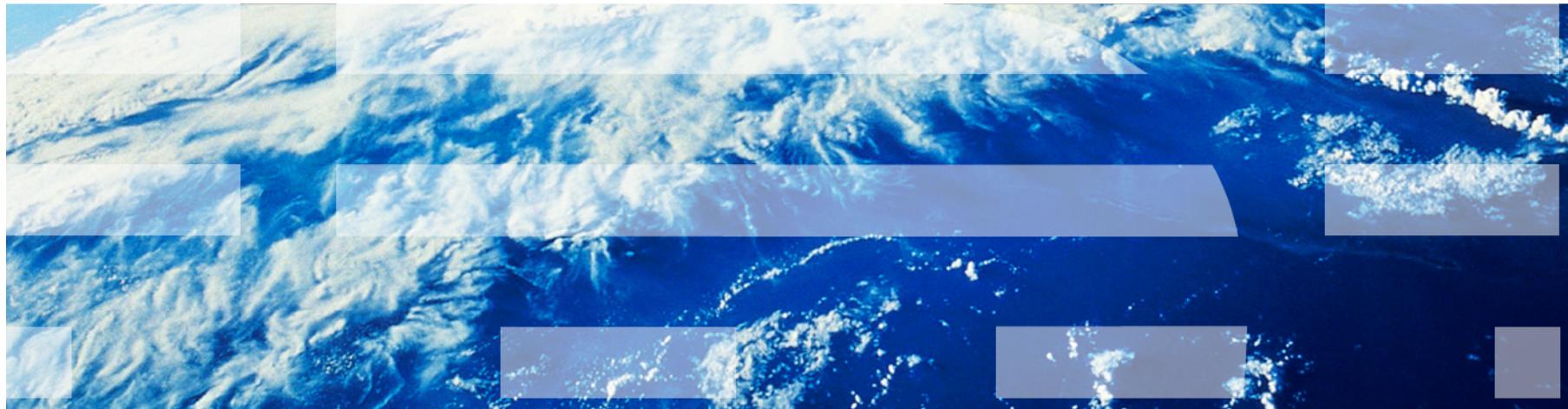


E6893 Big Data Analytics Lecture 3:

Big Data Analytics Algorithms

Ching-Yung Lin, Ph.D.

Adjunct Professor, Dept. of Electrical Engineering and Computer Science



September 22, 2023

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

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

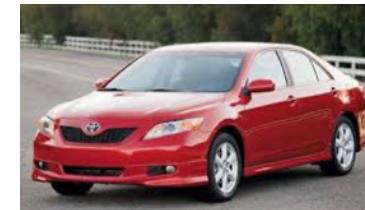
- Rule 1. Symbol has something like bull's head
- Rule 2. Big black portion in front of car.
- Rule 3.????

5

ML – Support Vector Machine

Feature Space

Positive SVs



Negative SVs

Classification example: using SVM to recognize a Toyota Camry

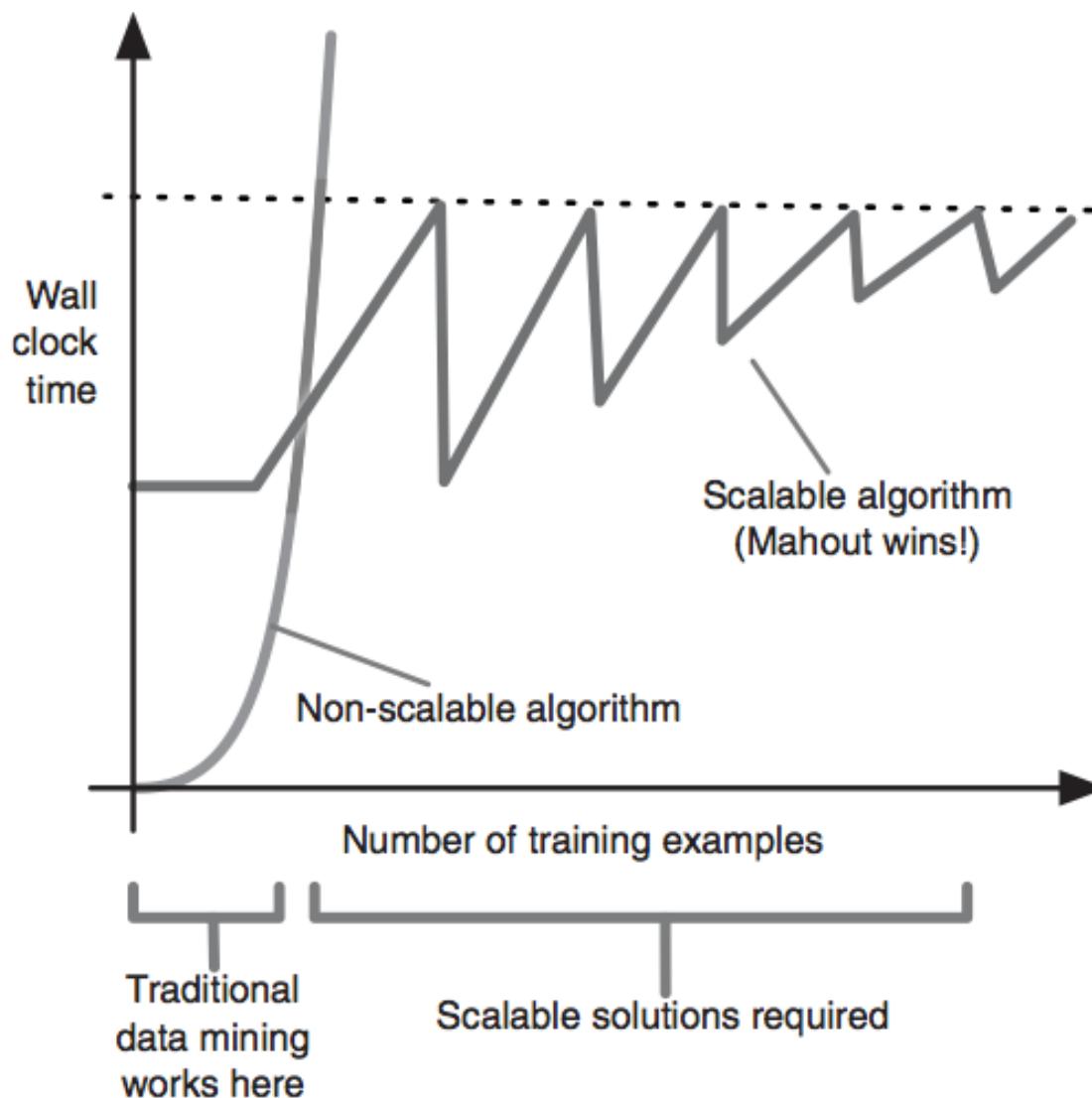
ML – Support Vector Machine



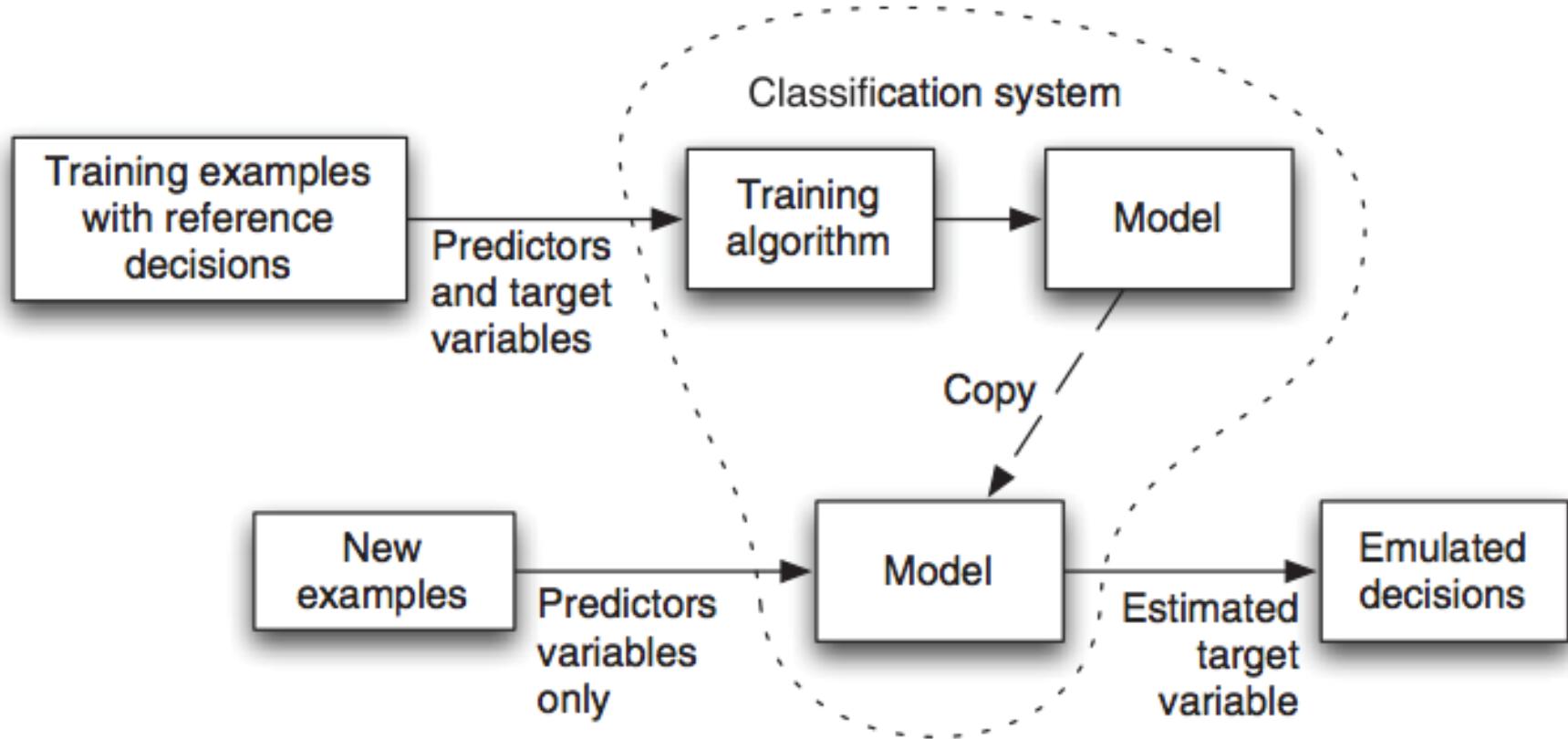
When to use Big Data System for classification?

