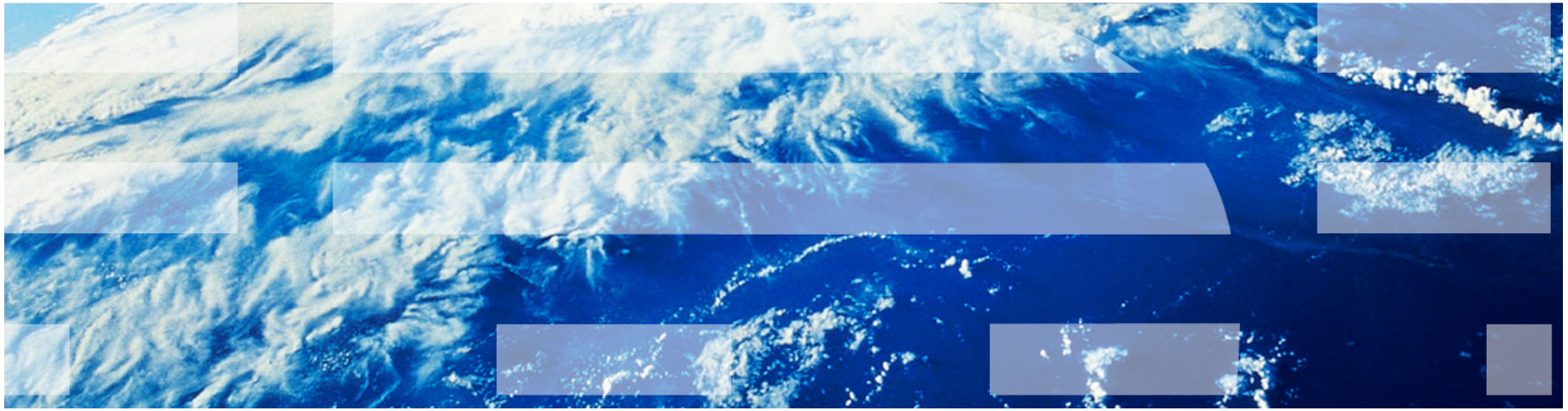
E6893 Big Data Analytics Lecture 4:

Big Data Analytics Algorithms — II

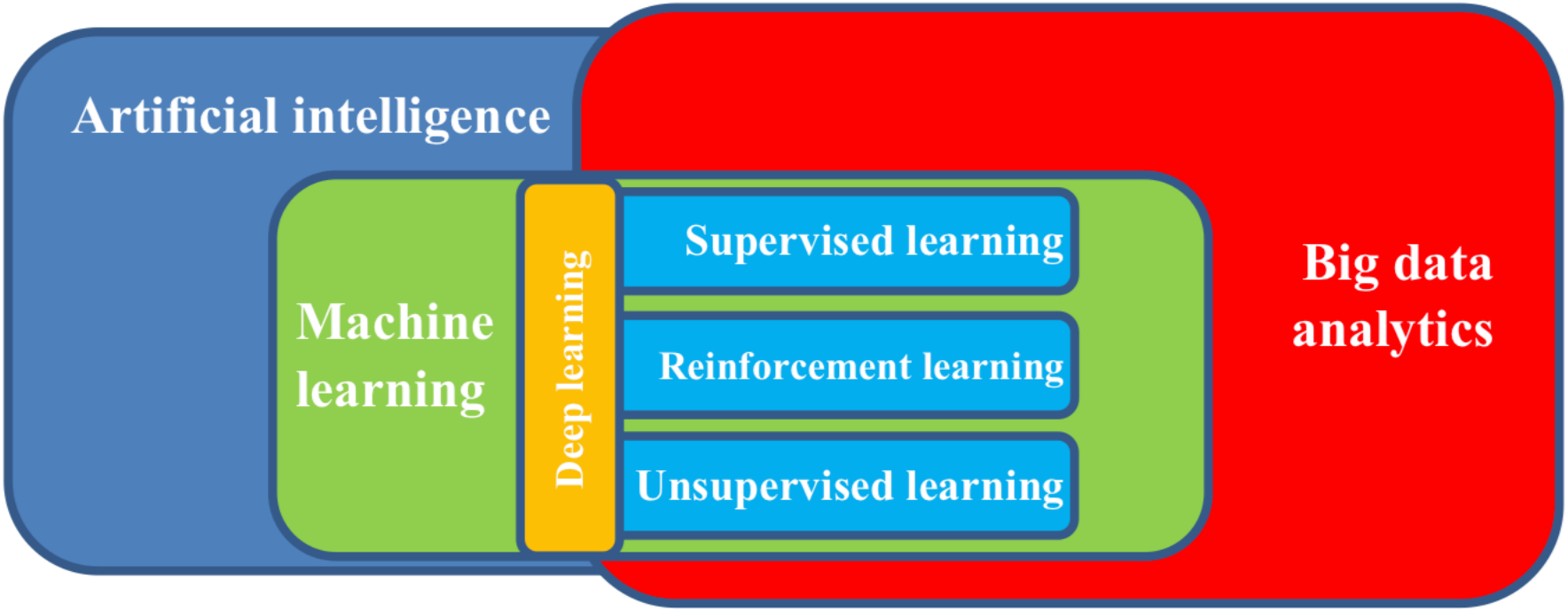
Ching-Yung Lin, Ph.D.

Adjunct Professor, Dept. of Electrical Engineering and Computer Science



September 27th, 2019

A schematic view of AI, ML, and Big Data Analytics



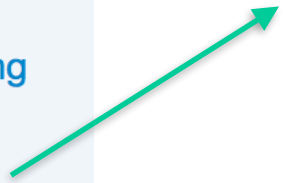
citation: http://www.fsb.org/wp-content/uploads/P011117.pdf

Spark ML Classification and Regression

Columbia University

MLlib: Main Guide

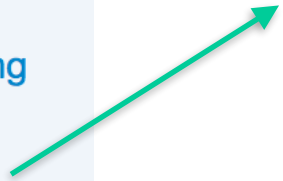
- Basic statistics
- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Frequent Pattern Mining
- Model selection and tuning
- Advanced topics



- Classification
  - Logistic regression
    - Binomial logistic regression
    - Multinomial logistic regression
  - Decision tree classifier
  - Random forest classifier
  - Gradient-boosted tree classifier
  - Multilayer perceptron classifier
  - Linear Support Vector Machine
  - One-vs-Rest classifier (a.k.a. One-vs-All)
  - Naive Bayes
- Regression
  - Linear regression
  - Generalized linear regression
    - Available families
  - Decision tree regression
  - Random forest regression
  - Gradient-boosted tree regression
  - Survival regression
  - Isotonic regression

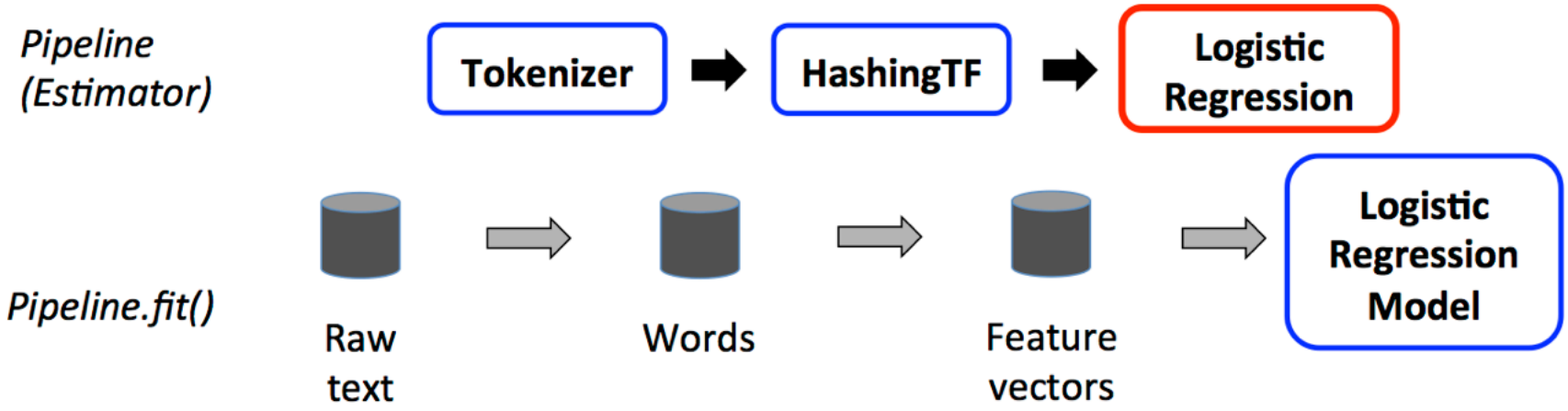
MLlib: Main Guide

- 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)

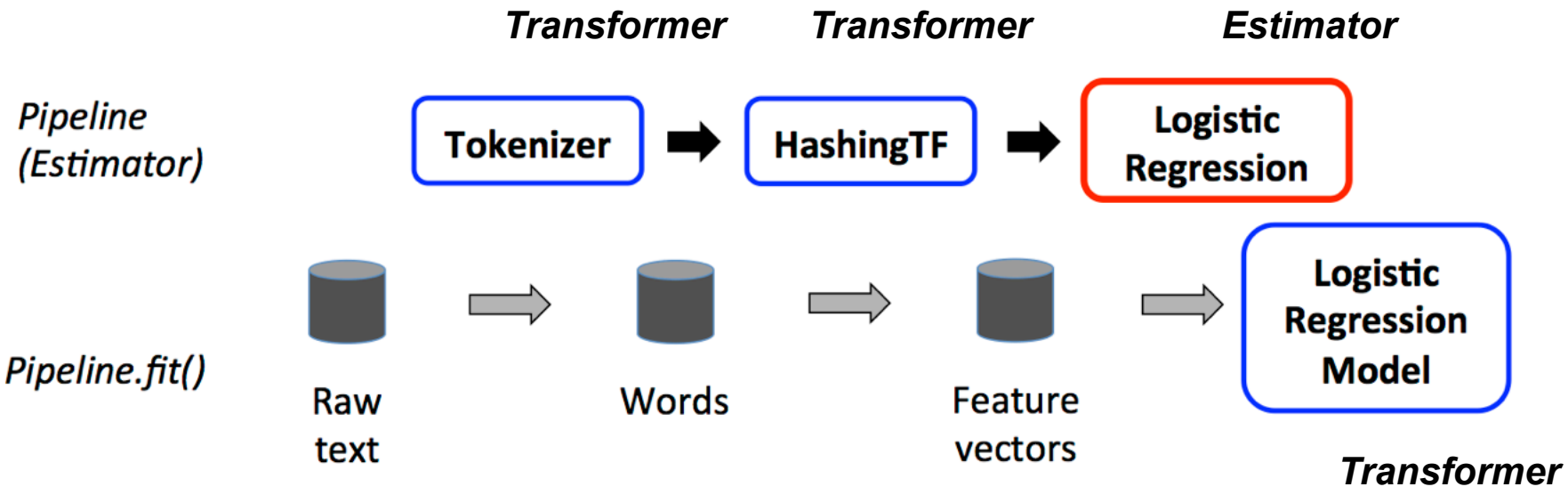
Spark ML Pipeline Example



Spark ML Pipeline terms

- **`DataFrame`**: This ML API uses `DataFrame` from Spark SQL as an ML dataset, which can hold a variety of data types. E.g., a `DataFrame` could have different columns storing text, feature vectors, true labels, and predictions.
- **`Transformer`**: A `Transformer` is an algorithm which can transform one `DataFrame` into another `DataFrame`. E.g., an ML model is a `Transformer` which transforms a `DataFrame` with features into a `DataFrame` with predictions.
- **`Estimator`**: An `Estimator` is an algorithm which can be fit on a `DataFrame` to produce a `Transformer`. E.g., a learning algorithm is an `Estimator` which trains on a `DataFrame` and produces a model.
- **`Pipeline`**: A `Pipeline` chains multiple `Transformer`s and `Estimator`s together to specify an ML workflow.
- **`Parameter`**: All `Transformer`s and `Estimator`s now share a common API for specifying parameters.

Spark ML Pipeline Example



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

```
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

Columbia University

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

Columbia University

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` t
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

Columbia University

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.

Term Frequency — Inverse Document Frequency (TF-IDF)

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

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

$$W_i = TF_i \cdot IDF_i = TF_i \cdot \frac{N}{DF_i} \quad \text{or} \quad W_i = TF_i \cdot \log\frac{N}{DF_i}$$

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

```
##      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()
```

Examples — using a news corpus

Reuters-21578 dataset: 22 files, each one has 1000 documents except the last one.

http://www.daviddlewis.com/resources/testcollections/ reuters21578/

Extraction code:

```
mvn -e -q exec:java
-Dexec.mainClass="org.apache.lucene.benchmark.utils.ExtractReuters"
-Dexec.args="reuters/ reuters-extracted/"
```

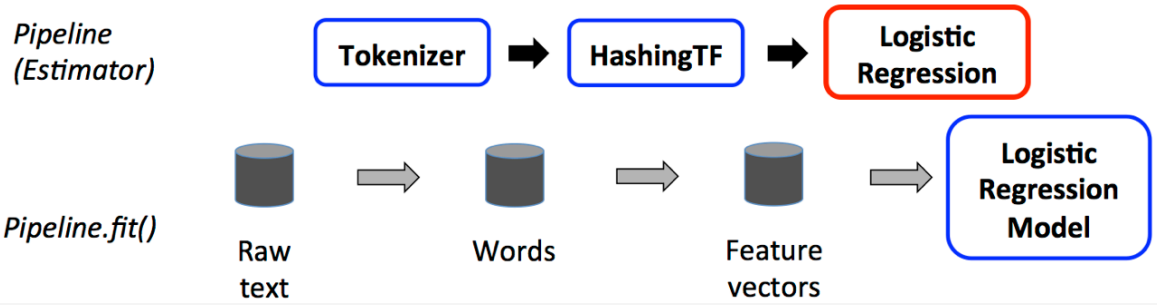
Using the extracted folder, run the `SequenceFileFromDirectory` class. You can use the launcher script from the Mahout root directory to do the same:

```
bin/mahout seqdirectory -c UTF-8
-i examples/reuters-extracted/ -o reuters-seqfiles
```

This will write the Reuters articles in the `SequenceFile` format. Now the only step left is to convert this data to vectors. To do that, run the `SparseVectorsFromSequence-Files` class using the Mahout launcher script:

