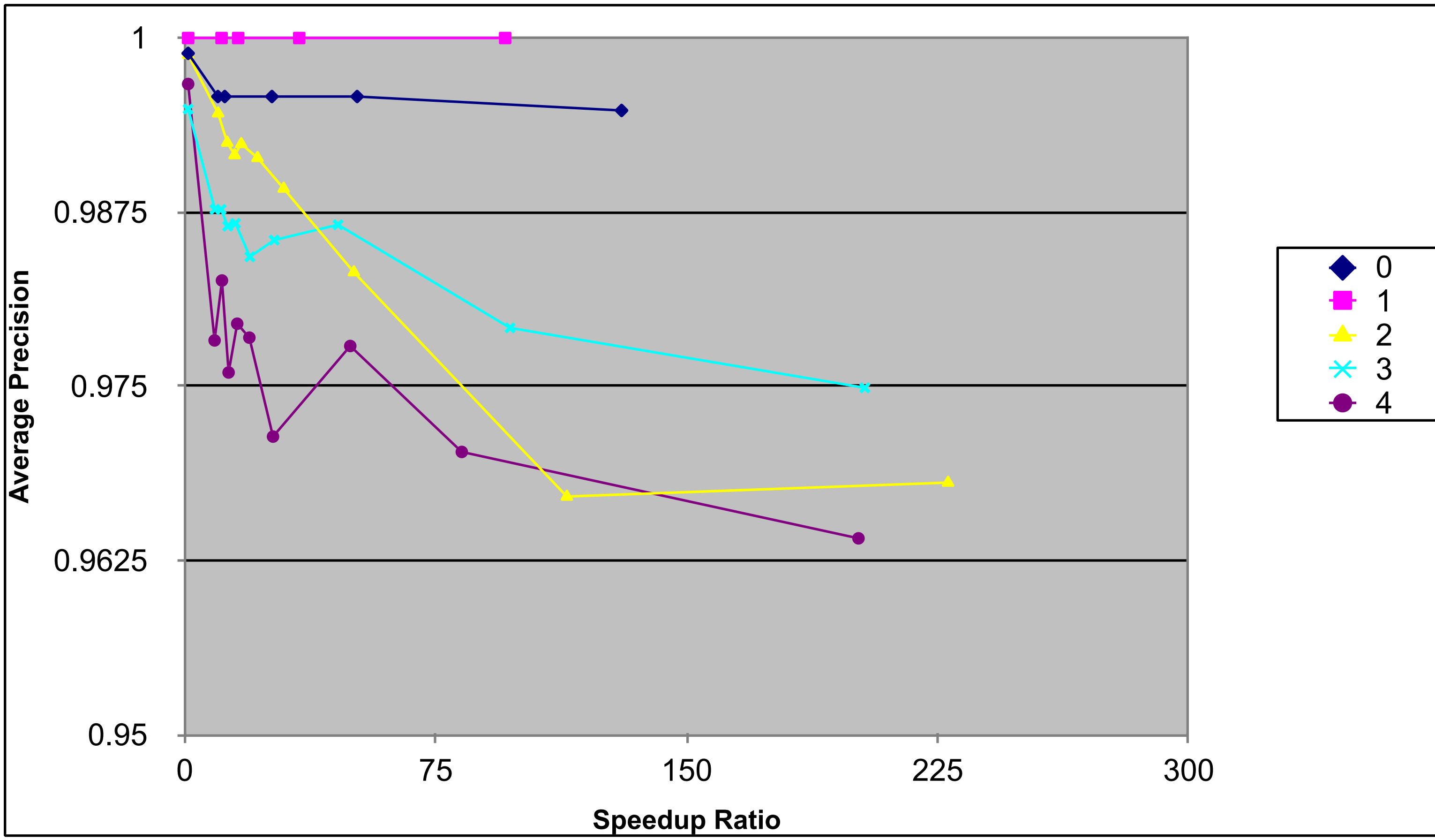
Experiments

- 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



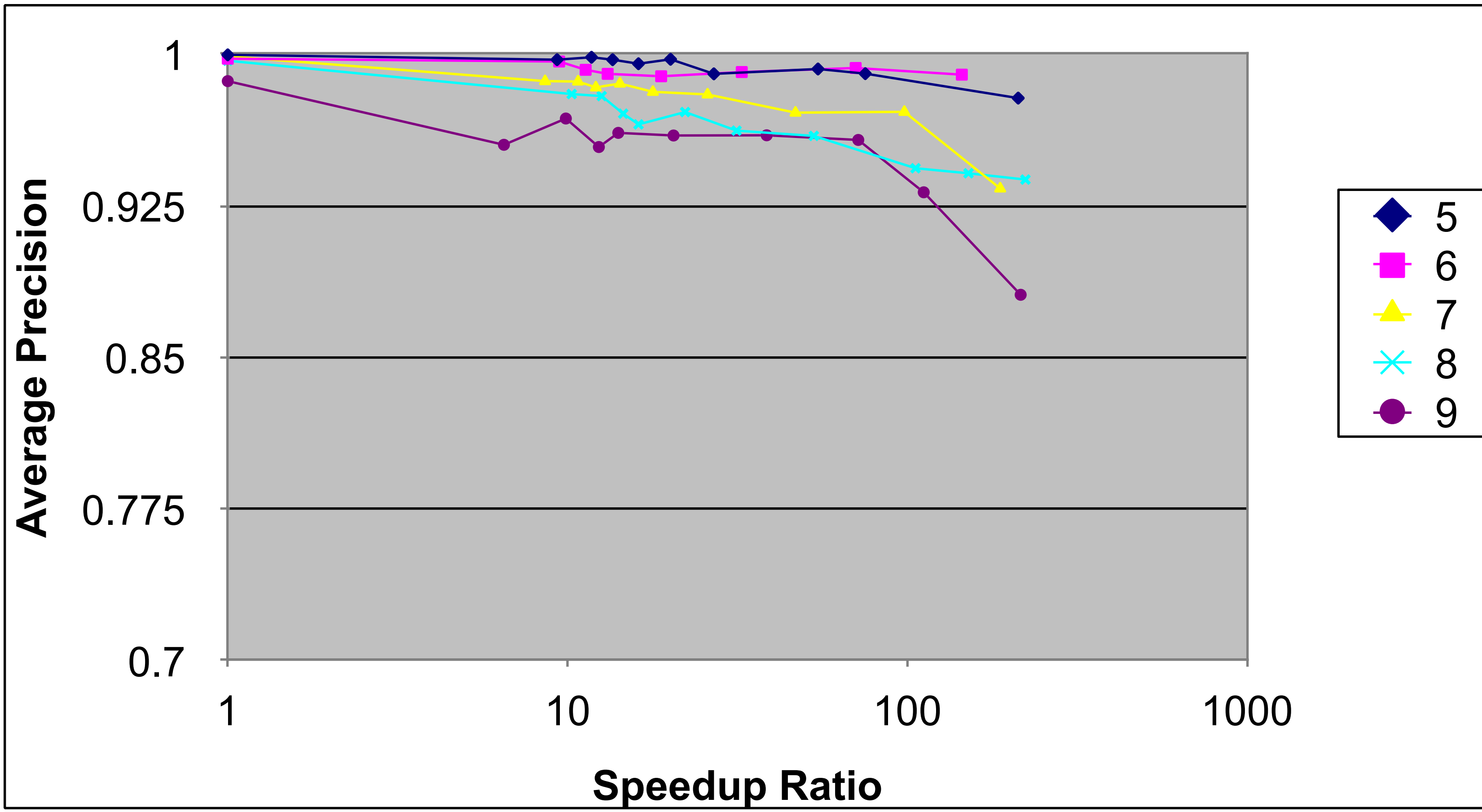