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



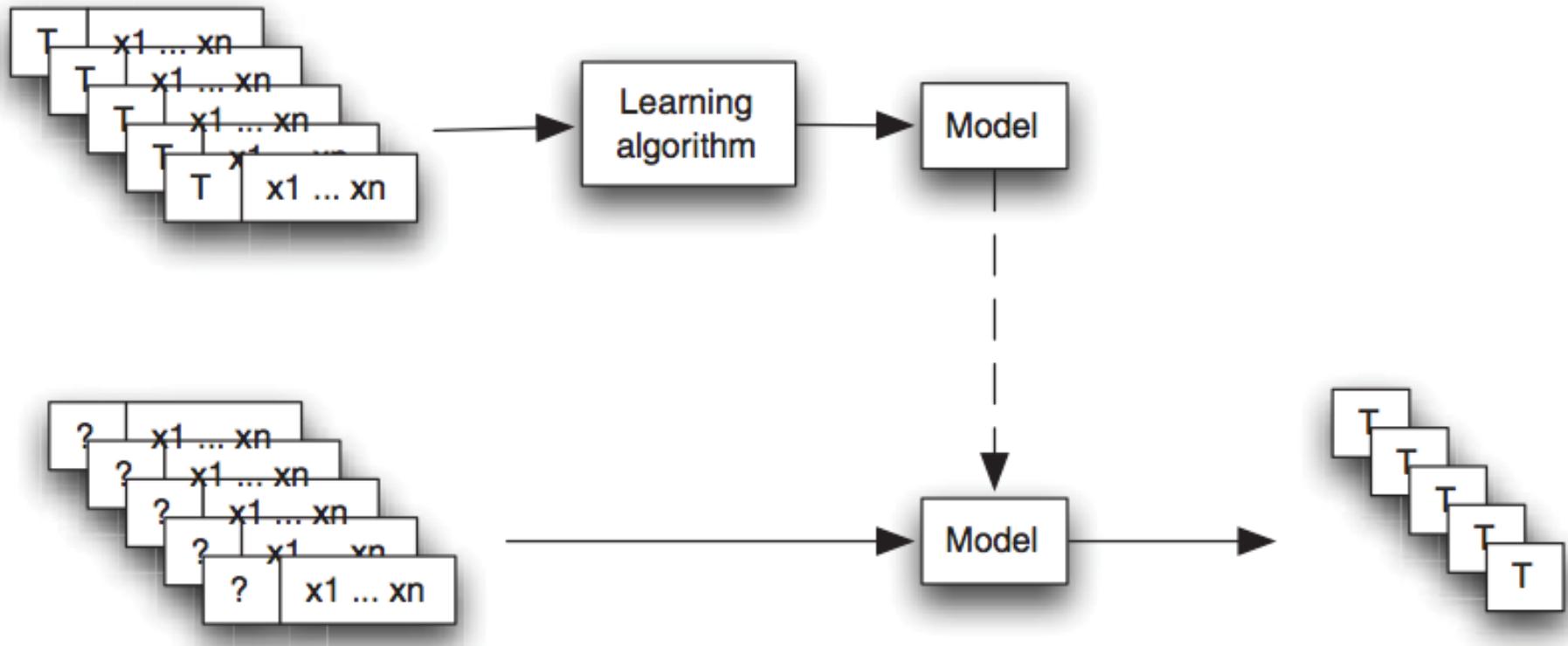
How does a classification system work?



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 <i>variable</i> 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

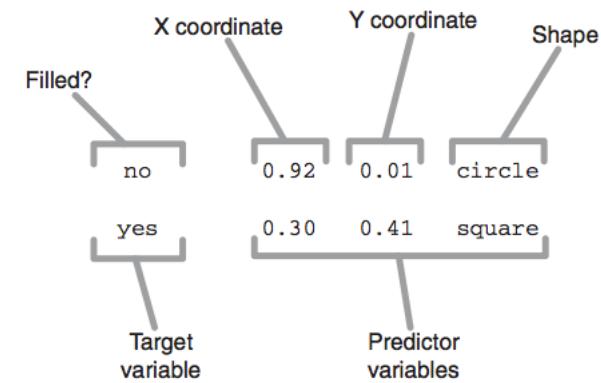
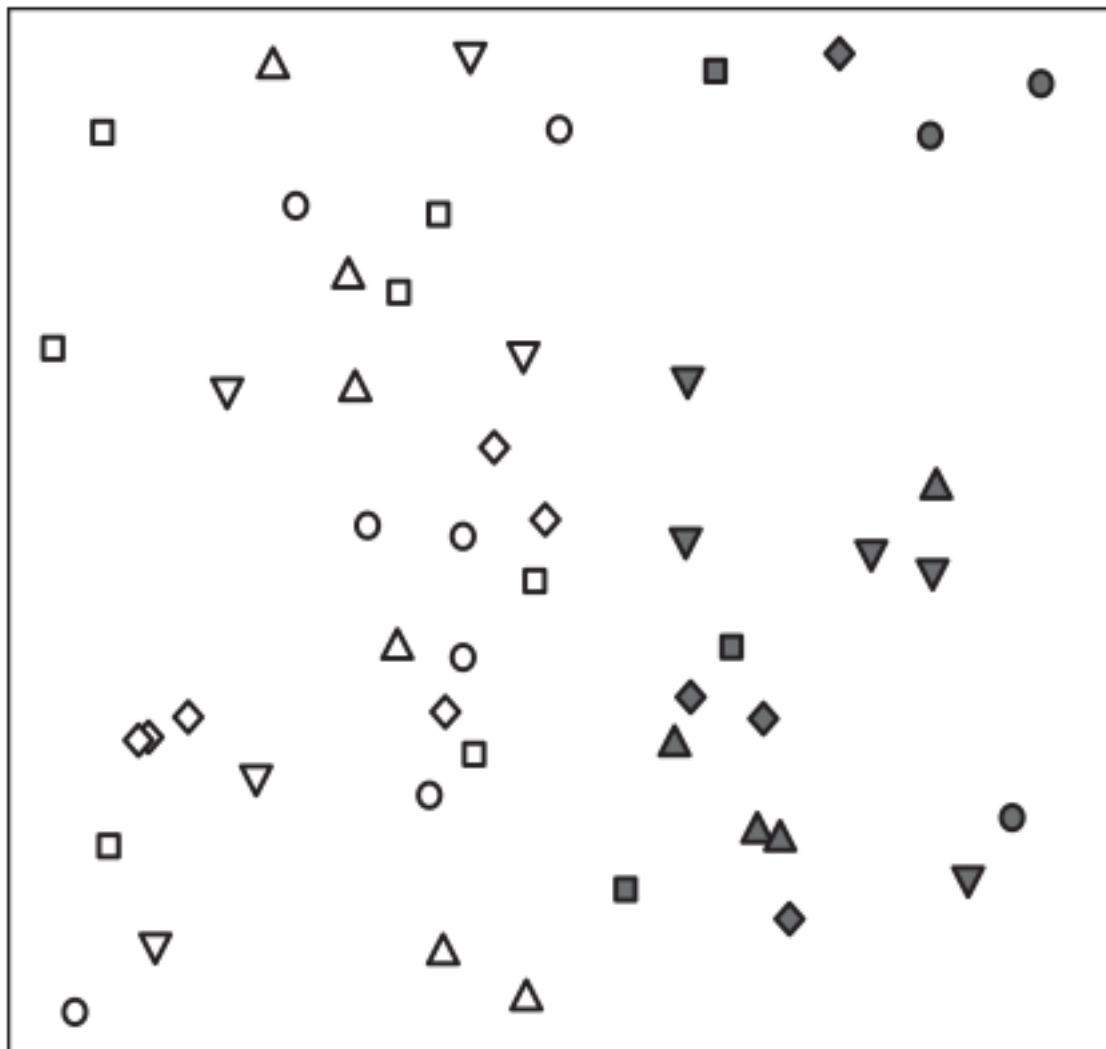
Classification algorithms are related to, but still quite different from, clustering algorithms such as the k-means algorithm described in previous chapters. Classification algorithms are a form of supervised learning, as opposed to unsupervised learning, which happens with clustering algorithms. A supervised 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

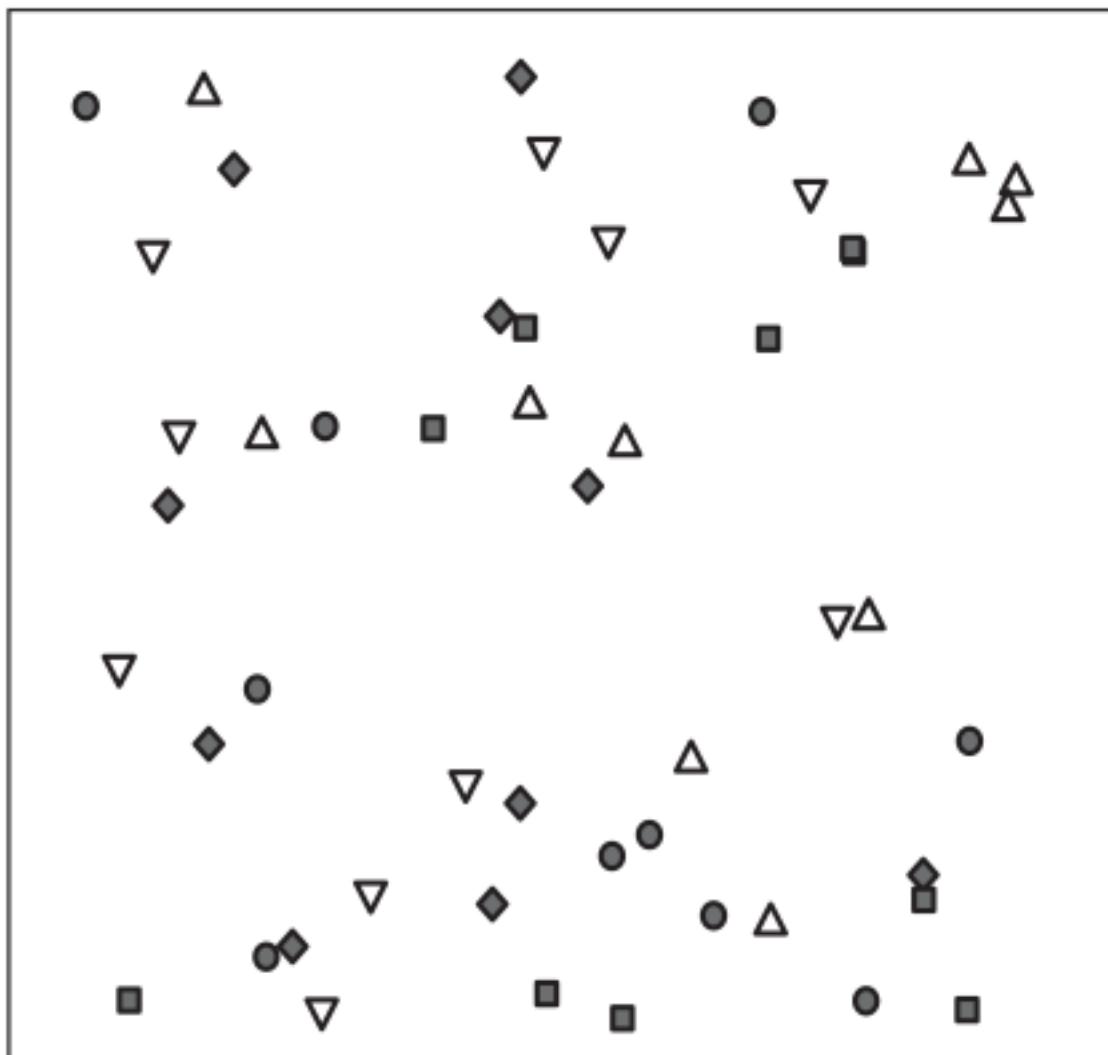
Stage	Step
1. Training the model	Define target variable. Collect historical data. Define predictor variables. Select a learning algorithm. Use the learning algorithm to train the model.
2. Evaluating the model	Run test data. Adjust the input (use different predictor variables, different algorithms, or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

Classification Example 1 — Color-Fill



Position looks promising, especially the x-axis ==> predictor variable.
 Shape seems to be irrelevant. Target variable is “color-fill” label.

