Some of the slides have been provided through the courtesy of Dr. Ching-Yung Lin at Columbia University
Remind -- Hadoop-related Apache Projects

- **Ambari™**: A web-based tool for provisioning, managing, and monitoring Hadoop clusters. It also provides a dashboard for viewing cluster health and ability to view MapReduce, Pig and Hive applications visually.
- **Avro™**: A data serialization system.
- **Cassandra™**: A scalable multi-master database with no single points of failure.
- **Chukwa™**: A data collection system for managing large distributed systems.
- **HBase™**: A scalable, distributed database that supports structured data storage for large tables.
- **Hive™**: A data warehouse infrastructure that provides data summarization and ad hoc querying.
- **Mahout™**: A Scalable machine learning and data mining library.
- **Pig™**: A high-level data-flow language and execution framework for parallel computation.
- **Spark™**: A fast and general compute engine for Hadoop data. Spark provides a simple and expressive programming model that supports a wide range of applications, including ETL, machine learning, stream processing, and graph computation.
- **Tez™**: A generalized data-flow programming framework, built on Hadoop YARN, which provides a powerful and flexible engine to execute an arbitrary DAG of tasks to process data for both batch and interactive use-cases.
- **ZooKeeper™**: A high-performance coordination service for distributed applications.
Key Components of Mahout

Collaborative Filtering
- User-Based Collaborative Filtering - single machine
- Item-Based Collaborative Filtering - single machine / MapReduce
- Matrix Factorization with Alternating Least Squares - single machine / MapReduce
- Matrix Factorization with Alternating Least Squares on Implicit Feedback - single machine / MapReduce
- Weighted Matrix Factorization, SVD++, Parallel SGD - single machine

Classification
- Logistic Regression - trained via SGD - single machine
- Naive Bayes/ Complementary Naive Bayes - MapReduce
- Random Forest - MapReduce
- Hidden Markov Models - single machine
- MultiLayer Perceptron - single machine

Clustering
- Canopy Clustering - single machine / MapReduce (deprecated, will be removed once Streaming k-Means is stable enough)
- k-Means Clustering - single machine / MapReduce
- Fuzzy k-Means - single machine / MapReduce
- Streaming k-Means - single machine / MapReduce
- Spectral Clustering - MapReduce
25 April 2014 - Goodbye MapReduce

The Mahout community decided to move its codebase onto modern data processing systems that offer a richer programming model and more efficient execution than Hadoop MapReduce. **Mahout will therefore reject new MapReduce algorithm implementations from now on.** We will however keep our widely used MapReduce algorithms in the codebase and maintain them.

We are building our future implementations on top of a **DSL for linear algebraic operations** which has been developed over the last months. Programs written in this DSL are automatically optimized and executed in parallel on **Apache Spark**.
Spark 1.1 and related efforts

Spark RDD (Resilient Distributed Dataset) API

Ooyala Job Server
Hive on Spark
Pig on Spark

Spark Streaming real-time
GraphX Graph (alpha)
MLLib machine learning
Spark SQL

RDD-Based Tables
RDD-Based Matrices
RDD-Based Graphs
DStream’s: Streams of RDD’s

HDFS, S3, Cassandra
YARN, Mesos, Standalone

Releases:
Spark 1.0(.2): Aug 05, 2014;
Spark 1.1(.0): Sept 11, 2014

http://www.meetup.com/spark-users/files/
Spark Concepts

Spark focuses on one such class of applications: those that reuse a working set of data across multiple parallel operations. This includes many iterative machine learning algorithms, as well as interactive data analysis tools.

Linear regression performance – Spark vs Hadoop

Spark MLlib v1.0 & v1.1

Sparse vector support
Decision trees
Linear algebra: SVD and PCA

**Algorithms**: SVD via Lanczos, multiclass support in decision tree, logistic regression with L-BFGS, nonnegative matrix factorization, streaming linear regression

**Feature extraction and transformation**: scaling, normalization, tf-idf, Word2Vec

**Statistics**: sampling (core), correlations, hypothesis testing, random data generation

**Performance and scalability**: major improvement to decision tree, tree aggregation

**Python API**: decision tree, statistics, linear methods
The R Project for Statistical Computing

PCA 5 vars
princomp(x = data, cor = cor)

(1-3) 60%

Clustering 4 groups
Factor 1 [41%]  Factor 3 [19%]

V. De Geneve
**RHive**

The RHive framework serves as a bridge between the R language and Hive. RHive delivers the rich statistical libraries and algorithms of R to data stored in Hadoop by extending Hive’s SQL-like query language (HiveQL) with R-specific functions. Through the RHive functions, you can use HiveQL to apply R statistical models to data in your Hadoop cluster that you have cataloged using Hive.