```
bin/mahout seq2sparse -i reuters-seqfiles/ -o reuters-vectors -ow
```

Spark ML Pipeline Example Code I

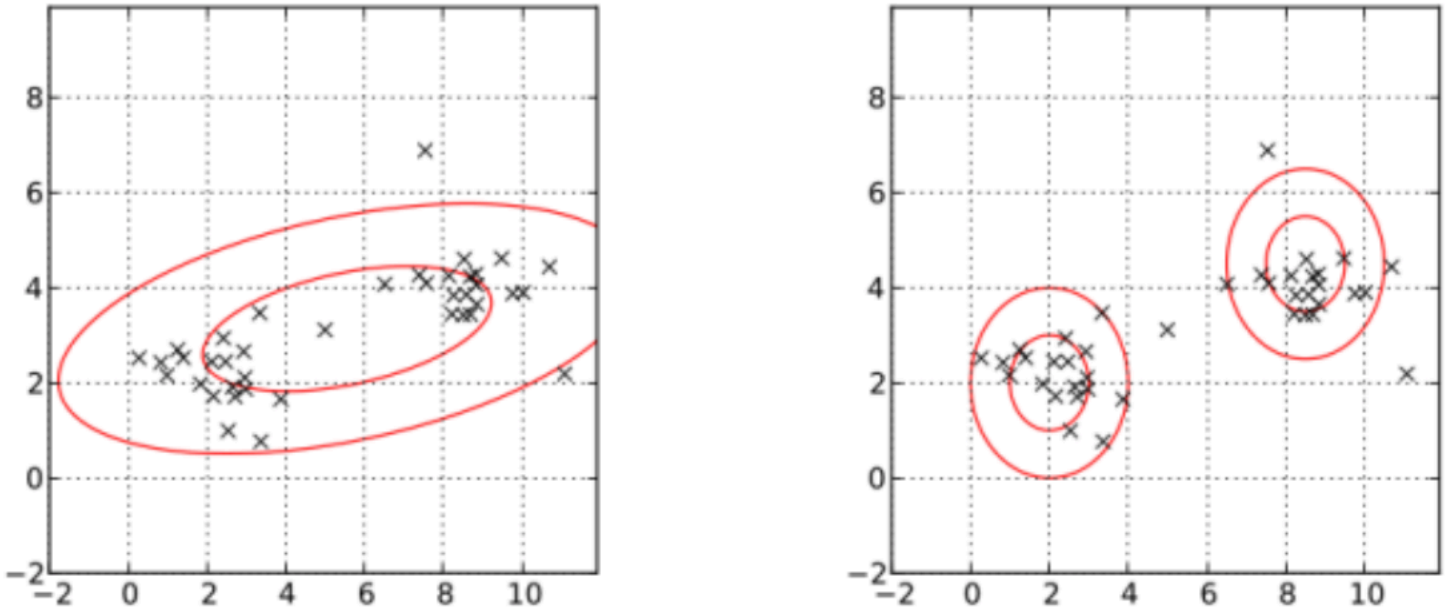



```
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer

# 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 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

Multi-dimensional Model

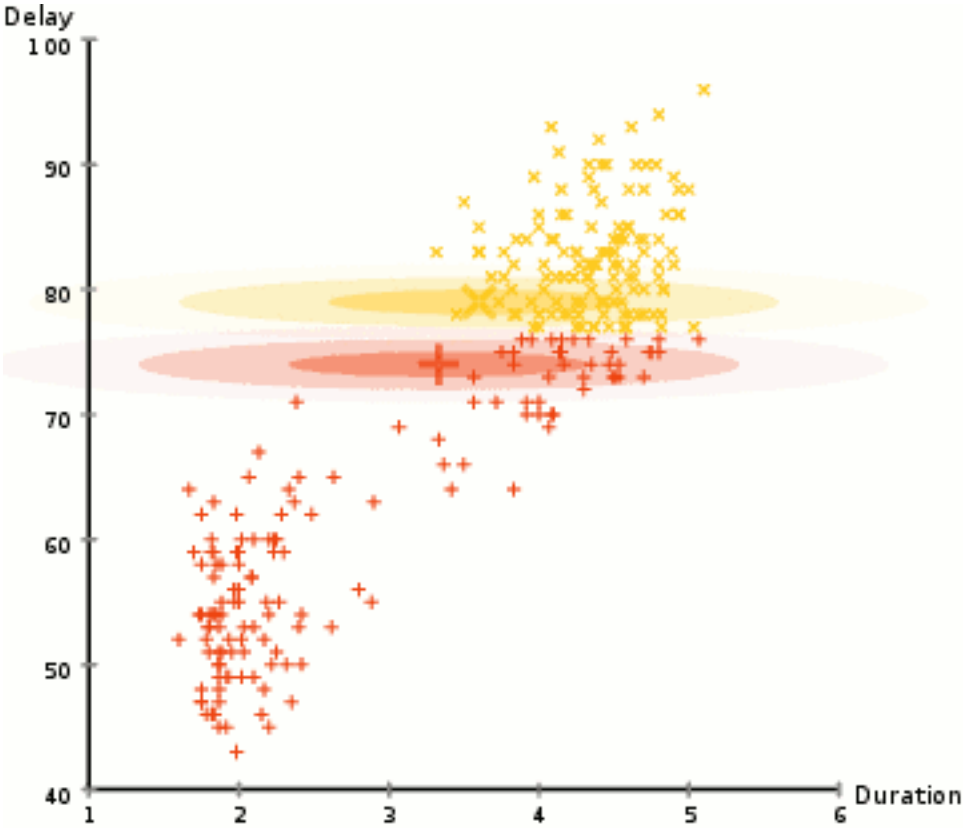
$$p(\vec{x}) = \sum_{i=1}^{K} \phi_i \mathcal{N}(\vec{x} \mid \vec{\mu}_i, \Sigma_i)$$

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

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

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

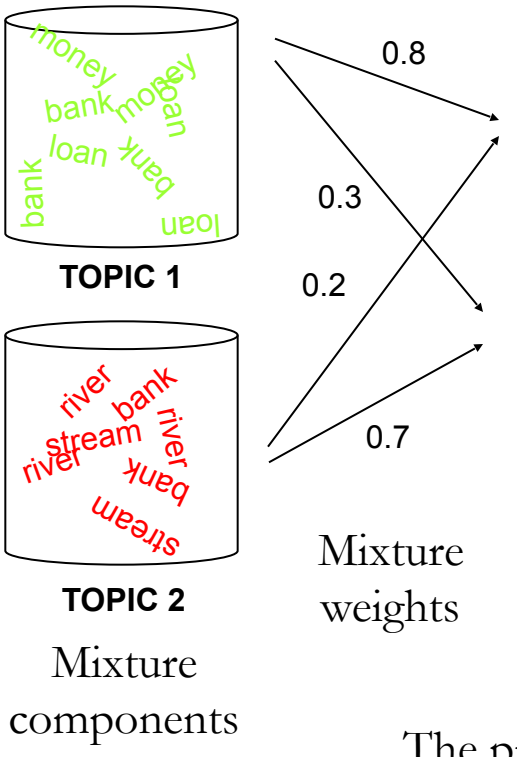
Param name	Type(s)	Default	Description
predictionCol	Int	"prediction"	Predicted cluster center
probabilityCol	Vector	"probability"	Probability of each cluster

```
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



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

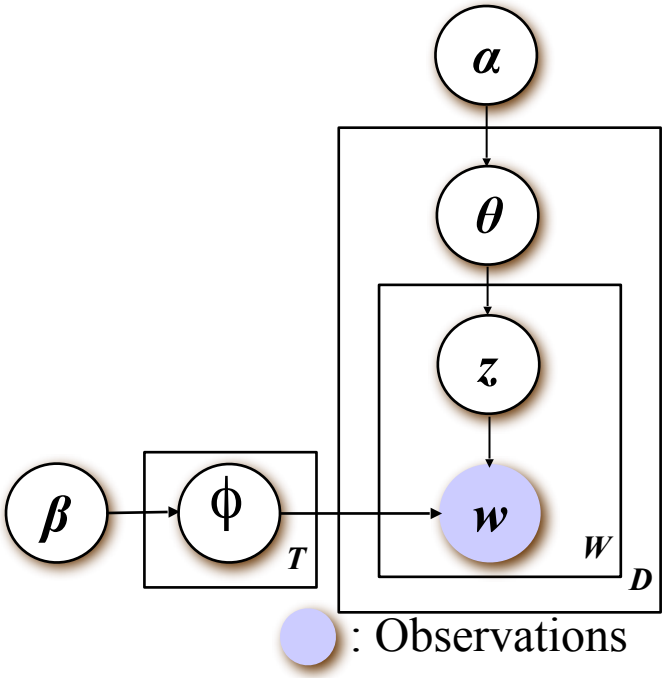
OUTPUT:

- 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

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)
```

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.