Classification Example 2 — Color-Fill (another feature)



Example of fundamental classification algorithms:

- Naive Bayesian
- Complementary Naive Bayesian
- Stochastic Gradient Descent (SGD)
- 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: <code>OnlineLogisticRegression</code> , <code>CrossFoldLearner</code> , <code>AdaptiveLogisticRegression</code>	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 $i = 1, 2, \dots, n$, do:
 - $w := w - \alpha \nabla Q_i(w)$.

Let's suppose we want to fit a straight line $y = w_1 + w_2x$ to a training set of two-dimensional points $(x_1, y_1), \dots, (x_n, y_n)$ using least squares. The objective function to be minimized is:

$$Q(w) = \sum_{i=1}^n Q_i(w) = \sum_{i=1}^n (w_1 + w_2x_i - y_i)^2.$$

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

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} := \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \alpha \left[\begin{array}{c} \sum_{i=1}^n 2(w_1 + w_2x_i - y_i) \\ \sum_{i=1}^n 2x_i(w_1 + w_2x_i - y_i) \end{array} \right].$$

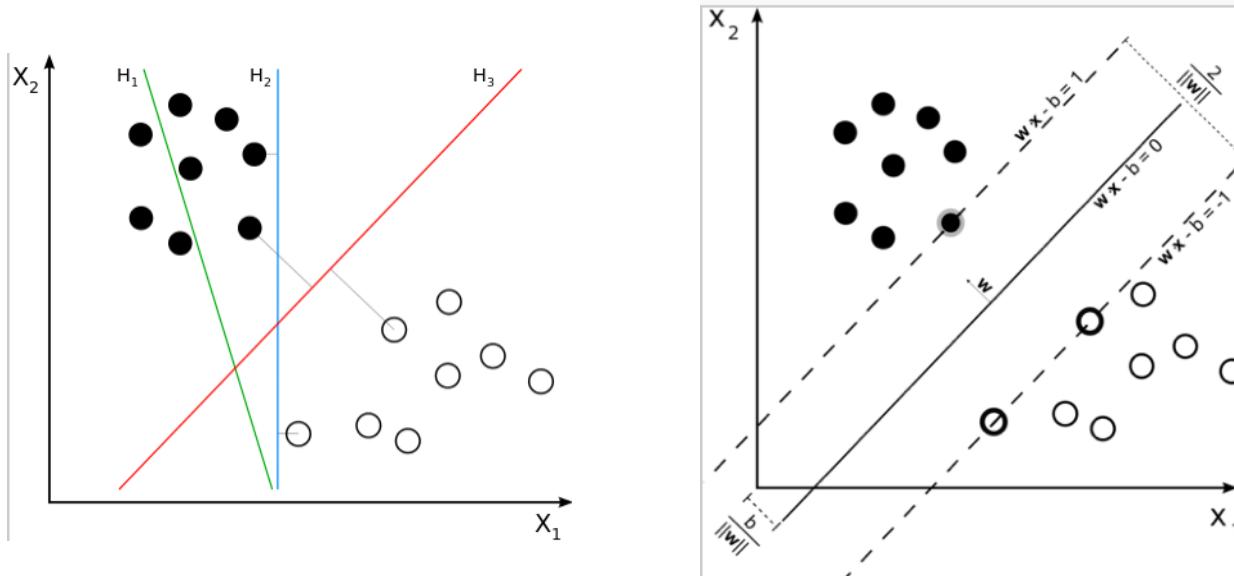
Characteristic of SGD

THE SGD ALGORITHM

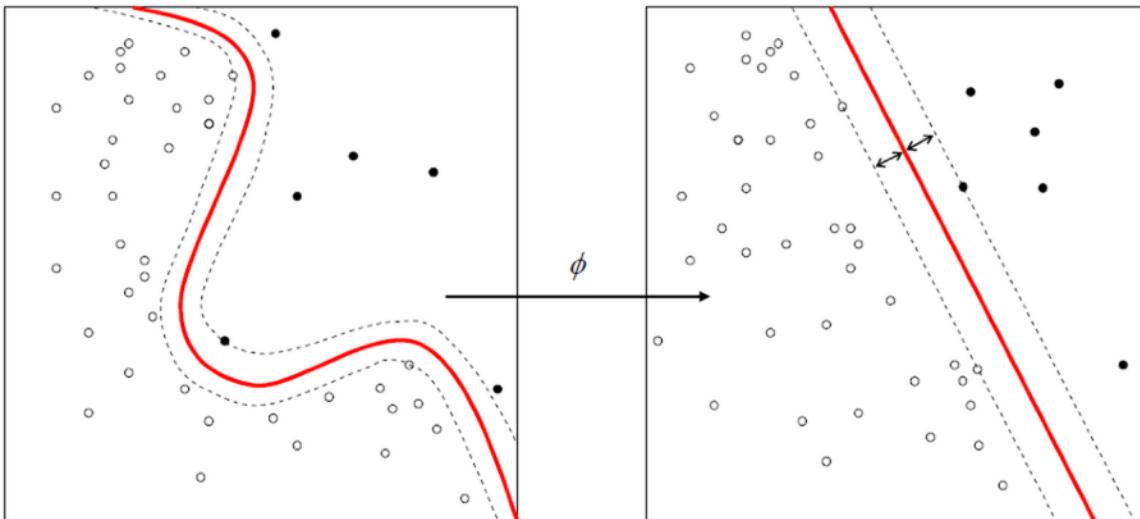
Stochastic gradient descent (SGD) is a widely used learning algorithm in which each training example is used to tweak the model slightly to give a more correct answer for that one example. This incremental approach is repeated over many training examples. With some special tricks to decide how much to 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

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

Test Set:

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

$$\text{evidence} = P(\text{male}) p(\text{height|male}) p(\text{weight|male}) p(\text{footsize|male}) + P(\text{female}) p(\text{height|female}) p(\text{weight|female}) p(\text{footsize|female})$$

$$P(\text{male}) = 0.5$$

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

$$p(\text{weight|male}) = 5.9881 \cdot 10^{-6}$$

$$p(\text{foot size|male}) = 1.3112 \cdot 10^{-3}$$

$$\text{posterior numerator (male)} = \text{their product} = 6.1984 \cdot 10^{-9}$$

$$P(\text{female}) = 0.5$$

$$p(\text{height|female}) = 2.2346 \cdot 10^{-1}$$

$$p(\text{weight|female}) = 1.6789 \cdot 10^{-2}$$

$$p(\text{foot size|female}) = 2.8669 \cdot 10^{-1}$$

$$\text{posterior numerator (female)} = \text{their product} = 5.3778 \cdot 10^{-4}$$

=> 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 $X = x_1, \dots, x_n$ 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.

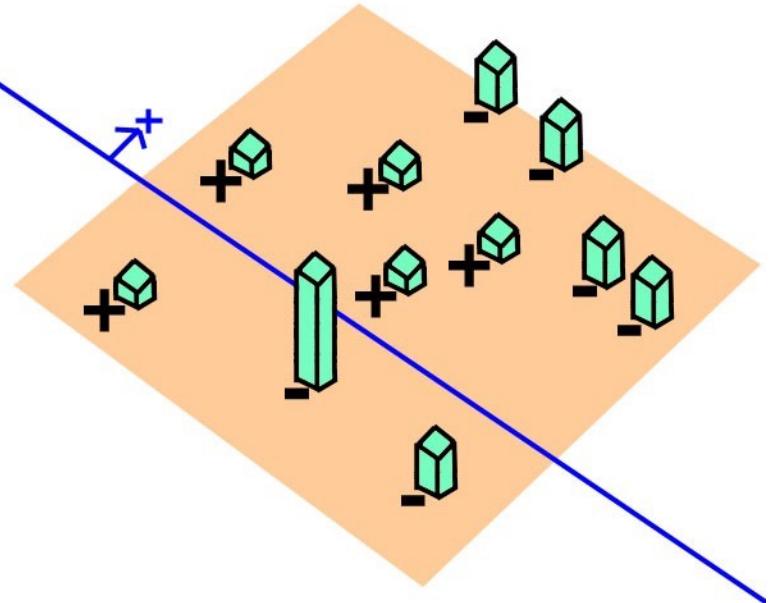
Random forest uses a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features.

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

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



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: $(x_1, c_1, t_1), \dots, (x_N, c_N, t_N)$ 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 $s = 1, \dots, S$

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

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

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

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

where Z_s is a normalization term.

End

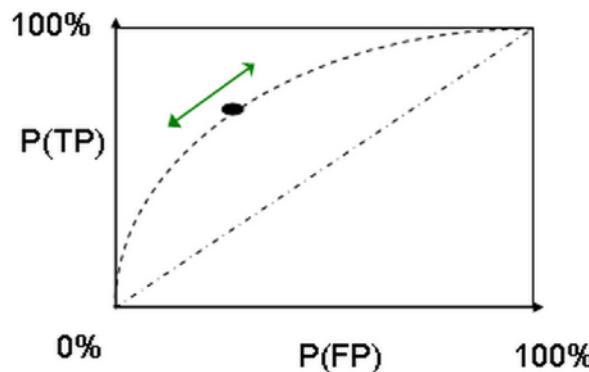
Find weak hypothesis by: $h_s = \arg \min_{h_j \in H} \varepsilon_j$.

