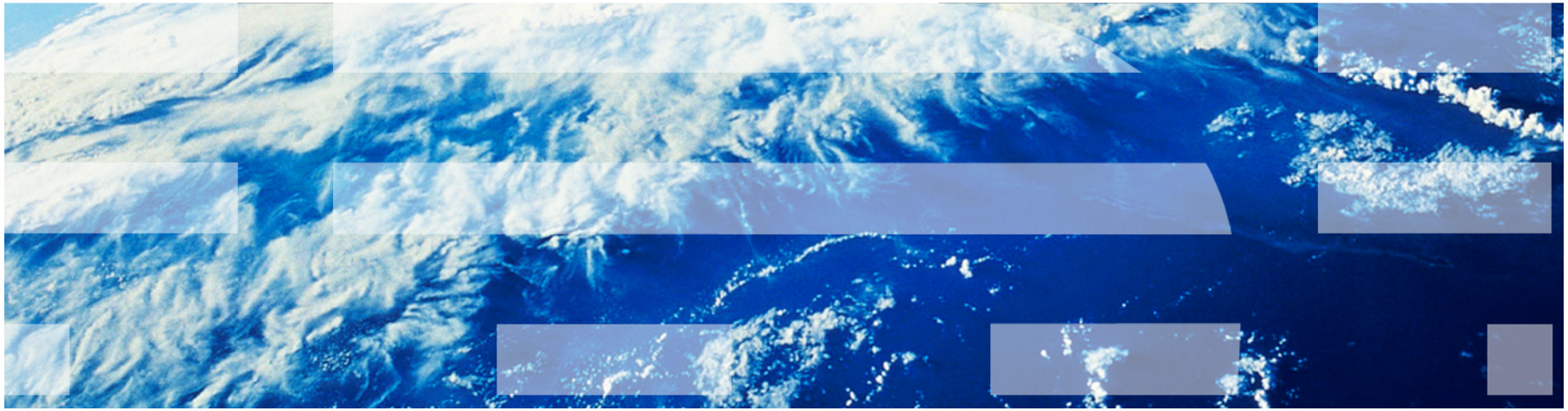
E6893 Big Data Analytics Lecture 6:

Big Data Analytics Algorithms — III

Ching-Yung Lin, Ph.D.

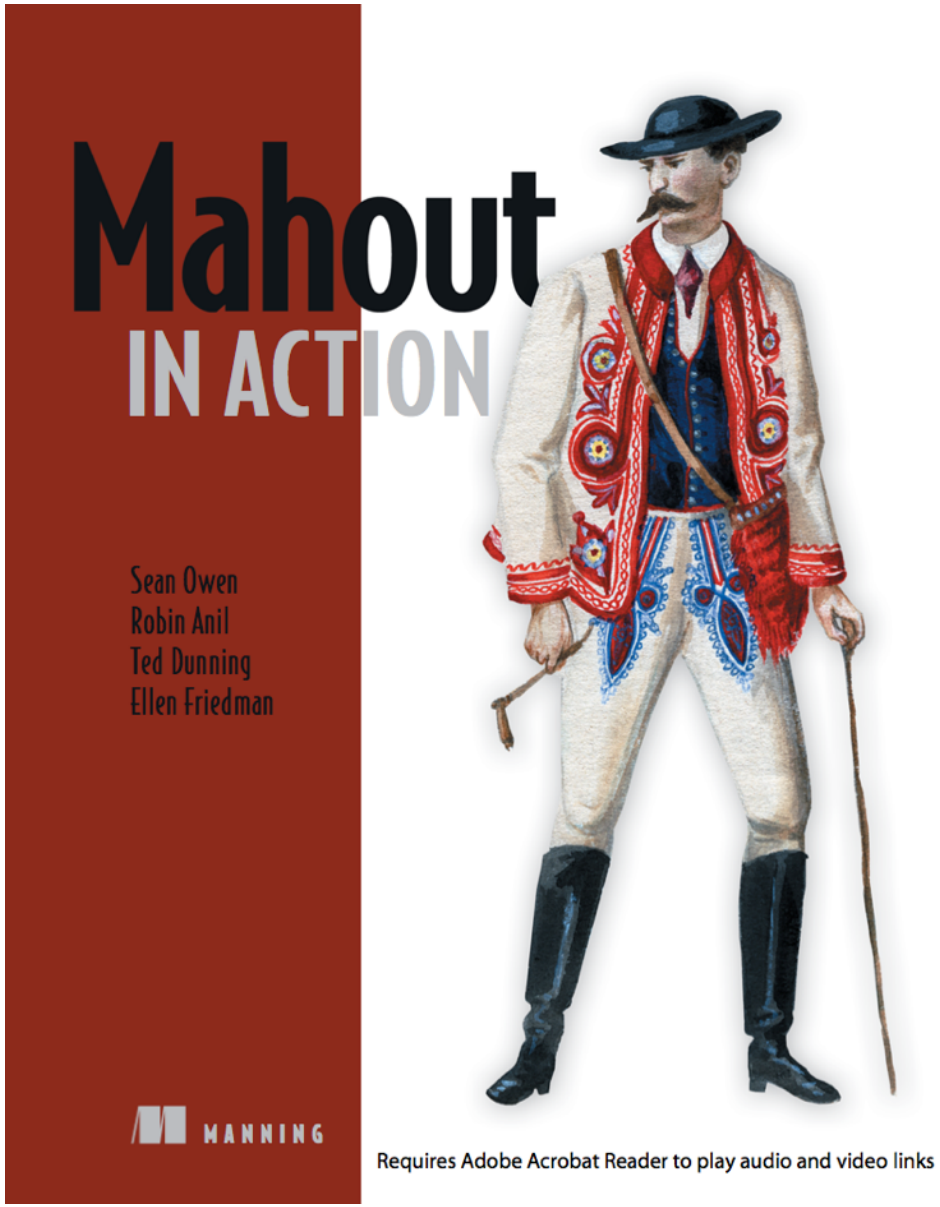
Adjunct Professor, Dept. of Electrical Engineering and Computer Science

IBM Distinguished Researcher and Chief Scientist, Graph Computing



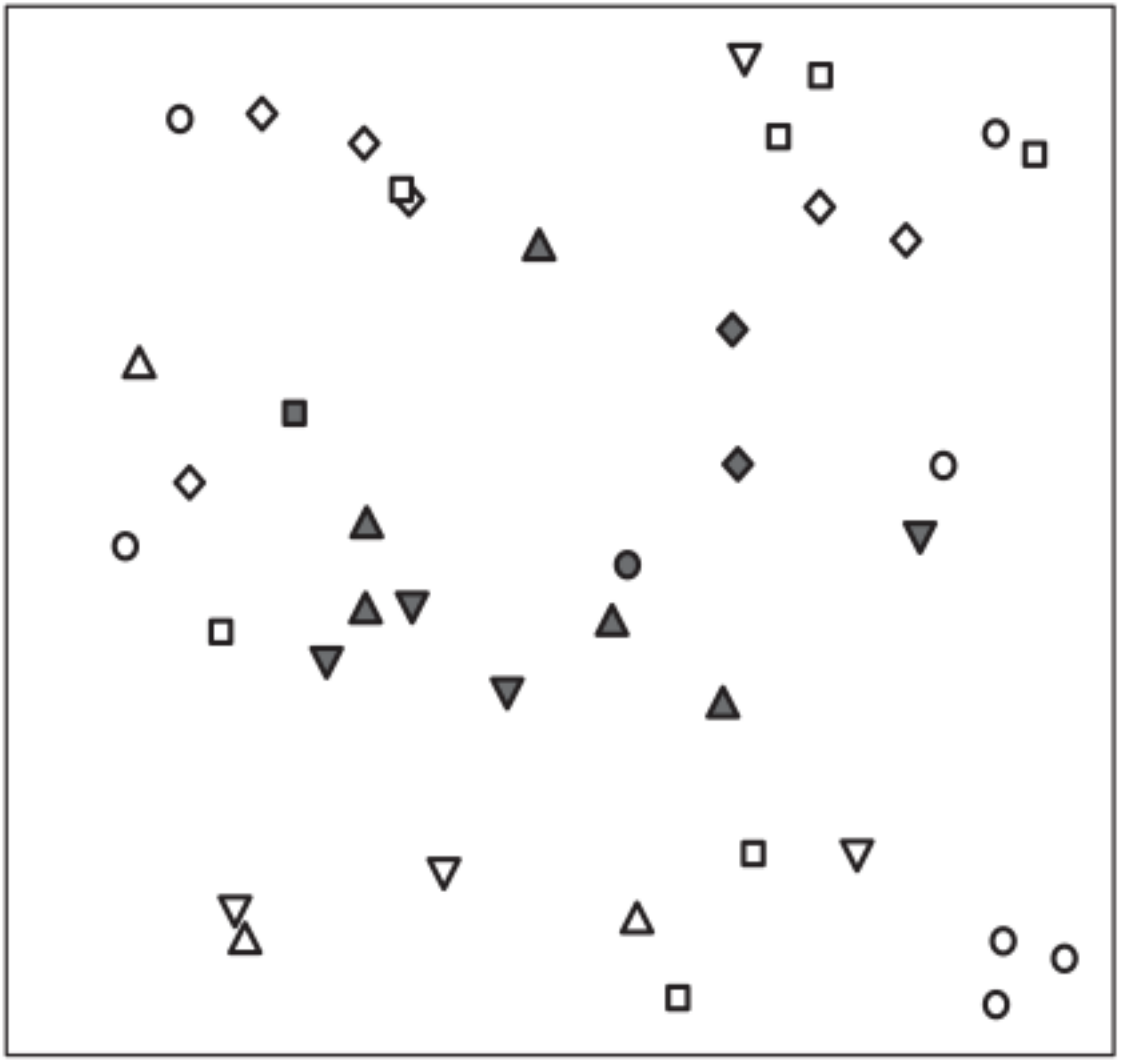
October 15th, 2015

Mahout reference book

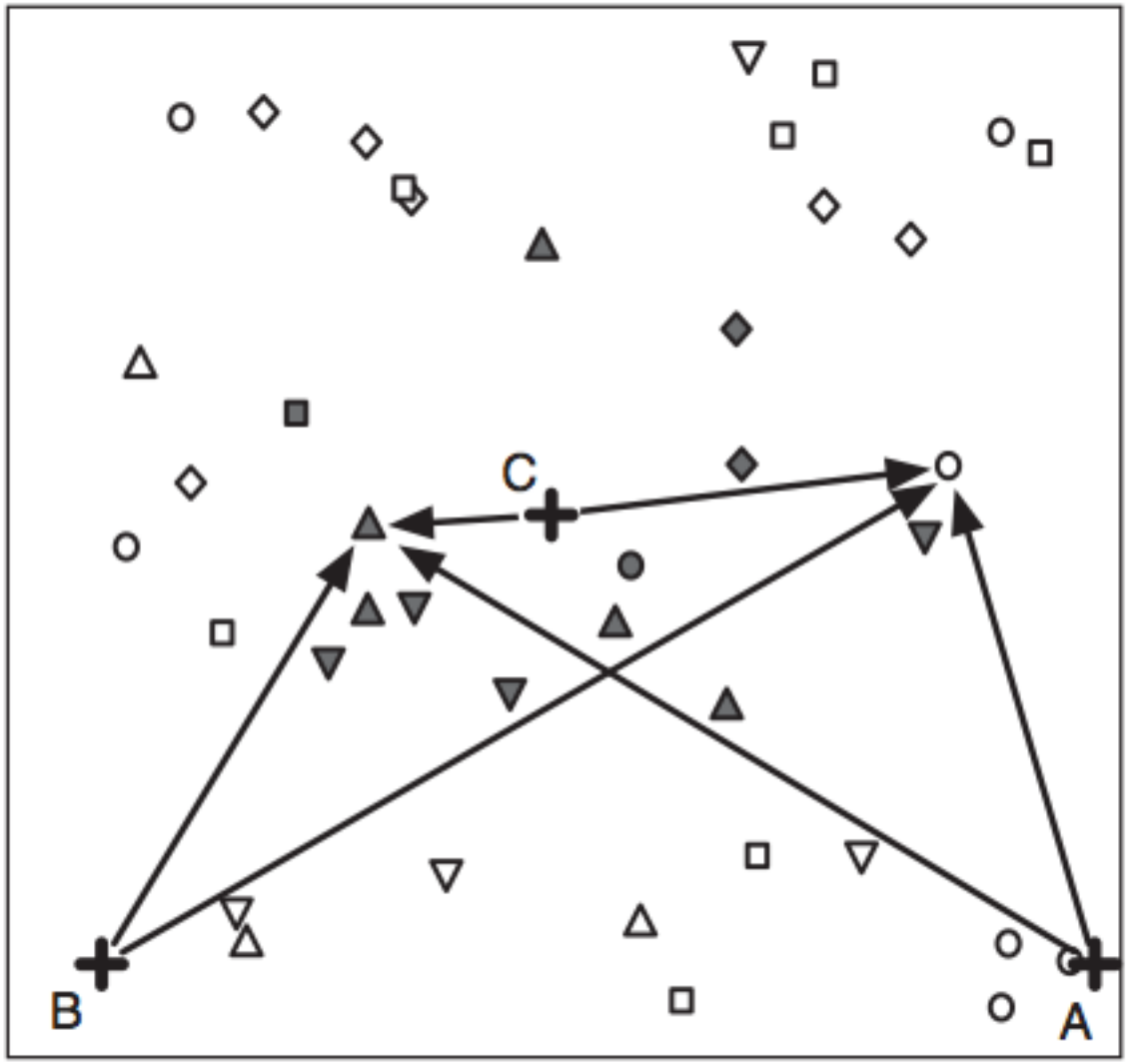


Requires Adobe Acrobat Reader to play audio and video links

Classification Example 3



What may be a good predictor?



Choose algorithm via Mahout

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

Stochastic Gradient Descent (SGD)

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

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

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

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

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

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

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

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

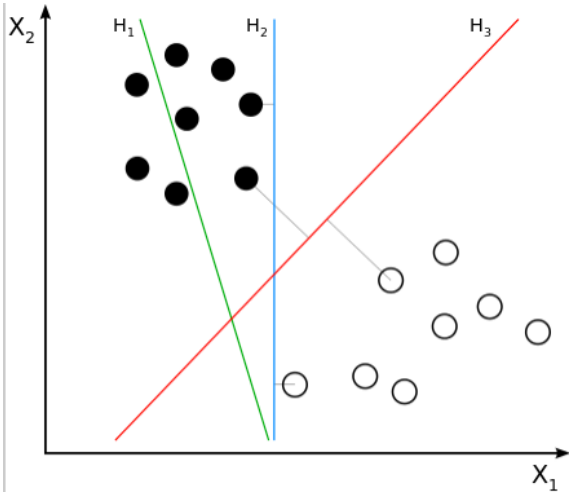
Characteristic of SGD

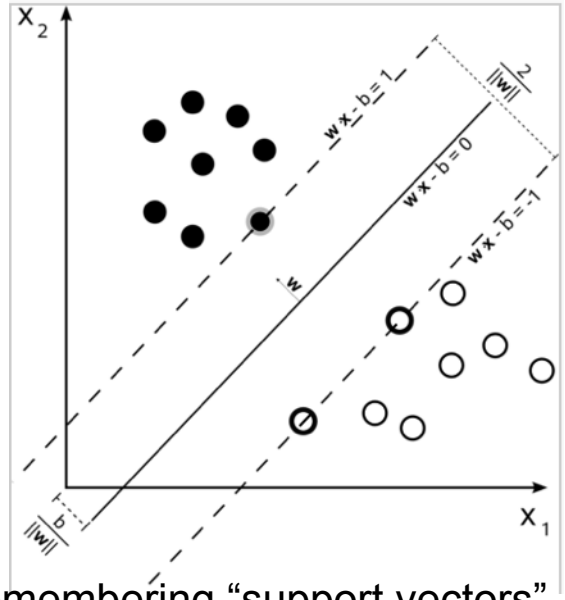
THE SGD ALGORITHM

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

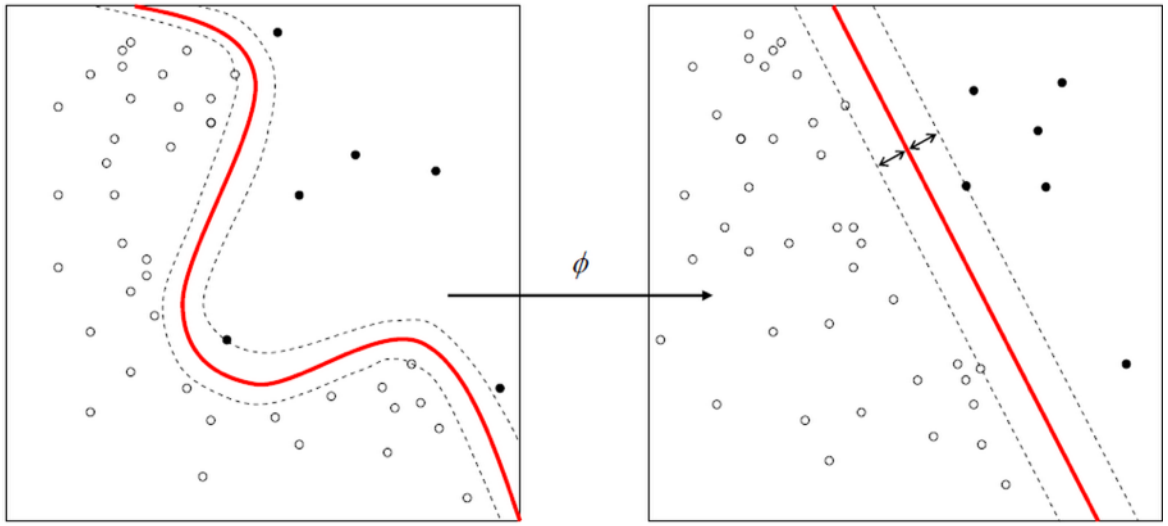
Importantly, because these algorithms do the same simple operation for each training example, they require a constant amount of memory. For this reason, each training example requires roughly the same amount of work. These properties make SGD-based algorithms scalable in the sense that twice as much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

Support Vector Machine (SVM)





maximize boundary distances; remembering “support vectors”



nonlinear kernels

Naive Bayes

Training set:

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

Classifier using Gaussian distribution assumptions:

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

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

Test Set:

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

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

$$P(male) = 0.5$$

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

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

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

==> female

Random Forest

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

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

For b = 1 through B:

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

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

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

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

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

Choosing a learning algorithm to train the model

- One low overhead classification method is the stochastic gradient descent (SGD) algorithm for logistic regression.
- This algorithm is sequential, but it's fast.