Machine Learning 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

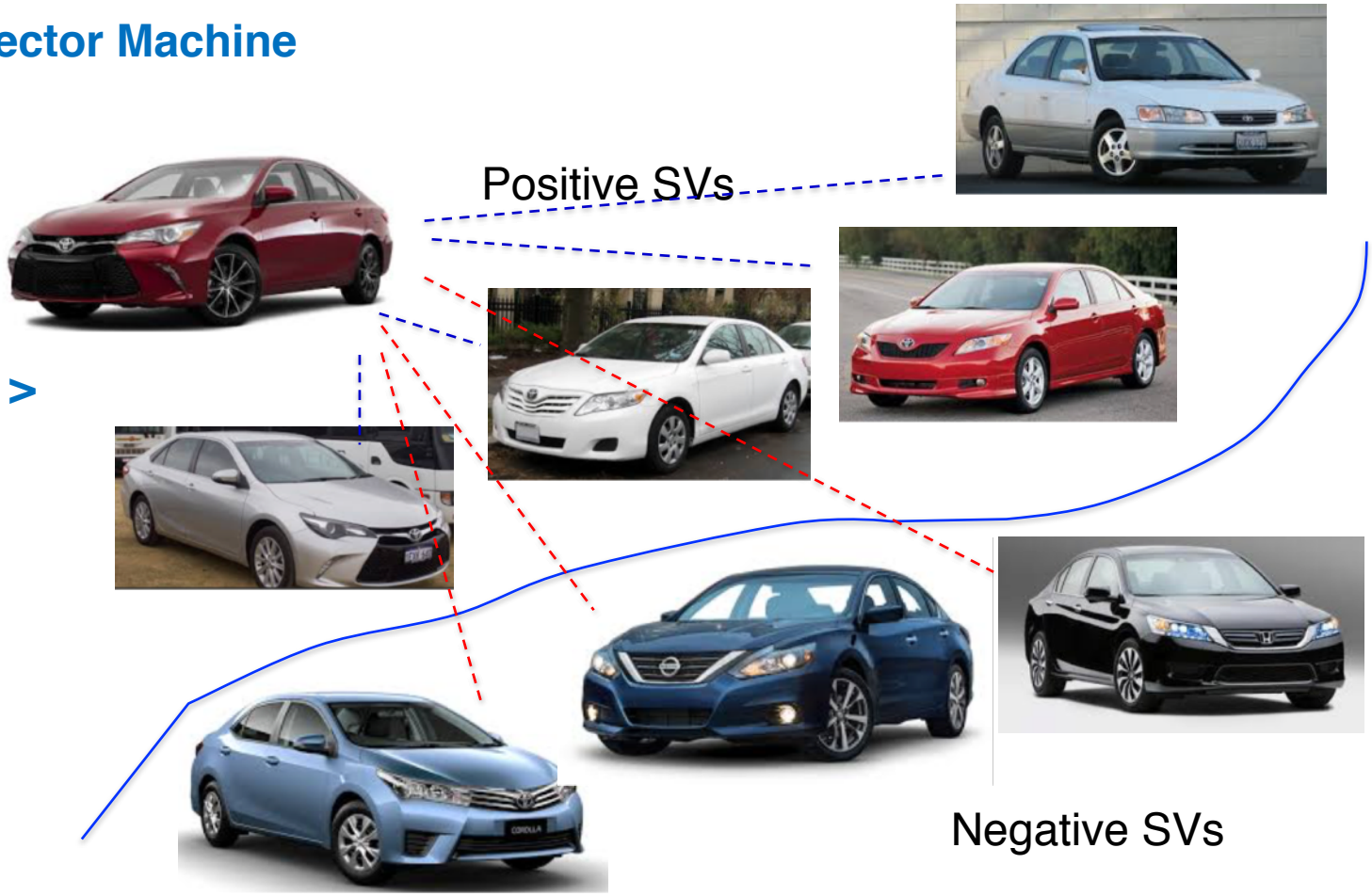
Feature Space

Positive SVs

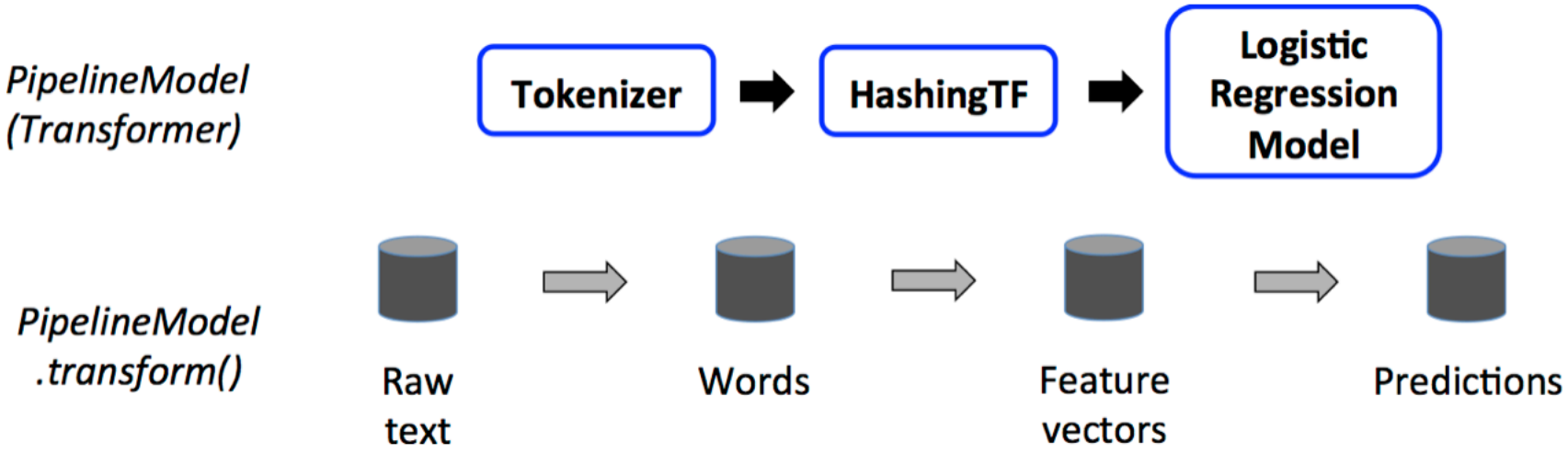
Negative SVs

Machine Learning example: using SVM to recognize a Toyota Camry

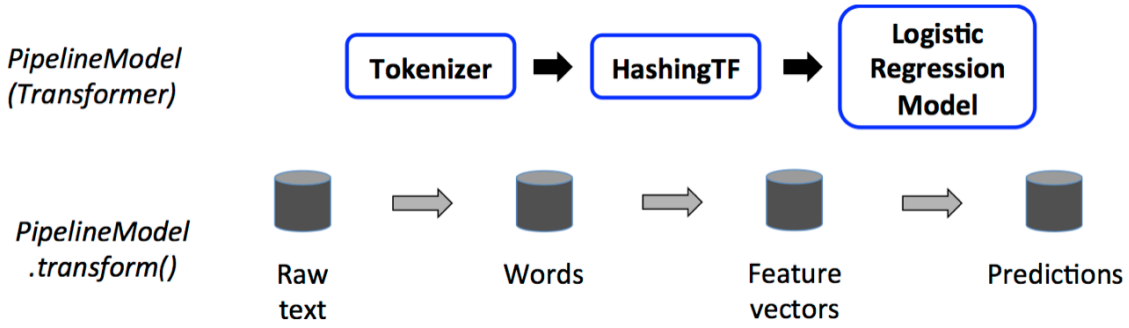
ML — Support Vector Machine



Spark ML Pipeline Example — classifier

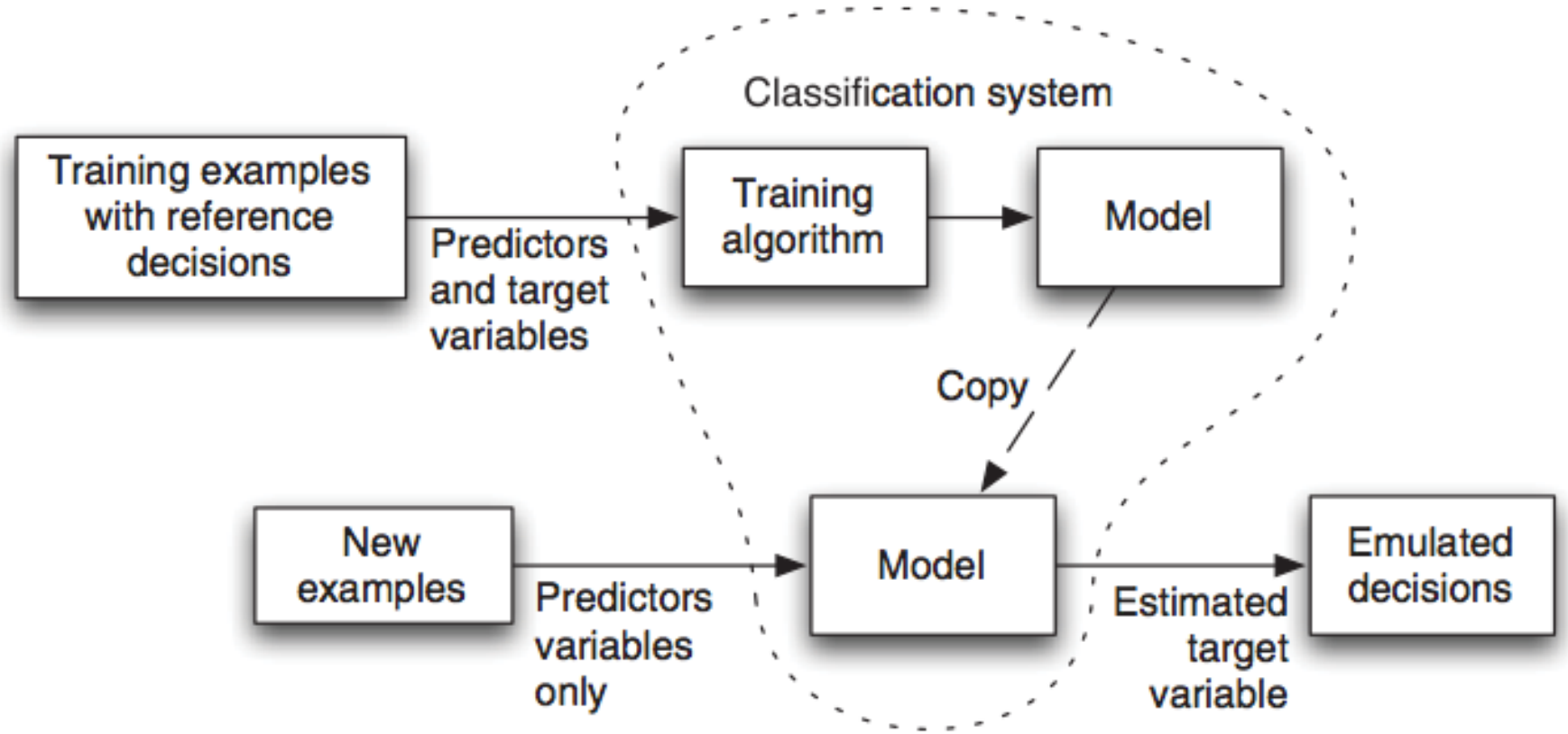


Spark ML Pipeline Example Code II




```
# 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))
```

How does a classification system work?



Fundamental classification algorithms

Example of fundamental classification algorithms:

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

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:

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

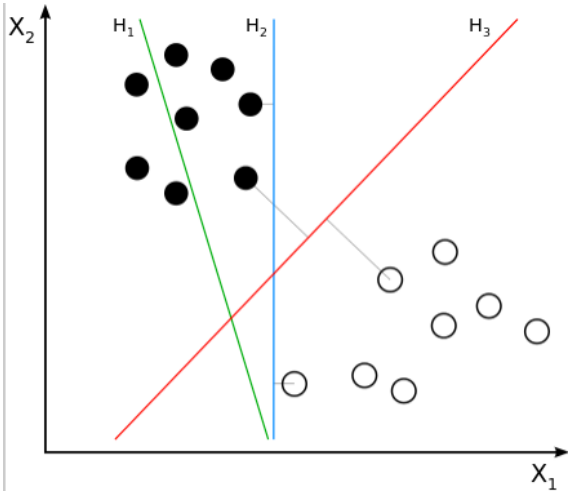
Characteristic of SGD

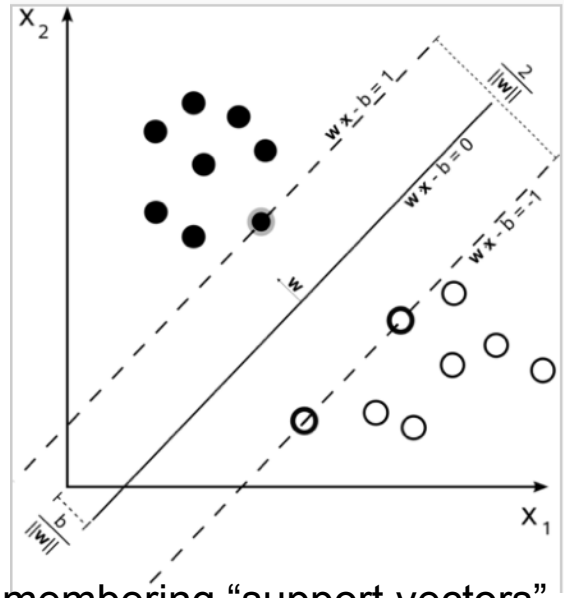
THE SGD ALGORITHM

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.

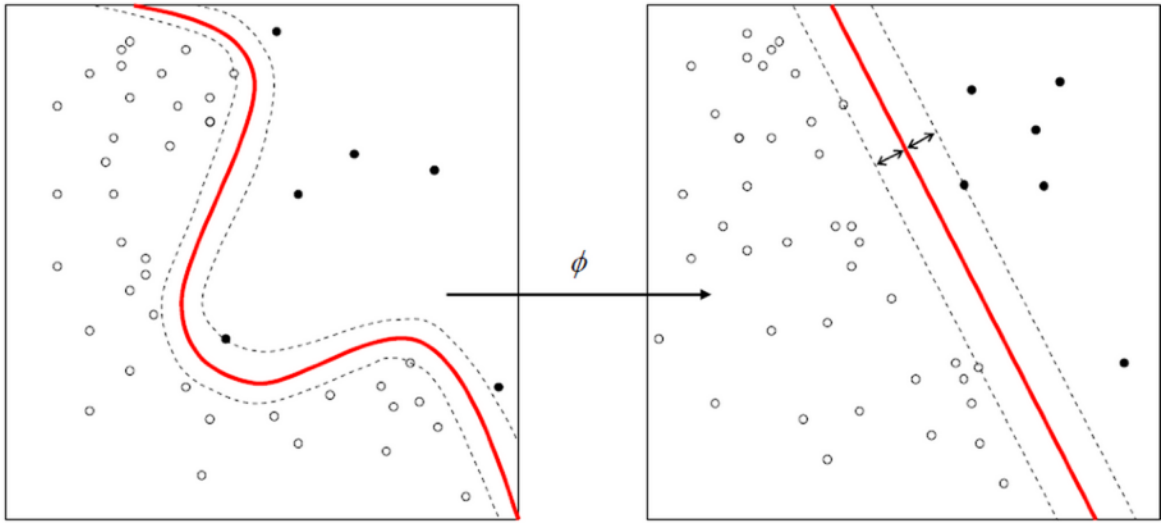
Importantly, because these algorithms do the same simple operation for each training example, they require a constant amount of memory. For this reason, each training example requires roughly the same amount of work. These properties make SGD-based algorithms scalable in the sense that twice as much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

Test Set:

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

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

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

$$P(male) = 0.5$$

$$p(\text{height}|\text{male}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(6-\mu)^2}{2\sigma^2}\right) \approx 1.5789,$$

$$p(\text{weight}|\text{male}) = 5.9881 \cdot 10^{-6}$$
$$p(\text{foot size}|\text{male}) = 1.3112 \cdot 10^{-3}$$
$$\text{posterior numerator (male)} = \text{their product} = 6.1984 \cdot 10^{-9}$$

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

==> 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.

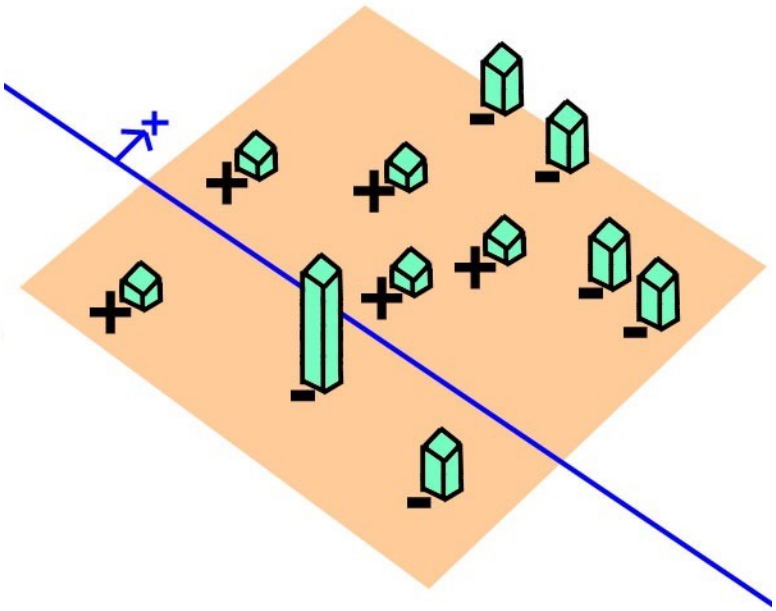
After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

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

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.

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

Each attribute is a simple classifier used in Adaboost.

- 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

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

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

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

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

where Z_s is a normalization term.

End

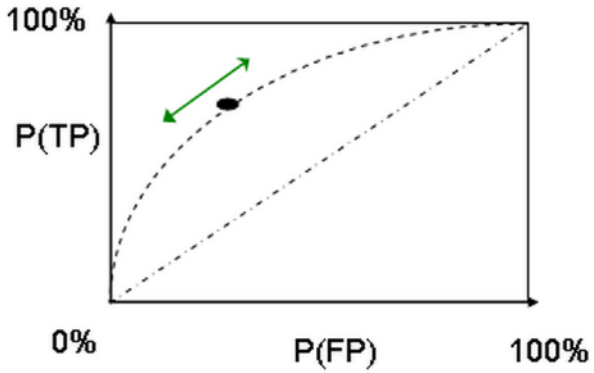
Find weak hypothesis by: $h_s = \arg\min_{h_j \in H} \varepsilon_j$.

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

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]]
...
```



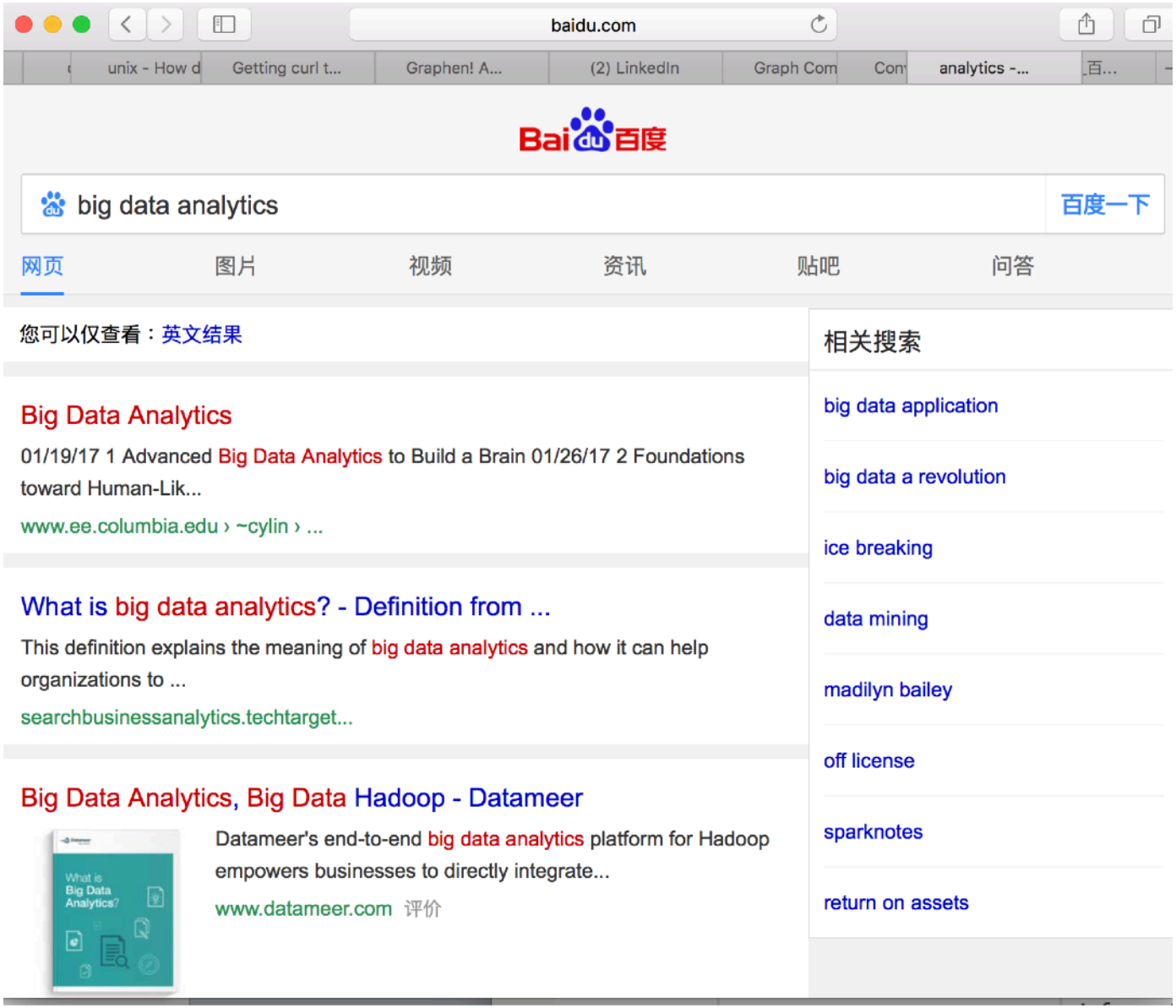
AUC (0 ~ 1):
- 1 — perfect
- 0 — perfectly wrong
- 0.5 — random

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.

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.

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());
```