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

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

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));           ← Counts the pairs
        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());
    }
}
```

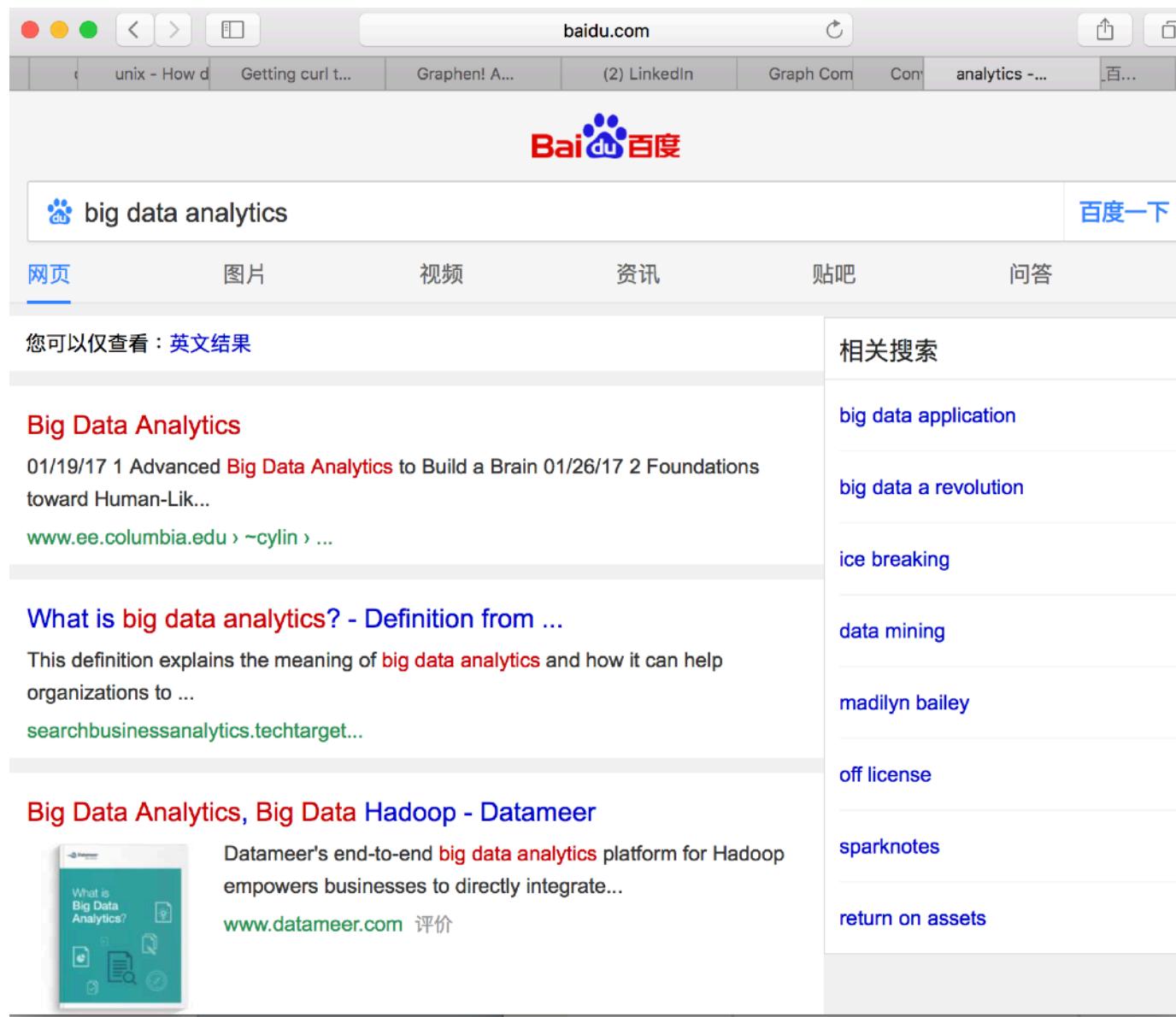
Reads and remembers values

Counts the pairs

Input contains target and model output

x2 gets true answer and score

Average Precision — commonly used in sorted results



The screenshot shows a Baidu search results page for the query "big data analytics". The results are categorized into several tabs: 网页 (Web Pages), 图片 (Images), 视频 (Videos), 资讯 (News), 贴吧 (Baidu Tieba), and 问答 (Answers). The "网页" tab is selected. The first result is a link to a Columbia University page about Big Data Analytics. The second result is a definition from TechTarget. The third result is a Datameer advertisement. On the right side of the page, there is a sidebar titled "相关搜索" (Related Searches) which lists several terms in blue: big data application, big data a revolution, ice breaking, data mining, madilyn bailey, off license, sparknotes, and return on assets.

'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,
 \Rightarrow 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.

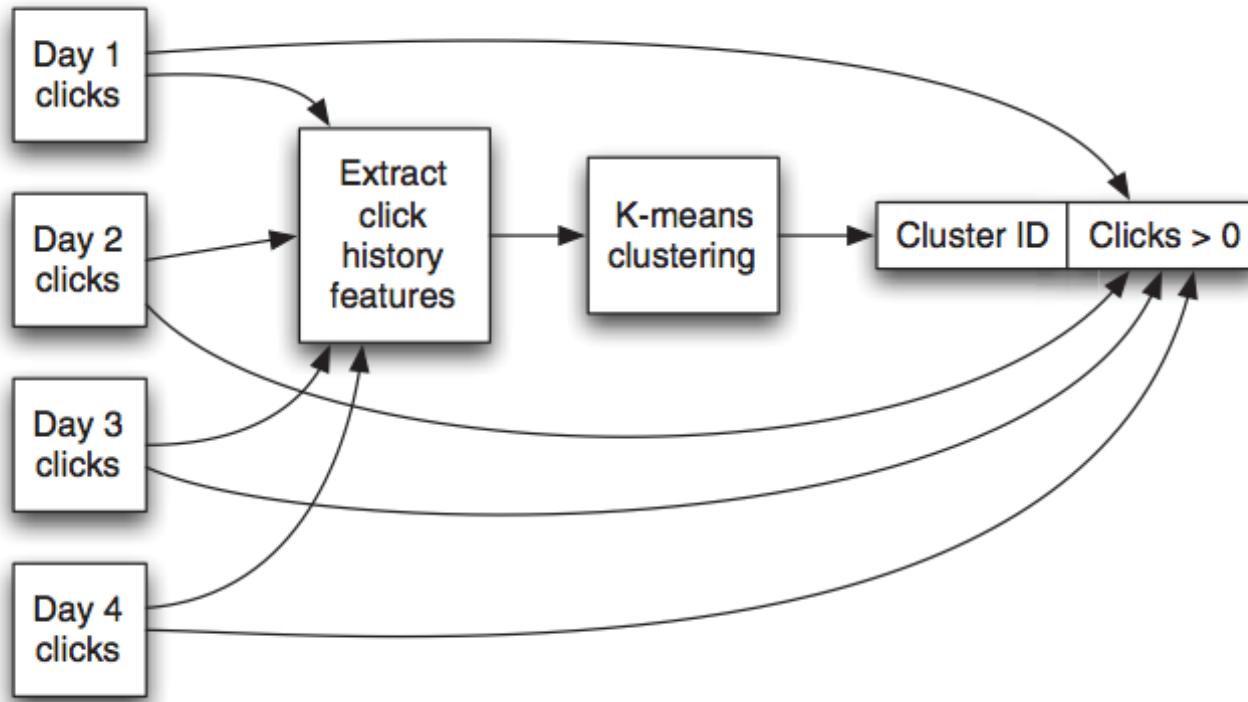
Example: Target Leak

```
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);
```

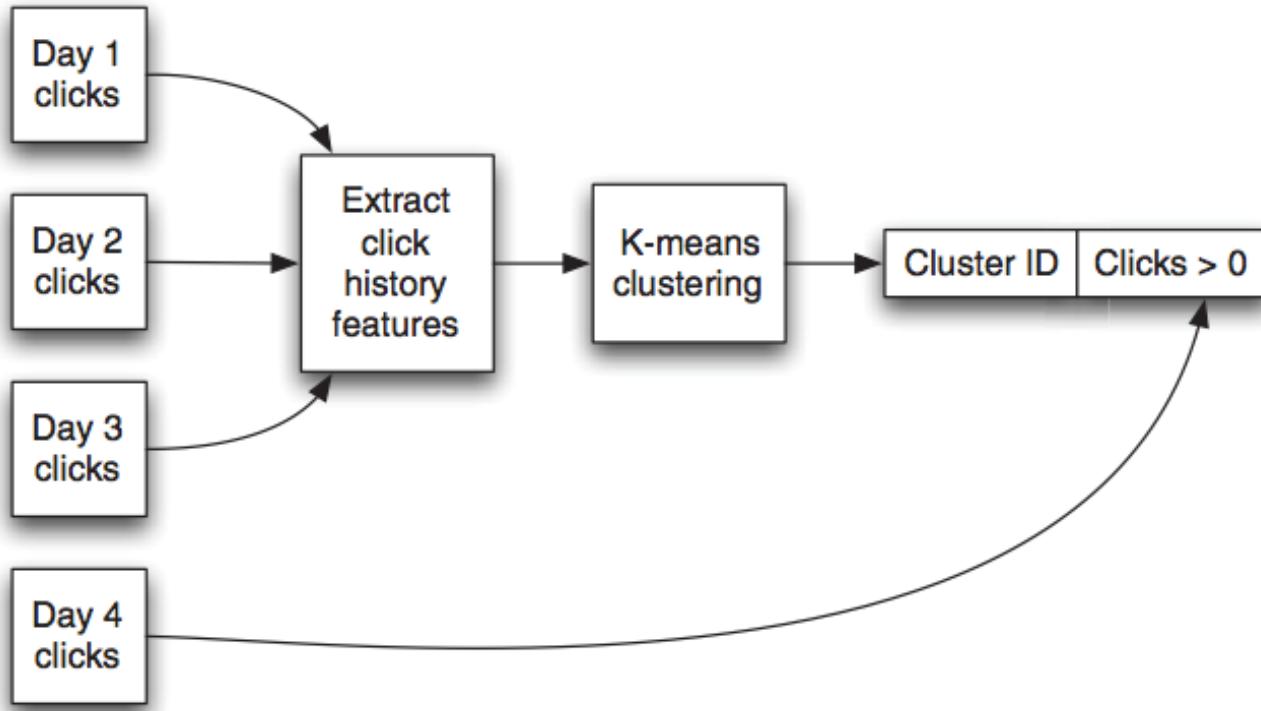
This date field is chosen so that all the documents from the same newsgroup appear to have come from the same month, but documents from different newsgroups come from different months.

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

*PipelineModel
(Transformer)*

Tokenizer

HashingTF

Logistic
Regression
Model

*PipelineModel
.transform()*



Raw
text

Words

Feature
vectors

Predictions

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

Examples

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

id	raw
0	[I, saw, the, red, balloon]
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, balloon]	[saw, red, balloon]
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'

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 of 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`:

<code>id</code>	<code>texts</code>
---	-----
0	<code>Array("a", "b", "c")</code>
1	<code>Array("a", "b", "b", "c", "a")</code>

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:

<code>id</code>	<code>texts</code>	<code> vector</code>
---	-----	-----
0	<code>Array("a", "b", "c")</code>	<code> (3,[0,1,2],[1.0,1.0,1.0])</code>
1	<code>Array("a", "b", "b", "c", "a")</code>	<code> (3,[0,1,2],[2.0,2.0,1.0])</code>

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)
```

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	<code>((262144, [51871, 63643, 174475, 253195], [1.0, 1.0, 2.2, 1.0]))</code>
3.3	false	2	bar	<code>((262144, [6031, 80619, 140467, 174475], [1.0, 1.0, 1.0, 3.3]))</code>
4.4	false	3	baz	<code>((262144, [24279, 140467, 174475, 196810], [1.0, 1.0, 4.4, 1.0]))</code>
5.5	false	4	foo	<code>((262144, [63643, 140467, 168512, 174475], [1.0, 1.0, 1.0, 5.5]))</code>