START RUNNING MAHOUT

```
$ $MAHOUT_HOME/bin/mahout
An example program must be given as the first argument.
Valid program names are:
  canopy: : Canopy clustering
  cat : Print a file or resource as the logistic regression models would see
     it
...
  runlogistic : Run a logistic regression model against CSV data
  ...
 trainlogistic : Train a logistic regression using stochastic gradient
     descent
 ...
```

CHECK MAHOUT'S BUILT-IN DATA

```
$ bin/mahout cat donut.csv
"x","y","shape","color","k","k0","xx","xy","yy","a","b","c","bias"
0.923307513352484,0.0135197141207755,21,2,4,8,0.852496764213146,...,1
0.711011884035543,0.909141522599384,22,2,3,9,0.505537899239772,...,1
...
0.67132937326096,0.571220482233912,23,1,5,2,0.450683127402953,...,1
0.548616112209857,0.405350996181369,24,1,5,3,0.300979638576258,...,1
0.677980388281867,0.993355110753328,25,2,3,9,0.459657406894831,...,1
$
```

The donut.csv data file in Example 3

Variable	Description	Possible values
x	The *x* coordinate of a point	Numerical from 0 to 1
y	The *y* coordinate of a point	Numerical from 0 to 1
shape	The shape of a point	Shape code from 21 to 25
color	Whether the point is filled or not	1=empty, 2=filled
k	The k-means cluster ID derived using only *x* and *y*	Integer cluster code from 1 to 10
k0	The k-means cluster ID derived using *x*, *y*, and color	Integer cluster code from 1 to 10
xx	The square of the *x* coordinate	Numerical from 0 to 1
xy	The product of the *x* and *y* coordinates	Numerical from 0 to 1
yy	The square of the *y* coordinate	Numerical from 0 to 1
a	The distance from the point (0,0)	Numerical from 0 to $\sqrt{2}$
b	The distance from the point (1,0)	Numerical from 0 to $\sqrt{2}$
c	The distance from the point (0.5,0.5)	Numerical from 0 to $(\sqrt{2})/2$
bias	A constant	1

Build a model using Mahout

```
$ bin/mahout trainlogistic --input donut.csv \
        --output ./model \
        --target color --categories 2 \
        --predictors x y  --types numeric \
        --features 20 --passes 100 --rate 50
...
color ~ -0.157*Intercept Term + -0.678*x + -0.416*y
      Intercept Term -0.15655
                   x -0.67841
                   y -0.41587
...
```

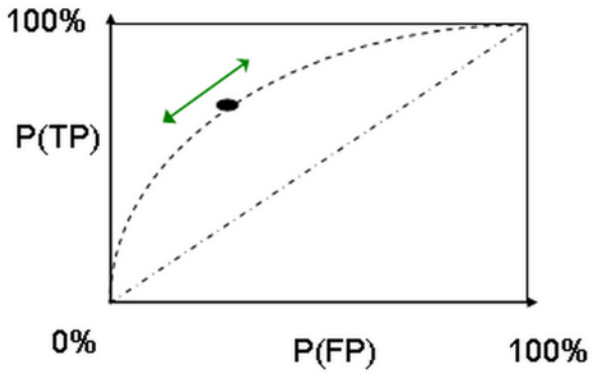
This command specifies that the input comes from the resource named donut.csv, that the resulting model is stored in the file ./model, that the target variable is in the field named `color` and that it has two possible values. The command also specifies that the algorithm should use variables `x` and `y` as predictors, both with numerical types. The remaining options specify internal parameters for the learning algorithm.

Trainlogistic program

Option	What it does
`--quiet`	Produces less status and progress output.
`--input <file-or-resource>`	Uses the specified file or resource as input.
`--output <file-for-model>`	Puts the model into the specified file.
`--target <variable>`	Uses the specified variable as the target.
`--categories <n>`	Specifies how many categories the target variable has.
`--predictors <v1> ... <vn>`	Specifies the names of the predictor variables.
`--types <t1> ... <tm>`	Gives a list of the types of the predictor variables. Each type should be one of `numeric`, `word`, or `text`. Types can be abbreviated to their first letter. If too few types are given, the last one is used again as necessary. Use `word` for categorical variables.
`--passes`	Specifies the number of times the input data should be reexamined during training. Small input files may need to be examined dozens of times. Very large input files probably don't even need to be completely examined.
`--lambda`	Controls how much the algorithm tries to eliminate variables from the final model. A value of 0 indicates no effort is made. Typical values are on the order of 0.00001 or less.
`--rate`	Sets the initial learning rate. This can be large if you have lots of data or use lots of passes because it's decreased progressively as data is examined.
`--noBias`	Eliminates the intercept term (a built-in constant predictor variable) from the model. Occasionally this is a good idea, but generally it isn't since the SGD learning algorithm can usually eliminate the intercept term if warranted.
`--features`	Sets the size of the internal feature vector to use in building the model. A larger value here can be helpful, especially with text-like input data.

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.

Build and test another model

```
$ bin/mahout trainlogistic --input donut.csv --output model \
        --target color --categories 2 \
        --predictors x y a b c  --types numeric \
        --features 20 --passes 100 --rate 50
...
color ~ 7.07*Intercept Term + 0.58*x + 2.32*y + 0.58*a + -1.37*b + -25.06*c
      Intercept Term 7.06759
                   a 0.58123
                   b -1.36893
                   c -25.05945
                   x 0.58123
                   y 2.31879
...
```

This model will reach a maximum value of 11.5 where c=0, and it will drop rapidly into negative territory once c> 0.3.

```
$  bin/mahout runlogistic --input donut.csv --model model \
          --auc --confusion
AUC = 1.00
confusion: [[27.0, 0.0], [0.0, 13.0]]
entropy: [[-0.1, -1.5], [-4.0, -0.2]]
```

Logistic Regression

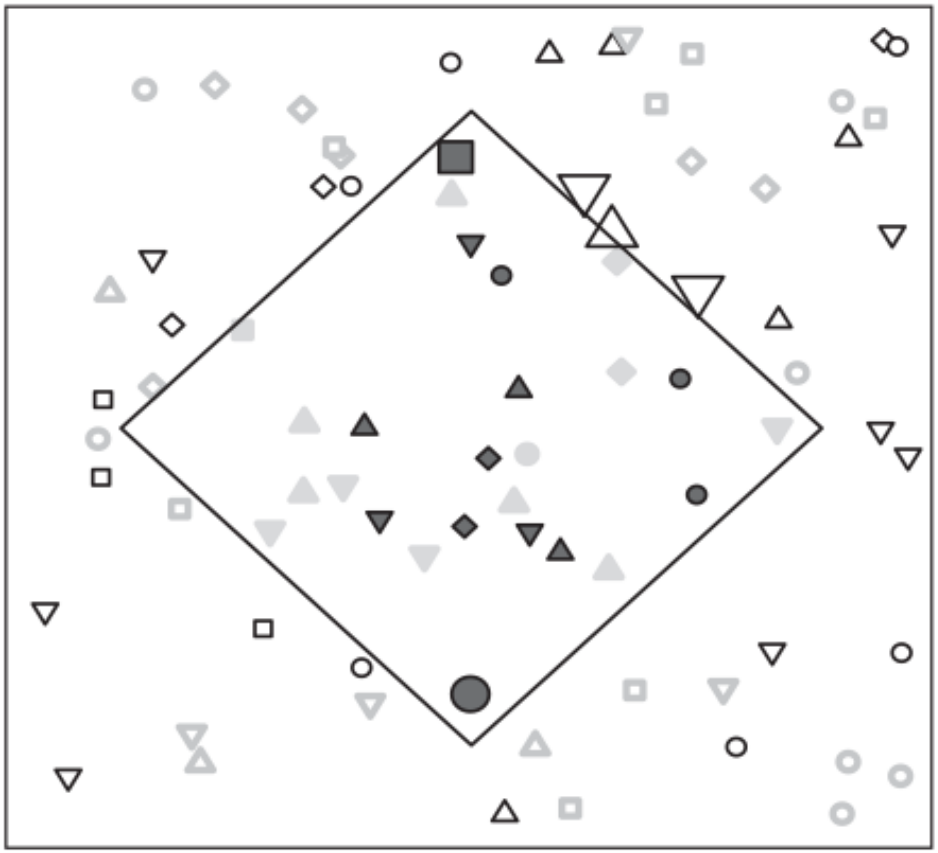
Columbia University

Logistic regression

Logistic regression describes a kind of classification model in which the predictor variables are combined with linear weights and then passed through a soft-limit function that limits the output to the range from 0 to 1. Logistic regression is closely related to other models such as a perceptron (where the soft limit is replaced by a hard limit), neural networks (where multiple layers of linear combination and soft limiting are used) and naive Bayes (where the linear weights are determined strictly by feature frequencies assuming independence). Logistic regression can't separate all possible classes, but in very high dimensional problems or where you can introduce new variables by combining other predictors, this is much less of a problem. The mathematical simplicity of logistic regression allows very efficient and effective learning algorithms to be derived.

Test using new data

```
$  bin/mahout runlogistic --input donut-test.csv --model model \
          --auc --confusion
AUC = 0.97
confusion: [[24.0, 2.0], [3.0, 11.0]]
entropy: [[-0.2, -2.8], [-4.1, -0.1]]
```



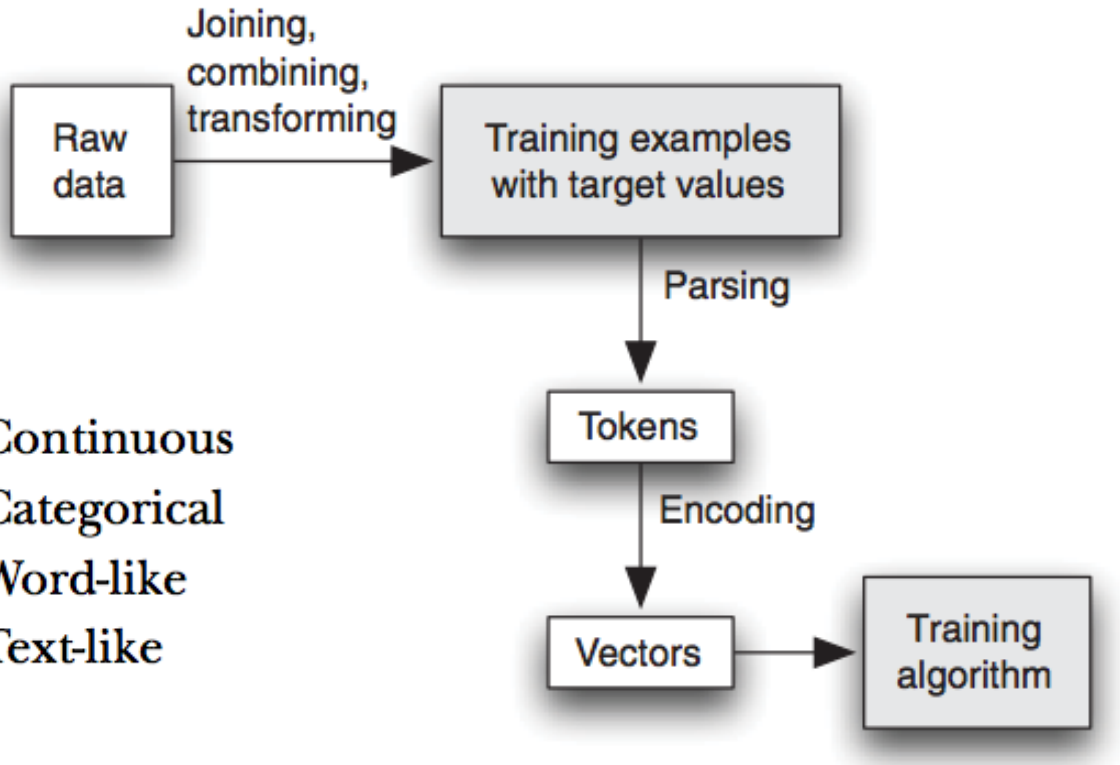
light color: original data
enlarged one: misclassification

Try another model

```
$ bin/mahout trainlogistic --input donut.csv --output model \
           --target color --categories 2 \
           --predictors x y a b  --types n --features 20 \
           --passes 100 --rate 50
...
color ~ 4.634*Intercept Term + -2.834*x + 5.558*y + -2.834*a + -6.000*b
      Intercept Term 4.63396
                   a -2.83439
                   b -5.99971
                   x -2.83439
                   y 5.55773
  ...
$  bin/mahout runlogistic --input donut-test.csv --model model \
             --auc --confusion
AUC = 0.91
confusion: [[27.0, 13.0], [0.0, 0.0]]
   entropy: [[-0.3, -0.4], [-1.5, -0.7]]
$
```

Only slightly worse. This indicates the information carried in c is also available in x, y, a, and b.

Train a classifier



1 *Preprocessing raw data*—Raw data is rearranged into records with identical fields. These fields can be of four types: continuous, categorical, word-like, or text-like in order to be classifiable.

2 *Converting data to vectors*—Classifiable data is parsed and vectorized using custom code or tools such as Lucene analyzers and Mahout vector encoders. Some Mahout classifiers also include vectorization code.

Preprocessing raw data — computational marketing example

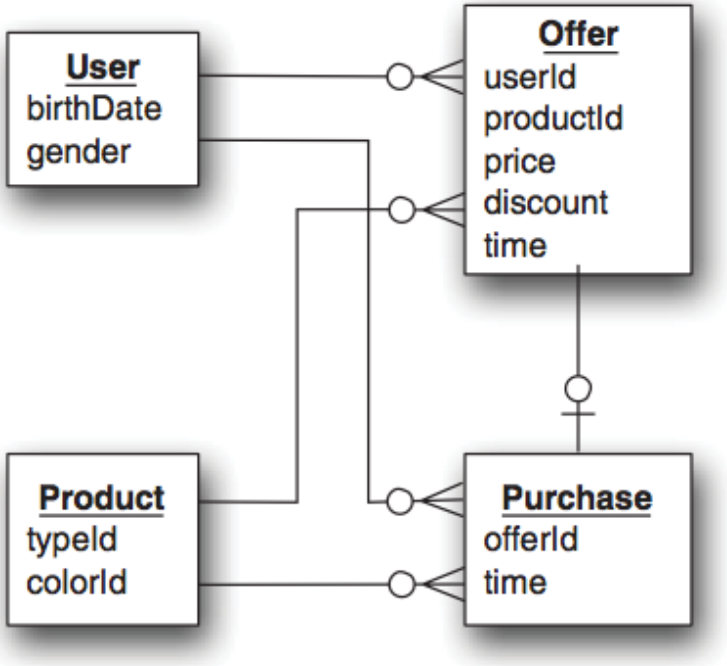
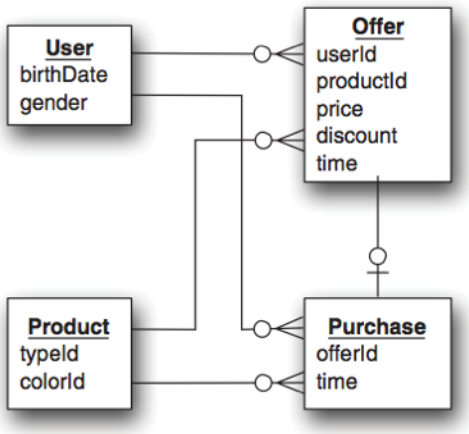


Table structure for a product sales example containing different types of data. Because no record in any table contains the data necessary in a training example for a classifier, this organization of raw data isn't directly usable as training data.