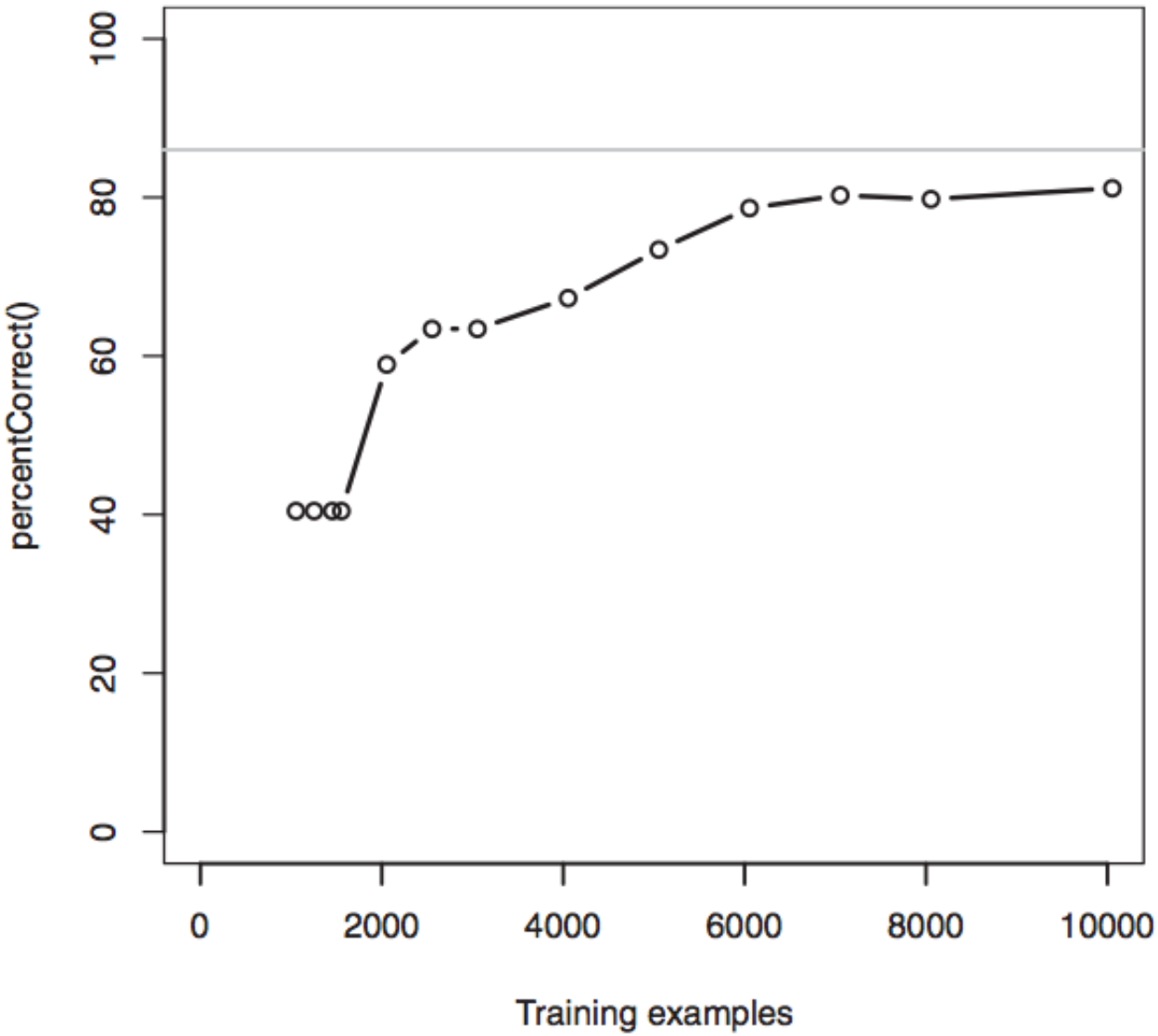
Reads and remembers values

Counts the pairs

Input contains target and model output

x2 gets true answer and score

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.

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

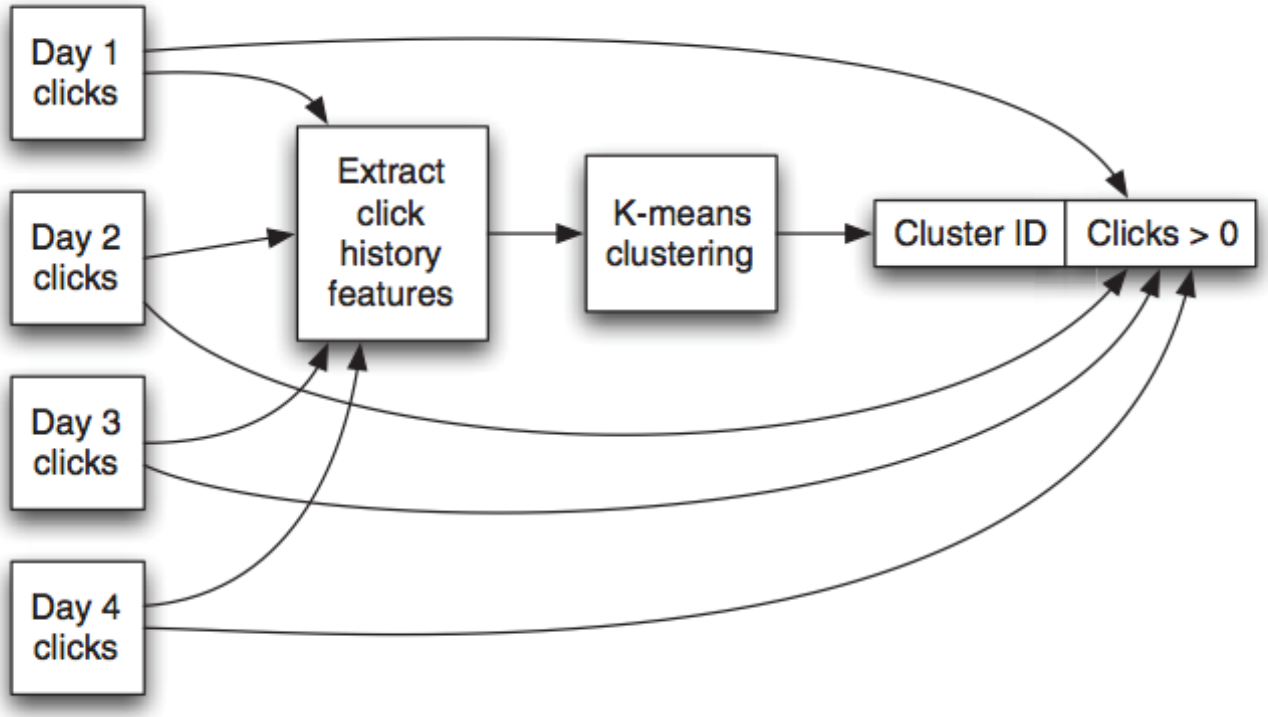
- A target leak is a bug that involves unintentionally providing data about the target variable in the section of the predictor variables.
- Don’t confused with intentionally including the target variable in the record of a training example.
- Target leaks can seriously affect the accuracy of the classification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

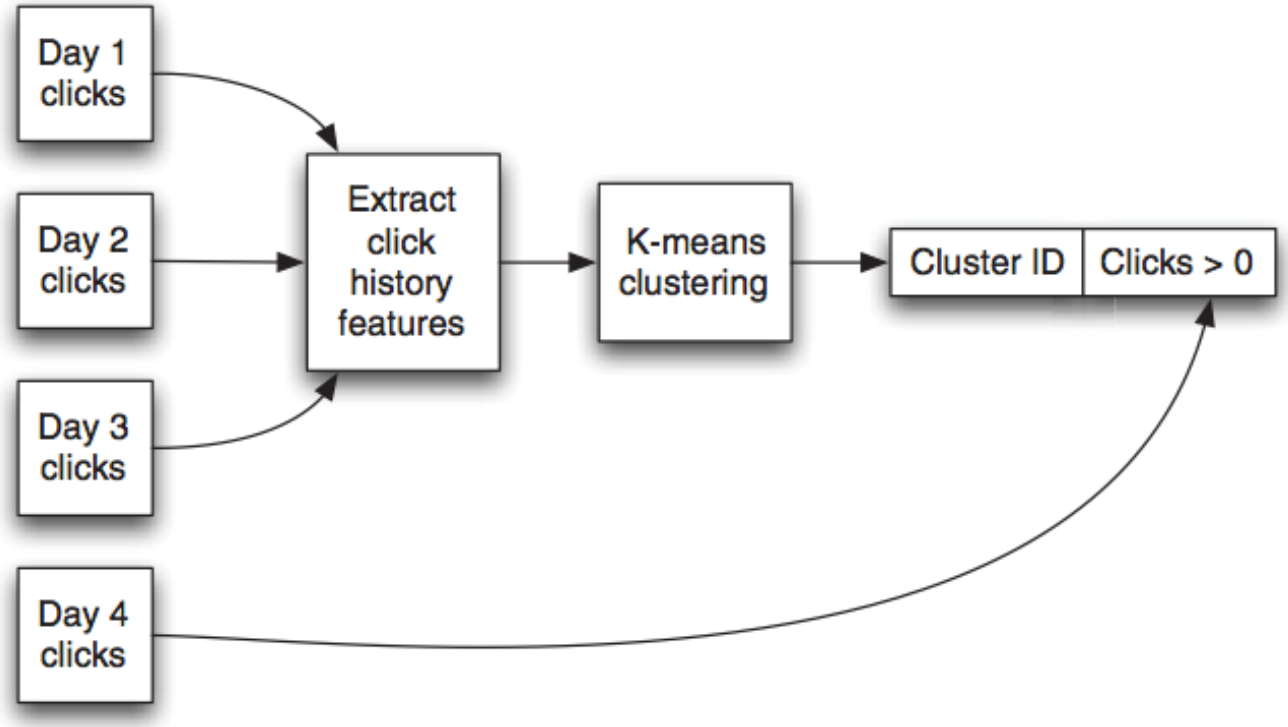
This date field is chosen so that all the documents from the same newsgroup appear to have come from the same month, but documents from different newsgroups come from different months.

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

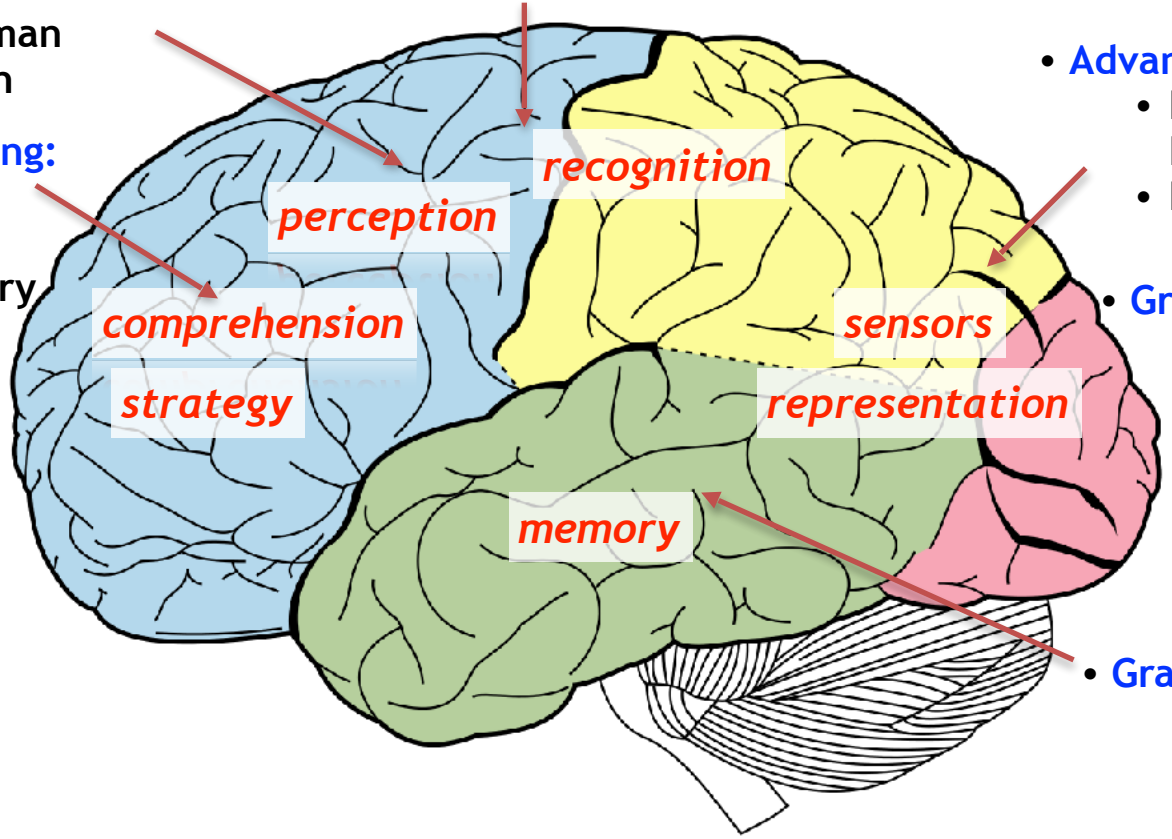


A good way to avoid a target leak: compute click history clusters based on days 1, 2, and 3, and derive the target variable (Clicks > 0) from day 4.