Results (Mnist 0-4)

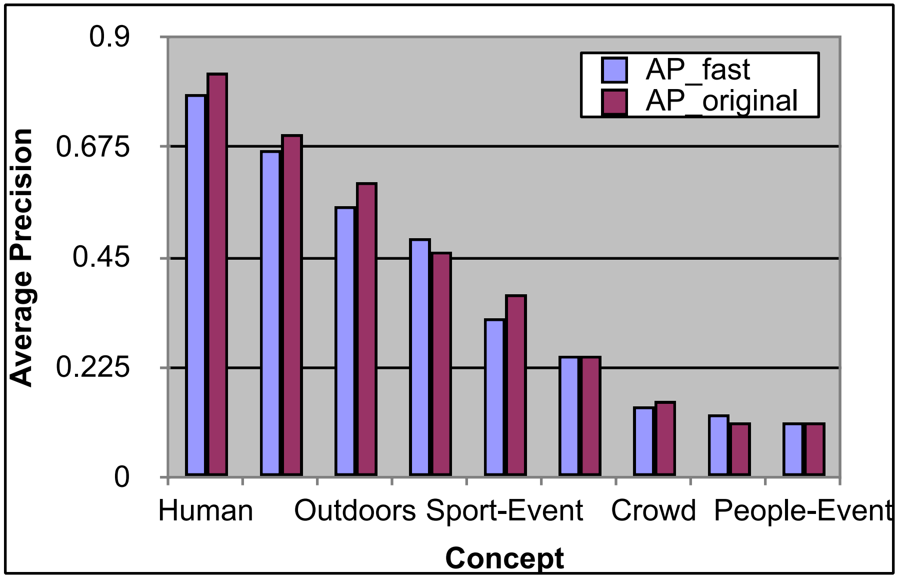


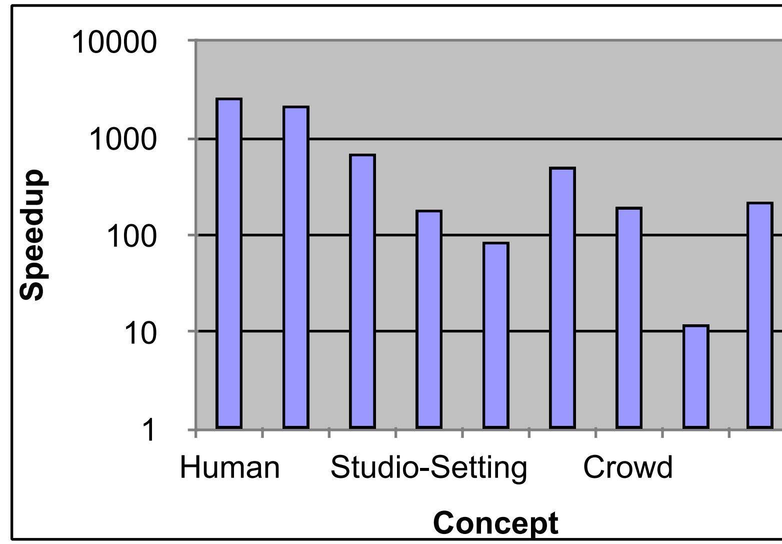
Columbia University

Results (Mnist 5-9)



Results (TREC 2003)