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

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

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

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 (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						

```
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"])

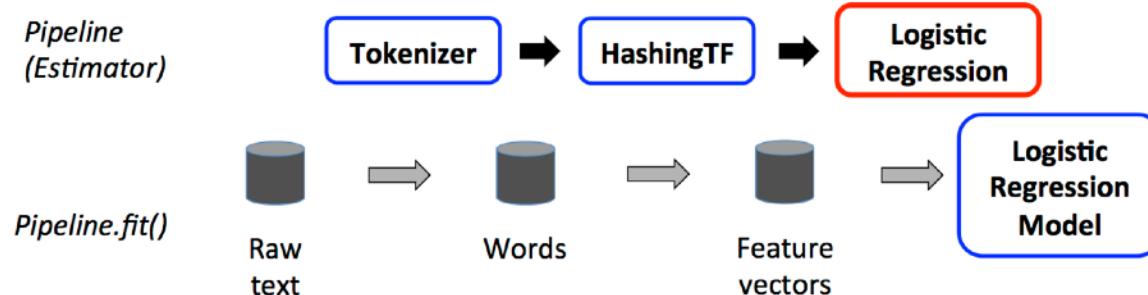
tokenizer = Tokenizer(inputCol="sentence", outputCol="words")
wordsData = tokenizer.transform(sentenceData)

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



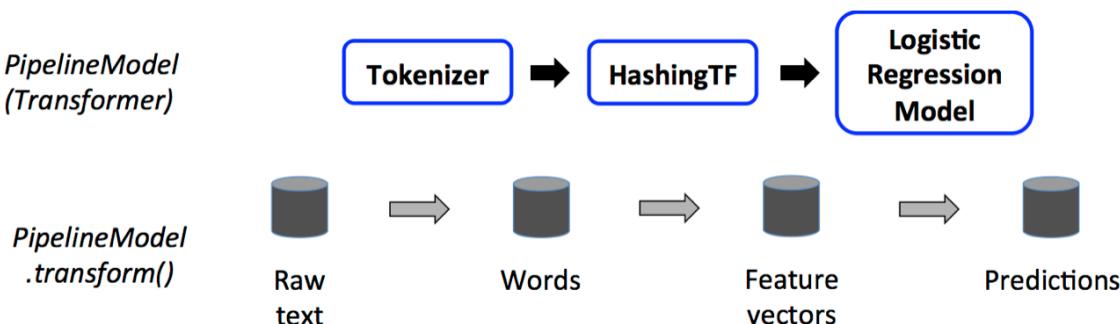
```

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])
  
```

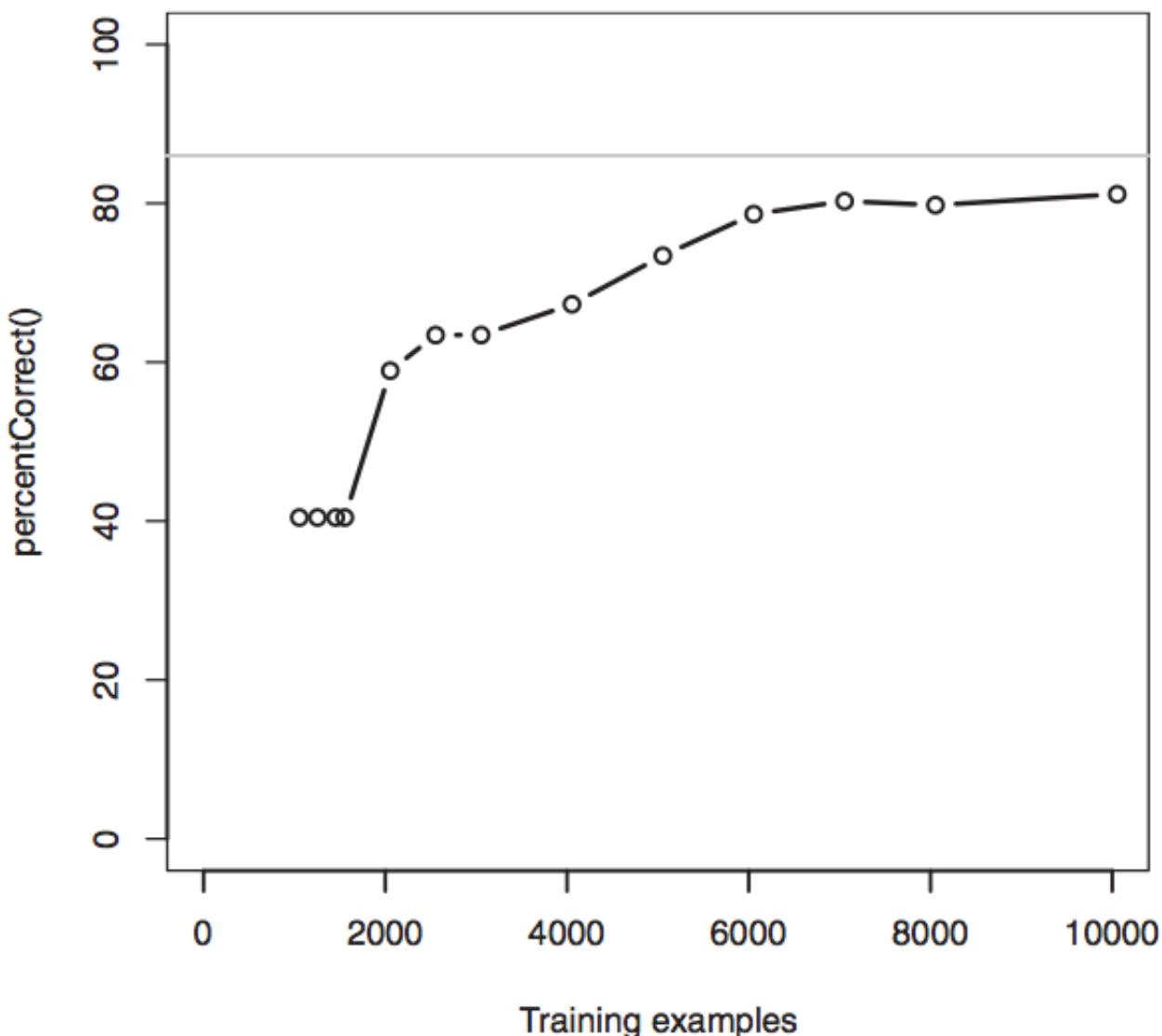
Spark ML Pipeline Example Code II



```
# 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.

Imperfect Learning for Autonomous Concept Modeling Learning

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

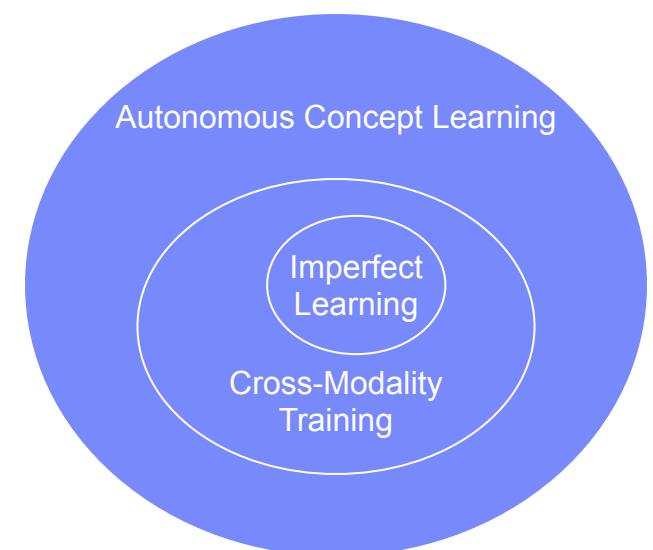
A solution for the scalability issues at training..

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.

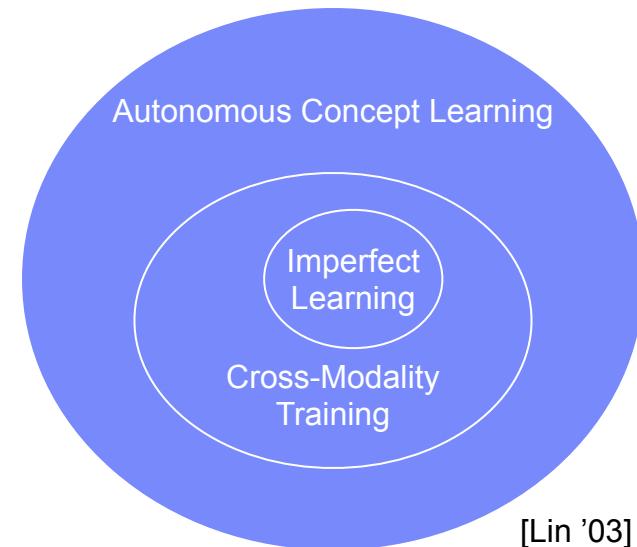
All (or almost all?) modeling/fusion techniques in our group used annotation for training
However, annotation is time- and cost- consuming.

Previous focuses were on improving the annotation efficiency – minimum GUI interaction, template matching, active learning, etc.

Is there a way to avoid annotation?

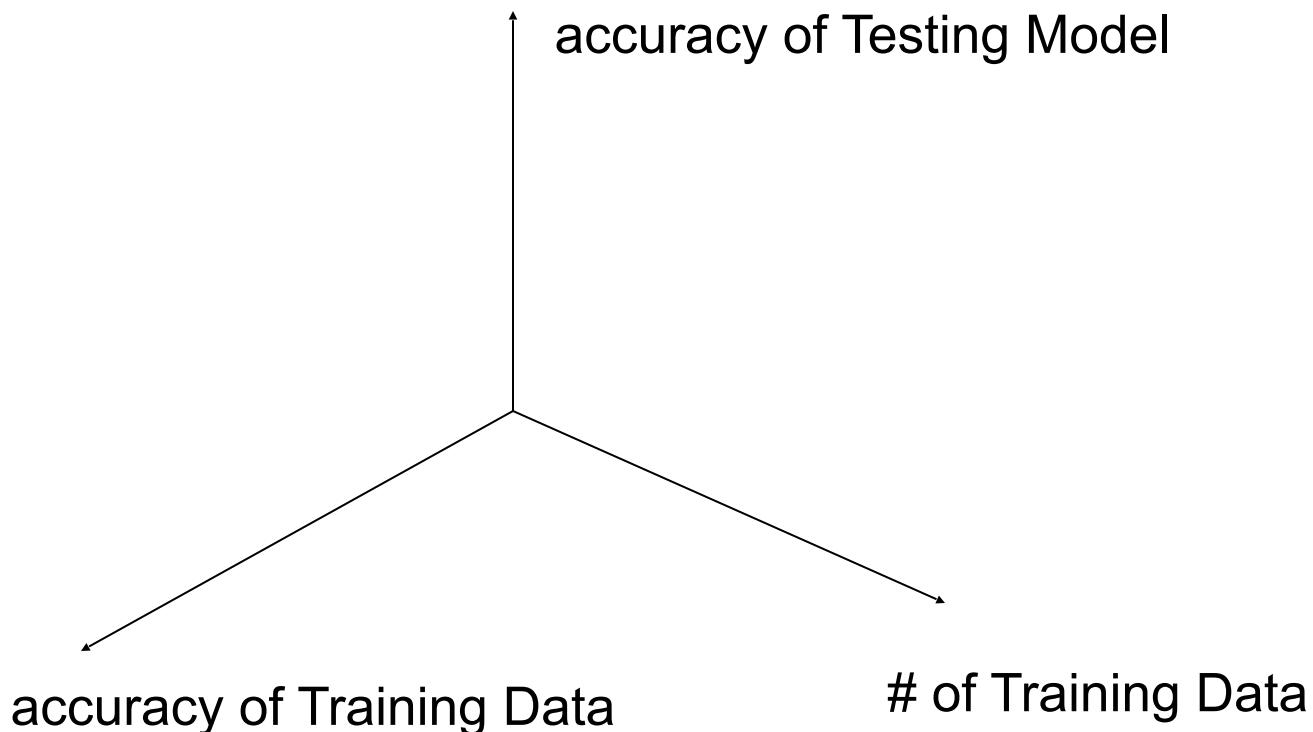
Use imperfect training examples that are obtained automatically/unsupervisedly from other learning machine(s).