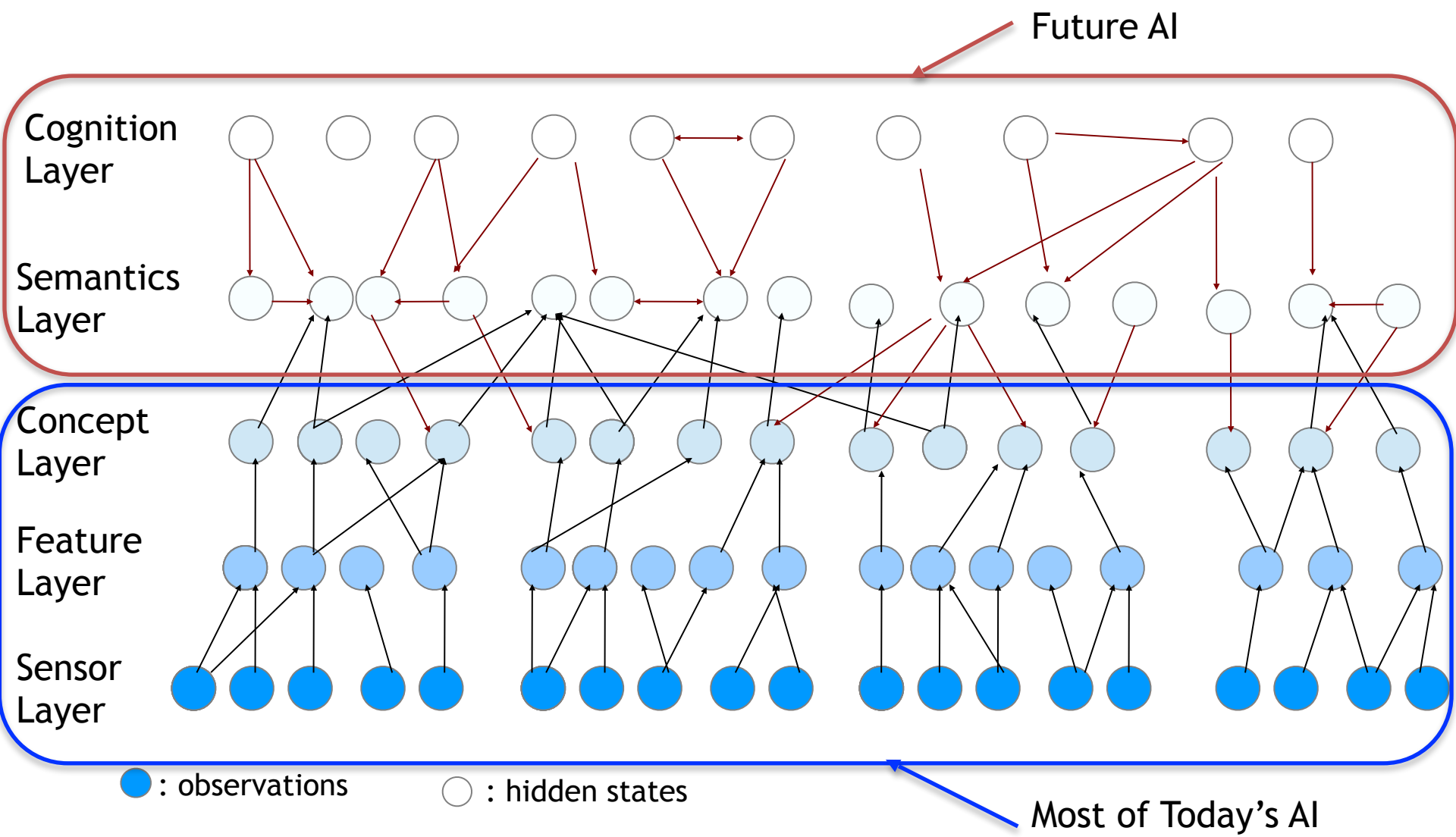
Future of AI ==> Full Function Brain Capability

Most of existing "AI" technology is only a key fundamental component.

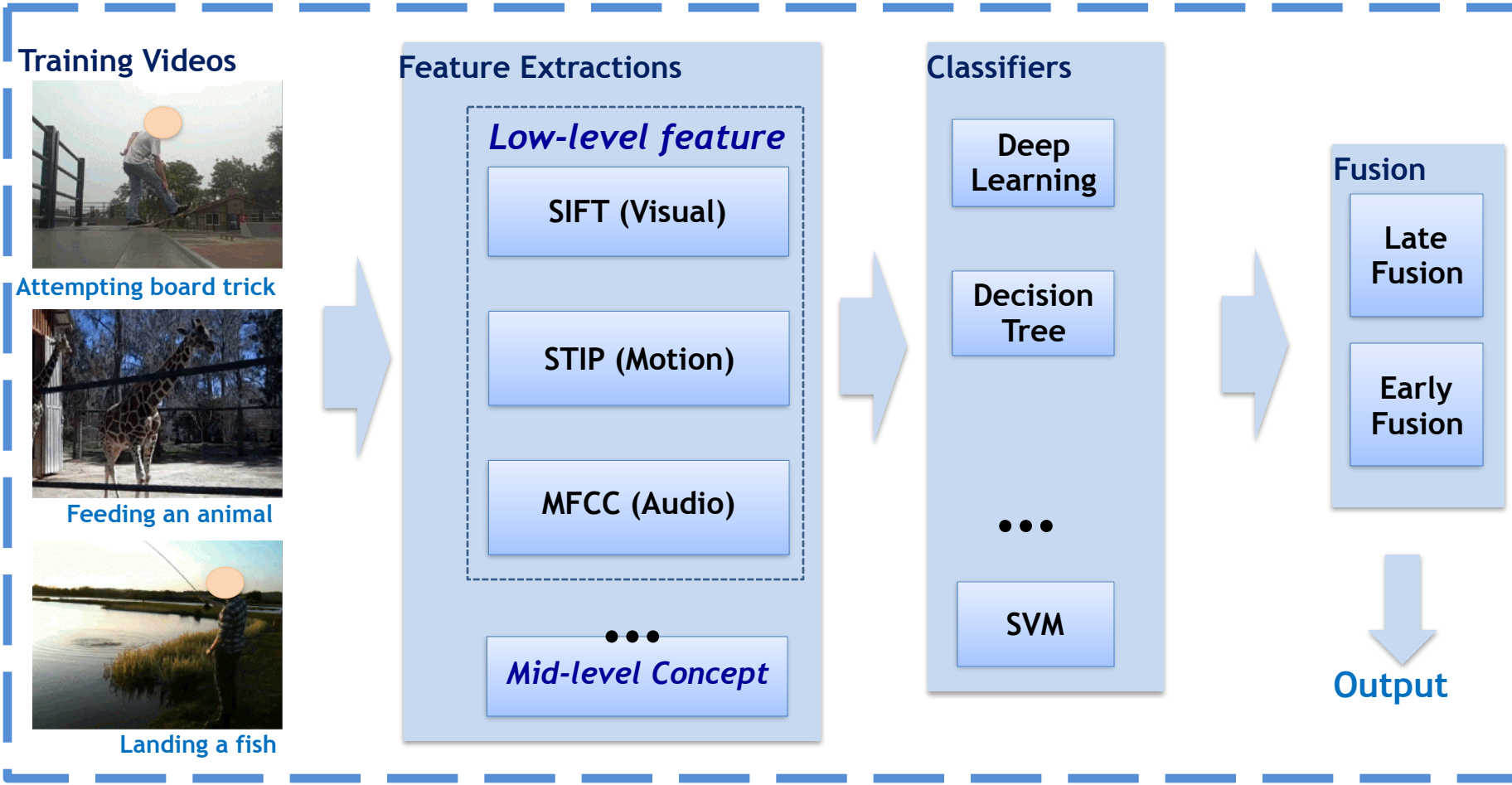
- **Machine Cognition:**
  - Robot Cognition Tools
  - Feeling
  - Robot-Human Interaction
- **Machine Learning:**
  - ML and Deep Learning
  - Autonomous Imperfect Learning
- **Machine Reasoning:**
  - Bayesian Networks
  - Game Theory Tools
- **Advanced Visualization:**
  - Dynamic and Interactive Viz.
  - Big Data Viz.
- **Graph Analytics:**
  - Network Analysis
  - Flow Prediction
- **Graph Database:**
  - Distributed Native Database