Converting data into training data for classifier

```
select
  now()-birthDate as age, gender,
  typeId, colorId, price, discount, offerTime,
  ifnull(purchase.time, 0, purchase.time - offer.time) as purchaseDelay,
  ifnull(purchase.time, 0, 1) as purchased
from
  offer
  join user using (userId)
  join product using (productId)
  left outer join purchase using (offerId);
```



original table

training data are converted into a single record

Converting classifiable data into vectors

Approach	Benefit	Cost	Where used?
Use one `Vector` cell per word, category, or continuous value	Involves no collisions and can be reversed easily	Requires two passes (one to assign cells and one to set values) and vectors may have different lengths	When dumping a Lucene index as `Vectors` for clustering
Represent `Vectors` implicitly as bags of words	Involves one pass and no collisions	Makes it difficult to use linear algebra primitives, difficult to represent continuous values, and you must format data in a special non-vector format	In naive Bayes
Use feature hashing	Involves one pass, vector size is fixed in advance, and it's a good fit for linear algebra primitives	Involves feature collisions, and the interpretation of the resulting model may be tricky	In `OnlineLogisticRegression` and other SGD learners

Feature hashing with Mahout APIs

Columbia University

ENCODING CONTINUOUS FEATURES

Mahout supports hashed feature encoding of continuous values through `ContinuousValueEncoder`. By default, the `ContinuousValueEncoder` only updates one location in the vector, but you can specify how many locations are to be updated by using its `setProbes()` method. To encode a value for one variable, a `Vector` is needed: `new RandomAccessSparseVector(lmp.getNumFeatures())`.

ENCODING CATEGORICAL AND WORD-LIKE FEATURES

```
FeatureVectorEncoder encoder =
    new StaticWordValueEncoder("variable-name");

for (DataRecord ex: trainingData) {
  Vector v = new RandomAccessSparseVector(10000);
  String word = ex.get("variable-name");
  encoder.addToVector(word, v);
  // use vector
}
```

Example — classifying 20 newsgroups data set (~ 20K docs)

Columbia University

NOTE The 20 newsgroups data set is a standard data set commonly used for machine learning research. The data is from transcripts of several months of postings made in 20 Usenet newsgroups from the early 1990s.

http://qwone.com/~jason/20Newsgroups/ *different from the book*

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

If you examine one of the files in the training data directory, such as 20news-bydate-train/sci.crypt/15524, you'll see something like this:

```
From: rdippold@qualcomm.com (Ron "Asbestos" Dippold)
Subject: Re: text of White House announcement and Q&As
Originator: rdippold@qualcom.qualcomm.com
Nntp-Posting-Host: qualcom.qualcomm.com
Organization: Qualcomm, Inc., San Diego, CA
Lines: 12

ted@nmsu.edu (Ted Dunning) writes:
>nobody seems to have noticed that the clipper chip *must* have been
>under development for considerably longer than the 3 months that
>clinton has been president. this is not something that choosing
...
```

The predictor features are either headers or body

Count types of the header lines

```
#!/bin/bash
export LC_ALL='C'
for file in 20news-bydate-train/*/*
do
  sed -E -e '/^$/,$d' -e 's/:.*//' -e '/^[[:space:]]/d' $file
done | sort | uniq -c | sort -nr
```

Header	Count	Comment	Action
Subject	11,314	Text	Keep
From	11,314	Sender of message	Keep
Lines	11,311	Number of lines in message	Keep
Organization	10,841	Related to sender	Try?
Distribution	2,533	Possible target leak	Try?
Nntp-Posting-Host	2,453	Related to sender	Try?
NNTP-Posting-Host	2,311	Same as previous except for capitalization	Try?
Reply-To	1,720	Possible cue, not common, experiment with this	Try?
Keywords	926	Describes content	Keep
Article-I.D.	673	Too specific	Drop
X-Newsreader	588	Software used by sender	Drop
Summary	391	Like "Keywords"	Keep
Originator	291	Similar to "From," but much less common	Drop
In-Reply-To	219	Probably just an ID number, rare	Drop

Header lines cont'd

Header	Count	Comment	Action
To	80	Possible target leak, rare	Drop
X-Disclaimer	64	Noise, rare	Drop
Disclaimer	56	Possible cue due to newsgroup paranoia level, rare	Drop
...		...	
Weather	1	Bizarre header	Drop
Orginization	1	Spelling errors in headers are surprising	Drop
Oganization	1	Well, maybe not	Drop
Oanization	1		Drop
Moon-Phase	1	Yes, really, one posting had this; open source is wonderful	!?!

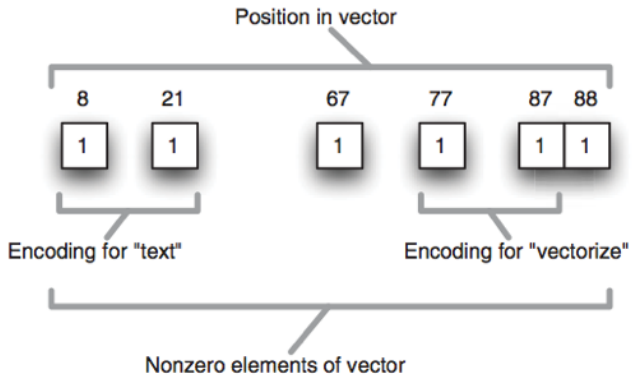
Tokenizing and vectoring text

```
FeatureVectorEncoder encoder = new StaticWordValueEncoder("text");
Analyzer analyzer =
    new StandardAnalyzer(Version.LUCENE_31);          <-- Break text into words

StringReader in = new StringReader("text to magically vectorize");
TokenStream ts = analyzer.tokenStream("body", in);
TermAttribute termAtt = ts.addAttribute(TermAttribute.class);

Vector v1 = new RandomAccessSparseVector(100);        <-- Encode into vector size 100
while (ts.incrementToken()) {
  char[] termBuffer = termAtt.termBuffer();
  int termLen = termAtt.termLength();

  String w = new String(termBuffer, 0, termLen);
  encoder.addToVector(w, 1, v1);                      <-- Add word w to vector v
}
System.out.printf("%s\n", new SequentialAccessSparseVector(v1));
```

This code produces a printable form of the vector, like this:

```
{8:1.0,21:1.0,67:1.0,77:1.0,87:1.0,88:1.0}
```

Mahout -> Encoder; Lucene -> parser



Training code — I

SETTING UP VECTOR ENCODERS

First, you need objects that will help convert text and line counts into vector values, as follows:

```
Map<String, Set<Integer>> traceDictionary =
  new TreeMap<String, Set<Integer>>();
FeatureVectorEncoder encoder = new StaticWordValueEncoder("body");
encoder.setProbes(2);
encoder.setTraceDictionary(traceDictionary);
FeatureVectorEncoder bias = new ConstantValueEncoder("Intercept");
bias.setTraceDictionary(traceDictionary);
FeatureVectorEncoder lines = new ConstantValueEncoder("Lines");
lines.setTraceDictionary(traceDictionary);
Dictionary newsGroups = new Dictionary();
Analyzer analyzer = new StandardAnalyzer(Version.LUCENE_31);
```

In this code, there are three different encoders for three different kinds of data. The first one, `encoder`, is used to encode the textual content in the subject and body of a posting. The second, `bias`, provides a constant offset that the model can use to encode the average frequency of each class. The third, `lines`, is used to encode the number of lines in a message.

Training code — II

CONFIGURING THE LEARNING ALGORITHM

You can configure the logistic regression learning algorithm as follows:

```
OnlineLogisticRegression learningAlgorithm =
    new OnlineLogisticRegression(
          20, FEATURES, new L1())
        .alpha(1).stepOffset(1000)
        .decayExponent(0.9)
        .lambda(3.0e-5)
        .learningRate(20);
```

The learning algorithm constructor accepts arguments specifying the number of categories in the target variable, the size of the feature vectors, and a regularizer. In addition, there are a number of configuration methods in the learning algorithm. The `alpha`, `decayExponent`, and `stepOffset` methods specify the rate and way that the learning rate decreases. The `lambda` method specifies the amount of regularization, and the `learningRate` method specifies the initial learning rate.

Training code — III

ACCESSING DATA FILES

Next, you need a list of all of the training data files, as follows:

```
List<File> files = new ArrayList<File>();
for (File newsgroup : base.listFiles()) {
  newsGroups.intern(newsgroup.getName());

   files.addAll(Arrays.asList(newsgroup.listFiles()));
 }

 Collections.shuffle(files);
 System.out.printf("%d training files\n", files.size());
```

This code puts all of the newsgroups' names into the dictionary. Predefining the contents of the dictionary like this ensures that the entries in the dictionary are in a stable and recognizable order. This helps when comparing results from one training run to another.

Training code — IV

PREPARING TO TOKENIZE THE DATA

Most of the data in these files is textual, so you can use Lucene to tokenize it. Using Lucene is better than simply splitting on whitespace and punctuation because you can use Lucene's `StandardAnalyzer` class, which treats special tokens like email addresses correctly.

You'll also need several variables to accumulate averages for progress output such as average log likelihood, percent correct, document line count, and the number of documents processed:

```
double averageLL = 0.0;
double averageCorrect = 0.0;
double averageLineCount = 0.0;
int k = 0;
double step = 0.0;
int[] bumps = new int[]{1, 2, 5};
double lineCount;
```

These variables will allow you to measure the progress and performance of the learning algorithm.

Training code — V

READING AND TOKENIZING THE DATA

```
for (File file : files) {
  BufferedReader reader = new BufferedReader(new FileReader(file));
  String ng = file.getParentFile().getName();
  int actual = newsGroups.intern(ng);
  Multiset<String> words = ConcurrentHashMultiset.create();

  String line = reader.readLine();
  while (line != null && line.length() > 0) {
    if (line.startsWith("Lines:")) {

      String count = Iterables.get(onColon.split(line), 1);
      try {
        lineCount = Integer.parseInt(count);
        averageLineCount += (lineCount - averageLineCount)
            / Math.min(k + 1, 1000);
      } catch (NumberFormatException e) {
        lineCount = averageLineCount;
      }
    }
    boolean countHeader = (
        line.startsWith("From:") || line.startsWith("Subject:")||
        line.startsWith("Keywords:")|| line.startsWith("Summary:"));
    do {
      StringReader in = new StringReader(line);
      if (countHeader) {
        countWords(analyzer, words, in);
      }
      line = reader.readLine();
    } while (line.startsWith(" "));
  }
  countWords(analyzer, words, reader);
  reader.close();
}
```

1. Identify newsgroup
2. Check for line count header
3. Count words in headers
4. Count words in body

Training code — VI

Columbia University

VECTORIZING THE DATA

With the data from the document in hand, you're ready to collect all of the features into a single feature vector that can be used by the classifier learning algorithm. Here's how to do that:

```
Vector v = new RandomAccessSparseVector(FEATURES);
bias.addToVector(null, 1, v);
lines.addToVector(null, lineCount / 30, v);
logLines.addToVector(null, Math.log(lineCount + 1), v);
for (String word : words.elementSet()) {
  encoder.addToVector(word, Math.log(1 + words.count(word)), v);
}
```

First, encode the `bias` (constant) term, which always has a value of `1`. This feature can be used by the learning algorithm as a threshold. Some problems can't be solved using logistic regression without such a term.

The line count is encoded both in raw form and as a logarithm. The division by 30 is done to put the line length into roughly the same range as other inputs so that learning will progress more quickly.

The body of the document is encoded similarly except that the weight applied to each word in the document is based on the log of the frequency of the word in the document, rather than the straight frequency. This is done because words occur more than once in a single document much more frequently than would be expected by the overall frequency of the word. The log of the frequency takes this into account.

Training code — VII

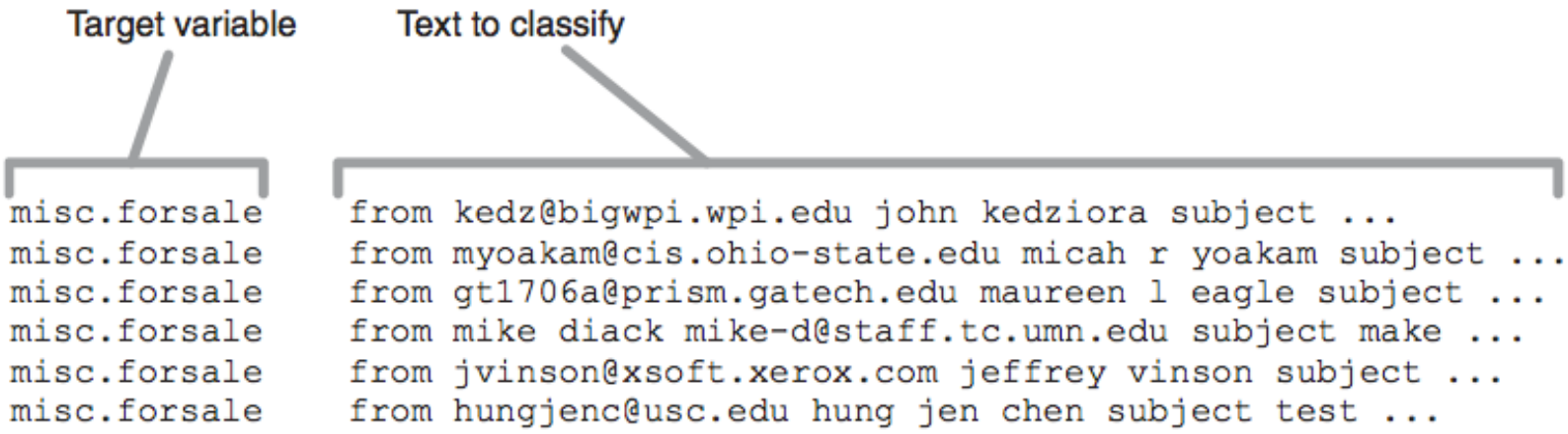
TRAINING THE SGD MODEL WITH THE ENCODED DATA

```
learningAlgorithm.train(actual, v);
k++;
int bump = bumps[(int) Math.floor(step) % bumps.length];
int scale = (int) Math.pow(10, Math.floor(step / bumps.length));
if (k % (bump * scale) == 0) {
  step += 0.25;
  System.out.printf("%10d %10.3f %10.3f %10.2f %s %s\n",
       k, ll, averageLL, averageCorrect * 100, ng,
       newsGroups.values().get(estimated));
}
learningAlgorithm.close();
```

Each time the number of examples reaches `bump * scale`, another line of output is produced to show the current status of the learning algorithm. As the learning progresses, the frequency of status reports will decrease. This means that when accuracy is changing more quickly, there will be more frequent updates.

Classifying using Naive Bayes

```
$ bin/mahout prepare20newsgroups -p 20news-bydate-train/ \
        -o 20news-train/ \
        -a org.apache.lucene.analysis.standard.StandardAnalyzer \
        -c UTF-8

no HADOOP_CONF_DIR or HADOOP_HOME set, running locally
INFO: Program took 3713 ms

$ bin/mahout prepare20newsgroups -p 20news-bydate-test \
        -o 20news-test \
        -a org.apache.lucene.analysis.standard.StandardAnalyzer \
        -c UTF-8
no HADOOP_CONF_DIR or HADOOP_HOME set, running locally
INFO: Program took 2436 ms
```



Training and testing naive Bayes classifier

```
bin/mahout trainclassifier -i 20news-train \
            -o 20news-model \
            -type cbayes \
            -ng 1 \
            -source hdfs
...
INFO: Program took 250104 ms
```

The result is a model stored in the 20news-model directory, as specified by the `-o` option. The `-ng` option indicates that individual words are to be considered instead of short sequences of words. The model consists of several files that contain the components of the model. These files are in a binary format and can't be easily inspected directly, but you can use them to classify the test data with the help of the `testclassifier` program.