**RHadoop**

Another open source framework available to R programmers is RHadoop, a collection of packages intended to help manage the distribution and analysis of data with Hadoop. Three packages of note — rmr2, rhdfs, and rhbase — provide most of RHadoop’s functionality:

- **rmr2**: The rmr2 package supports translation of the R language into Hadoop-compliant MapReduce jobs (producing efficient, low-level MapReduce code from higher-level R code).

- **rhdfs**: The rhdfs package provides an R language API for file management over HDFS stores. Using rhdfs, users can read from HDFS stores to an R data frame (matrix) and similarly write data from these R matrices back into HDFS storage.

- **rhbase**: rhbase packages provide an R language API as well, but their goal in life is to deal with database management for HBase stores, rather than HDFS files.
Recommender — Inputs

<table>
<thead>
<tr>
<th>User</th>
<th>Item</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>101</td>
<td>5.0</td>
</tr>
<tr>
<td>1</td>
<td>102</td>
<td>3.0</td>
</tr>
<tr>
<td>1</td>
<td>103</td>
<td>2.5</td>
</tr>
<tr>
<td>2</td>
<td>101</td>
<td>2.0</td>
</tr>
<tr>
<td>2</td>
<td>102</td>
<td>2.5</td>
</tr>
<tr>
<td>2</td>
<td>103</td>
<td>5.0</td>
</tr>
<tr>
<td>2</td>
<td>104</td>
<td>2.0</td>
</tr>
<tr>
<td>3</td>
<td>101</td>
<td>2.5</td>
</tr>
<tr>
<td>3</td>
<td>104</td>
<td>4.0</td>
</tr>
<tr>
<td>3</td>
<td>105</td>
<td>4.5</td>
</tr>
<tr>
<td>3</td>
<td>107</td>
<td>5.0</td>
</tr>
<tr>
<td>4</td>
<td>101</td>
<td>5.0</td>
</tr>
<tr>
<td>4</td>
<td>103</td>
<td>3.0</td>
</tr>
</tbody>
</table>
User-based Recommendation — Scenario I

**ADULT:** I’m looking for a CD for a teenager.

**EMPLOYEE:** OK, what does this teenager like?

**ADULT:** Oh, you know, what all the young kids like these days.

**EMPLOYEE:** What kind of music or bands?

**ADULT:** It’s all noise to me. I don’t know.

**EMPLOYEE:** Uh, well... I guess lots of young people are buying this boy band album here by New 2 Town?

**ADULT:** Sold!
User-based Recommendation Algorithms

for every item \( i \) that \( u \) has no preference for yet
for every other user \( v \) that has a preference for \( i \)
    compute a similarity \( s \) between \( u \) and \( v \)
    incorporate \( v \)'s preference for \( i \), weighted by \( s \), into a running average
return the top items, ranked by weighted average

for every other user \( w \)
    compute a similarity \( s \) between \( u \) and \( w \)
    retain the top users, ranked by similarity, as a neighborhood \( n \)
for every item \( i \) that some user in \( n \) has a preference for,
    but that \( u \) has no preference for yet
for every other user \( v \) in \( n \) that has a preference for \( i \)
    compute a similarity \( s \) between \( u \) and \( v \)
    incorporate \( v \)'s preference for \( i \), weighted by \( s \), into a running average
Example Recommender Code via Mahout

class RecommenderIntro {
    public static void main(String[] args) throws Exception {
        DataModel model =
            new FileDataModel (new File("intro.csv"));  // Load data file

        UserSimilarity similarity =
            new PearsonCorrelationSimilarity (model);
        UserNeighborhood neighborhood =
            new NearestNUserNeighborhood (2, similarity, model);

        Recommender recommender = new GenericUserBasedRecommender (model, neighborhood, similarity);

        List<RecommendedItem> recommendations =
            recommender.recommend(1, 1);

        for (RecommendedItem recommendation : recommendations) {
            System.out.println(recommendation);
        }
    }
}
Process and output of the example

**Recommendation for Person 1:**

- Item 104 > Item 106
- Item 107 is not favored
Update data

FileDataModel supports update files. These are just more data files that are read after the main data file, and that overwrite any previously read data. New preferences are added and existing ones are updated. Deletes are handled by providing an empty preference value string.

For example, consider the following update file:

```
1,108,3.0
1,103,
```

This says, “update (or create) user 1’s preference for item 108, and set the value to 3.0” and “remove user 1’s preference for item 103.”