These machines can be built based on other modalities or prior machines on related dataset domain.



Proposition

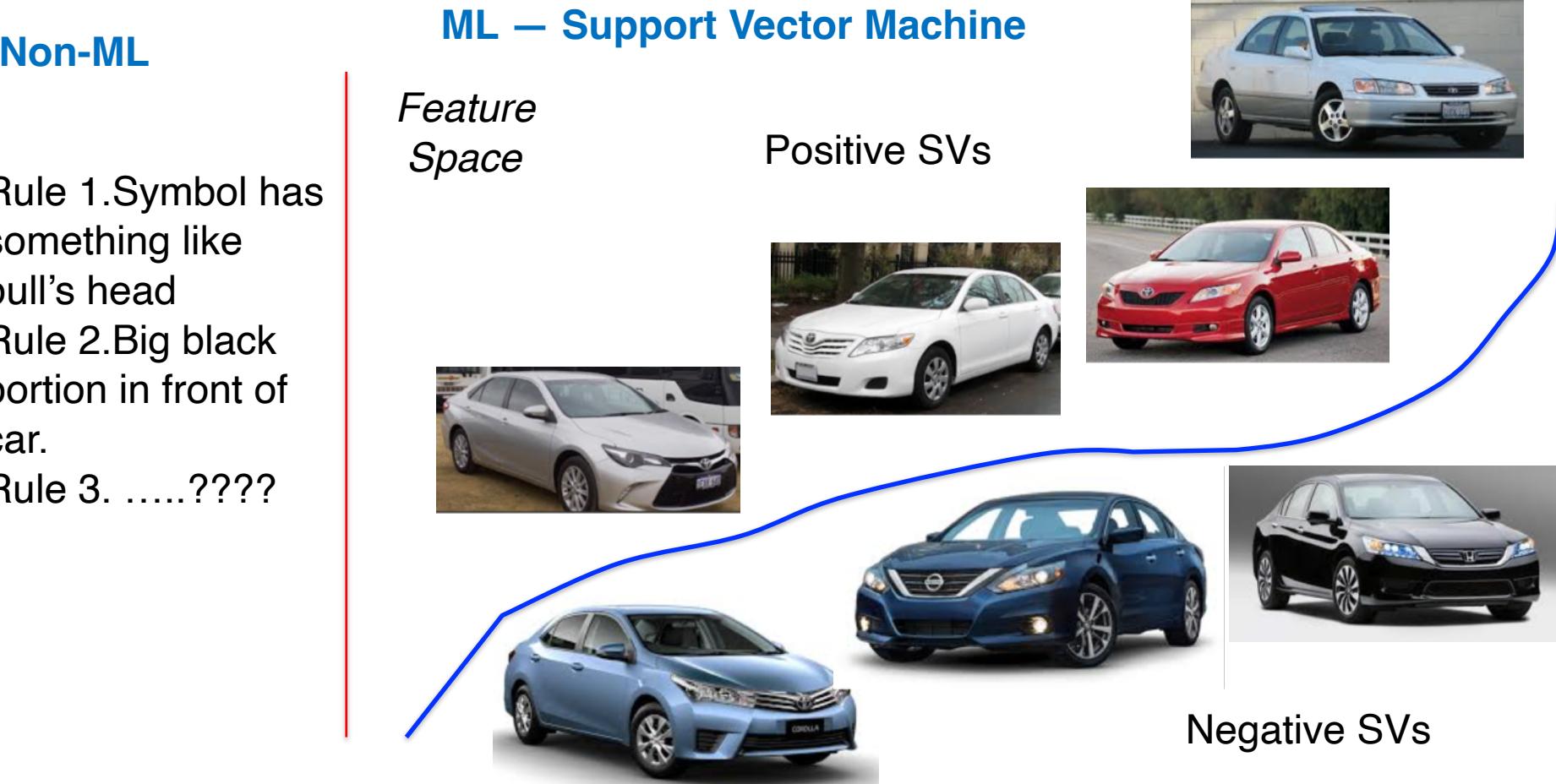
Supervised Learning ← Time consuming; Spend a lot of time to do the annotation
Unsupervised continuous learning ← When will it beat the supervised learning?



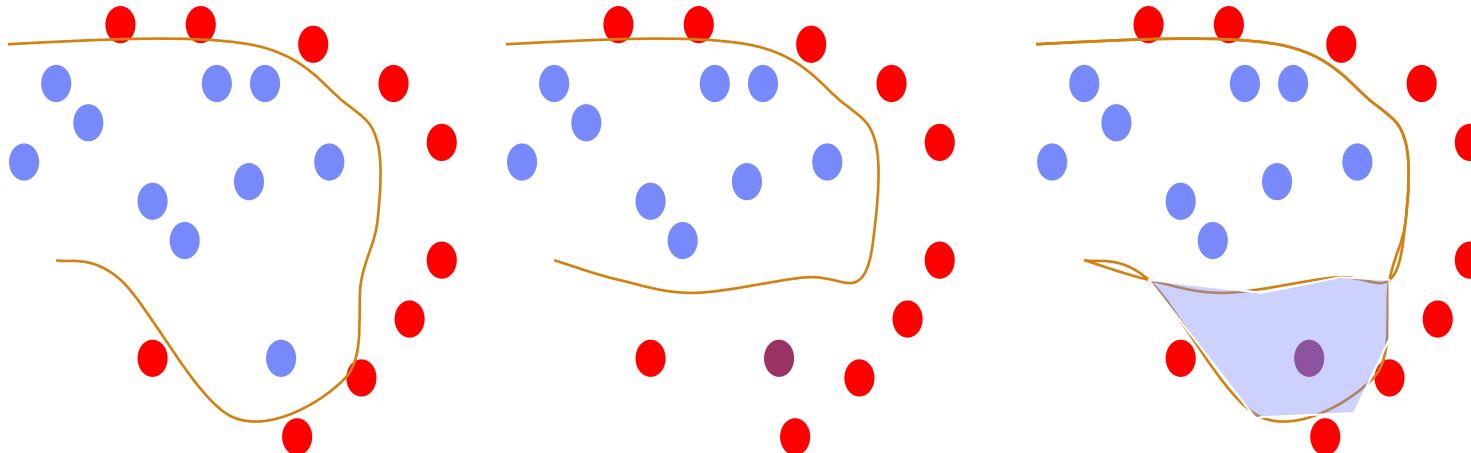
Recap: Classification example: using SVM to recognize a Toyota Camry

Non-ML

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



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

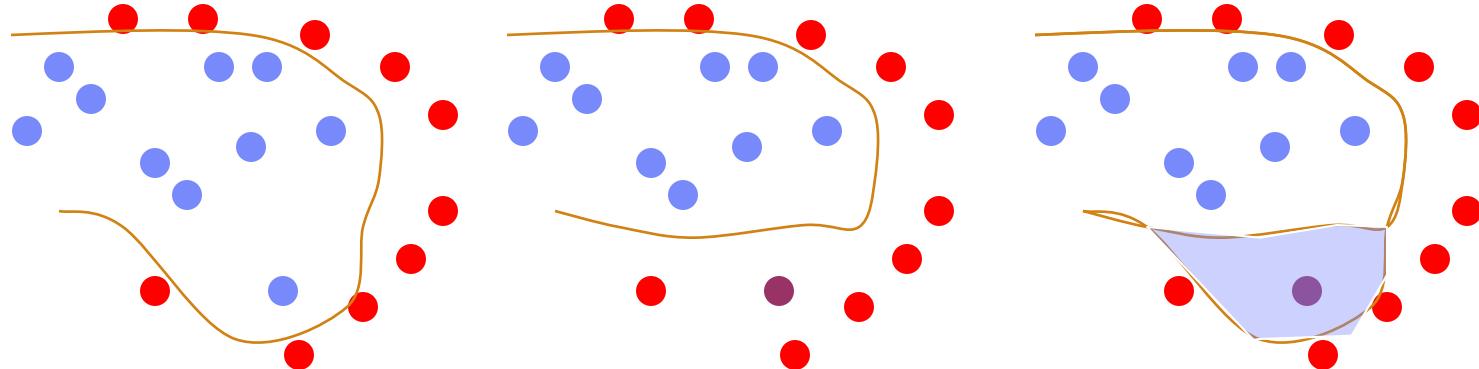
Assume we have ten positive examples and ten negative examples. if 1 positive example is wrong (false positive), how will it affect SVM? Will the system break down? Will the accuracy decrease significantly?

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

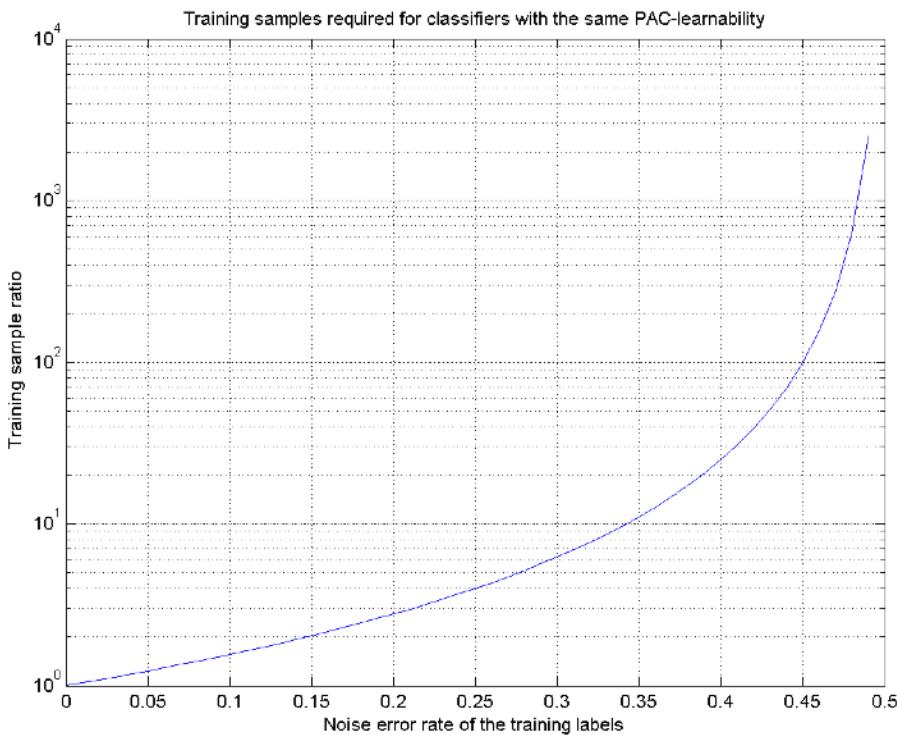
- From Hessienberg's Uncertainty Theory, everything is random. It is not measurable. Thus, we can assume a random distribution of positive ones and negative ones.
- Assume there are two Gaussians in the feature space. One is positive. The other one is negative.
- Let's assume two situations. The first 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 use 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 makes it possible to adapt and be 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 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 $1-\delta$ 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 $(1-\varepsilon)^m$ which we want to be less than δ . In other words, we want $(1-\varepsilon)^m \leq \delta$. Since for any $0 <= x < 1$, $(1-x) \leq e^{-x}$, we then have:

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

Sample Size v.s. VC dimension

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

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

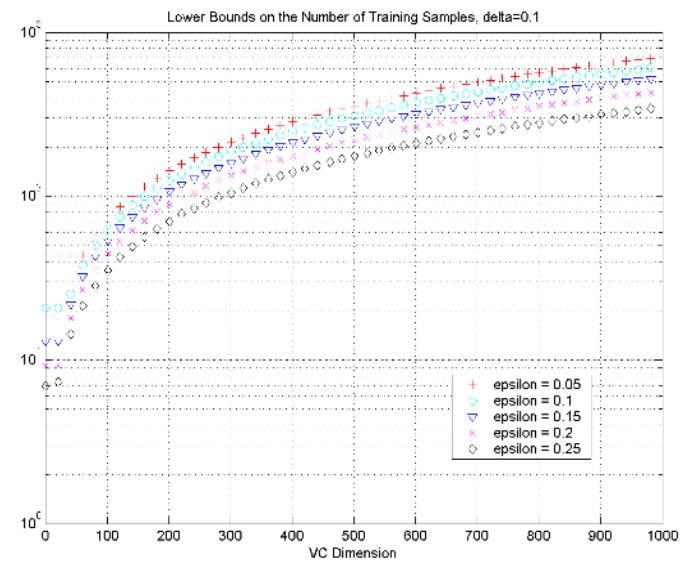
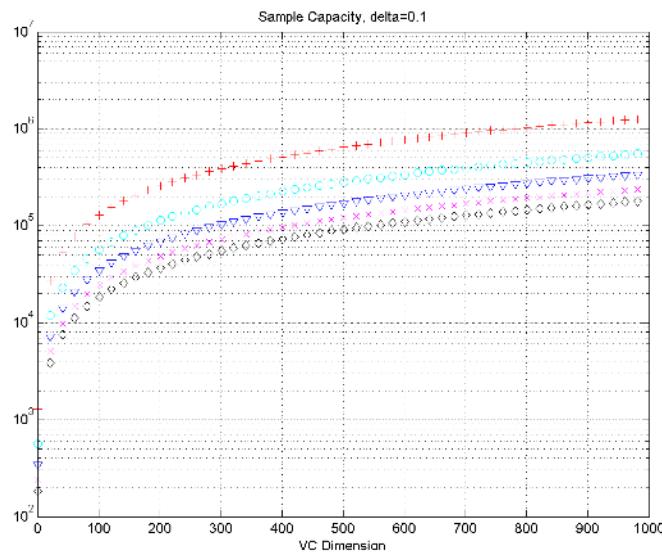
any consistent function $A: Sc\mathbf{C}$ is a learning function for \mathbf{C} , and, for $0 < \epsilon < 1/2$, m has to be larger than or equal to a lower bound,

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

For any m smaller than the lower bound, there is no function $A: Sc\mathbf{H}$, for any hypothesis space \mathbf{H} , is a learning function for \mathbf{C} . The sample space of \mathbf{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 referred as sample capacity, which guarantees the learnability of training samples.



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

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

and at most

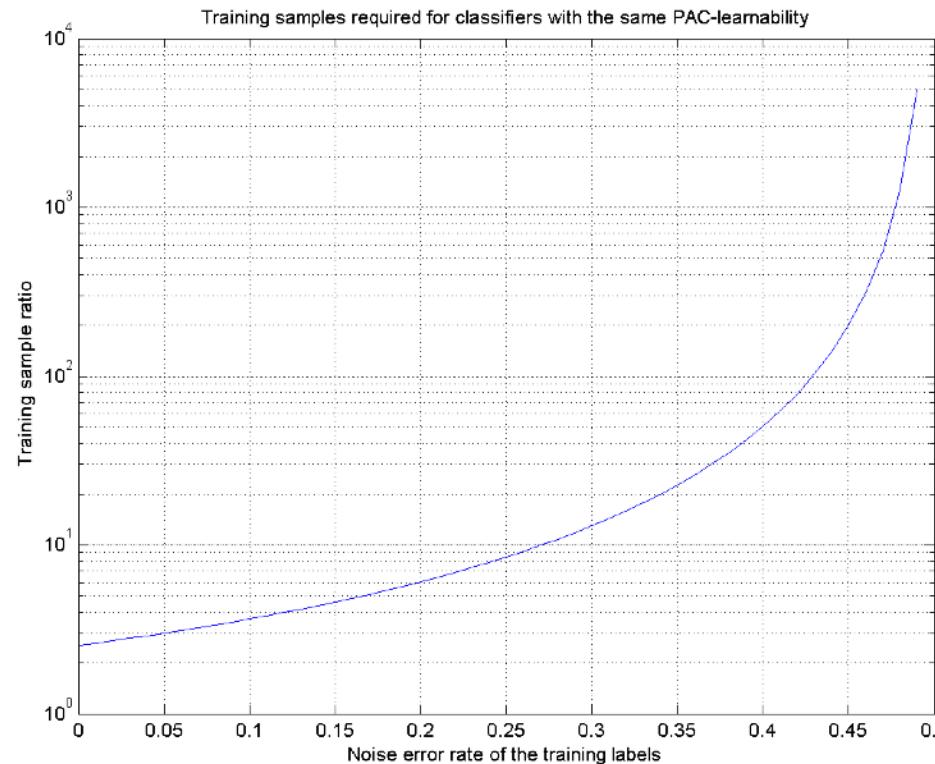
$$\frac{\ln(N/\delta)}{\epsilon \cdot (1 - \exp(-\frac{1}{2}(1 - 2\eta)^2)))}$$

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

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

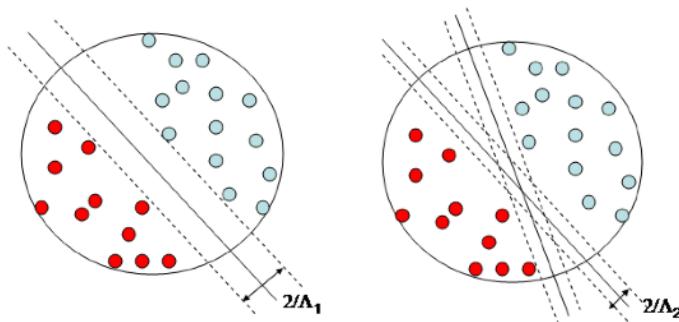
Training samples required when learning from noisy examples

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

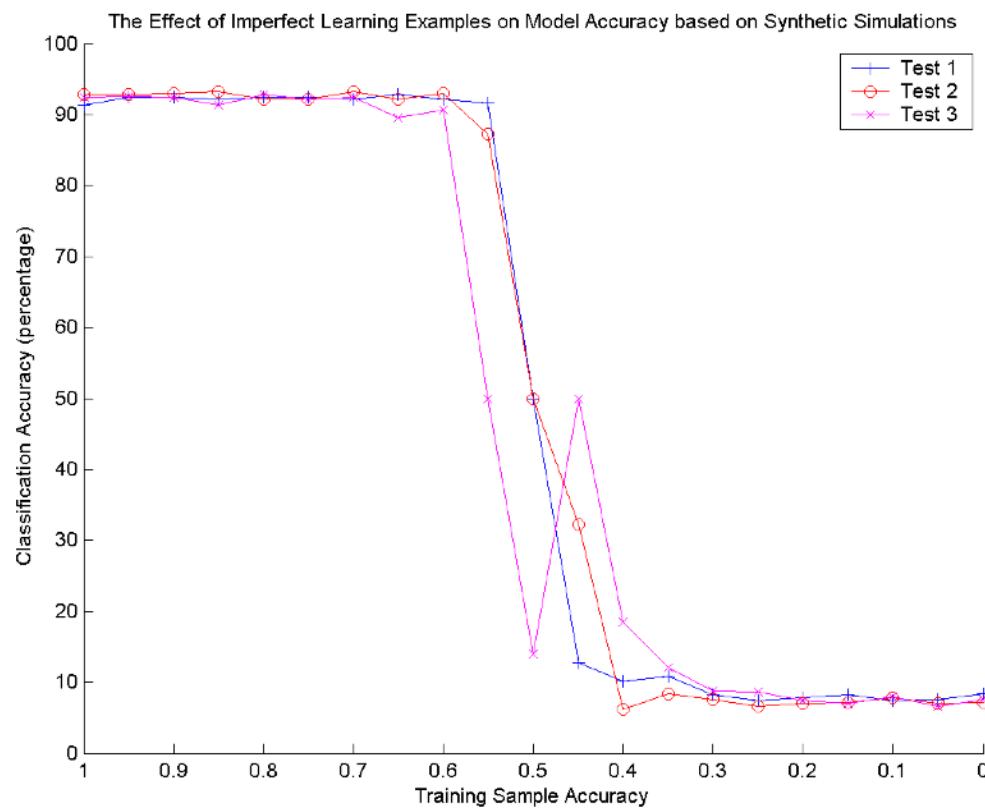
For an SVM, we can find the bounded VC dimension:



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

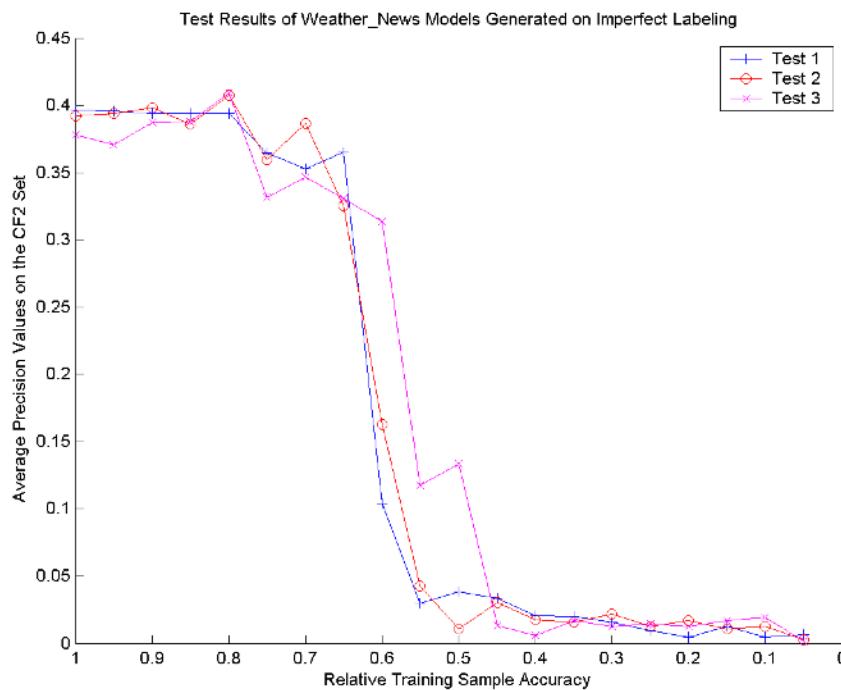
Experiments - 1

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).



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.

Homework #1 (Due 10/6/2023, 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 (11/3/23) — 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



Graphen Core

Full-brain AI Platform and Knowledge Agents empower leaders across industries.



Graphen Finance

Utilize AI to predict risks, monitor operations, and find leads.



Graphen Drugomics

AI understanding and simulating Life Functions to develop drugs.