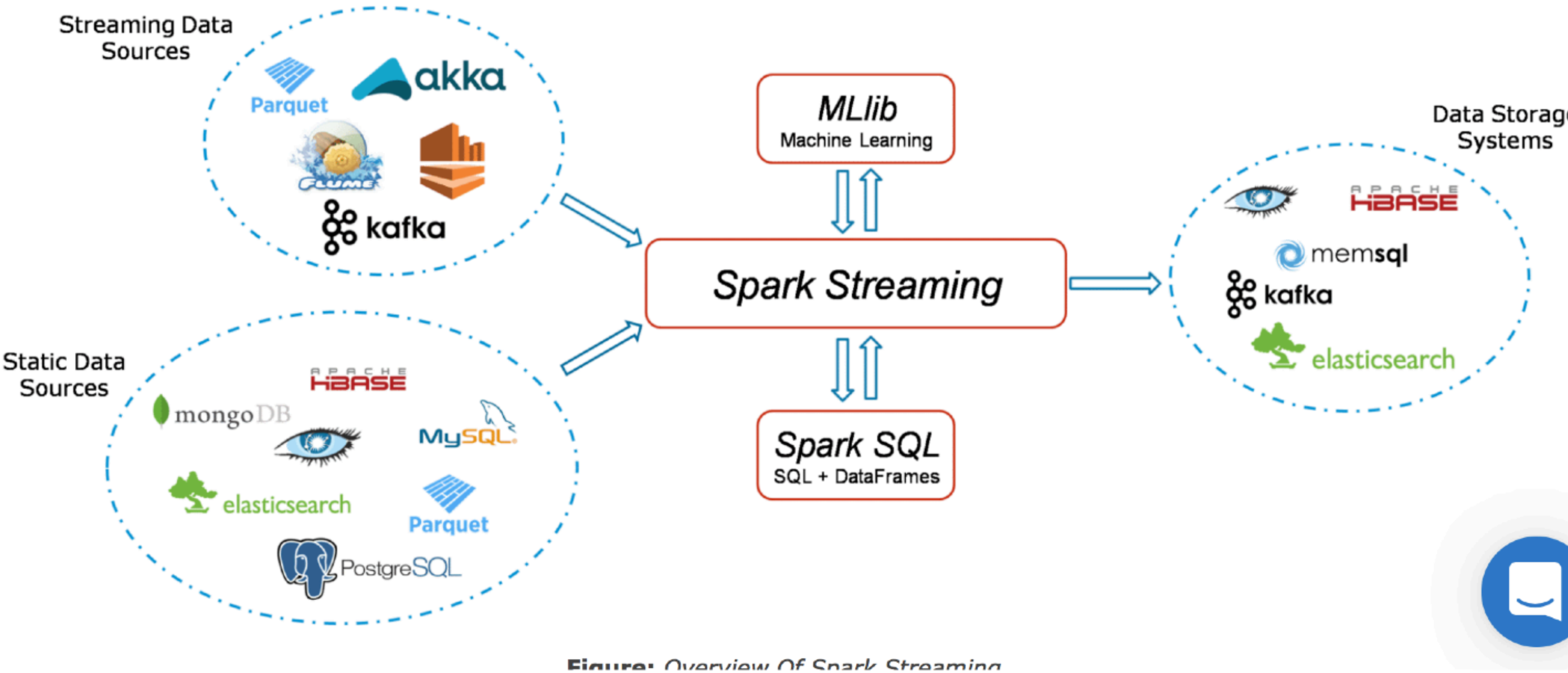
Summary of Complexity Reduction

- 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

Spark Streaming

Spark Streaming

Spark Streaming



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

Spark Streaming Example

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.

```python
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`).

```python
# 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.

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

```python
# 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
```

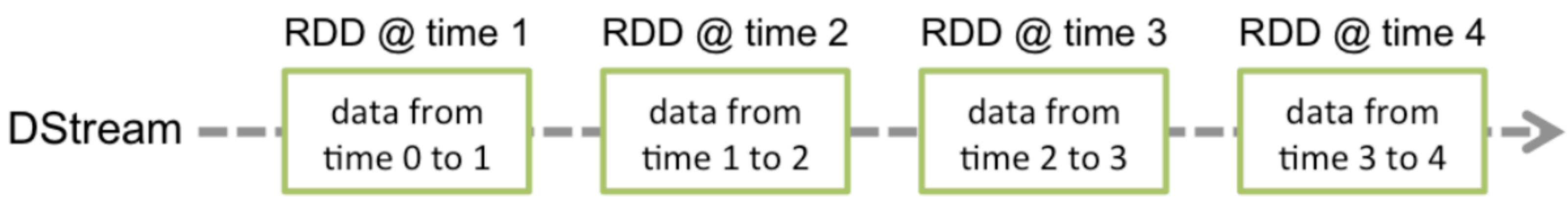
Then, any lines typed in the terminal running the netcat server will be counted and printed on screen every second. It will look something like the following.

Scala Java **Python**

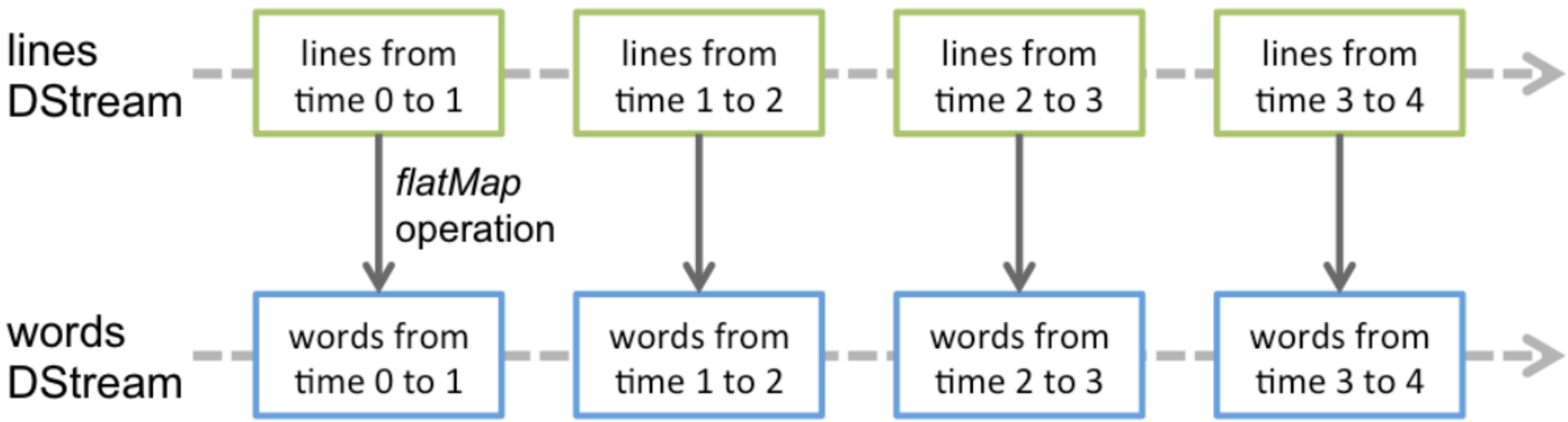
```
# TERMINAL 1:
# Running Net
cat

$ 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)
...
```