These update files must simply exist in the same directory as the main data file, and their names must begin with the same prefix, up to the first period. For example, if the main data file is foo.txt.gz, the update files might be named foo.1.txt.gz and foo.2.txt.gz.
User Similarity Measurements

- Pearson Correlation Similarity
- Euclidean Distance Similarity
- Cosine Measure Similarity
- Spearman Correlation Similarity
- Tanimoto Coefficient Similarity (Jaccard coefficient)
- Log-Likelihood Similarity
Pearson Correlation Similarity

\[ \rho_{X,Y} = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y} \]

- \( \mu_X = E[X] \)
- \( \mu_Y = E[Y] \)
- \( \sigma_X^2 = E[(X - E[X])^2] = E[X^2] - E[X]^2 \)
- \( \sigma_Y^2 = E[(Y - E[Y])^2] = E[Y^2] - E[Y]^2 \)
- \( E[(X - \mu_X)(Y - \mu_Y)] = E[(X - E[X])(Y - E[Y])] = E[XY] - E[X] E[Y], \)

<table>
<thead>
<tr>
<th>Item 101</th>
<th>Item 102</th>
<th>Item 103</th>
<th>Correlation with user 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>5.0</td>
<td>3.0</td>
<td>2.5</td>
</tr>
<tr>
<td>User 2</td>
<td>2.0</td>
<td>2.5</td>
<td>5.0</td>
</tr>
<tr>
<td>User 3</td>
<td>2.5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>User 4</td>
<td>5.0</td>
<td>-</td>
<td>3.0</td>
</tr>
<tr>
<td>User 5</td>
<td>4.0</td>
<td>3.0</td>
<td>2.0</td>
</tr>
</tbody>
</table>
On Pearson Similarity

Three problems with the Pearson Similarity:

1. Not take into account of the number of items in which two users’ preferences overlap. (e.g., 2 overlap items ==> 1, more items may not be better.)
2. If two users overlap on only one item, no correlation can be computed.
3. The correlation is undefined if either series of preference values are identical.

Adding Weighting WEIGHTED as 2nd parameter of the constructor can cause the resulting correlation to be pushed towards 1.0, or -1.0, depending on how many points are used.
Euclidean Distance Similarity

\[ d(p, q) = d(q, p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}. \]

\[ \text{Similarity} = \frac{1}{1 + d} \]

<table>
<thead>
<tr>
<th></th>
<th>Item 101</th>
<th>Item 102</th>
<th>Item 103</th>
<th>Distance</th>
<th>Similarity to user 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>5.0</td>
<td>3.0</td>
<td>2.5</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>User 2</td>
<td>2.0</td>
<td>2.5</td>
<td>5.0</td>
<td>3.937</td>
<td>0.203</td>
</tr>
<tr>
<td>User 3</td>
<td>2.5</td>
<td>-</td>
<td>-</td>
<td>2.500</td>
<td>0.286</td>
</tr>
<tr>
<td>User 4</td>
<td>5.0</td>
<td>-</td>
<td>3.0</td>
<td>0.500</td>
<td>0.667</td>
</tr>
<tr>
<td>User 5</td>
<td>4.0</td>
<td>3.0</td>
<td>2.0</td>
<td>1.118</td>
<td>0.472</td>
</tr>
</tbody>
</table>
Cosine Similarity

\[
similarity = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}
\]

Cosine similarity and Pearson similarity get the same results if data are normalized (mean == 0).
Spearman Correlation Similarity

\[ \rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} \]

\[ d_i = x_i - y_i \]

Pearson value on the relative ranks

<table>
<thead>
<tr>
<th>Variable $X_i$</th>
<th>Position in the ascending order</th>
<th>Rank $x_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>2</td>
<td>$\frac{2 + 3}{2} = 2.5$</td>
</tr>
<tr>
<td>1.2</td>
<td>3</td>
<td>$\frac{2 + 3}{2} = 2.5$</td>
</tr>
<tr>
<td>2.3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>18</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item 101</th>
<th>Item 102</th>
<th>Item 103</th>
<th>Correlation to user 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>3.0</td>
<td>2.0</td>
<td>1.0</td>
</tr>
<tr>
<td>User 2</td>
<td>1.0</td>
<td>2.0</td>
<td>3.0</td>
</tr>
<tr>
<td>User 3</td>
<td>1.0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>User 4</td>
<td>2.0</td>
<td>-</td>
<td>1.0</td>
</tr>
<tr>
<td>User 5</td>
<td>3.0</td>
<td>2.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Tanimoto (Jaccard) Coefficient Similarity

\[ J(A, B) = \frac{|A \cap B|}{|A \cup B|} \]