To run the naive Bayes model on the test data, you can use the following command:

```
bin/mahout testclassifier -d 20news-test \
            -m 20news-model \
            -type cbayes \
            -ng 1 \
            -source hdfs \
            -method sequential
```

Here the `-m` option specifies the directory that contains the model built in the previous step. The `-method` option specifies that the program should be run in a sequential mode rather than using Hadoop. For small data sets like this one, sequential operation is preferred. For larger data sets, parallel operation becomes necessary to keep the runtime reasonably small.

See Training Results

```
bin/mahout testclassifier -d 20news-train -m 20news-model\
   -type cbayes -ng 1 -source hdfs -method sequential
...
Correctly Classified Instances          :      11075   97.8876%
Incorrectly Classified Instances        :        239    2.1124%
Total Classified Instances              :      11314
```

When given this familiar data, the model is able to get nearly 98 percent correct, which is much too good to be true on this particular problem. The best machine learning researchers only claim accuracies for their systems around 84–86 percent.

Classification results

```
Confusion Matrix
-------------------------------------------------
a b c d e f g h i j k l m n o p q r s t <--Classified as
388                                     | 397 a = rec.sport.baseball
  386                                   | 396 b = sci.crypt
    396                                 | 399 c = rec.sport.hockey
      347                               | 364 d = talk.politics.guns
        377                             | 398 e = soc.religion.christian
  12      304           18              | 393 f = sci.electronics
            281     14  43        21    | 394 g = comp.os.ms-windows.misc
              313       22 16           | 390 h = misc.forsale
     26 69      83 41                   | 251 i = talk.religion.misc
        45        225 13            11  | 319 j = alt.atheism
                    334           32    | 395 k = comp.windows.x
                      367               | 376 l = talk.politics.mideast
          19 23 15      307 13          | 392 m =comp.sys.ibm.pc.hardware
                      16 335            | 385 n = comp.sys.mac.hardware
                           371          | 394 o = sci.space
                             393        | 398 p = rec.motorcycles
                            12 364      | 396 q = rec.autos
            11      22            305   | 389 r = comp.graphics
      102                         160   | 310 s = talk.politics.misc
                                    362 | 396 t = sci.med
Default Category: unknown: 20
```

```
-------------------------------------------------------
Correctly Classified Instances          :       6398    84.9442%
Incorrectly Classified Instances        :       1134    15.0558%
Total Classified Instances              :       7532
=======================================================
```

Evaluating Classifier using Mahout

- One useful differentiator of Mahout is its ability to provide immediate and ongoing assessments of performance by allowing a user to pass target variable scores and reference values to an evaluator one at a time to get on-the-fly performance feedback.

- This is different from most other systems that must process all of the scores and target values in batch mode to evaluate performance.

What is a good classifier?

Percentage correct is not necessarily the best way to evaluate and tune a system.

Data from two hypothetical classifiers to show some of the limitations of just looking at percent correct. The columns show the frequency of each possible model output. Each row contains data for a particular correct value, and the answer with the highest score is in bold. Model 1 is never quite right, but may still be useful, whereas model 2 is like a stopped clock.

Correct Value	Model 1			Model 2		
	A	B	C	A	B	C
A	0.45	0.50	0.05	0.01	0.01	0.99
B	0.50	0.45	0.05	0.01	0.01	0.99
C	0.05	0.50	0.45	0.01	0.01	0.99

Model 1 is more useful than Model 2. Why?

Recognizing the different in cost of errors

Example: The cost of a false alarm may be much less than the cost of a false negative.

Cancer
Missile

Example: The cost of a false negative may be less than the cost of a false alarm.

Spam

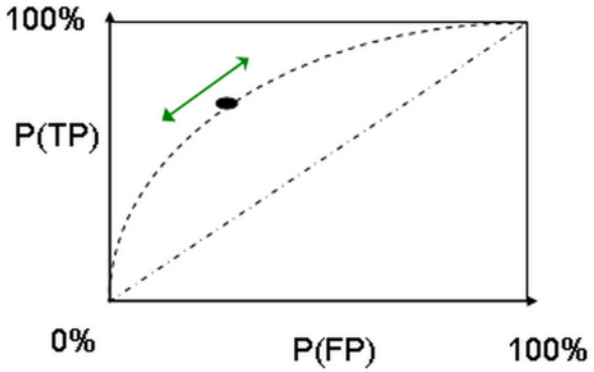
The classifier evaluation API

Metric	Supported by class
Percent correct	CrossFoldLearner
Confusion matrix	ConfusionMatrix, Auc
Entropy matrix	Auc
AUC	Auc, OnlineAuc, CrossFoldLearner, AdaptiveLogisticRegression
Log likelihood	CrossFoldLearner

Class	Description
`Auc`	
`OnlineAuc`	
`OnlineSummarizer`	
`ConfusionMatrix`	
`AbstractVectorClassifier`	`OnlineSummarizer`
`CrossFoldLearner`	`auc(),percentCorrect(),aloglikelihood()`
`AdaptiveLogisticRegression`	`CrossFoldLearnerauc()`

AUC and Online AUC

AUC



`OnlineAuc` differs from `Auc` in that `OnlineAuc` makes estimates of AUC available at any time during the loading of data with little overhead, though the insertion of each element is slightly more expensive. The algorithms used in `Auc` and `OnlineAuc` provide essentially identical accuracy, but `Auc.auc()` is more expensive to run than `OnlineAuc.auc()` because it involves sorting several thousand samples.

Code to calculate AUC and Online AUC

```
Auc x1 = new Auc();
OnlineAuc x2 = new GlobalOnlineAuc();
BufferedReader in = new BufferedReader(new FileReader(inputFile));
int lineCount = 0;
String line = in.readLine();
while (line != null) {
  lineCount++;
  String[] pieces = line.split(",");
  double score = Double.parseDouble(pieces[0]);
  int target = Integer.parseInt(pieces[1]);
  x1.add(target, score);
  x2.addSample(target, score);
  if (lineCount%500 == 0) {
    System.out.printf("%10d\t%10.3f\t%10d\t%.3f\n",
                lineCount, score, target, x2.auc());
  }
  line = in.readLine();
}

System.out.printf("%d lines read\n", lineCount);
System.out.printf("%10.2f = batch estimate\n", x1.auc())
System.out.printf("%10.2f = on-line estimate\n", x2.auc());
```

1 **x2 computes AUC as input is read**

2 **xl and x2 results are essentially identical**

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

Entropy Matrix

In an *entropy matrix*, the rows correspond to actual target values, and the columns correspond to model outputs just as with a confusion matrix. The difference is that the elements of the matrix are the averages of the log of the probability score for each true or estimated category combination. Using the average log in this way is strongly motivated by mathematical principles based on approximating a probability distribution optimally. This average log probability expresses how unusual a particular classifier result is by comparing the classifier's likelihood of classifying into a certain category against how common that category is overall.

```
Matrix entropy = x1.entropy();
for (int i = 0; i < entropy.rowSize(); i++) {
  for (int j = 0; j < entropy.columnSize(); j++) {
    System.out.printf("%10.2f ", entropy.get(i, j));
  }
  System.out.printf("\n");
}
```

Average Log Likelihood

Log likelihood is a way of scoring a model based on how certain or uncertain the model is about the answers it produces. Log likelihood requires that a model produce a probability score as an output. A model gets credit if it gives very low probabilities to incorrect answers and a high probability to the correct answer. A particularly useful feature of log-likelihood evaluation is that a model is given partial credit for narrowing the set of possible answers even if the model doesn't settle on exactly the right answer.

```
OnlineSummarizer summarizeLogLikelihood = new OnlineSummarizer();
for (int i = 0; i < 1000; i++) {
  Example x = Example.readExample();
  double ll = classifier.logLikelihood(x.target, x.features);
  summarizeLogLikelihood.add(ll);
}

System.out.printf(
    "Average log-likelihood = %.2f (%.2f, %.2f) (25%%-ile,75%%-ile)\n",
    summarizeLogLikelihood.getMean(),
    summarizeLogLikelihood.getQuartile(1),
    summarizeLogLikelihood.getQuartile(2));
```

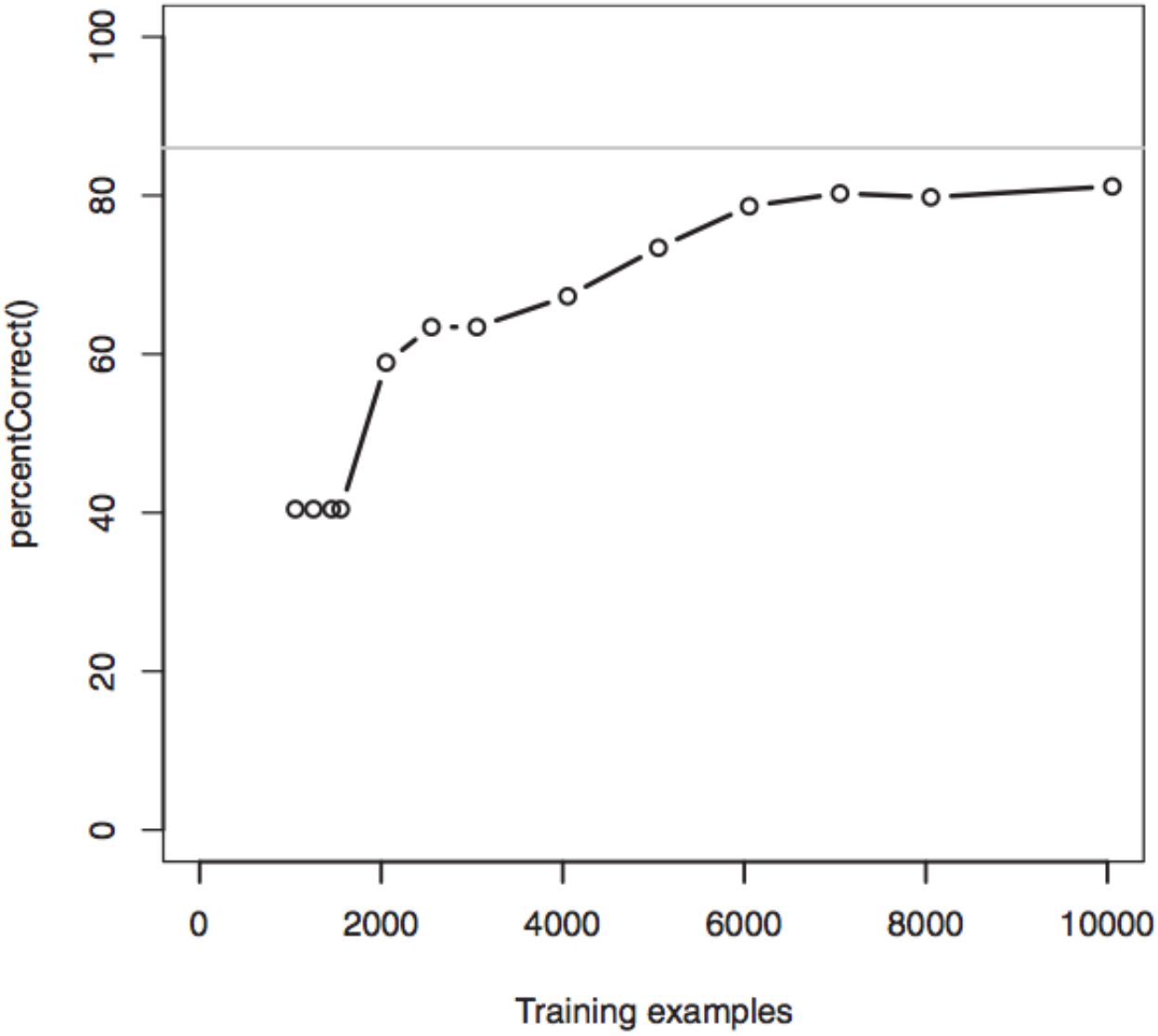
Characteristics of Log Likelihood

Log likelihood has the desirable mathematical property that you can only maximize a log-likelihood score if your model exactly reproduces the actual probability distribution for the target variable. Log likelihood and AUC complement each other. Log likelihood can be used with multiple target categories, whereas AUC is limited to binary outputs. AUC, on the other hand, can accept any kind of score; log likelihood requires scores that reflect probability estimates. AUC has an absolute scale from 0 to 1 that can be used to compare different models, whereas the scale for log likelihood is open-ended.

Log Likelihood has a maximum value of 0 and no bound on how far negative it can go.

For highly accurate classifiers, the value of average log likelihood should be close to the average percent correct for the classifier,

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

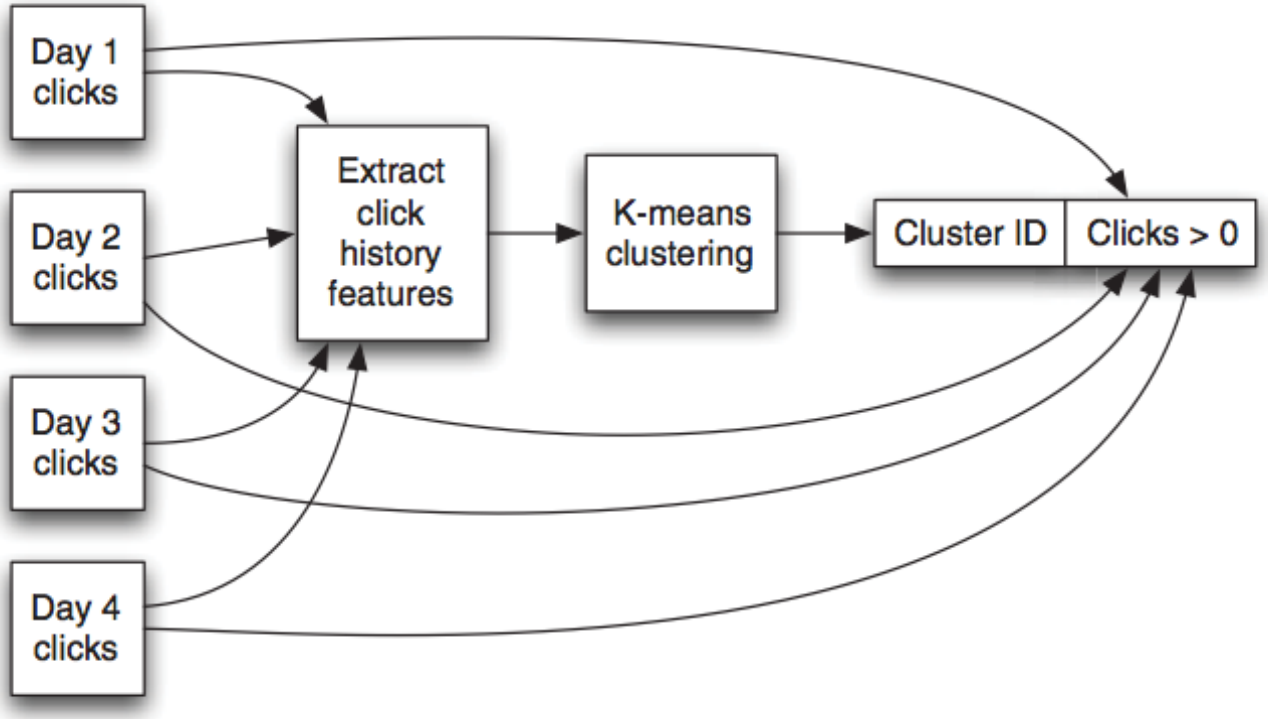
When working with real data and real classifiers it's almost a rule that the first attempts to build models will fail, occasionally spectacularly. Unlike normal software engineering, the failures of models aren't usually as dramatic as a null pointer dereference or out-of-memory exception. Instead, a 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.