Graphen Genomics

Making human knowing Biologically Digitized-Self, and enabling Personalized Treatment.



Graphen Automotive

Advanced AI Car Doctor and Assistant.



Graphen Robotics

Smarter AI Machines for Humans.



Graphen Energy

AI helps energy providers realize smart grids with sustainable energy.



Graphen Security

Foundations help organizations with self-defense AI cybersecurity.

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

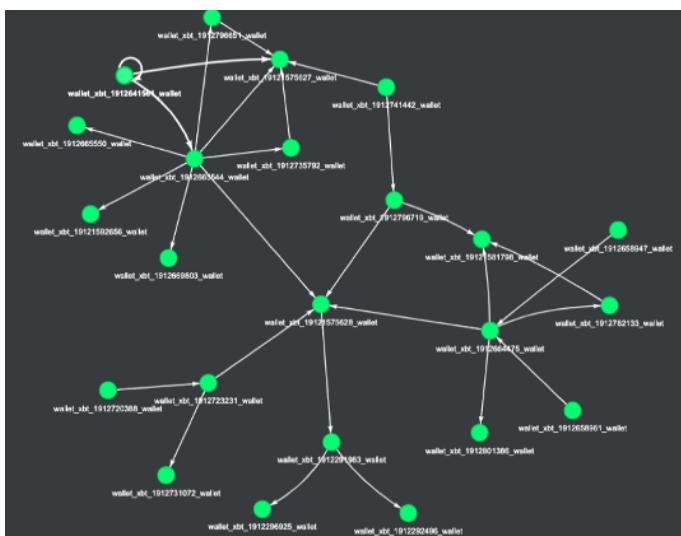


<https://www.graphen.ai/products/Ava.html>

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



防詐高風險交易

系統名稱	案件性質	交易日期	交易網址	帳戶名稱	帳號	交易地點	交易方式	交易類型	（轉賬）對方銀行代碼	（轉賸）對方銀行帳戶	交易金額	交易備註	發送代號	審核階段	審核人員	最後審核時間
FDS	21-1131	2021/05/19 09:40:22	78546823	陳虹輝	18720008078701	TW	本行網銀	轉出	018	181878070982134	180,000		F2	L2審核中	Crystal Wang	2021/05/19 09:42:51
FDS	21-1132	2021/05/19 09:40:32	78546741	胡莎	1843000801015701	TW	銀行網銀	轉入	052	131703201901999	3,000,000	驗修	E2	L2審核中	Crystal Wang	2021/05/19 09:40:52
FDS	21-1133	2021/05/19 10:15:52	78546871	周冰乳	18720008982194	TW	他行網銀	轉出			1,000,000	確認	P2_E6	L2待審核		2021/05/19 10:14:10
FDS_Excellur	21-1134	2021/05/19 09:42:23	78546823	陳虹輝	18720007078701	TW	ATM	對外銀行	009	1261070720007940	500,000	陳虹輝,05	P3_E6	L1審核中	Edison Cheng	2021/05/19 09:43:21
FDS	21-1135	2021/05/19 09:45:51	78547121	王綱名	1223000100079571	TW	本行網銀	轉出			200,000	王綱名,05	E2	L1審核中	Alex Wu	2021/05/19 09:45:24
FDS	21-1136	2021/05/19 11:18:52	78547407	施三信	1810000800007675	TW	他行網銀	轉入			1,000,000	4月	P3_E6	L1審核中	Edison Cheng	2021/05/19 11:21:42
FDS	21-1137	2021/05/19 12:14:47	78547079	王廷威	1430000800007947	TW	ATM	日盛銀行	005		150,000		P3_E6	L1審核中	Alex Wu	2021/05/19 09:40:12
FDS	21-1138	2021/05/19 13:40:02	78547765	廖盈輝	1800000800006887	TW	ATM	海打銀行	005		300,000		P3_E6	尚未作		
FDS	21-1139	2021/05/19 13:40:57	78547070	陳駿昇	1420000300009107	TW	本行網銀	轉出	022	1028479040007397	500,000	陳駿昇	E2	L1審核中	Edison Cheng	2021/05/19 13:55:42
FDS	21-1140	2021/05/19 16:15:02	78547167	張天培	1420000800007662	TW	他行網銀	轉出			2,000,000	張先生	P2_E1	尚未作		

Online Banks

Crypto Currencies



Credit Cards

Area 3 ‘Healthy Life’ Tasks List:

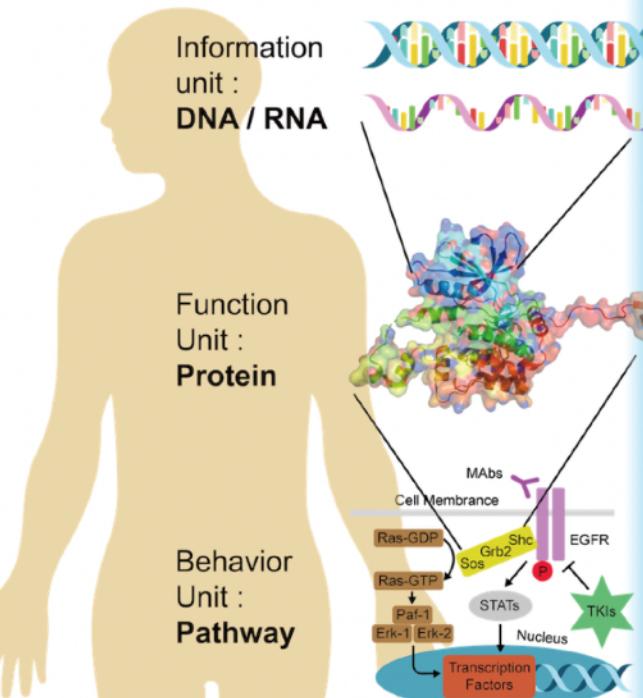
- C1: Precision Health — Gene and Protein Analysis of Network, Pathway, and Biomarkers
- C2: Large-Scale System for Human Genome Analysis
- C3: Secure Patient Data
- C4: Medical Image Analysis
- C5: Drugable Targets for Precision Medicine
- C6: Virus Mutations and Function Prediction
- C7: Microbe and 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

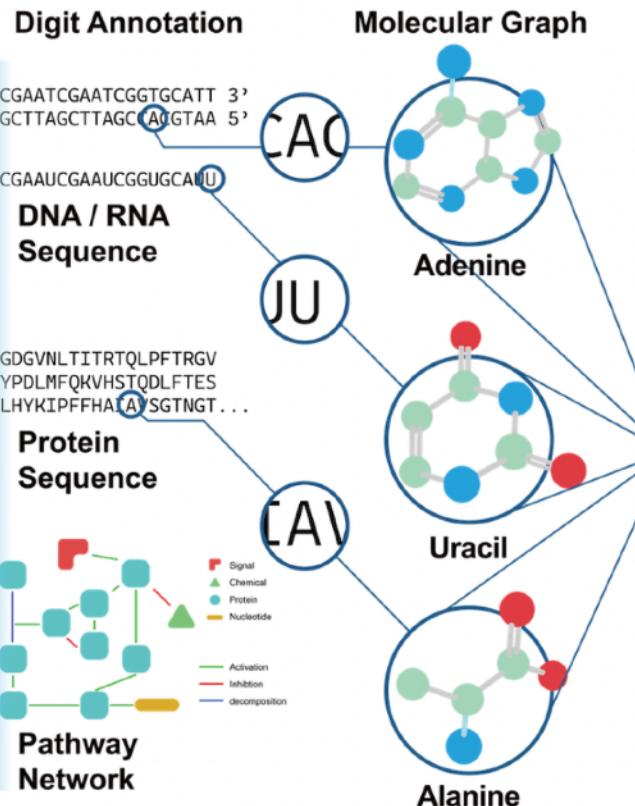
<https://www.graphen.ai/products/atom.html>



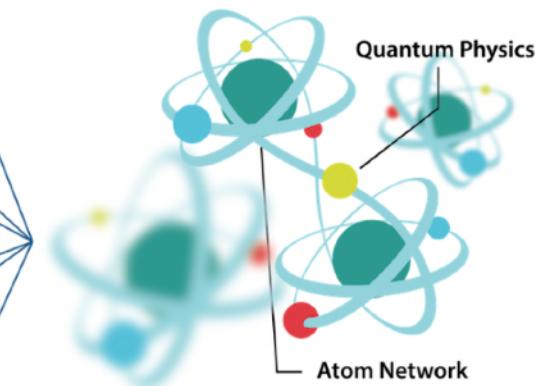
Biological Materials



Digitalize Bioinformation



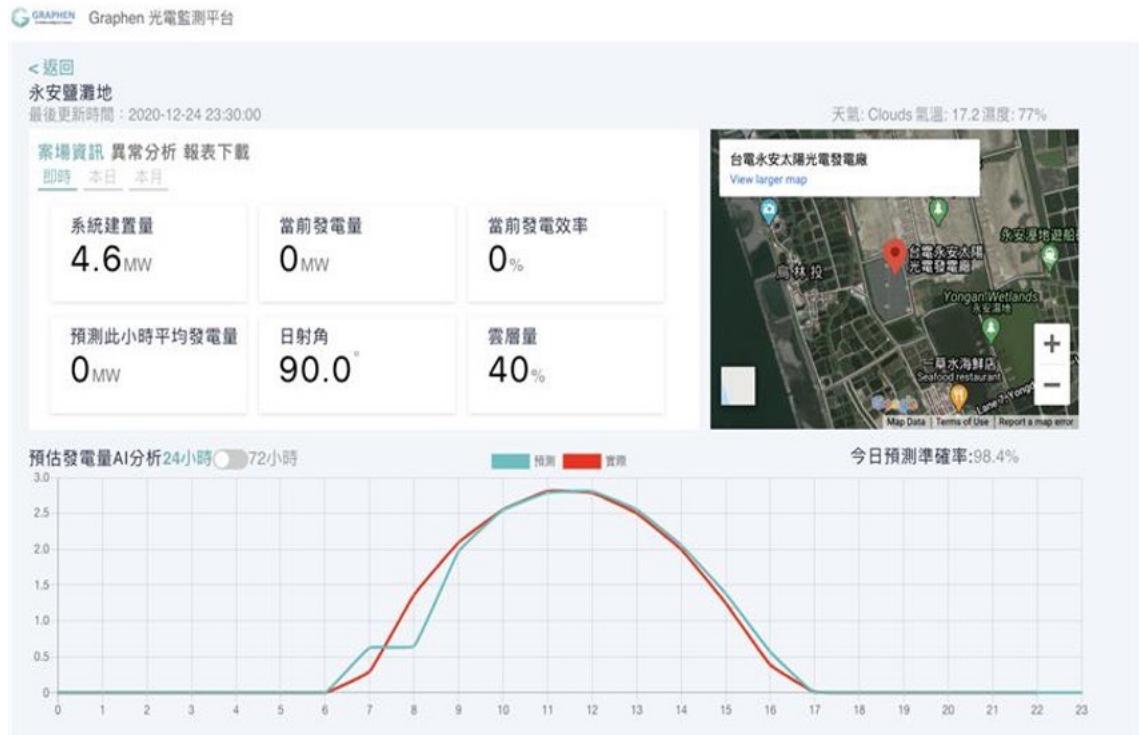
Atomotive Forces



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?