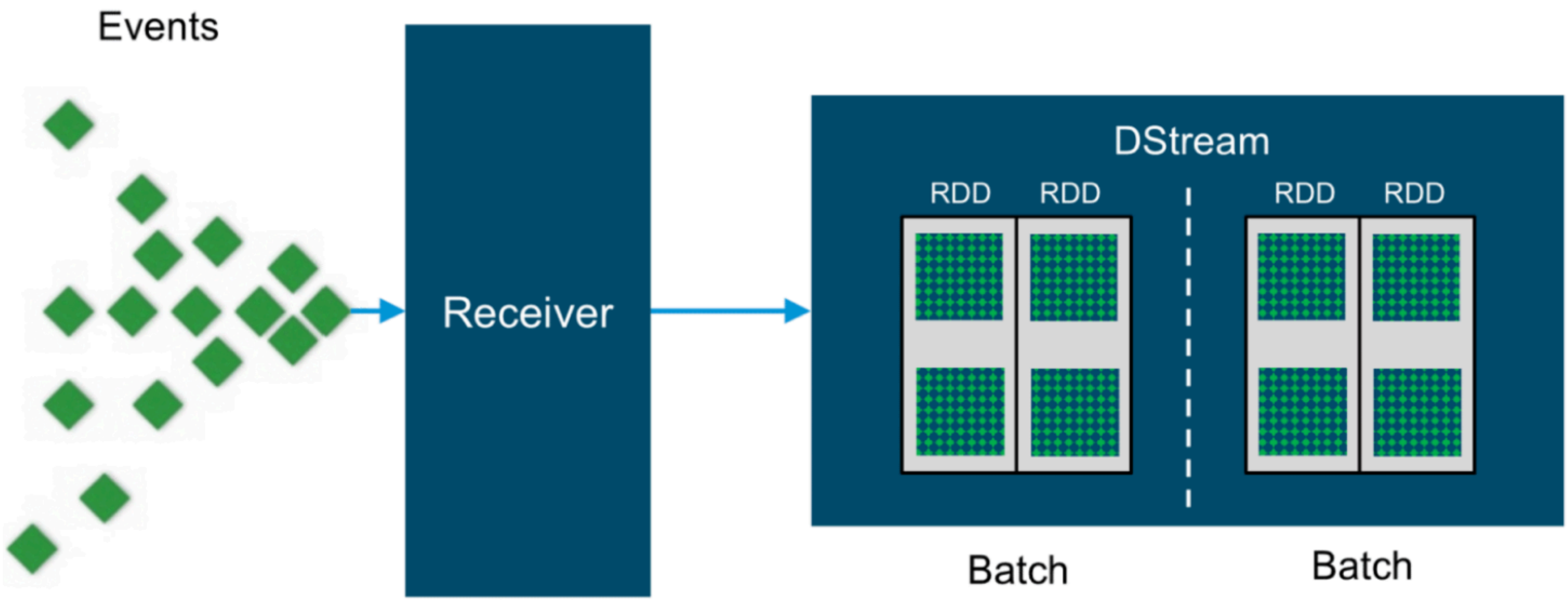

Discretized Streams



Discretized Streams

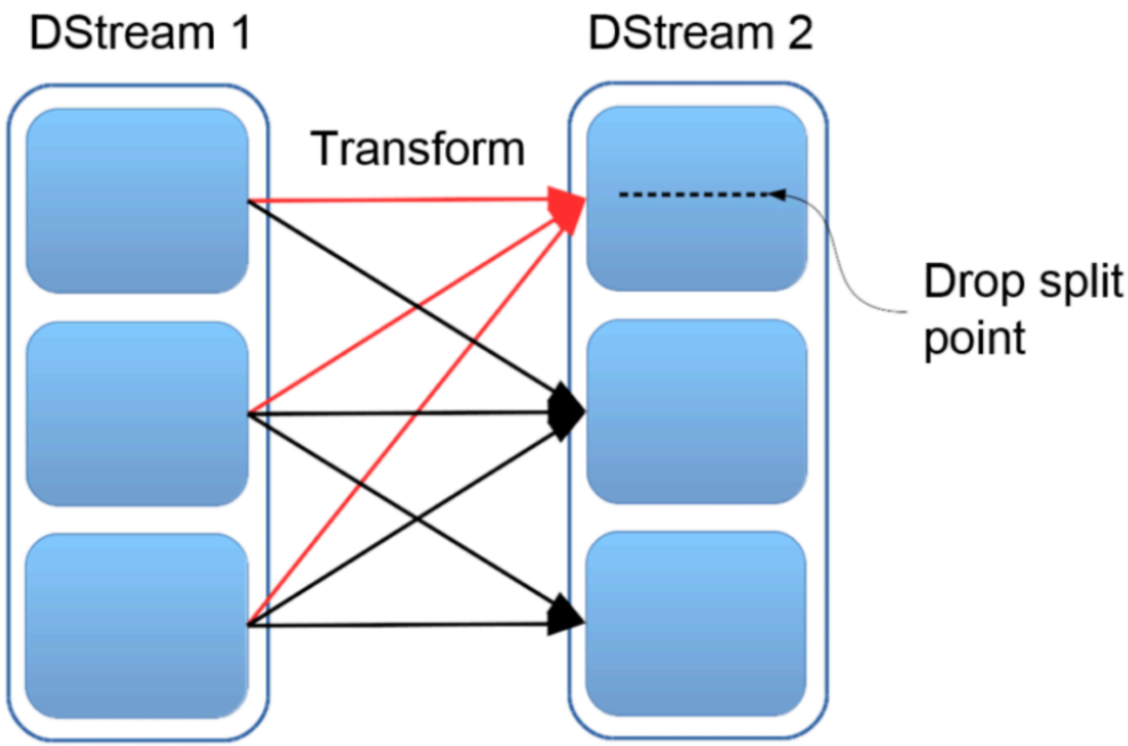


Discretized Streams



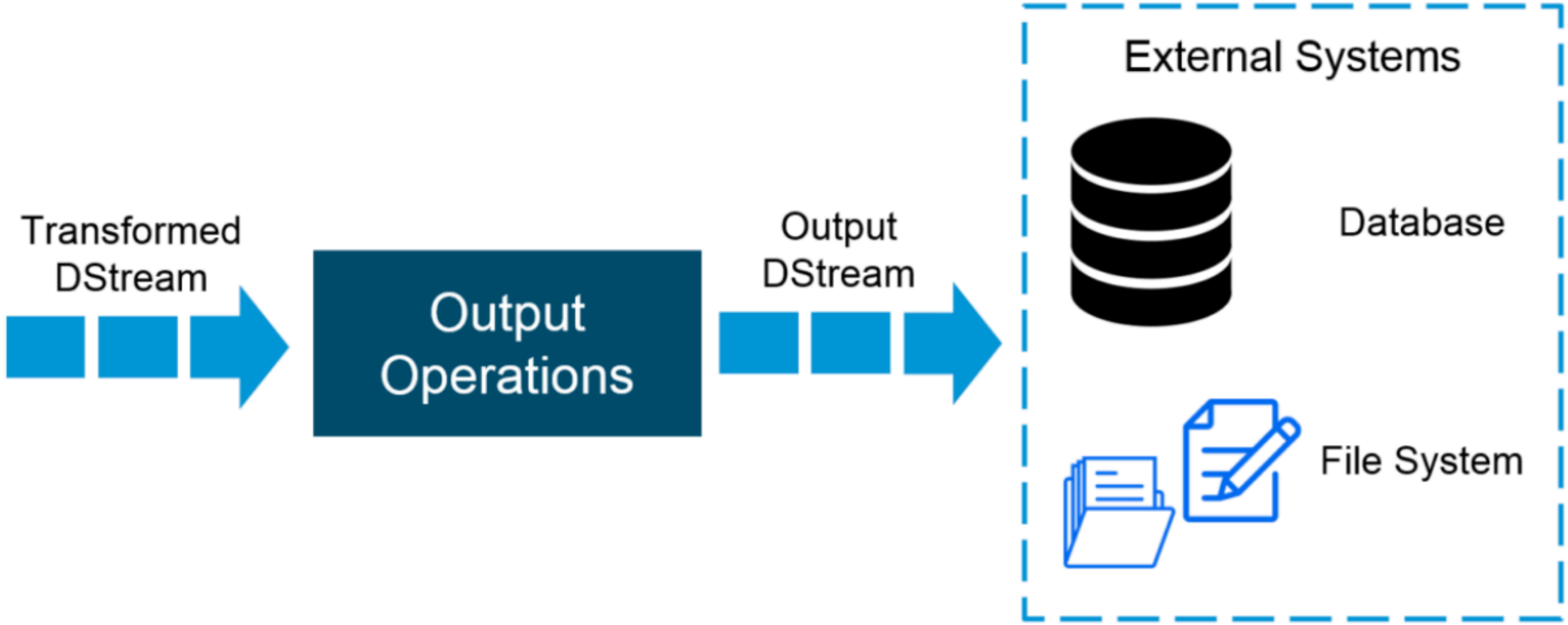
https://www.edureka.co/blog/spark-streaming/