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);
```

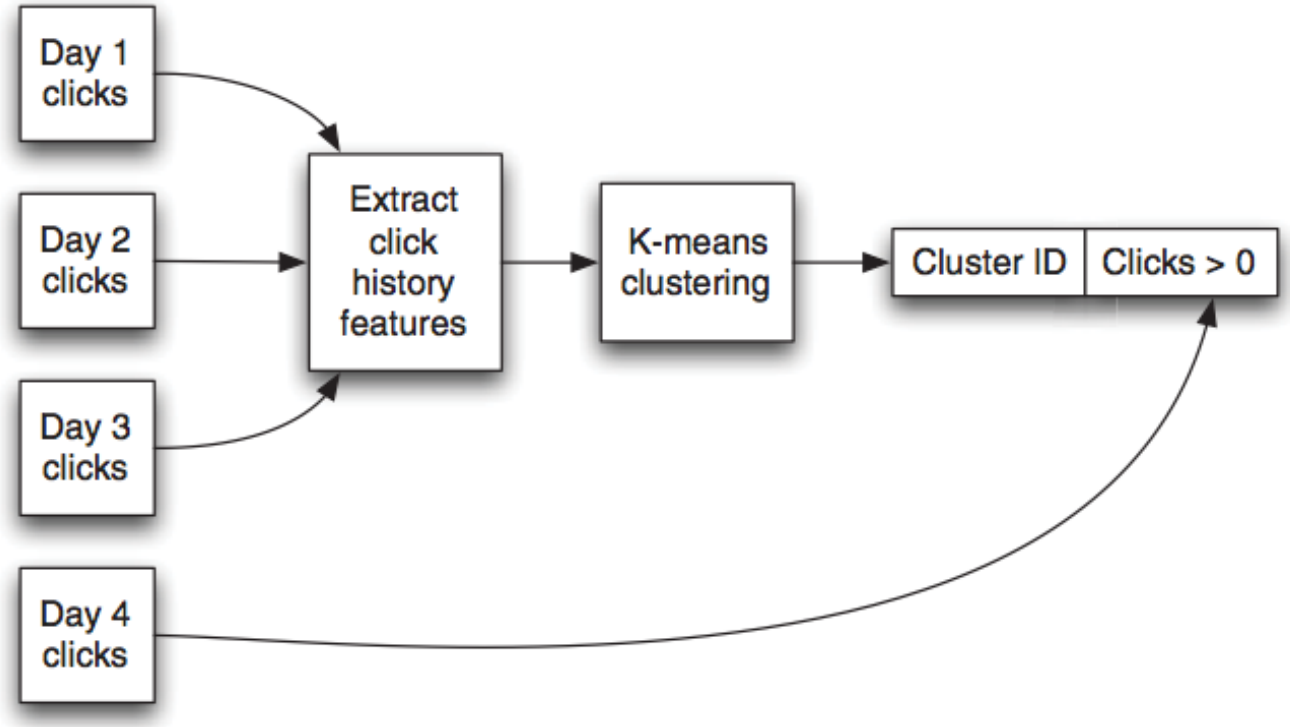
This date field is chosen so that all the documents from the same newsgroup appear to have come from the same month, but documents from different newsgroups come from different months.

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.

Deploy a Classifier

The deployment process can be broken down into these steps:

- Scope out the problem
- Optimize feature extraction as needed
- Optimize vector extraction as needed
- Deploy the scalable classifier service

Optimize feature extraction as needed

At small to medium scale—up to a few million examples—any technique will work, and proven data management and extraction technologies like relational databases will serve well.

As scale increases, you're likely to have to commit to a parallel extraction architecture, such as Hadoop, or a commercially supported system like Aster Data, Vertica, or Greenplum. If you choose to use Hadoop, you'll also have to decide what general approach you'll use to implement your feature extraction code. Good options include Apache Hive (http://hive.apache.org/), Apache Pig (http://pig.apache.org), Cascading (http://www.cascading.org/), or even raw MapReduce programs written in Java.

Optimize vector encoding as needed

Columbia University

If you have less than about 100 million training examples per classifier after feature extraction and downsampling, consider a sequential model builder such as SGD. If you have more than about five million training examples, you'll also need to optimize your data parsing and feature vector encoding code fairly carefully. This requirement may involve parsing without string conversions, multithreaded input conversion, and caching strategies to maximize the speed with which data can be fed to the learning algorithm.

For more than about 10 million training examples, you may want to consider using a `CrossFoldLearner` directly with predefined learning-rate parameters.

Determine 'how big is big?'

- How many training examples are there?
- What is the batch size for classification?
- What is the required response time for classification batches?
- How many classifications per second, in total, need to be done?
- What is the expected peak load for the system?

Estimating peak load

To estimate peak load, it's convenient to simply assume that a day has 20,000 seconds instead of the actual 86,400 seconds that make up a day. Using an abbreviated day of 20,000 seconds is a rule of thumb that allows you to convert from average rates to peak rates at plausible levels of transaction burstiness.

Estimating long-term average transaction rates is usually pretty easy, but you need to know the peak rate to be supported when you design your system. Using this rule of thumb allows you to make a so-called Fermi estimate for the required number of transactions per second that your system will need to handle. Enrico Fermi was a renowned physicist who was remarkably good at making estimates like this, and his name has become attached to such back-of-the-envelope computations.

The number of training examples

Typically, Mahout becomes interesting once you exceed 100,000 training examples. There is, however, no lower bound on how much data can be used. When you have 10 million or more training examples, most other data-mining solutions are unable to complete training, whereas above 100 million or a billion training examples there are very few systems available that can build usable models. The number of training examples, along with model types, are the primary determinants of the training time.

Classification batch size

It's common for a number of classifications to be done to satisfy a single request. This requirement often occurs because multiple alternatives need to be evaluated. For instance, in an ad placement system, there are typically thousands of advertisements that each have models to be evaluated to determine which advertisement to show on a web page. The batch size in this case would be the number of advertisements that need to be evaluated. A few applications have very large batches of more than 100,000 models.

At the other extreme, a fraud-detection server might only have a single model whose value needs to be computed, because the question of whether a single transaction is fraudulent involves only one decision: fraud or not fraud. This situation is very different from an advertisement-selection server, where a click-prediction model must be computed for each of many possible advertisements.

Maximum Response Time and Required Throughput

The maximum response time determines how quickly a single batch of classifications must be evaluated. This, in turn, defines a lower bound on the required classification rate. This rate is only a lower bound because if there are enough transactions per second, it may be necessary to support an even higher rate.

Typically, it's pretty easy to build a system that supports 1,000 classifications per second with small classification batch sizes because such a rate is small relative to what current computers can do. In fact, the Thrift-based server described later in this chapter can easily meet this requirement. Until batch sizes exceed 100 to 1,000 classifications per batch, you should be able to support roughly that transaction rate, because costs will be dominated by network time. At batch sizes of 10,000 or more, response times will start to climb substantially. At larger batch sizes and with subsecond response times required, you'll probably need to start scaling horizontally or restructuring how the classifications are done.

Balancing big versus fast

Columbia University

AT TRAINING TIME

Running the large-scale joins necessary to create classifiable data often dominates all other aspects of training. You can use standard Hadoop tools for that, but care needs to be taken to make the joins run efficiently.

AT CLASSIFICATION TIME

When you come to integrating your classifier into your system, your overall classification throughput needs will drive your selection of what you need to optimize in the classifier. Most Mahout classifiers are fast enough that this will only matter in extreme systems, but you need to check whether you're in the extreme category. If not, you're ready to roll with the standard methods, but if you're extreme, you'll need to budget time to stretch the normal performance limits.

If you require a moderate number (10s to 1000s) of classifications per evaluation and have more stringent latency and throughput requirements, you'll need to augment the basic classification service with a multithreaded evaluation of multiple models or inputs, but otherwise the basic design should still suffice.

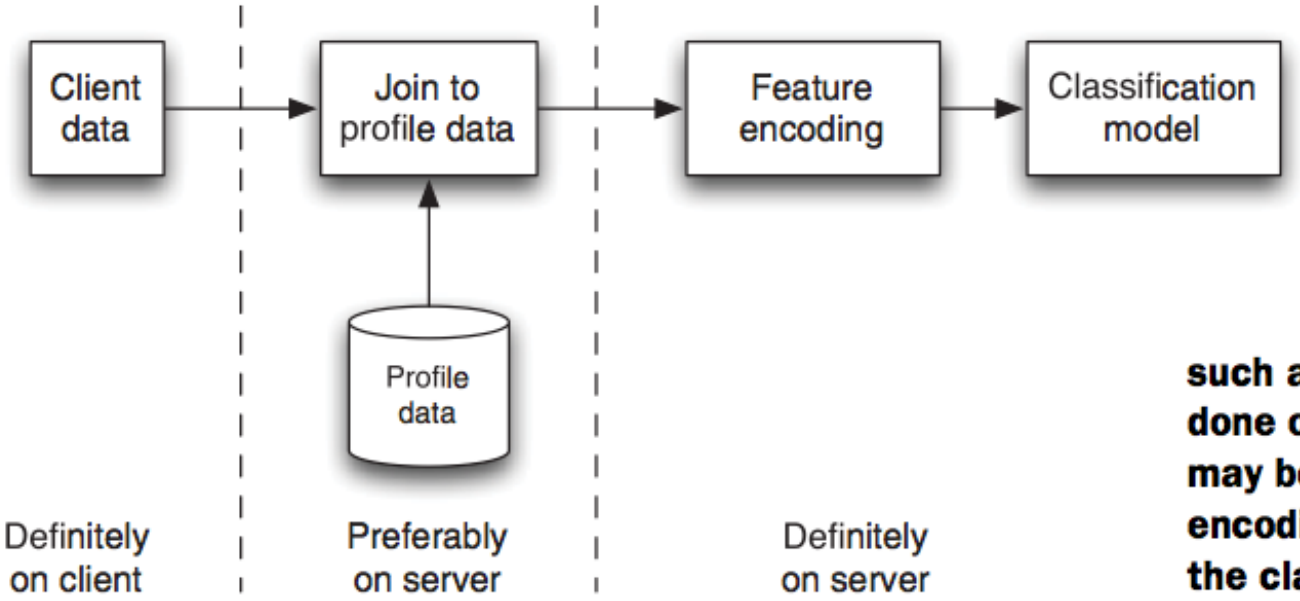
Speed Up reading and encoding

For many applications, the methods for reading, parsing, and encoding data as feature vectors that were discussed in previous chapters are good enough. It's important to note, however, that the SGD classifiers in Mahout only support training on a single machine rather than in parallel. This limitation can lead to slow training times if you have vats of training data. Look at a profiler and you'll see that most of the time used in the Mahout SGD model training API is spent in preparing the data for the training algorithm, not in doing the actual learning itself.

To speed this up, you have to go to a lower level than is typical in Java programs. The `String` data type, for instance, is handy in that it comes with good innate support for Unicode and lots of libraries for parsing, regex matching, and so on. Unfortunately, this happy generality also has costs in terms of complexity. It's common for training data to be subject to much more stringent assumptions about character set and content than is typical for general text.

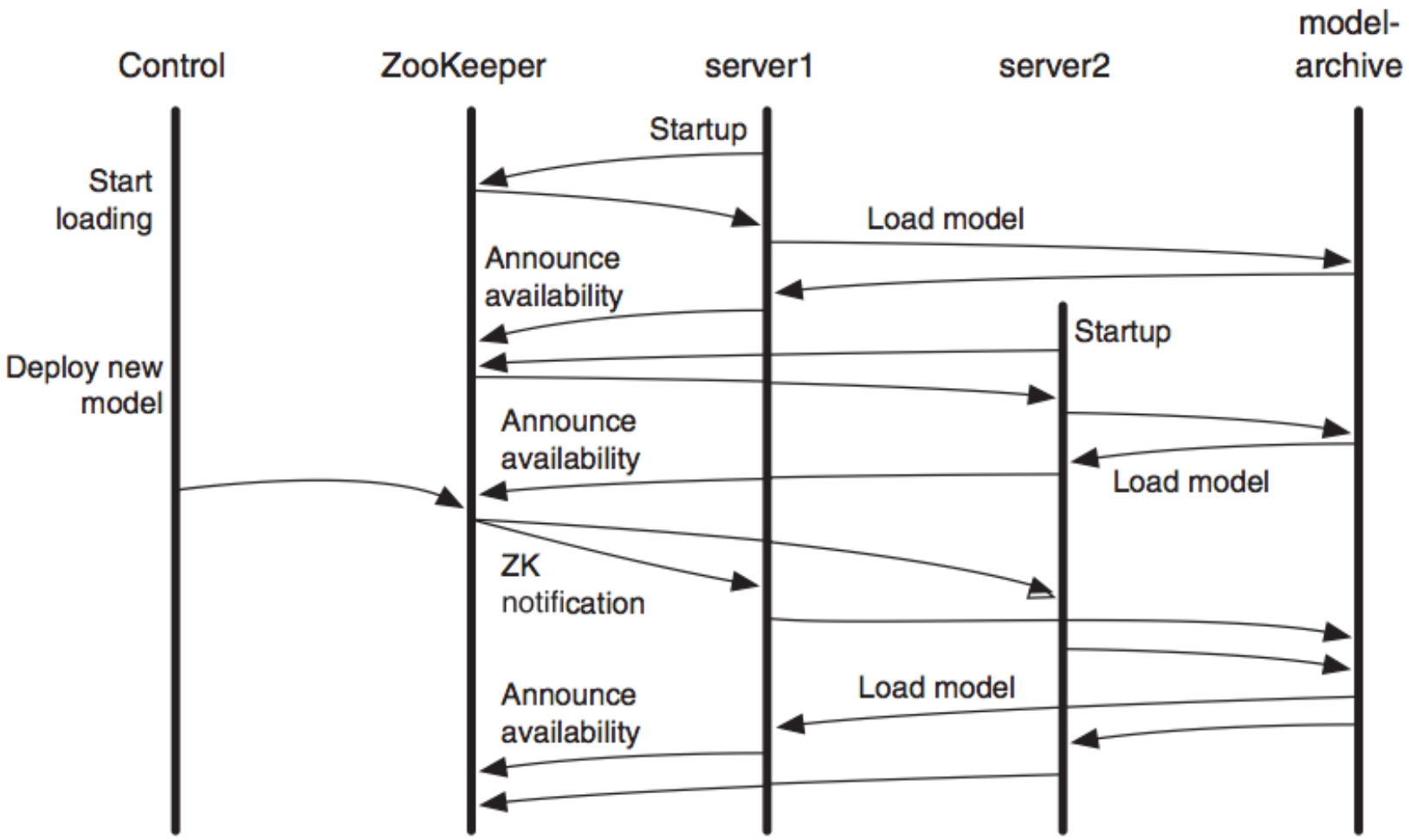
Another major speed bump that training programs encounter is a copying style of processing. In this style, data is often processed by creating immutable strings that represent lines of input. These lines are segmented into lists of new immutable strings representing input fields. Then, if these fields represent text, they're further segmented into new immutable strings during tokenization. The tokens in the text may be stemmed, which involves creating yet another generation of immutable strings.

Integrating Mahout Classifiers



Joins with profile data, such as user profiles, are preferably done on the server, but these joins may be done by the client. Feature encoding should definitely be done on the classification server.

Model deployments

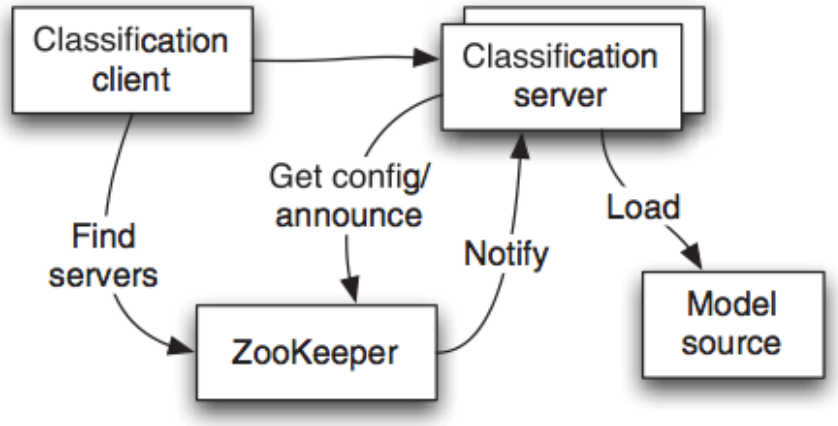


Apache ZooKeeper allows you to connect to a small cluster of servers that provide an API to access what looks a lot like an ordinary filesystem.
Zookeepr does simple create, replace, and delete function and a variety of change notifications and consistency guarantees.

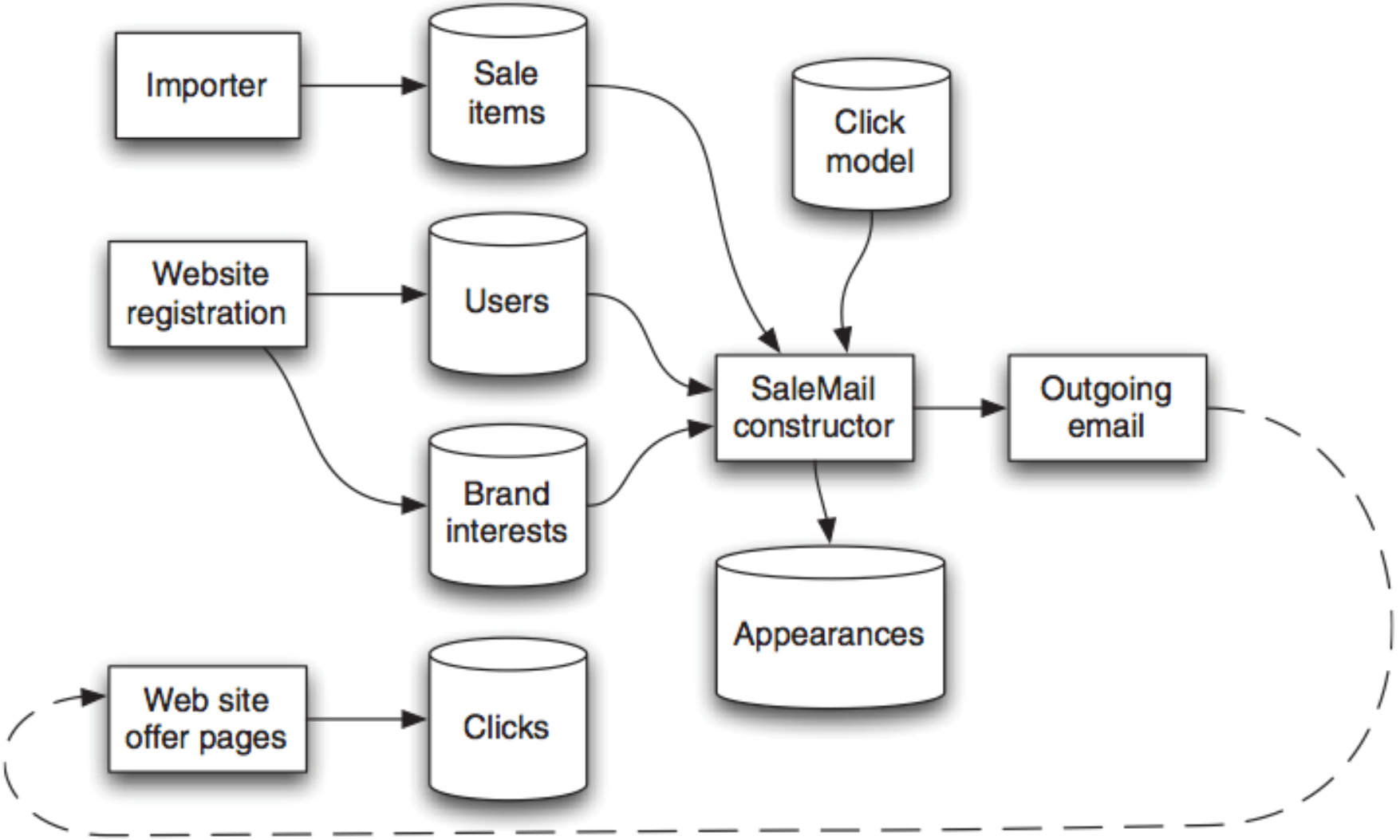
Example: Thrift-based classification server

Communication between the classification client and server will be handled in this example using Apache Thrift (http://thrift.apache.org/). Thrift is an Apache project that allows client-server applications to be built very simply.

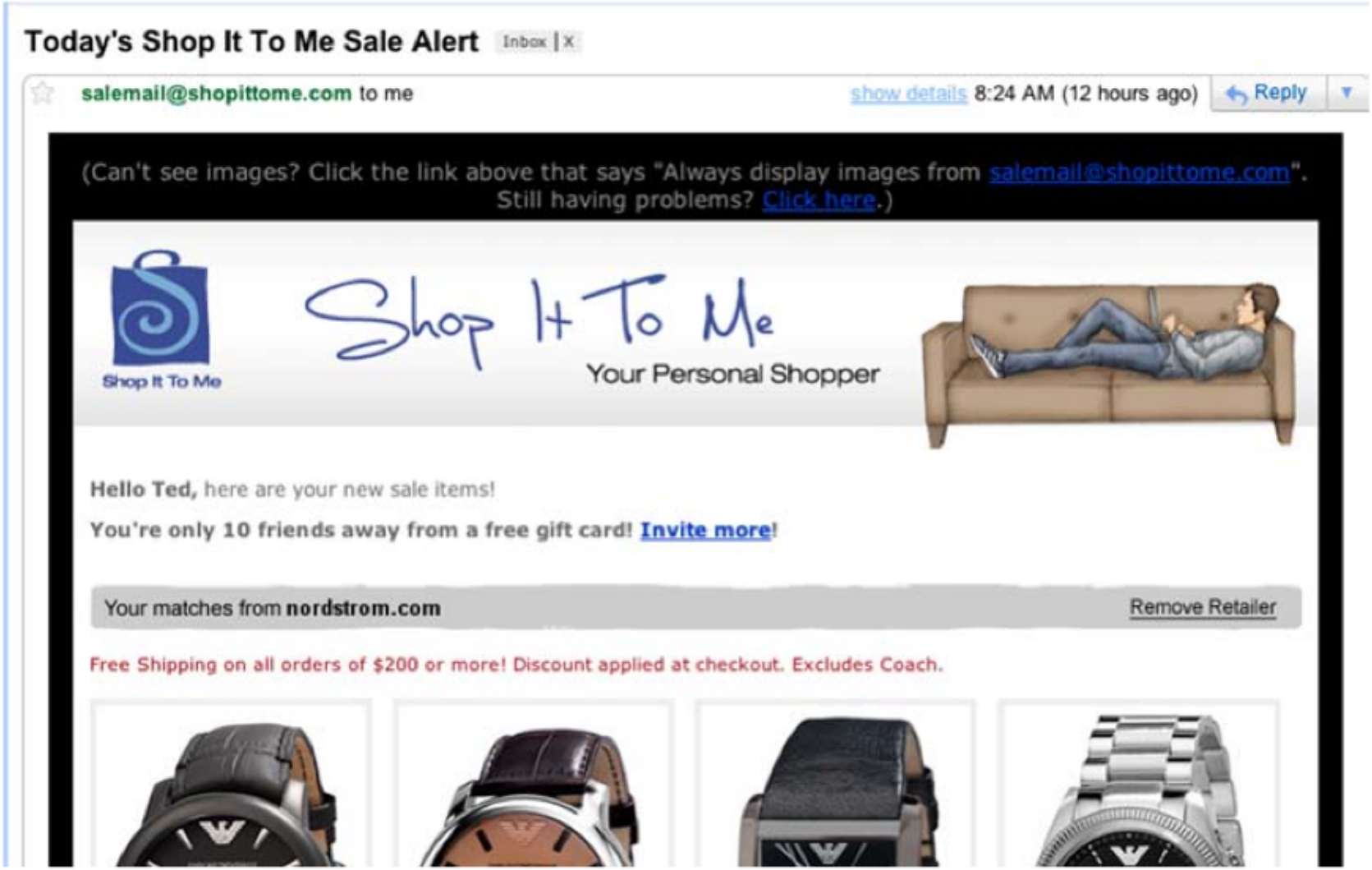
The coordination of multiple servers and between server and client will be handled in this example using Apache ZooKeeper (http://zookeeper.apache.org/).



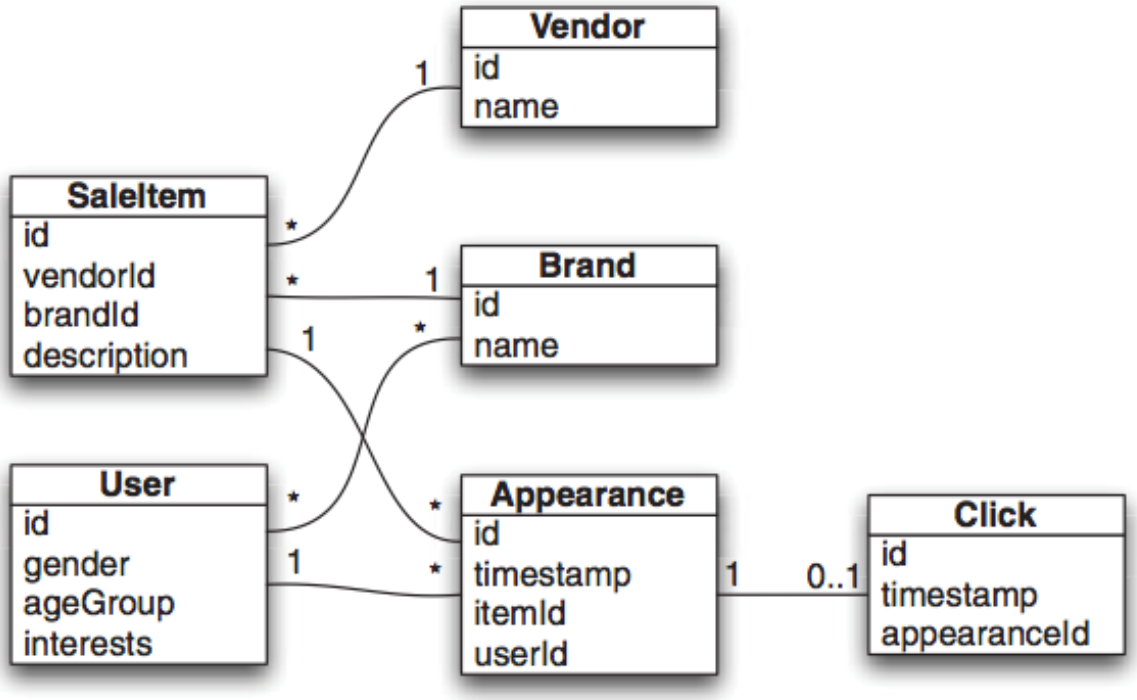
Use Case: Shop It To Me



User Interface

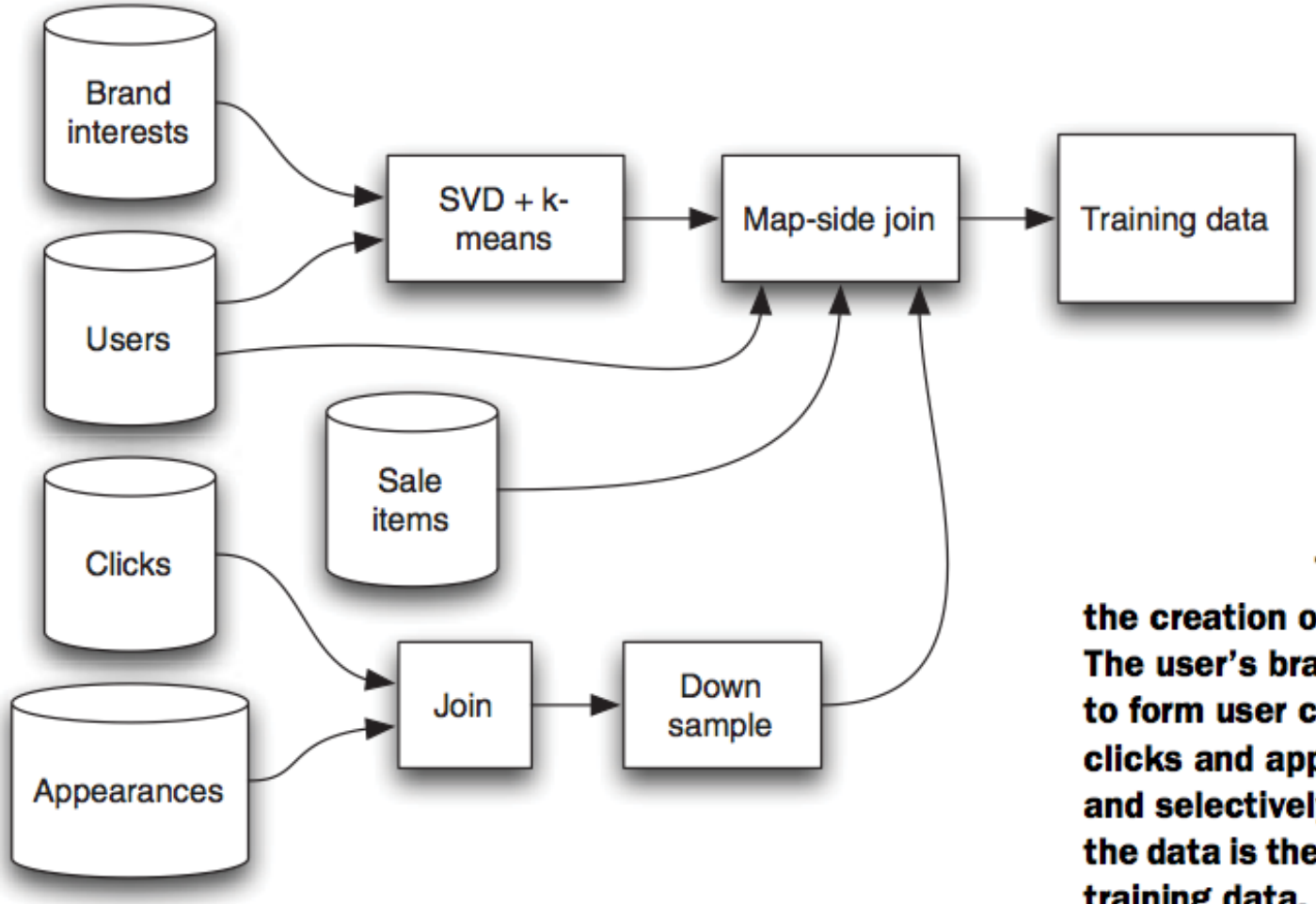


Database Tables



Simplified UML view of database tables used to support the data flow at Shop It To Me. Each sale item has a unique vendor and brand but can appear many times. Likewise, a user may indicate interest in multiple brands but will see many items. Each appearance may result in at most one click.

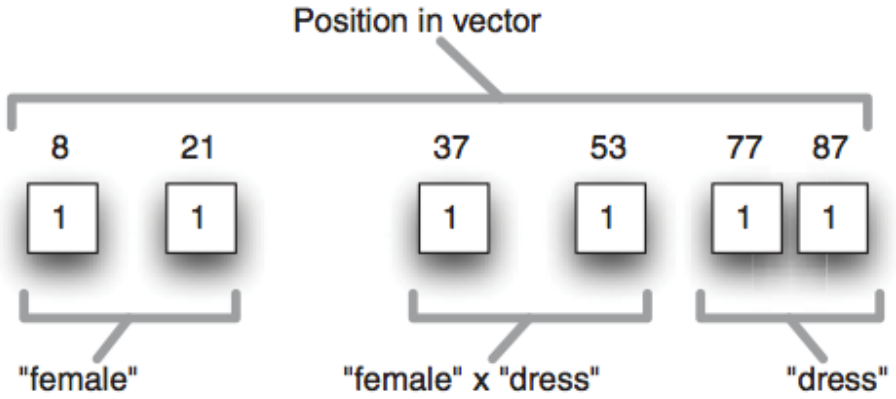
Training the Model



The data flow during the creation of model training data. The user's brand interests are used to form user clusters. In parallel, clicks and appearances are joined and selectively downsampled. All of the data is then joined to produce the training data.

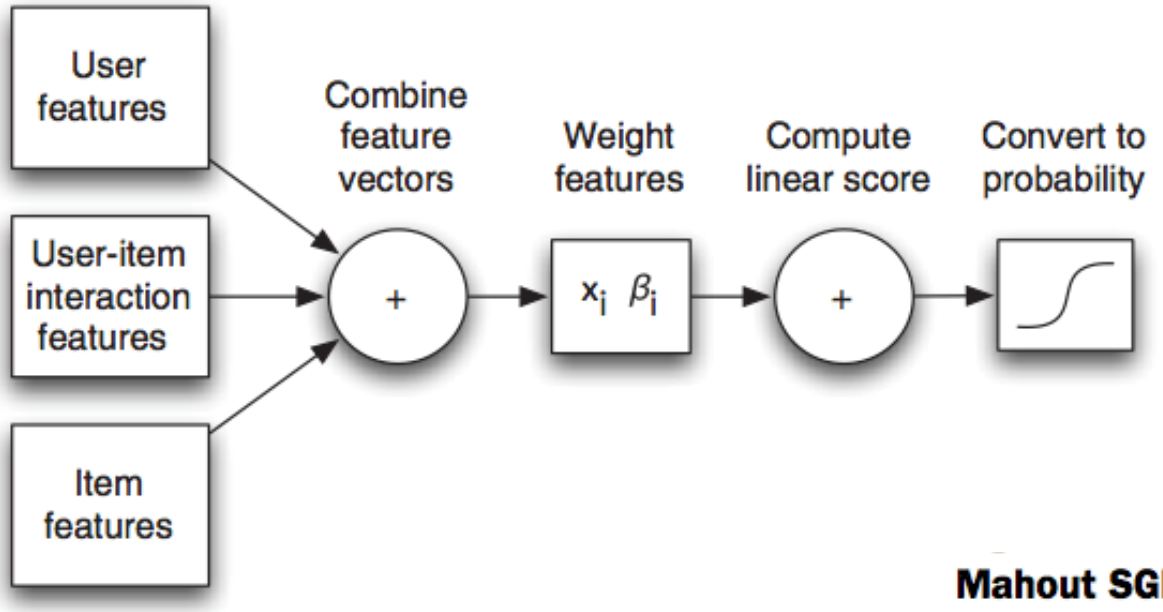
Feature vector encoding

Fields in a possible training record for a simplified email marketing, click-through prediction model. The record contains user-specific, item-specific, and user-item interaction variables. Real models of this sort have many more fields than are shown in this conceptual example.



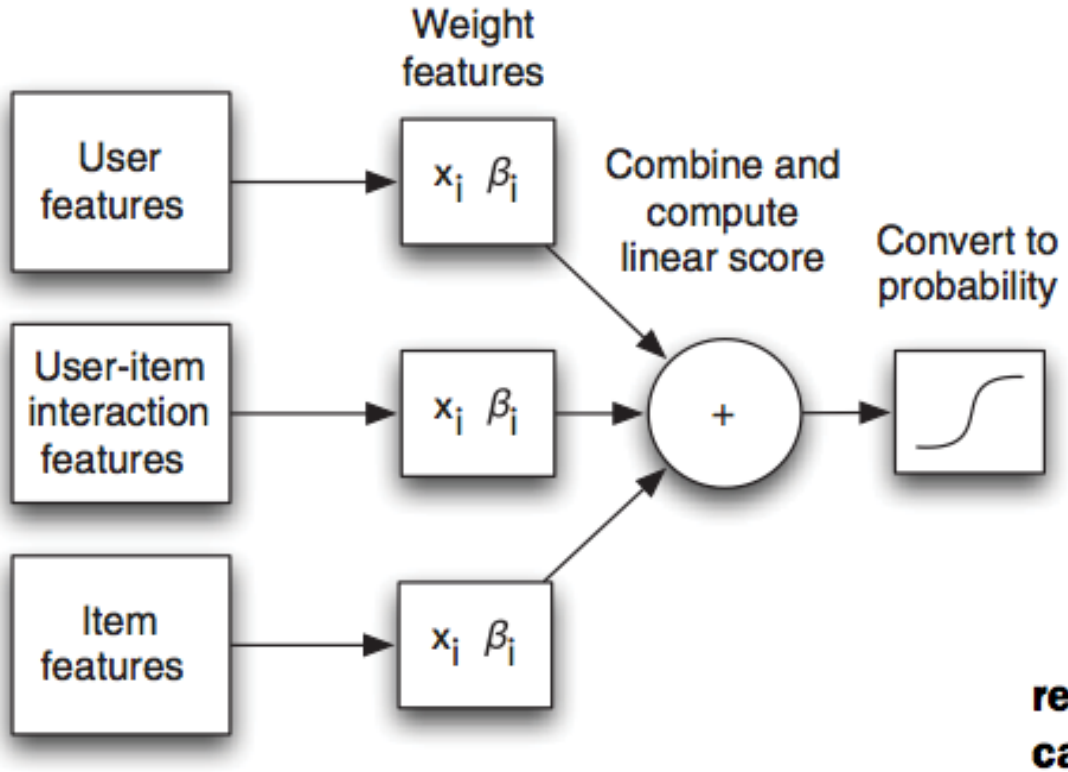
Interaction features need to be hashed to locations different from the locations of their component features.

Linear Combination of feature vectors



How model scoring works for the Mahout SGD logistic regression model

Model can be rearranged



How model scoring can be rearranged to allow precomputation and caching of intermediate results

Using caching and partial model evaluation

Columbia University