Evolution of Artificial Intelligence

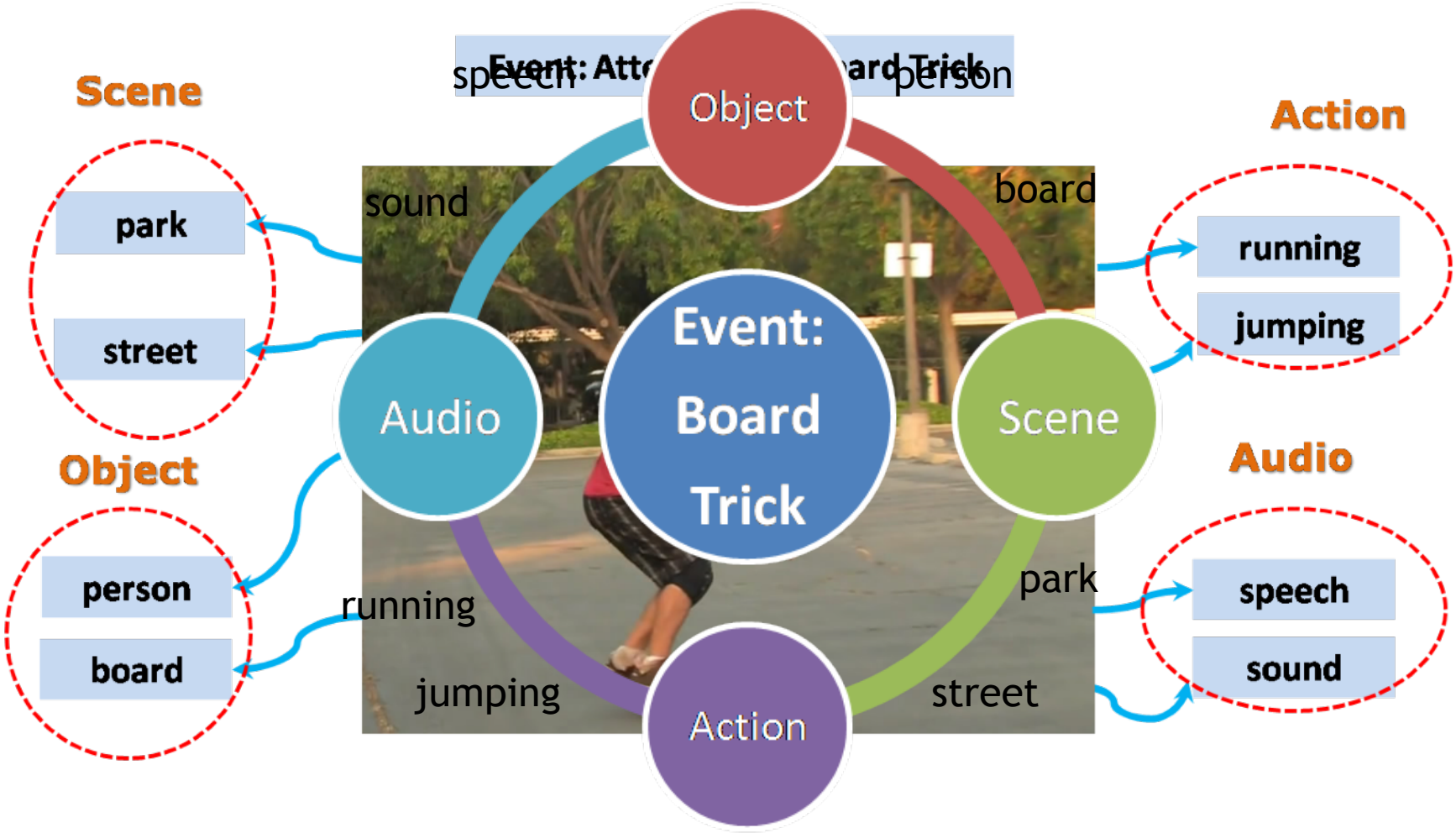


Event Detection Baseline

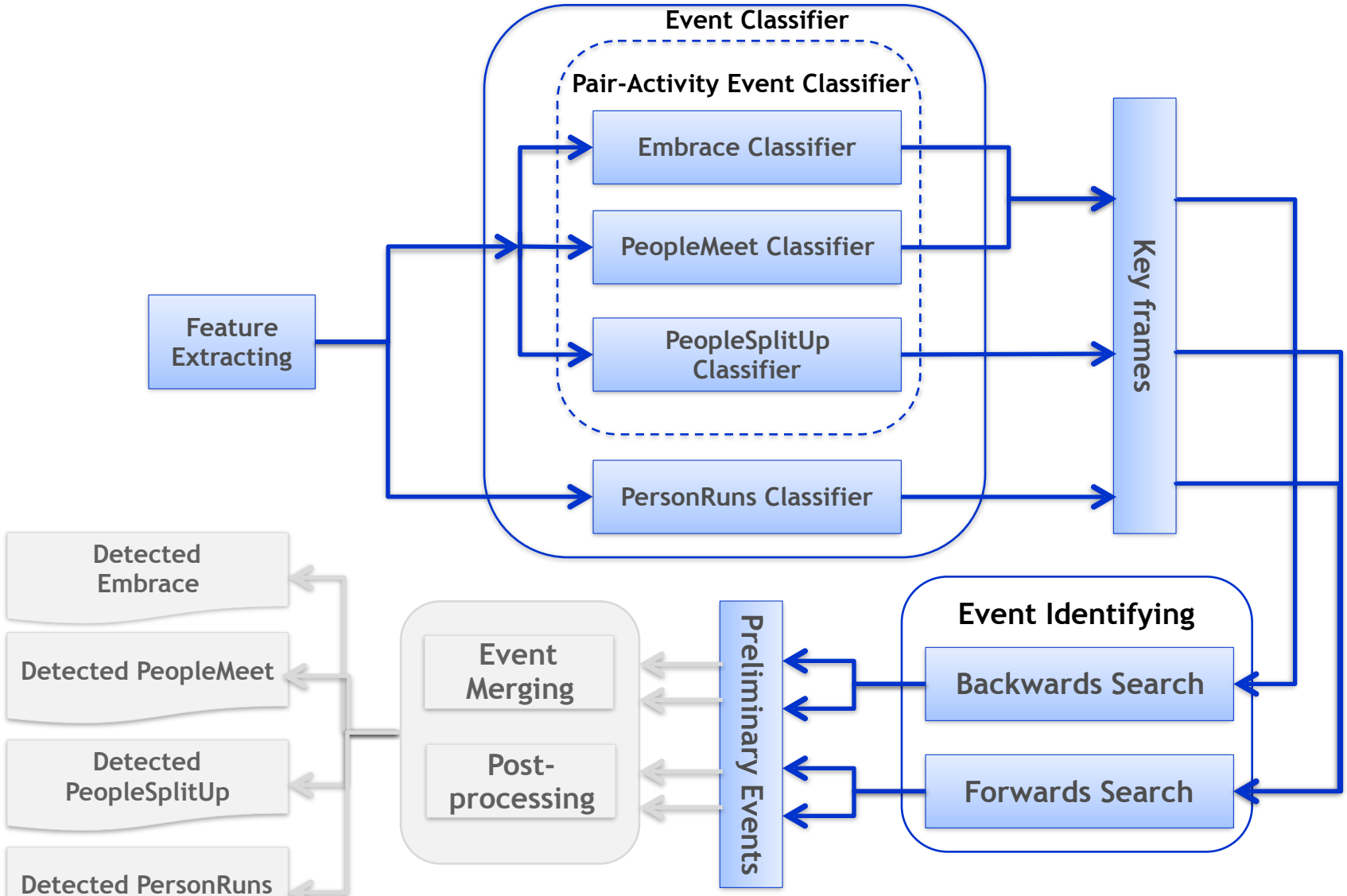


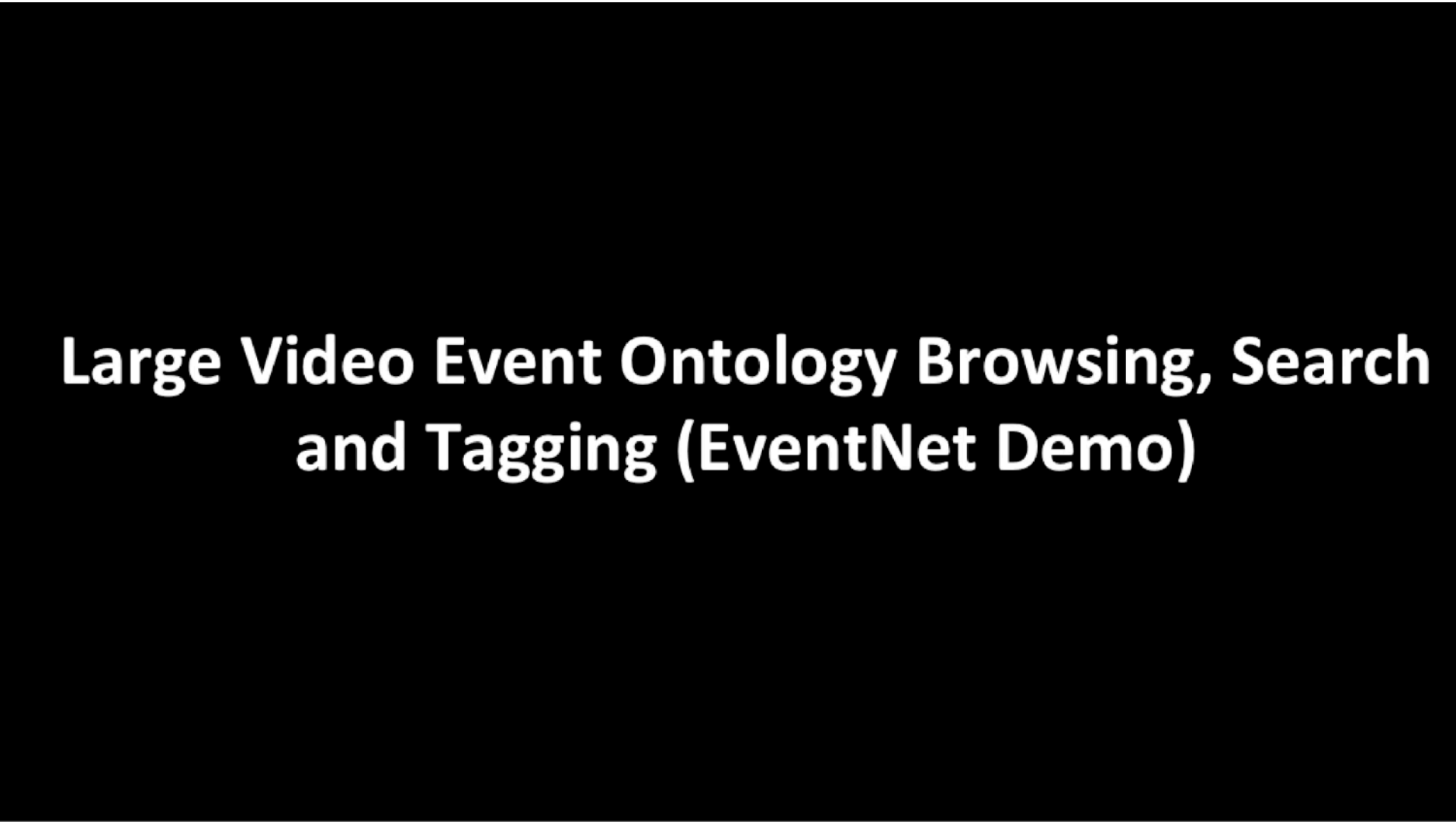
Mid-level Feature Representation

- Decompose an event into concepts



Events Classification Framework





Examples of our Previous work on Abnormal Video Event Analysis

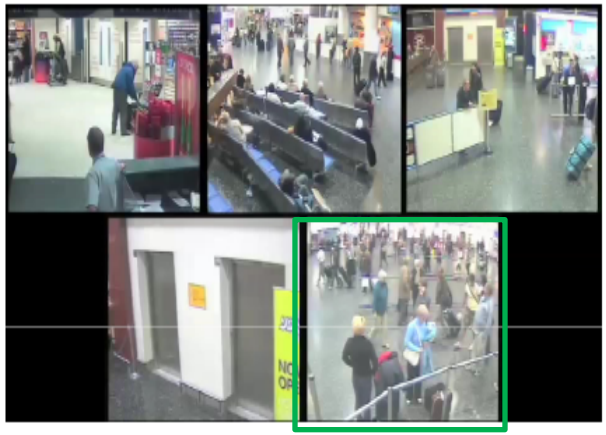
Event: Abnormal Behavior (Surveillance Video)

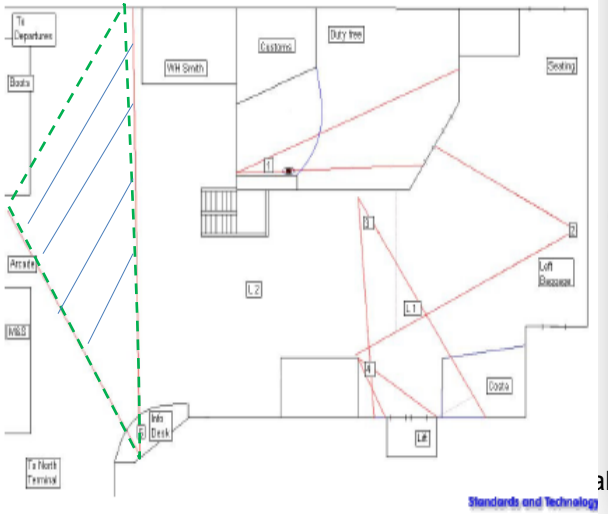
TRECVID Surveillance Event Detection (SED) Evaluation 2008-2016

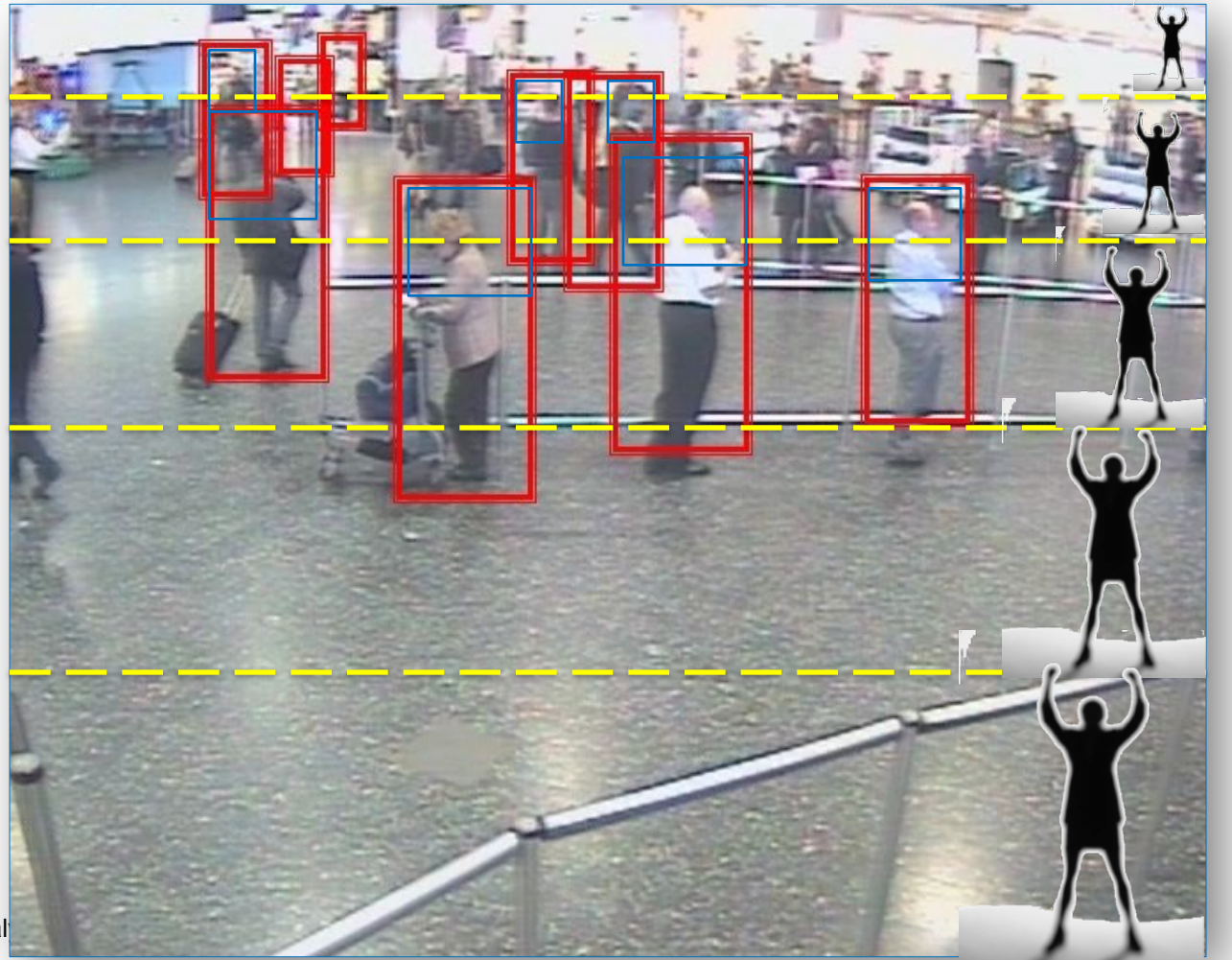
Event: Making a bomb (Consumer Video)

TRECVID Multimedia Event Detection (MED) Evaluation 2010-2016

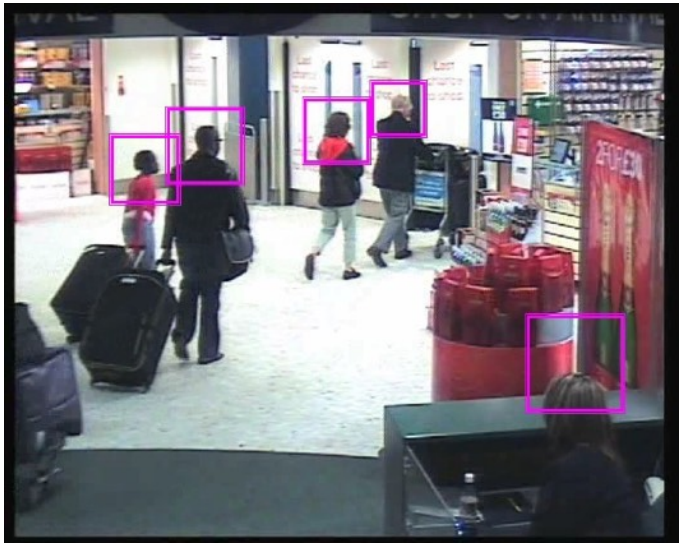
Detection and Tracking of Head, Shoulder, and Body

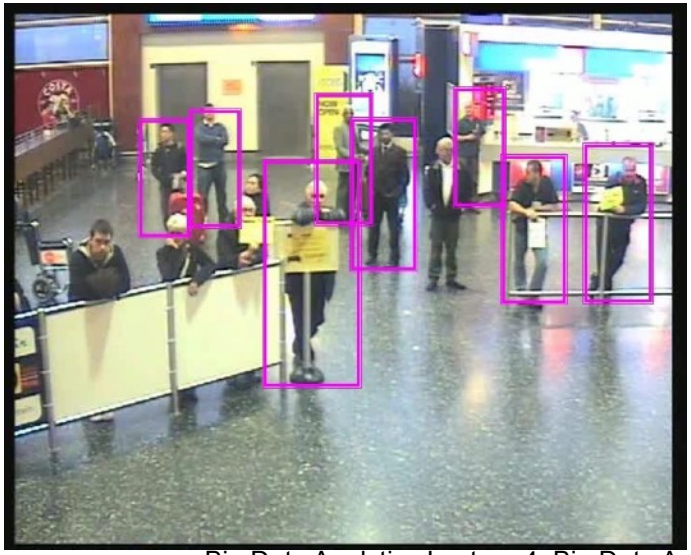


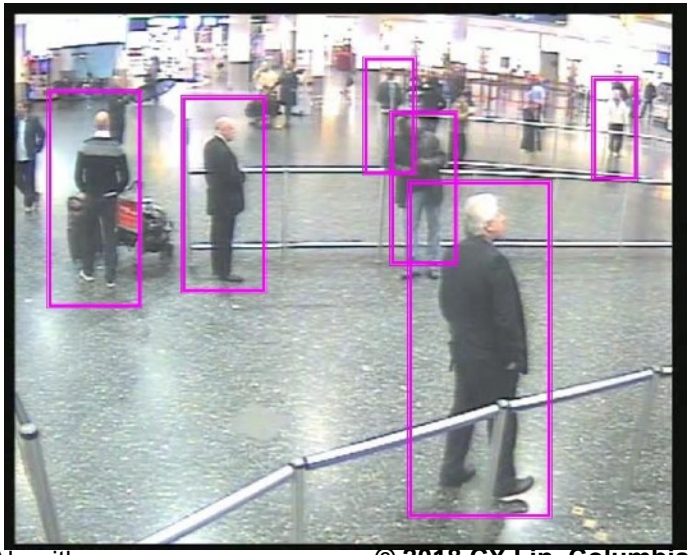




Detection Results







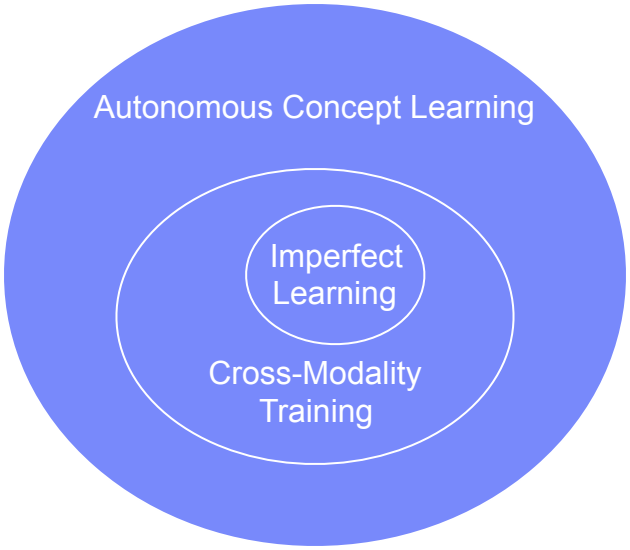
Imperfect Learning for Autonomous Concept Modeling Learning

Reference: C.-Y. Lin et al., SPIE EI West, 2005

A solution for the scalability issues at training..