DStream Transforms



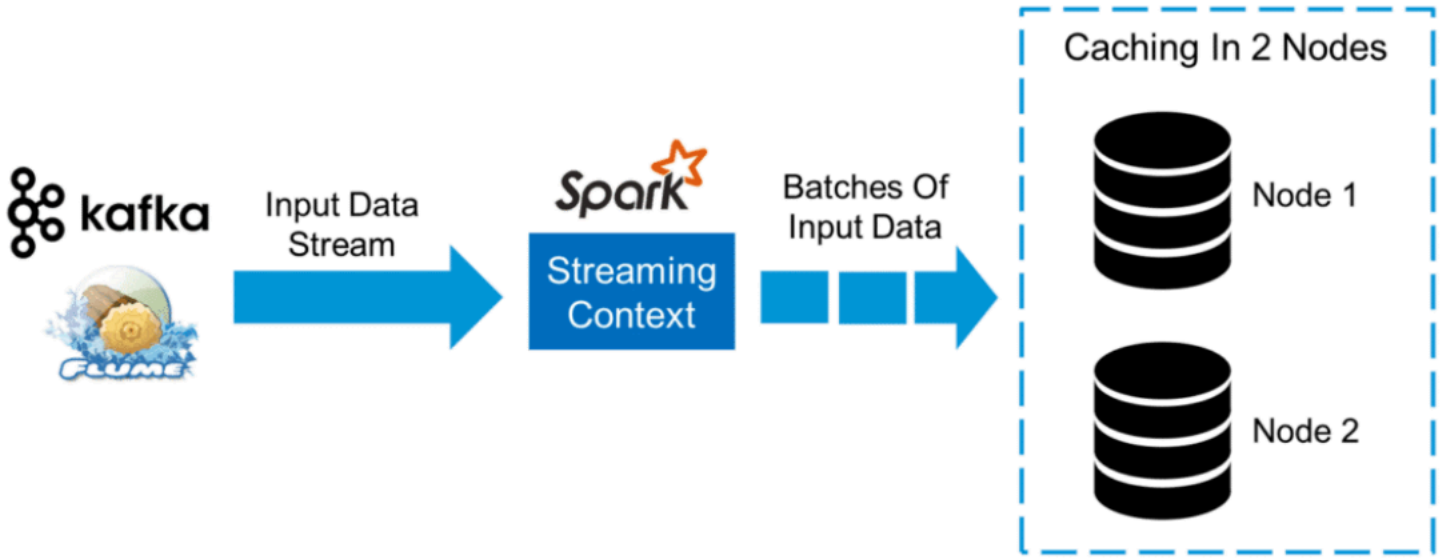
https://www.edureka.co/blog/spark-streaming/

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

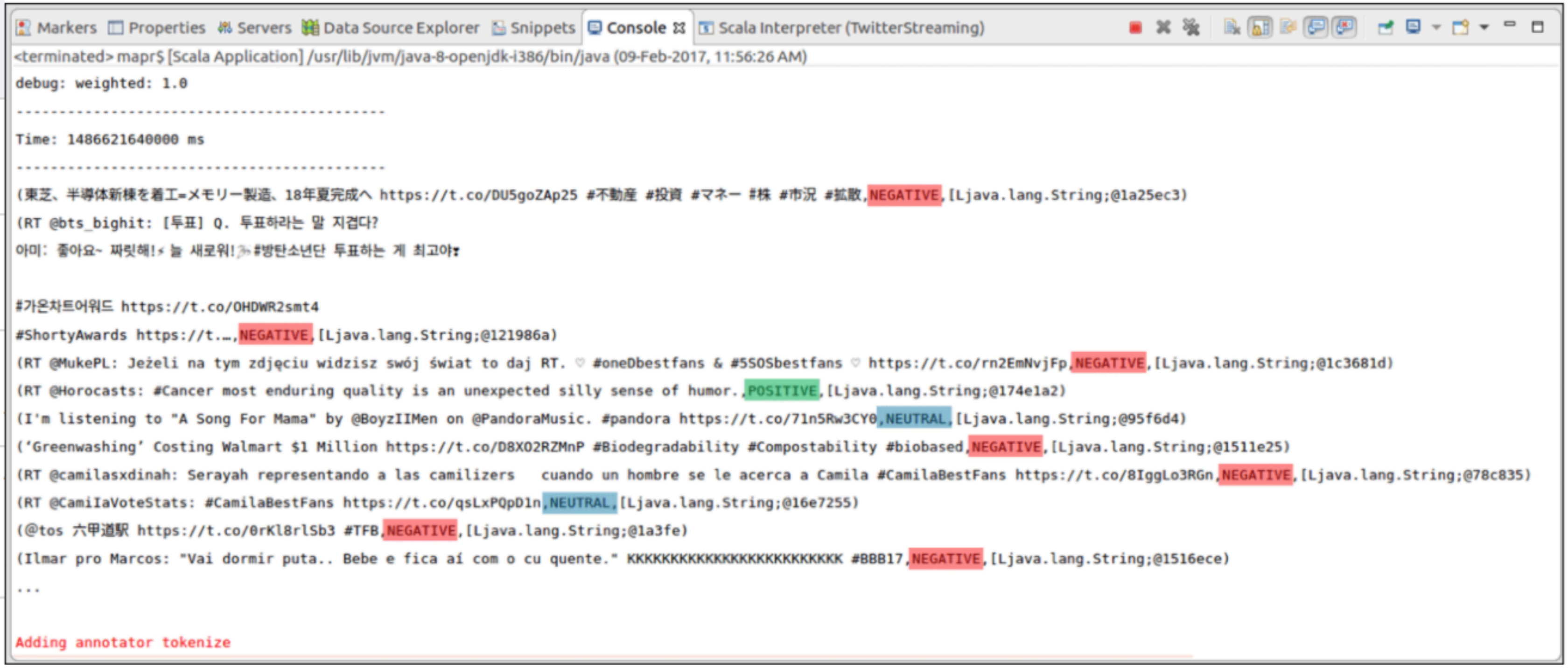
```
//Input DStream transformation using flatMap
val tags = stream.flatMap { status => Get Text From The Hashtags }

//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.