```
public class ModelEvaluator {
  private OnlineLogisticRegression model;
  private List<Item> items = Lists.newArrayList();
  private Map<Item, Double> itemCache = Maps.newHashMap();
  private Map<Long, Double> interactionCache = Maps.newHashMap();
  private FeatureEncoder encoder = new FeatureEncoder();
  public List<ScoredItem> topItems(User u, int limit) {
    Vector userVector =
      new RandomAccessSparseVector(model.numFeatures());
    encoder.addUserFeatures(u, userVector);
    double userScore = model.classifyScalarNoLink(userVector);
    PriorityQueue<ScoredItem> r =
          new PriorityQueue<ScoredItem>();
    for (Item item : items) {
      Double itemScore = itemCache.get(item);
      if (itemScore == null) {
        Vector v = new RandomAccessSparseVector(model.numFeatures());
        encoder.addItemFeatures(item, v);
        itemScore = model.classifyScalarNoLink(v);
        itemCache.put(item, itemScore);
      }
      long code = encoder.interactionHash(u, item);
      Double interactionScore = interactionCache.get(code);
      if (interactionScore == null) {
        Vector v = new RandomAccessSparseVector(model.numFeatures());
        encoder.addInteractions(u, item, v);
        interactionScore = model.classifyScalarNoLink(v);
        interactionCache.put(code, interactionScore);
      }
      double score = userScore + itemScore
                   + interactionScore;
      r.add(new ScoredItem(score, item));
      while (r.size() > limit) {
        r.poll();
      }
    }
    return Lists.newArrayList(r);
  }
}
```

1. Receive from other code
2. Score user only once
3. Accumulate best priority queue
4. Compute score if not found
5. Look for cached interaction effect
6. Combine components

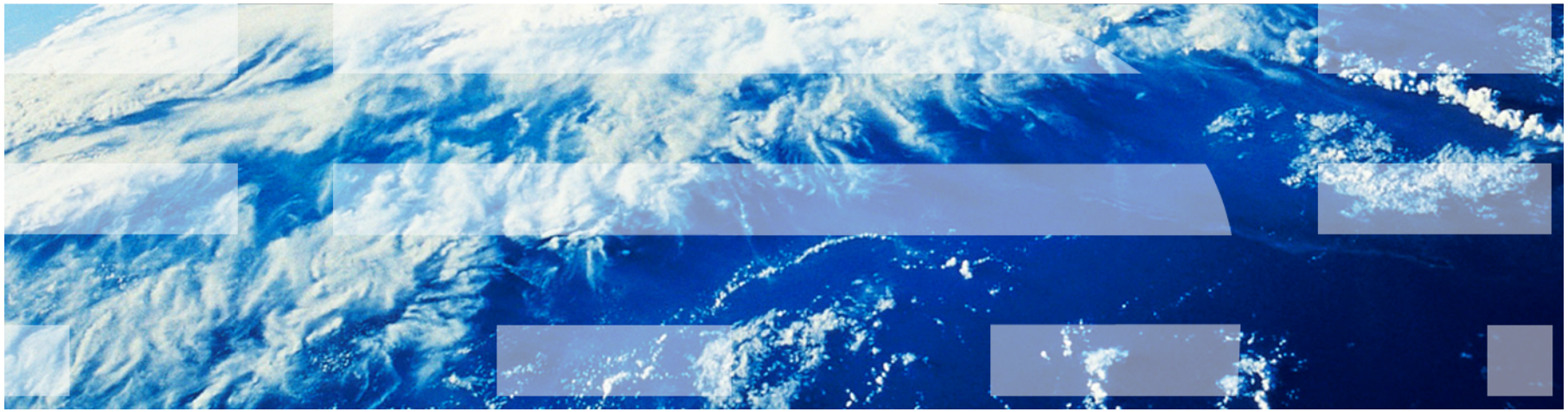
Look for cached item score

Questions?

E6893 Big Data Analytics:

Demo Session for Classification

Ruichi Yu



Oct 30, 2014

Mahout Classification

Mahout provides:

1. Naive Bayes
2. Hidden Markov Models
3. Logistic Regression
4. Random Forests

Naive Bayes classification:

Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features

Naive Bayes classification example:

Twenty Newsgroups Classification Example

Naive Bayes classification example:

The 20 newsgroups dataset is a collection of approximately 20,000 newsgroup documents, evenly across 20 different newsgroups. Mahout CBayes classifier to create a model that would classify a new document into one of the 20 newsgroups.

Prerequisites:

Mahout
Maven

Part 1: Use existed model

1. Download Mahout:
https://mahout.apache.org/general/downloads.html

2. Download Mahout-trunk:
git clone git://git.apache.org/mahout.git mahout-trunk

3. For Maven users please include the following snippet in your pom under mahout-trunk forder:

```
<dependency>
    <groupId>org.apache.mahout</groupId>
    <artifactId>mahout-core</artifactId>
    <version>${mahout.version}</version>
</dependency>
```

Part 1: Use existed model

4. If running Hadoop in cluster mode, start the hadoop daemons by executing the following commands:
$ cd $HADOOP_HOME/bin
$ ./start-all.sh

Running locally:
$ export MAHOUT_LOCAL=true

5. Before running, please make sure you have already set up javahome
export JAVA_HOME=/Library/Java/Home

6. In the trunk directory of Mahout, compile and install Mahout:
   $ cd $MAHOUT_HOME
   $ mvn -DskipTests clean install

Part 1: Use existed model

7. Run the 20 newsgroups example script by executing:

$ ./examples/bin/classify-20newsgroups.sh

8. Please select the algorithm you would like to use. Here we choose 1.

Then you can see the results.

Part 1: Use existed model

```
      0       2       0       0       |  413          j     = rec.sport.baseball
0       0       0       1       0       0       1       0       0       3       394
      0       0       0       1       |  400          k     = rec.sport.hockey
0       1       1       0       0       1       0       0       0       1       0
      1       1       0       0       |  417          l     = sci.crypt
2       4       0       14      6       7       12      2       2       2       0
      0       2       2       2       |  437          m     = sci.electronics
3       1       0       0       1       0       0       3       0       0       0
      0       1       1       0       |  385          n     = sci.med
0       1       0       1       2       1       0       1       0       1       1
      2       2       2       2       |  414          o     = sci.space
3       0       0       1       1       0       0       1       1       0       1
7     1       0       3       0       |  428          p     = soc.religion.christian
0       1       0       0       0       0       0       0       0       0       0
      365     0       0       0       |  369          q     = talk.politics.mideast
0       0       0       0       0       0       0       0       1       0       0
      2       323     0       5       |  334          r     = talk.politics.guns
34      0       0       1       0       0       0       0       0       2       1
      3       2       172     6       |  253          s     = talk.religion.misc
1       0       1       0       0       0       0       1       0       0       0
      4       11      4       275     |  301          t     = talk.politics.misc

=========================================================
Statistics
---------------------------------------------------------
Kappa                                     0.8549
Accuracy                                 88.9064%
Reliability                              84.4083%
Reliability (standard deviation)          0.2217
Weighted precision                        0.8891
Weighted recall                           0.8891
Weighted F1 score                         0.8864
```

Part 2: Train your own model

1. Set up your path:(very important)

export MAHOUT_HOME=/Users/Rich/Documents/Courses/Fall2014/BigData/mahout-distribution-0.9/mahout-trunk/bin

export MAHOUT_CONF_DIR=/Users/Rich/Documents/Courses/Fall2014/BigData/mahout-distribution-0.9/mahout-trunk/src/conf

2. Build your working directory

export WORK_DIR=/Users/Rich/Documents/Courses/Fall2014/BigData/mahout-distribution-0.9/WorkDir

mkdir -p ${WORK_DIR}

Part 2: Train your own model

3. Download and extract the 20news-bydate.tar.gz from the 20newsgroups dataset to the working directory:

```
curl http://people.csail.mit.edu/jrennie/20Newsgroups/20news-bydate.tar.gz -o ${WORK_DIR}/20news-bydate.tar.gz

$ mkdir -p ${WORK_DIR}/20news-bydate
$ cd ${WORK_DIR}/20news-bydate && tar xzf ../20news-bydate.tar.gz && cd .. && cd ..
$ mkdir ${WORK_DIR}/20news-all
$ cp -R ${WORK_DIR}/20news-bydate/*/* ${WORK_DIR}/20news-all
```

Part 2: Train your own model

4. Convert the full 20 newsgroups dataset into a < Text, Text > :
SequenceFile is a hadoop class which allows us to write arbitary (key, value) pairs into it

Important Hint here:
Please use the full path of mahout!!

/Users/Rich/Documents/Courses/Fall2014/BigData/mahout-distribution-0.9/mahout-trunk/bin/mahout seqdirectory -i ${WORK_DIR}/20news-all -o ${WORK_DIR}/20news-seq -ow

Part 2: Train your own model

5. Convert and preprocesses the dataset into a < Text, VectorWritable > SequenceFile containing term frequencies for each document:

```
/Users/Rich/Documents/Courses/Fall2014/BigData/
mahout-distribution-0.9/mahout-trunk/bin/mahout
seq2sparse -i ${WORK_DIR}/20news-seq -o $
{WORK_DIR}/20news-vectors -lnorm -nv -wt tfidf
```

Part 2: Train your own model

6.Split the preprocessed dataset into training and testing sets:

```
/Users/Rich/Documents/Courses/Fall2014/BigData/mahout-distribution-0.9/mahout-trunk/bin/mahout split -i ${WORK_DIR}/20news-vectors/tfidf-vectors --trainingOutput ${WORK_DIR}/20news-train-vectors --testOutput ${WORK_DIR}/20news-test-vectors  --randomSelectionPct 40 --overwrite --sequenceFiles -xm sequential
```

Part 2: Train your own model

7.Train the classifier:

Important Hint here:
abc is the path you store the labelindex. You can change it to other name

/Users/Rich/Documents/Courses/Fall2014/BigData/mahout-distribution-0.9/mahout-trunk/bin/mahout trainnb -i ${WORK_DIR}/20news-train-vectors -el -o ${WORK_DIR}/model -li ${WORK_DIR}/abc -ow -c

Part 2: Train your own model

8. Test the classifier:
/Users/Rich/Documents/Courses/Fall2014/BigData/mahout-distribution-0.9/mahout-trunk/bin/mahout testnb -i ${WORK_DIR}/20news-test-vectors -m ${WORK_DIR}/model -l ${WORK_DIR}/abc -ow -o ${WORK_DIR}/20news-testing -c

Part 2: Train your own model

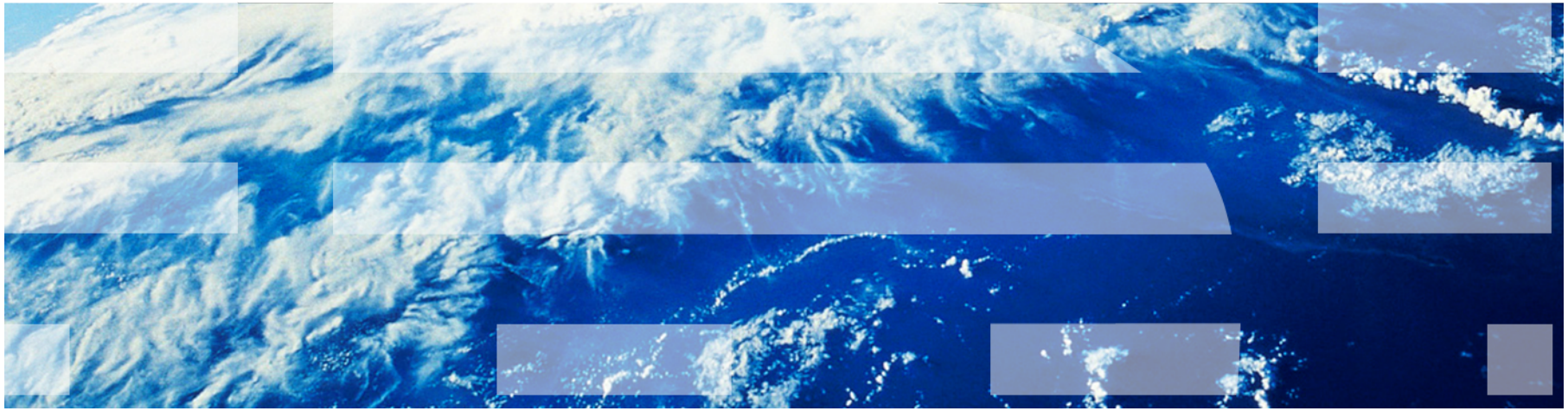
```
0       0       0       0       0       0       0       1       1       5       426
        0       0       0       1       |  435            k     = rec.sport.hockey
0       2       1       0       0       0       0       0       0       1       0
        0       2       0       1       |  406            l     = sci.crypt
1       2       0       5       6       2       10      3       2       2       4
        0       1       1       2       |  388            m     = sci.electronics
1       0       0       0       1       2       2       1       0       0       0
        0       0       2       2       |  394            n     = sci.med
1       1       0       0       1       0       2       1       0       0       1
        2       0       0       1       |  381            o     = sci.space
8       1       1       2       1       0       1       0       1       0       0
2       4       1       4       0       |  381            p     = soc.religion.christian
0       1       0       0       0       0       0       0       0       2       0
        390     2       0       2       |  397            q     = talk.politics.mideast
1       0       0       0       1       2       2       1       0       2       1
        1       361     2       9       |  390            r     = talk.politics.guns
21      0       0       0       0       0       1       0       0       1       1
        4       5       187     4       |  239            s     = talk.religion.misc
2       0       1       1       0       0       1       2       0       3       0
        4       16      3       295     |  335            t     = talk.politics.misc