Autonomous Learning of Video Concepts through Imperfect Training Labels:

- Develop theories and algorithms for supervised concept learning from imperfect annotations -- imperfect learning
- Develop methodologies to obtain imperfect annotation – learning from cross-modality information or web links
- Develop algorithms and systems to generate concept models – novel generalized Multiple-Instance Learning algorithm with Uncertain Labeling Density



What is Imperfect Learning?

Definitions from Machine Learning Encyclopedia:

Supervised learning:

a machine learning technique for creating a function from training data.

The training data consists of pairs of input objects and desired outputs.

The output of the function can be a continuous value (called regression), or can predict a class label of the input object (called classification).

Predict the value of the function for any valid input object after having seen only a small number of training examples.

The learner has to generalize from the presented data to unseen situations in a "reasonable" way.

Unsupervised learning:

a method of machine learning where a model is fit to observations.

It is distinguished from supervised learning by the fact that there is no a priori output.

A data set of input objects is gathered. Unsupervised learning then typically treats input objects as a set of random variables.

A joint density model is then built for the data set.

Proposed Definition of Imperfect Learning:

A supervised learning technique with imperfect training data.

The training data consists of pairs of input objects and desired outputs. There may be error or noise in the desired output of training data.

The input objects are typically treated as a set of random variables.

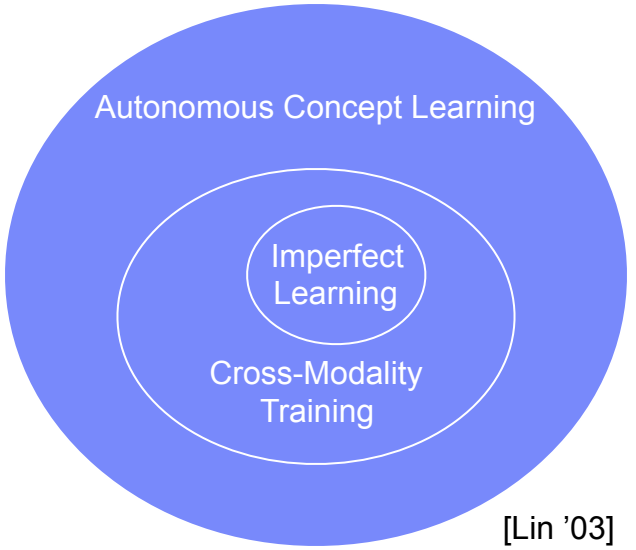
Why do we need Imperfect Learning?

Annotation is a Must for Supervised Learning.

- All (or almost all?) modeling/fusion techniques in our group used annotation for training
- However, annotation is time- and cost- consuming.
- Previous focuses were on improving the annotation efficiency – minimum GUI interaction, template matching, active learning, etc.

Is there a way to avoid annotation?

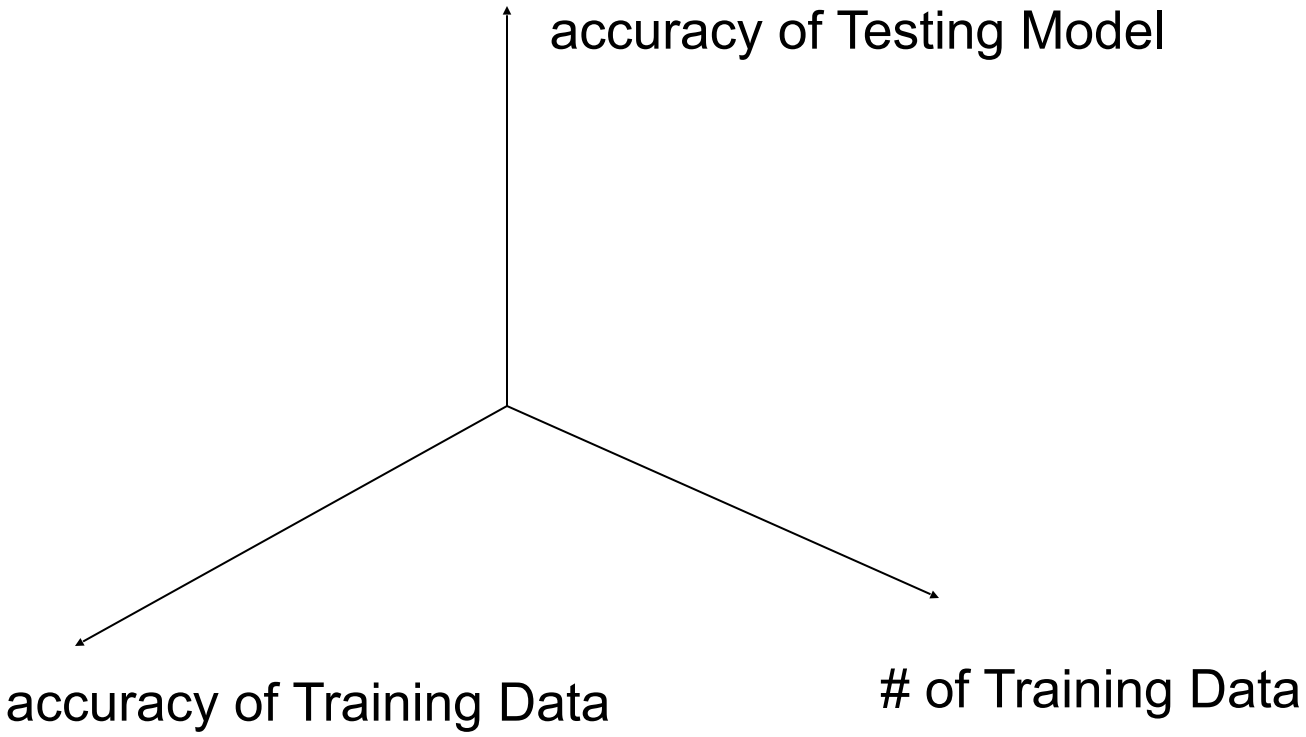
- Use imperfect training examples that are obtained automatically/unsupervisedly from other learning machine(s).
- These machines can be built based on other modalities or prior machines on related dataset domain.



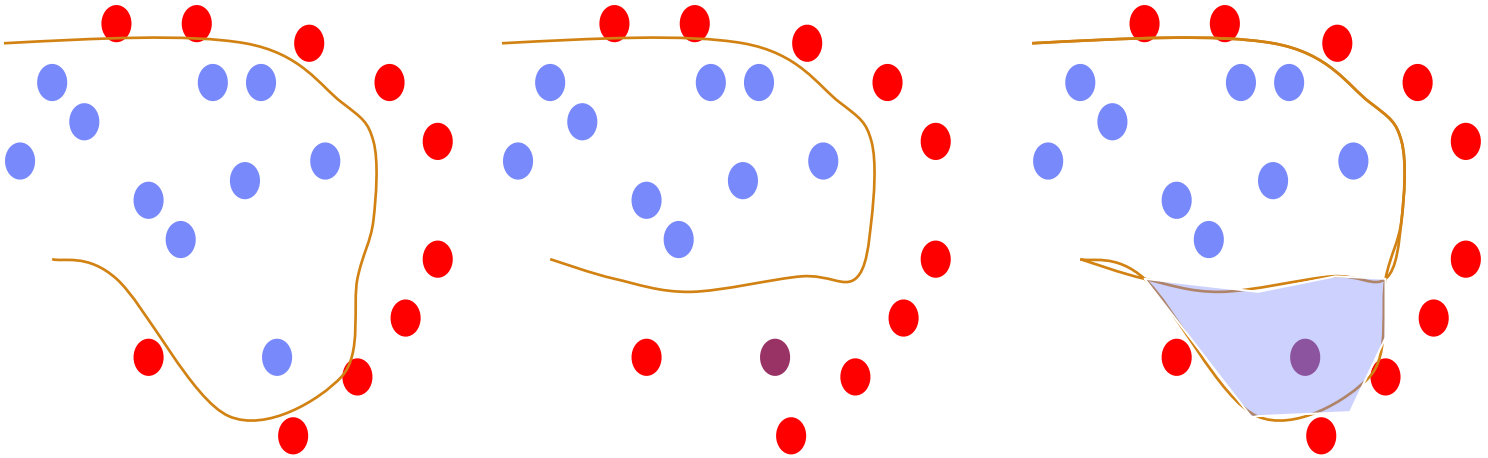
[Lin '03]

Proposition

Supervised Learning ← Time consuming; Spend a lot of time to do the annotation
Unsupervised continuous learning ← When will it beat the supervised learning?



The key objective of this paper – can concept models be learned from imperfect labeling?



Example: The effect of imperfect labeling on classifiers (left -> right: perfect labeling, imperfect labeling, error classification area)

False positive Imperfect Learning

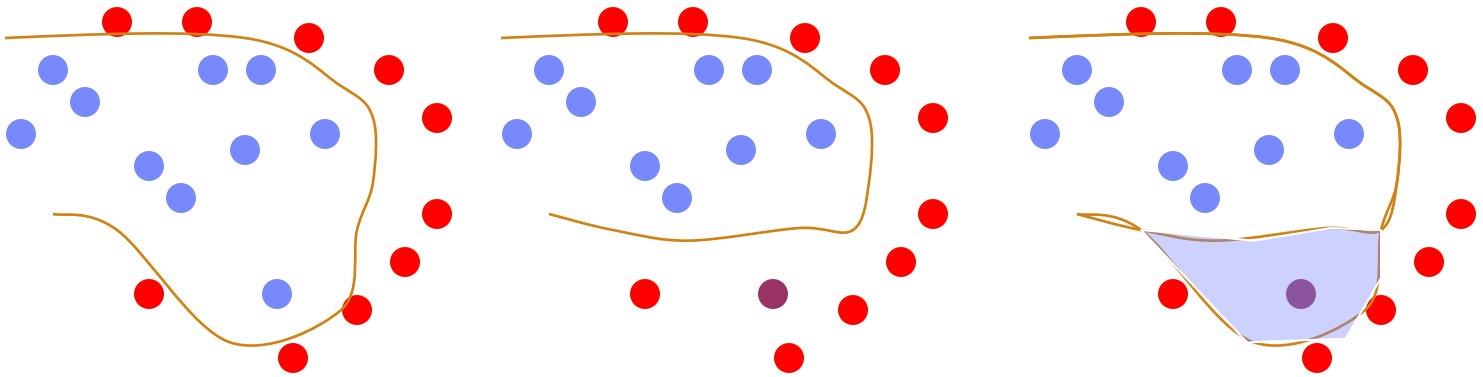
Assume we have ten positive examples and ten negative examples. if 1 positive example is wrong (false positive), how will it affect SVM? Will the system break down? Will the accuracy decrease significantly?

If the ratio change, how is the result?

Does it depend on the testing set?

If time goes by and we have more and more training data, how will it affect? In what circumstance, the effect of false positive will decrease? In what situation, the effect of false positive will still be there?

Assume the distribution of features of testing data is similar to the training data. When will it



Imperfect Learning

If learning example is not perfect, what will be the result?

If you teach something wrong, what will be the consequence?

Case 1: False positive only

Case 2: False positive and false negative

Case 3: Learning example has confidence value

From Hessienberg's Uncertainty Theory

- From Hessienberg's Uncertainty Theory, everything is random. It is not measurable. Thus, we can assume a random distribution of positive ones and negative ones.
- Assume there are two Gaussians in the feature space. One is positive. The other one is negative.
- Let's assume two situations. The first one: every positive is from positive and every negative is from negative. The second one: there may be some random mistake in the negative.
- Also, let's assume two cases. 1. There are overlap between two Gaussians. 2. There are not. So, maybe these can be derived to become a variable based on mean and sigma.
- If the training samples of SVM are random, how will be the result? Is it predictable with a closed mathematical form?
- How about using linear example in the beginning and then use the random examples next?

False Positive Samples

- Will false positive examples become support vectors? Very likely. We can also assume a r.v. here.

- Maybe we can also using partially right data ➔ Having more weighting on positive ones.
- Then for the uncertain ones ➔having fewer chance to become support vector

- Will it work if, when support vector is picked, we take the uncertainty as a probability? Or, should we compare it to other support vectors? This can be an interesting issue. It's like human brain. The first one you learn, you remember it. The later ones you may forget about it. The more you learn the more it will be picked. The fewer it happens, it will be more easily forgotten. Maybe I can even develop a theory to simulate human memory.

- Uncertainty can be a time function. Also, maybe the importance of support vector can be a time function. So, sometimes machine will forget things. ➔ This make it possible to adapt and adjustable to outside environment.

- Maybe I can develop a theory of continuous learning

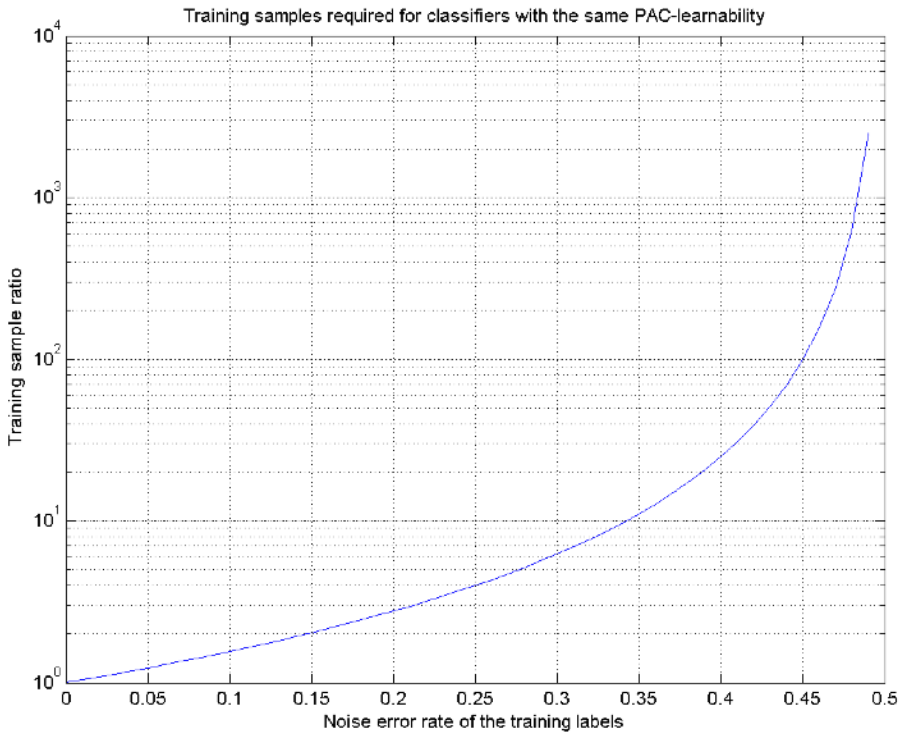
- Or, continuous learning based on imperfect memory

- In this way, the learning machine will be affected mostly by the current data. For those 'old' data, it will put less weighting ➔ may reflect on the distance function.