```
Markers  Properties  Servers  Data Source Explorer  Snippets  Console  Scala Interpreter (TwitterStreaming)
<terminated> mapr$ [Scala Application] /usr/lib/jvm/java-8-openjdk-i386/bin/java (09-Feb-2017, 11:56:26 AM)
debug: weighted: 1.0
-------------------------------------------
Time: 1486621640000 ms
-------------------------------------------
(東芝、半導体新棟を着工=メモリー製造、18年夏完成へ https://t.co/DU5goZAp25 #不動産 #投資 #マネー #株 #市況 #拡散,NEGATIVE,[Ljava.lang.String;@1a25ec3)
(RT @bts_bighit: [투표] Q. 투표하라는 말 지겹다?
아미: 좋아요~ 짜릿해! 늘 새로워! #방탄소년단 투표하는 게 최고야

#가온차트어워드 https://t.co/OHDWR2smt4
#ShortyAwards https://t.…,NEGATIVE,[Ljava.lang.String;@121986a)
(RT @MukePL: Jeżeli na tym zdjęciu widzisz swój świat to daj RT. ♡ #oneDbestfans & #5SOSbestfans ♡ https://t.co/rn2EmNvjFp,NEGATIVE,[Ljava.lang.String;@1c3681d)
(RT @Horocasts: #Cancer most enduring quality is an unexpected silly sense of humor.,POSITIVE,[Ljava.lang.String;@174e1a2)
(I'm listening to "A Song For Mama" by @BoyzIIMen on @PandoraMusic. #pandora https://t.co/71n5Rw3CY0,NEUTRAL,[Ljava.lang.String;@95f6d4)
('Greenwashing' Costing Walmart $1 Million https://t.co/D8XO2RZMnP #Biodegradability #Compostability #biobased,NEGATIVE,[Ljava.lang.String;@1511e25)
(RT @camilasxdinah: Serayah representando a las camilizers   cuando un hombre se le acerca a Camila #CamilaBestFans https://t.co/8IggLo3RGn,NEGATIVE,[Ljava.lang.String;@78c835)
(RT @CamiIaVoteStats: #CamilaBestFans https://t.co/qsLxPQpD1n,NEUTRAL,[Ljava.lang.String;@16e7255)
(@tos 六甲道駅 https://t.co/0rKl8rlSb3 #TFB,NEGATIVE,[Ljava.lang.String;@1a3fe)
(Ilmar pro Marcos: "Vai dormir puta.. Bebe e fica aí com o cu quente." KKKKKKKKKKKKKKKKKKKKKKKK #BBB17,NEGATIVE,[Ljava.lang.String;@1516ece)
...