=======================================================
Statistics
-------------------------------------------------------
Kappa                                        0.8576
Accuracy                                    89.2947%
Reliability                                 84.8391%
Reliability (standard deviation)             0.216
Weighted precision                           0.8941
Weighted recall                              0.8929
Weighted F1 score                            0.8908
```

E6893 Big Data Analytics:

Demo Session II: Mahout working with Eclipse and Maven for Collaborative Filtering

Aonan Zhang

Dept. of Electrical Engineering



October 9th, 2014

The Apache Mahout™ project's goal is to build a scalable machine learning library

Applicable Models

Latest release version 0.9 has

- User and Item based recommenders
- Matrix factorization based recommenders
- K-Means, Fuzzy K-Means clustering
- Latent Dirichlet Allocation
- Singular Value Decomposition
- Logistic regression classifier
- (Complementary) Naive Bayes classifier
- Random forest classifier
- High performance java collections
- A vibrant community

1. Download Eclipse

- https://www.eclipse.org/downloads/

2. Install Maven

- -Help -Install New Software
- -Add -Name: m2eclipse -Location: http://download.eclipse.org/technology/m2e/releases (Google "install m2eclipse")

1. Collaborative Filtering

Users

Items

1		2		4		5
	3	1		2		
5		3		4	3	
	1		1	2		3
	5		4			

1. Start a Maven project:

- -File -New -Other -Maven Project
- -maven-archetype-quickstart

2. Add Mahout dependency in pom.xml (your version might be 0.9)

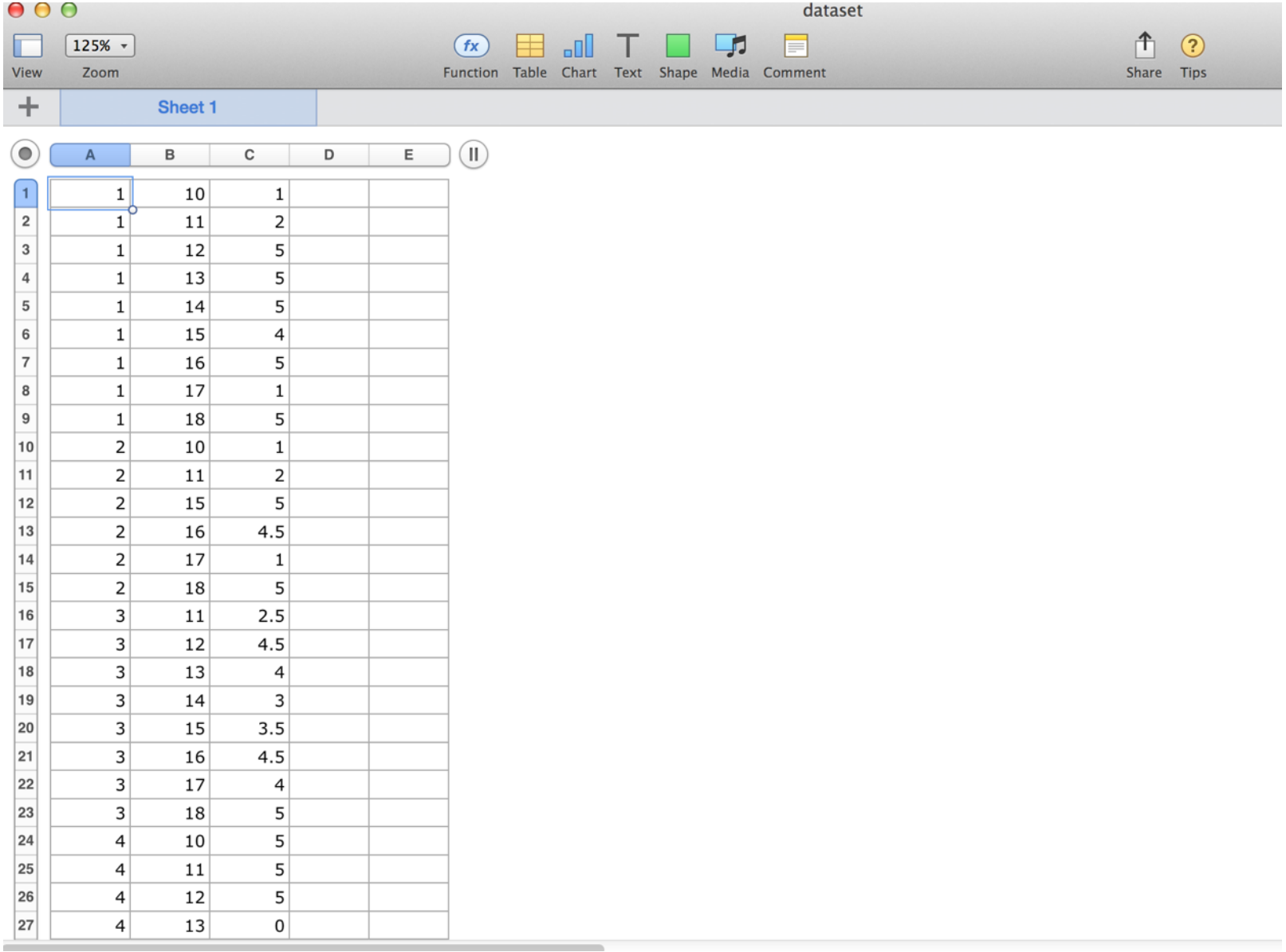
```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSche
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.prediction</groupId>
  <artifactId>RecommendApp</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>RecommendApp</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.apache.mahout</groupId>
      <artifactId>mahout-core</artifactId>
      <version>0.7</version>
</dependency>

    <dependency>
```

3. Copy data file into the project

- Go to https://mahout.apache.org/users/recommender/userbased-5-minutes.html and download the data

- Create data/dataset.csv

Mahout Working with Eclipse and Maven



	A	B	C	D	E
1	1	10	1		
2	1	11	2		
3	1	12	5		
4	1	13	5		
5	1	14	5		
6	1	15	4		
7	1	16	5		
8	1	17	1		
9	1	18	5		
10	2	10	1		
11	2	11	2		
12	2	15	5		
13	2	16	4.5		
14	2	17	1		
15	2	18	5		
16	3	11	2.5		
17	3	12	4.5		
18	3	13	4		
19	3	14	3		
20	3	15	3.5		
21	3	16	4.5		
22	3	17	4		
23	3	18	5		
24	4	10	5		
25	4	11	5		
26	4	12	5		
27	4	13	0		

4. Create a recommender: edit App.java

```
22 public class App
23 {
24     public static void main( String[] args ) throws IOException, TasteException
25     {
26         DataModel model = new FileDataModel(new File("data/dataset.csv"));
27         UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
28         UserNeighborhood neighborhood = new ThresholdUserNeighborhood(0.1, similarity, model);
29         UserBasedRecommender recommender = new GenericUserBasedRecommender(model, neighborhood, similarity);
30         List<RecommendedItem> recommendations = recommender.recommend(2, 3);
31         for (RecommendedItem recommendation : recommendations) {
32           System.out.println(recommendation);
33         }
34     }
35 }
```

5. Import packages

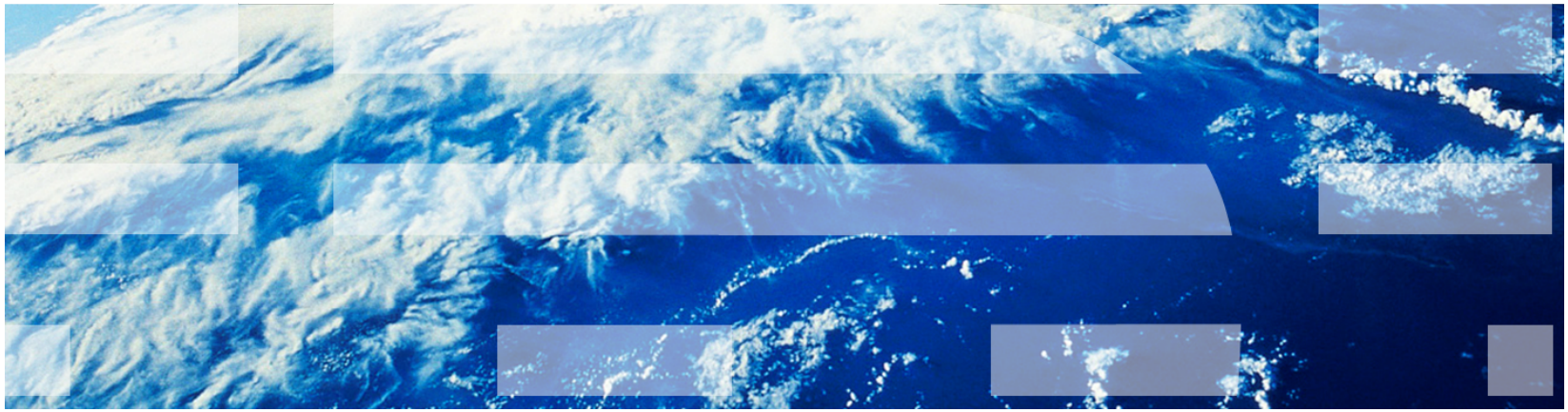
```
3 import java.io.File;
4 import java.io.IOException;
5 import java.util.List;
6
7 import org.apache.mahout.cf.taste.common.TasteException;
8 import org.apache.mahout.cf.taste.impl.model.file.FileDataModel;
9 import org.apache.mahout.cf.taste.impl.neighborhood.ThresholdUserNeighborhood;
10 import org.apache.mahout.cf.taste.impl.recommender.GenericUserBasedRecommender;
11 import org.apache.mahout.cf.taste.impl.similarity.PearsonCorrelationSimilarity;
12 import org.apache.mahout.cf.taste.model.DataModel;
13 import org.apache.mahout.cf.taste.neighborhood.UserNeighborhood;
14 import org.apache.mahout.cf.taste.recommender.RecommendedItem;
15 import org.apache.mahout.cf.taste.recommender.UserBasedRecommender;
16 import org.apache.mahout.cf.taste.similarity.UserSimilarity;
17
```

6. Run eclipse and finish!

```
RecommendedItem[item:12, value:4.8328104]
RecommendedItem[item:13, value:4.6656213]
RecommendedItem[item:14, value:4.331242]
```

- You may also want to evaluate the recommender. See https://mahout.apache.org/users/recommender/userbased-5-minutes.html for details

Demo Session 3: Mahout for Recommendation



Apache Mahout: What?

Scalable machine learning library.

Mahout and its associated frameworks are Java-based and therefore platform-independent, so you should be able to use it with any platform that can run a modern JVM.
Note that Mahout requires Java 6.
-Mahout in Action

Download Link: http://mahout.apache.org/general/downloads.html

Latest Release: 0.9 - mahout-distribution-0.9.tar.gz

MacOS:
brew install mahout

Say, we want to run collaborative filtering:

- Collaborative filtering—producing recommendations based on, and only based on, knowledge of users' relationships to items.
- These techniques require no knowledge of the properties of the items themselves. This is, in a way, an advantage.
- This recommender framework doesn't care whether the items are books, theme parks, flowers, or even other people, because nothing about their attributes enters into any of the input.

Example: Collaborative Filtering

Listing 2.1. Recommender input file, intro.csv

```
1,101,5.0
1,102,3.0
1,103,2.5

2,101,2.0
2,102,2.5
2,103,5.0
2,104,2.0

3,101,2.5
3,104,4.0
3,105,4.5
3,107,5.0

4,101,5.0
4,103,3.0
4,104,4.5
4,106,4.0

5,101,4.0
5,102,3.0
5,103,2.0
5,104,4.0
5,105,3.5
5,106,4.0
```

User 1 has preference 3.0 for item 102

User ID, item ID, preference value

To run locally:
export MAHOUT_LOCAL="any value"

bin/mahout recommenditembased -s SIMILARITY_LOGLIKELIHOOD -i /path/to/input/file -o /path/to/output/folder/ —numRecommendations 1

Note: The output folder should not exist.

Output:

1 [104:2.8088317]
2 [105:3.5743618]
3 [103:4.336442]
4 [105:3.6903737]
5 [107:3.663558]

The recommender engine recommended book 104 to user 1, book 105 to user and so on…

To run on hadoop:

unset MAHOUT_LOCAL

export HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop/

MAHOUT_CONF_DIR=/Users/bhavdeepsethi/Downloads/mahout-distribution-0.9/conf
(if not installed from brew)

Running with Hadoop- Problem. 0.9 does not have support for Hadoop 2.x

Support will come in Mahout 1.0

https://issues.apache.org/jira/browse/MAHOUT-1329

So we have to build from source:

http://mahout.apache.org/developers/buildingmahout.html

Pre-requistie: Git and Maven

MacOS:
brew install git
brew install maven

Ubuntu:
sudo apt-get install git
sudo apt-get install maven

Steps:
git clone git://git.apache.org/mahout.git mahout-trunk

mvn clean package -Dhadoop2.version=2.5.1 -Dhbase.version=0.98.6.1-hadoop2 -DskipTests

Data Set
http://www.grouplens.org/system/files/ml-100k.zip

hdfs dfs -mkdir /input

hdfs dfs -put ~Downloads/ml-100k/u.data /input

hdfs dfs -ls /input/

Run:
hadoop jar /Users/bhavdeepsethi/CU/BigDataAnalytics/mahout-again/mahout-trunk/mrlegacy/target/mahout-mrlegacy-1.0-SNAPSHOT-job.jar org.apache.mahout.cf.taste.hadoop.item.RecommenderJob -s SIMILARITY_COOCCURRENCE --input /input/u.data --output outputNew

Apache Mahout: Example

Output:

```
1	[845:5.0,550:5.0,546:5.0,25:5.0,531:5.0,529:5.0,527:5.0,31:5.0,515:5.0,514:5.0]
2	[546:5.0,288:5.0,11:5.0,25:5.0,531:5.0,527:5.0,515:5.0,508:5.0,496:5.0,483:5.0]
3
[137:5.0,284:5.0,508:4.8327274,248:4.826923,285:4.80597,845:4.754717,124:4.7058825,319:4.703242,293:4
.6792455,591:4.6629214]
4	[748:5.0,1296:5.0,546:5.0,568:5.0,538:5.0,508:5.0,483:5.0,475:5.0,471:5.0,876:5.0]
5	[732:5.0,550:5.0,9:5.0,546:5.0,11:5.0,527:5.0,523:5.0,514:5.0,511:5.0,508:5.0]
6	[739:5.0,9:5.0,546:5.0,11:5.0,25:5.0,531:5.0,528:5.0,527:5.0,526:5.0,521:5.0]
7	[879:5.0,845:5.0,751:5.0,750:5.0,748:5.0,746:5.0,742:5.0,739:5.0,735:5.0,732:5.0]
```

Each line represents the recommendation for a user. The first number is the user id and the 10 number pairs represents a movie id and a score.
If we are looking at the first line for example, it means that for the user 1, the 10 best recommendations are for the movies 845, 550, 546, 25 ,531, 529, 527, 31, 515, 514.

Source:

Example:
kMeans Clustering:

Data:
Download: http://archive.ics.uci.edu/ml/databases/synthetic_control/synthetic_control.data

hdfs dfs -mkdir -p /user/<username/whoami>/testdata
hdfs dfs -put synthetic_control.data testdata/synthetic_control.data

hadoop jar examples/target/mahout-examples-1.0-SNAPSHOT-job.jar org.apache.mahout.clustering.syntheticcontrol.kmeans.Job

Any Questions?