- Our goal is to have a very large training set. Remember a lot of things. So, we need to learn to forget.

Imperfect Learning: theoretical feasibility

- Imperfect learning can be modeled as the issue of noisy training samples on supervised learning.
- Learnability of concept classifiers can be determined by probably approximation classifier (pac-learnability) theorem.
- Given a set of "fixed type" classifiers, the pac-learnability identifies a minimum bound of the number of training samples required for a fixed performance request.
- If there is noise on the training samples, the above mentioned minimum bound can be modified to reflect this situation.
- The ratio of required sample is independent of the requirement of classifier performance.
- Observations: practical simulations using SVM training and detection also verify this theorem.



A figure of theoretical requirement of the number of sample needed for noisy and perfect training samples

PAC-identifiable

- PAC-identifiable: PAC stands for *probably approximate correct*. Roughly, it tells us a class of concepts **C** (defined over an input space with examples of size **N**) is PAC learnable by a learning algorithm **L**, if for arbitrary small δ and ε, and for all concepts *c* in **C**, and for all distributions **D** over the input space, there is a *1*-δ probability that the hypothesis *h* selected from space **H** by learning algorithm **L** is approximately correct (has error less than ε).

$$\Pr_D(\Pr_X(h(x) \neq c(x)) \geq \varepsilon) \leq \delta$$

- Based on the PAC learnability, assume we have *m* independent examples. Then, for a given hypothesis, the probability that m examples have not been misclassified is $(1-e)^m$ which we want to be less than δ. In other words, we want $(1-e)^m <= \delta$. Since for any $0 <= x <1$, $(1-x) <= e^{-x}$, we then have:

$$m \geq \frac{1}{\varepsilon} \ln(\frac{1}{\delta})$$

Sample Size v.s. VC dimension

Theorem 2 Let **C** be a nontrivial, well-behaved concept class. If the VC dimension of **C** is *d*, where $d < \infty$, then for $0 < e < 1$ and

$$m \geq \max\left(\frac{4}{\varepsilon}\log_2\frac{2}{\delta}, \frac{8d}{\varepsilon}\log_2\frac{13}{\varepsilon}\right)$$

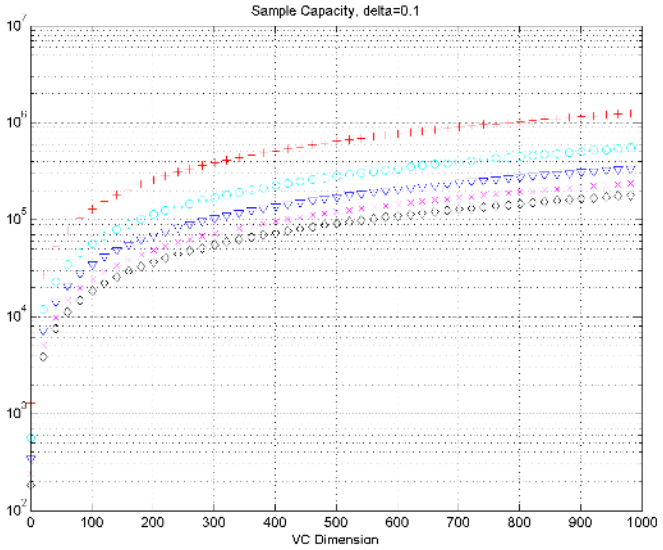
any consistent function A: Sc**C** is a learning function for **C,** and,
for $0 < e < 1/2$, m has to be larger than or equal to a lower bound,

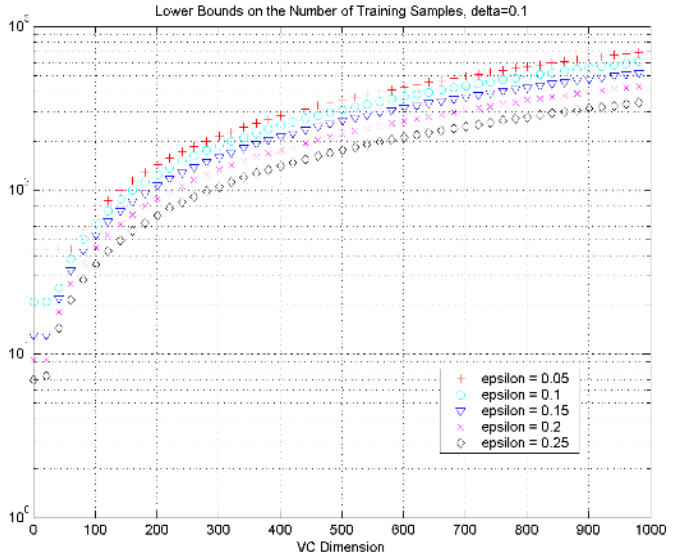
$$m \geq \max\left[\frac{1-\varepsilon}{\varepsilon}\ln\left(\frac{1}{\delta}\right), d\cdot(1-2\varepsilon(1-\delta)+2\delta))\right]$$

For any *m* smaller than the lower bound, there is no function A: Sc**H**, for any hypothesis space **H**, is a learning function for **C**. The sample space of **C**, denoted SC, is the set of all

How many training samples are required?

- Examples of training samples required in different error bounds for PAC-identifiable hypothesis. This figure shows the upper bounds and lower bounds at Theorem 2. The upper bound is usually refereed as sample capacity, which guarantees the learnability of training samples.





Noisy Samples

Theorem 4 Let $h < 1/2$ be the rate of classification noise and N the number of rules in the class **C**. Assume $0 < e, h < 1/2$. Then the number of examples, m, required is at least

$$m \geq \max\left[\frac{\ln(2\delta)}{\ln(1-\varepsilon(1-2\eta))}, \log_2 N \cdot (1-2\varepsilon(1-\delta)+2\delta))\right]$$

and at most

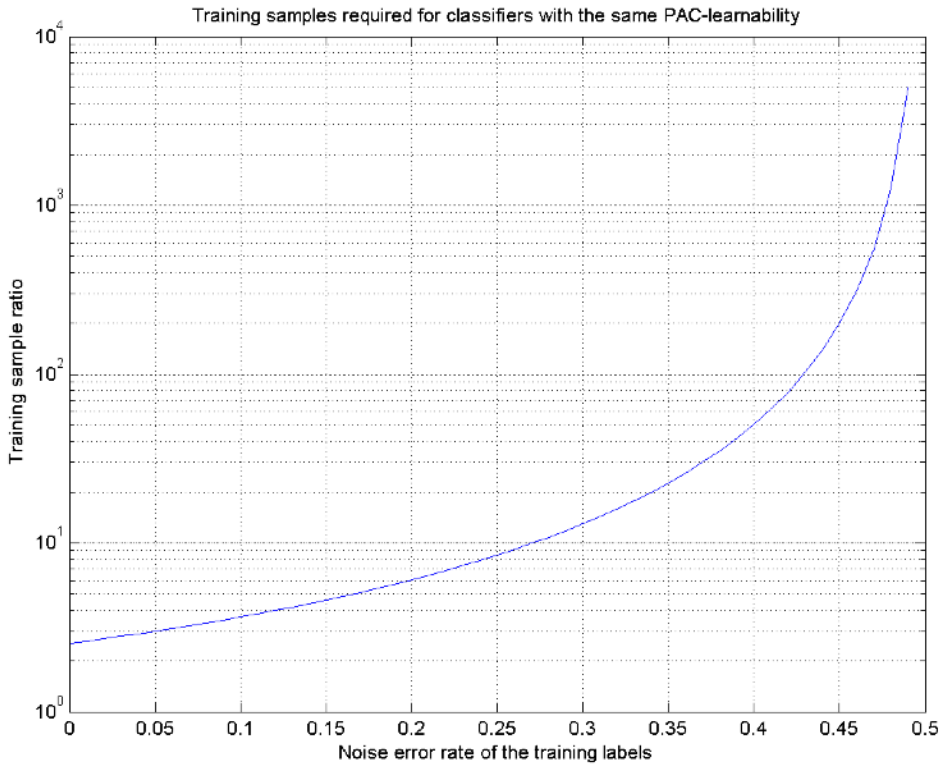
$$\frac{\ln(N/\delta)}{\varepsilon \cdot (1-\exp(-\frac{1}{2}(1-2\eta)^2))}$$

r is the ratio of the required noisy training samples v.s. the noise-free training samples

$$r_\eta = (1-\exp(-\tfrac{1}{2}(1-2\eta)^2))^{-1}$$

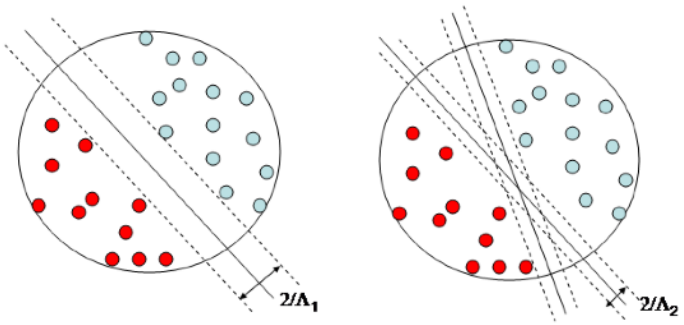
Training samples required when learning from noisy examples

Ratio of the training samples required to achieve PAC-learnability under the noisy and noise-free sampling environments. This ratio is consistent on different error bounds and VC dimensions of PAC-learnable hypothesis.



Learning from Noisy Examples on SVM

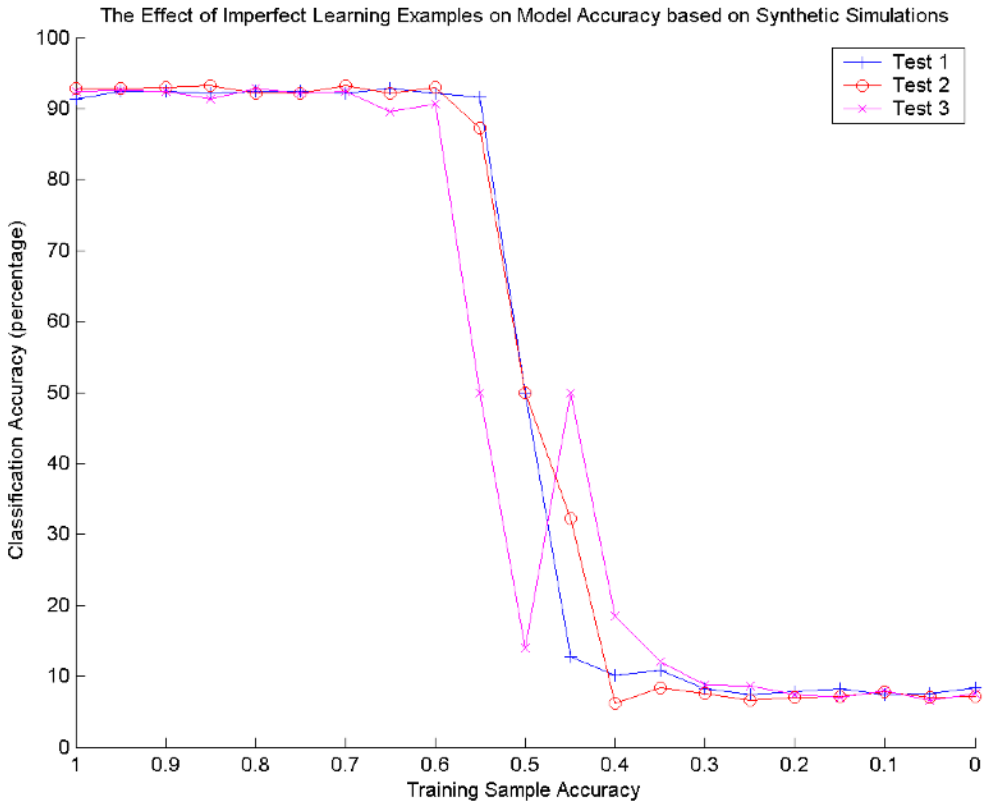
For an SVM, we can find the bounded VC dimension:



$$d \leq \min(\Lambda^2 R^2 + 1, n + 1)$$

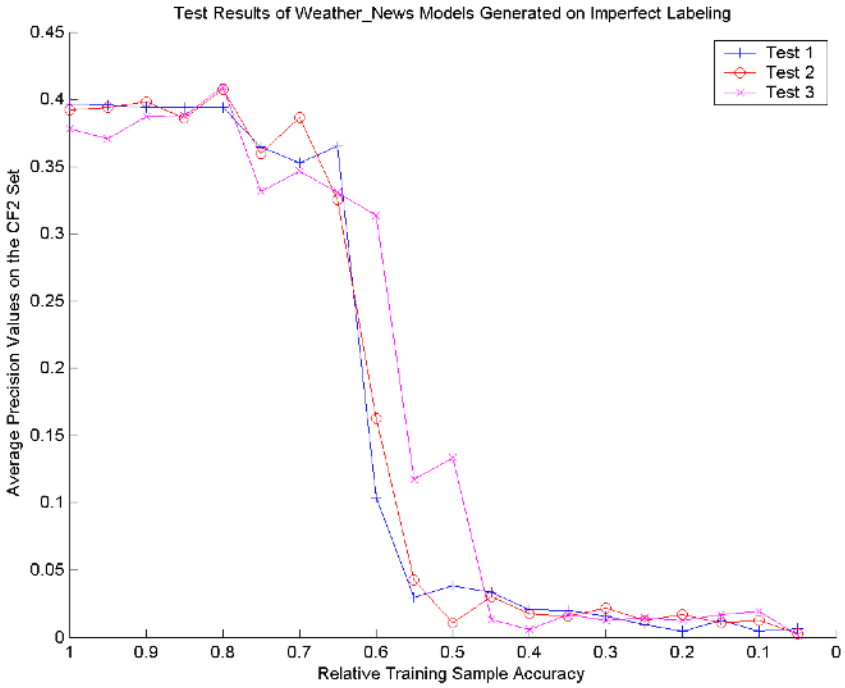
Experiments - 1

Examples of the effect of noisy training examples on the model accuracy. Three rounds of testing results are shown in this figure. We can see that model performance does not have significant decrease if the noise probability in the training samples is larger than 60% - 70%. And, we also see the reverse effect of the training samples if the mislabeling probability is larger than 0.5.



Experiments – 2:

Experiments of the effect of noisy training examples on the visual concept model accuracy. Three rounds of testing results are shown in this figure. We simulated annotation noises by randomly change the positive examples in manual annotations to negatives. Because *perfect* annotation is not available, accuracy is shown as a relative ratio to the manual annotations in [10]. In this figure, we see the model accuracy is not significantly affected for small noises. A similar drop on the training examples is observed at around 60% - 70% of annotation accuracy (i.e., 30% - 40% of missing annotations).



Conclusion

Imperfect learning is possible.

In general, the performance of SVM classifiers do not degrade too much if the manual annotation accuracy is larger than about 70%.

Continuous Imperfect Learning shall have a great impact in autonomous learning scenarios.

Questions?