Adding annotator tokenize
```

Positive

Neutral

Negative

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/

Homework #1 (Due 10/4/2024, 5pm)

See TA's instruction:

Task 1: Clustering (35%)

Task 2: Classification (35%)

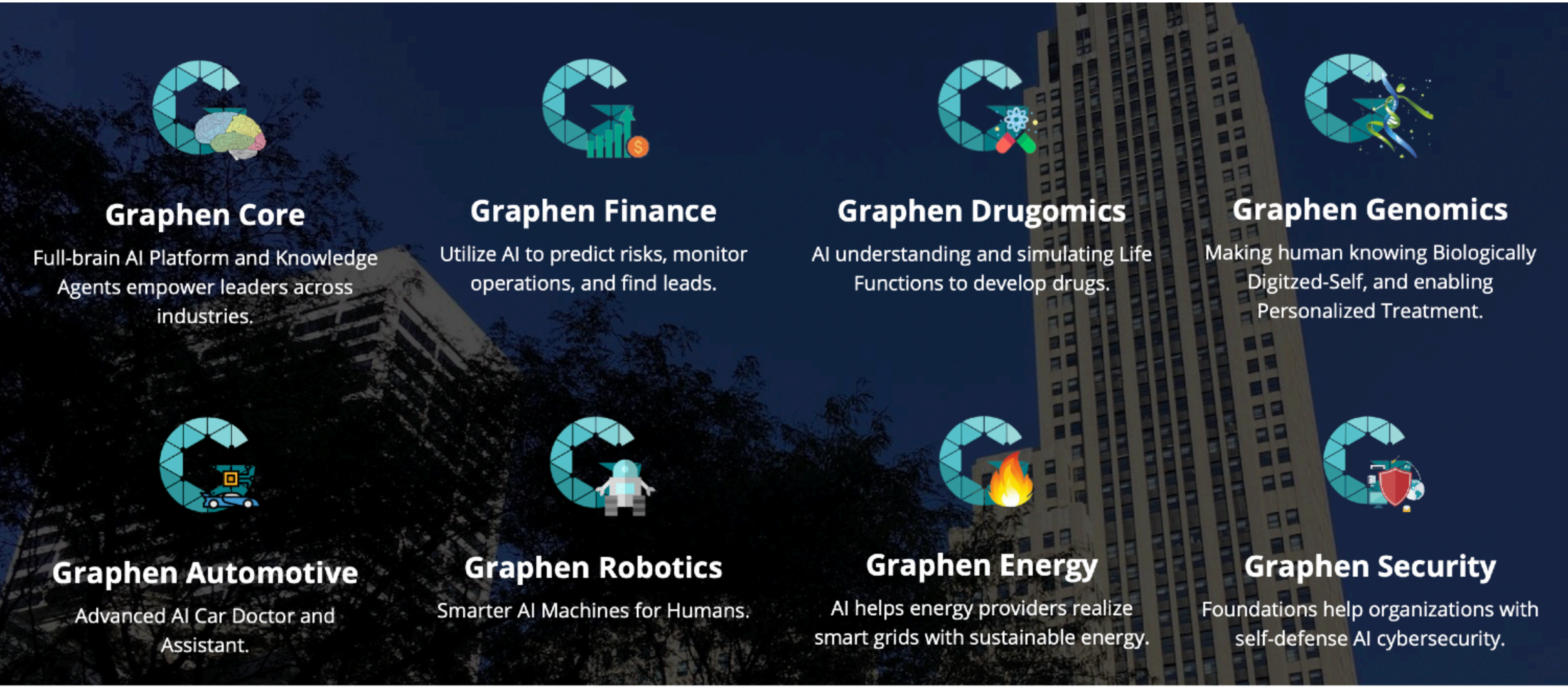
Task 3: Hadoop System Monitoring (30%)

Start brainstorming your final project

- Start finding your teammates.

- Proposal (11/8/24) — preparing about 5-7 pages of slides (each item 1/5 of the proposal score):
  - Goal — novel? challenging?
  - Data — 3Vs? New dataset? Existing dataset?
  - Methods — planning of methodologies and algorithms? Feasible?
  - System — an overview of system. What will be implemented?
  - Schedule — what to achieve by what time, and by whom?

Example: Big Data Analytics & AI Application Areas



Area 1 'Cognitive Machine' Tasks List:

A1: Deep Video Understanding (Visual + Knowledge) — Face Recognition, Feeling Recognition, and Interaction

A2: Deep Video Understanding (Language + Knowledge) — Speech Recognition, Gesture Recognition, and Feeling Recognition

A3: Deep Video Understanding — Event and Story Understanding

A4: Humanized Conversation — Personality-Based Conversations

A5: Autonomous Robot Learning of Physical Environment

A6: Autonomous Task Learning via Mimicking

A7: Digital Human - Creation and Facial Expression

A8: Digital Human - Action

A9: Digital Human - Text-to-Audio, Lip Sync, and Audio-to-Text

A10: Human and Digital Human Interactions

A11: Feeling and Art Recognition

A12: Creative Writing & Story Telling

A13: Knowledge Learning & Construction

A14: Dreams — Simulating Brain functions while sleeping

A15: Self-Consciousness, Ethics, and Morality

Digital Human Examples

Graphen Ava

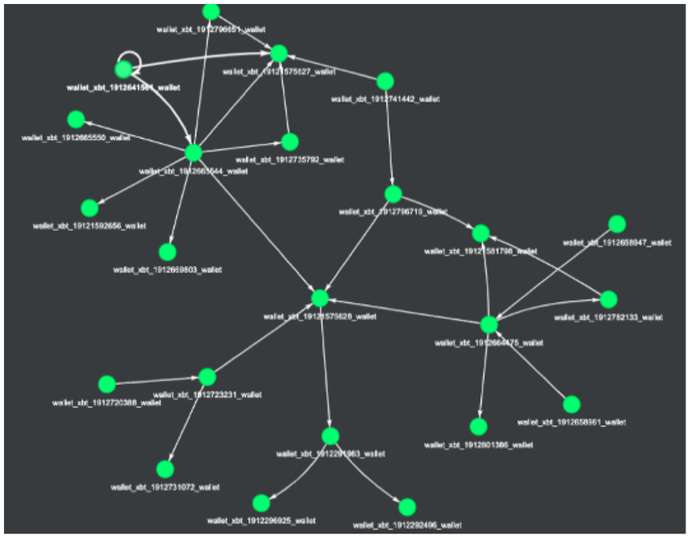
Meta

https://www.graphen.ai/products/Ava.html

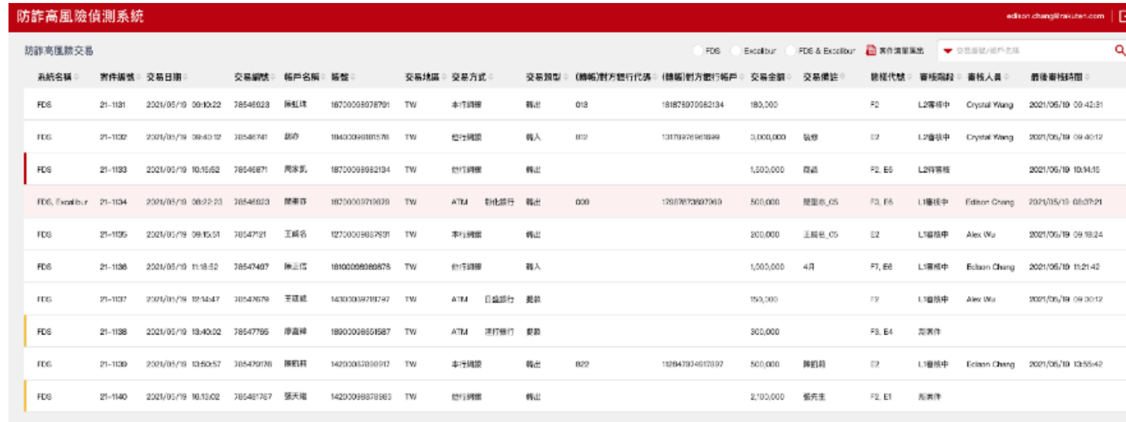
Area 2 'Finance Advisor' Tasks List:

B1: Market Intelligence — Constructing Financial Knowledge Graphs
B2: Market Intelligence — Company Environmental, Societal, and Governance Performance
B3: Market Intelligence — Event Linkage and Impact Prediction
B4: Market Intelligence — Alpha Generation from Alternative Sources
B5: Advance KYC — Customer Profiling based on Personality, Needs, and Value
B6: Advanced KYC — Customer Behavior Prediction
B7: Investment Strategy — AI Trader (Foreign Exchange)
B8: Investment Strategy — AI Trader (Stock Markets)
B9: Investment Strategy — Automatic Dynamic Asset Allocation
B10: Customer Interaction — Customer Communication Strategies
B11: Customer Interaction — Insurance Product Sales & Marketing Strategy
B12: Automatic Story Telling for Marketing
B13: Automatic Market Competition Analysis
B14: Automatic Consumer Sales Leads Finding
B15: Human Capital Growth Recommendations

Real-Time Fraud Analysis Examples



Crypto Currencies



Online Banks

Credit Cards

Area 3 'Healthy Life' Tasks List:

C1: Precision Health — Gene and Protein Analysis of Network, Pathway, and Biomarkers

C2: Large-Scale System for Human Genome Analysis

C3: Secure Patient Data

C4: Medical Image Analysis

C5: Drugable Targets for Precision Medicine

C6: Virus Mutations and Function Prediction

C7: Microbe and Disease Knowledge Graph

C8: Disease Symptoms Knowledge Graphs

C9: Virtual Doctor

C10: Knowledge Graphs for Gene Interaction and Disease Similarity

C11: Biomedical Knowledge Construction and Extraction

C12: Generating Gene Therapy

C13: Molecular Drug Synthesis

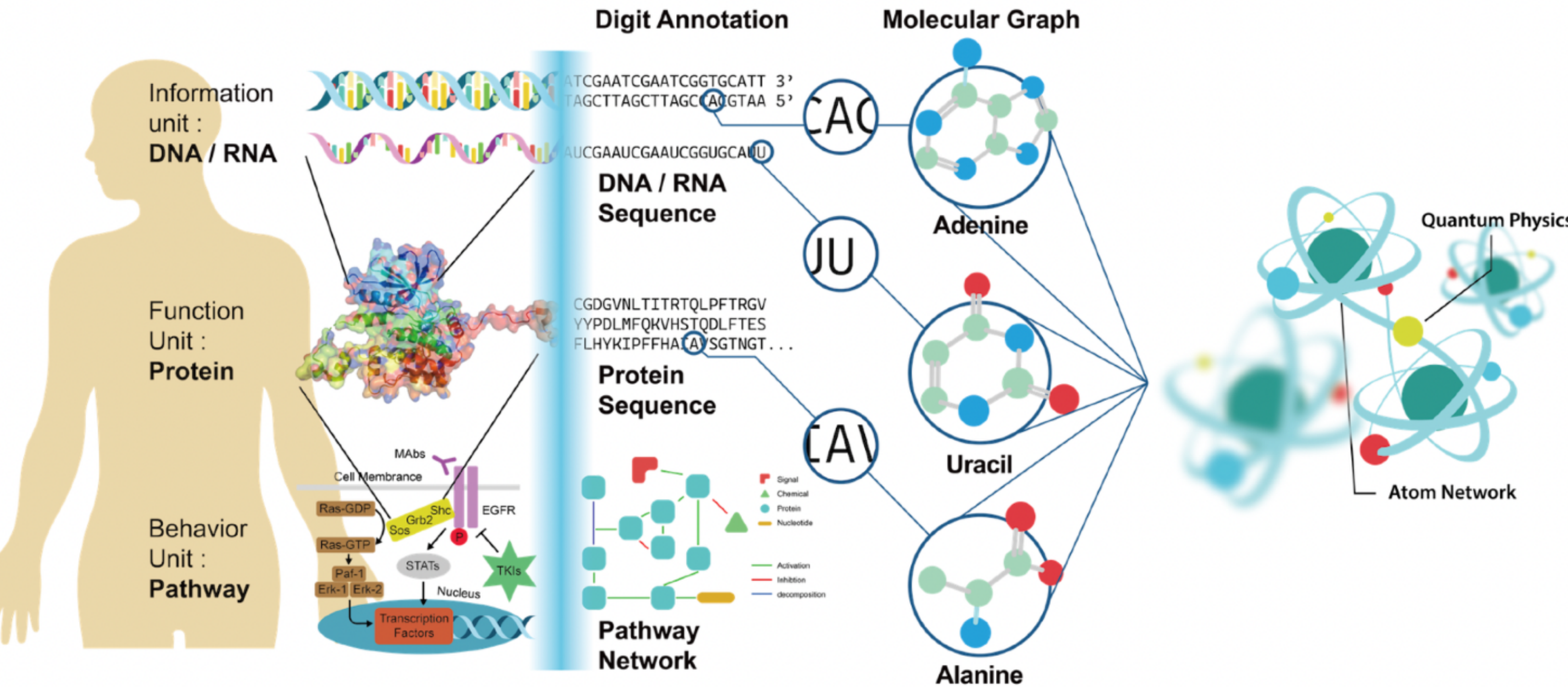
C14: Protein Interaction Predictor

C15: Aging Impacts

Digital Biology Examples

https://www.graphen.ai/products/atom.html

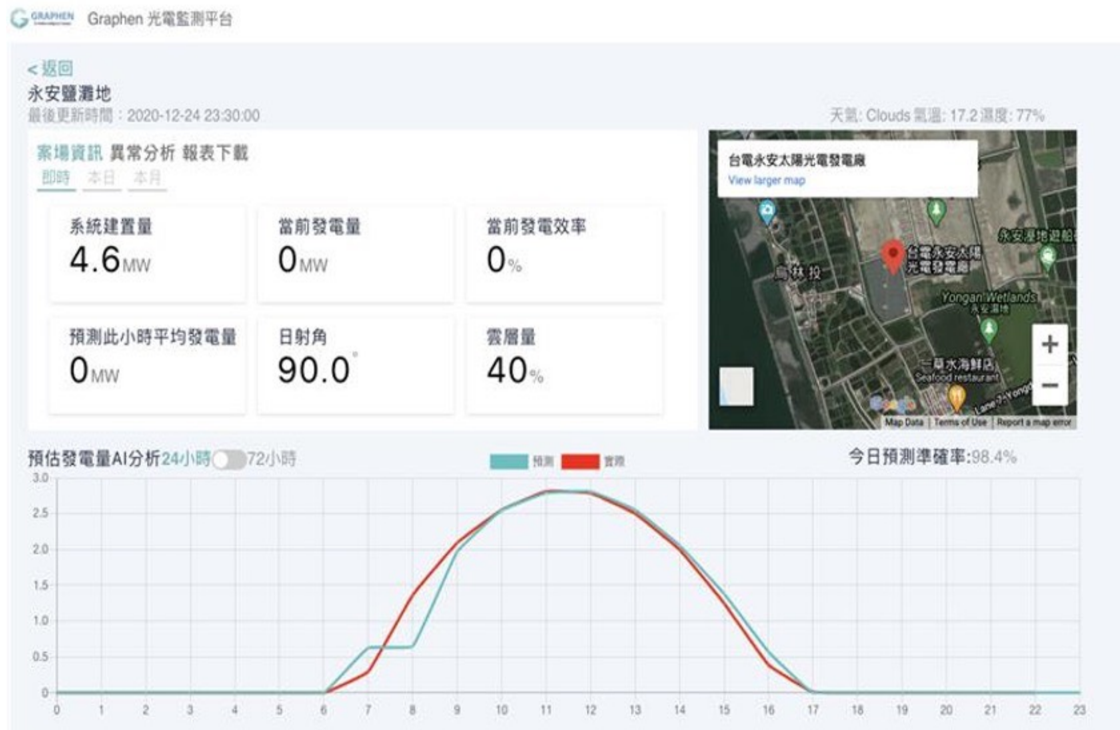
Biological Materials **Digitalize Bioinformation** **Atomotive Forces**



Area 4 'Green Earth' Tasks List:

D1: Distributed Solar Power Load Forecasting and Predictive Maintenance
D2: Distributed Wind Power Load Forecasting and Predictive Maintenance
D3: Power Flow Optimization
D4: Smart Grid Pricing Strategy
D5: Cybersecurity of Smart Grid
D6: Stimulating Crop Growth
D7: Electronic Car Sensing and Predictive Maintenance
D8: Autonomous Driving
D9: Smart City of Connected Cars
D10: Social Policy Monitoring
D11: International Relationships and Policy Monitoring
D12: Mobile Cognition
D13: AI Chip Design
D14: Visual Exploration in Immersive Environment
D15: Computer Vision Enhanced Immersive Environment

Green Earth Examples



- **Renewable Energy Prediction**
- **Power System Anomaly Detection**
- **Predictive Maintenance**
- **Dispatching System**

https://www.youtube.com/watch?v=9PTlqCCMX-0

9/20/2022

Questions?