Discard preference values

<table>
<thead>
<tr>
<th>Item 101</th>
<th>Item 102</th>
<th>Item 103</th>
<th>Item 104</th>
<th>Item 105</th>
<th>Item 106</th>
<th>Item 107</th>
<th>Similarity to user 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>1.0</td>
</tr>
<tr>
<td>User 2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>0.75</td>
</tr>
<tr>
<td>User 3</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0.17</td>
</tr>
<tr>
<td>User 4</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0.4</td>
</tr>
<tr>
<td>User 5</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0.5</td>
</tr>
</tbody>
</table>
Log-Likelihood Similarity

Assess how unlikely it is that the overlap between the two users is just due to chance.

\[
D = -2 \ln \left( \frac{\text{likelihood for null model}}{\text{likelihood for alternative model}} \right) \\
= -2 \ln(\text{likelihood for null model}) + 2 \ln(\text{likelihood for alternative model})
\]

<table>
<thead>
<tr>
<th>Item 101</th>
<th>Item 102</th>
<th>Item 103</th>
<th>Item 104</th>
<th>Item 105</th>
<th>Item 106</th>
<th>Item 107</th>
<th>Similarity to user 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User 1</strong></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>0.90</td>
</tr>
<tr>
<td><strong>User 2</strong></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>0.84</td>
</tr>
<tr>
<td><strong>User 3</strong></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0.55</td>
</tr>
<tr>
<td><strong>User 4</strong></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>0.16</td>
</tr>
<tr>
<td><strong>User 5</strong></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0.55</td>
</tr>
</tbody>
</table>
Performance measurements

- Spearman: 0.8
- Tanimoto: 0.82
- Log-Likelihood: 0.73
- Euclidean: 0.75
- Pearson (weighted): 0.77
- Pearson: 0.89

<table>
<thead>
<tr>
<th></th>
<th>Item 1</th>
<th>Item 2</th>
<th>Item 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual</td>
<td>3.0</td>
<td>5.0</td>
<td>4.0</td>
</tr>
<tr>
<td>Estimate</td>
<td>3.5</td>
<td>2.0</td>
<td>5.0</td>
</tr>
<tr>
<td>Difference</td>
<td>0.5</td>
<td>3.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Average difference</td>
<td>(\frac{(0.5 + 3.0 + 1.0)}{3} = 1.5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Root-mean-square</td>
<td>(\sqrt{(0.5^2 + 3.0^2 + 1.0^2) / 3} = 1.8484)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DataModel model = new GroupLensDataModel (new File("ratings.dat"));
RecommenderEvaluator evaluator =
    new AverageAbsoluteDifferenceRecommenderEvaluator ();
RecommenderBuilder recommenderBuilder = new RecommenderBuilder () {
    @Override
    public Recommender buildRecommender(
        DataModel model) throws TasteException {
        UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
        UserNeighborhood neighborhood =
            new NearestNUserNeighborhood(100, similarity, model);
        return new GenericUserBasedRecommender(
            model, neighborhood, similarity);
    }
};
double score = evaluator.evaluate(
    recommenderBuilder, null, model, 0.95, 0.05);
System.out.println(score);

10 nearest neighbors: 0.98
100 nearest neighbors: 0.89
500 nearest neighbors: 0.75

95% of training; 5% of testing
Selecting the number of neighbors

Based on number of neighbors

```
new NearestNUserNeighborhood(100, similarity, model);
```

Based on a fixed threshold, e.g., 0.7 or 0.5

```
new ThresholdUserNeighborhood(0.7, similarity, model)
```
ADULT: I’m looking for a CD for a teenage boy.

EMPLOYEE: What kind of music or bands does he like?

ADULT: He wears a Bowling In Hades T-shirt all the time and seems to have all of their albums. Anything else you’d recommend?

EMPLOYEE: Well, about everyone I know that likes Bowling In Hades seems to like the new Rock Mobster album.
Item-based recommendation algorithm

for every item \(i\) that \(u\) has no preference for yet
for every item \(j\) that \(u\) has a preference for
compute a similarity \(s\) between \(i\) and \(j\)
add \(u\)'s preference for \(j\), weighted by \(s\), to a running average
return the top items, ranked by weighted average
public Recommender buildRecommender(DataModel model) throws TasteException {
    ItemSimilarity similarity = new PearsonCorrelationSimilarity(model);
    return new GenericItemBasedRecommender(model, similarity);
}

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>PearsonCorrelationSimilarity</td>
<td>0.75</td>
</tr>
<tr>
<td>PearsonCorrelationSimilarity + weighting</td>
<td>0.75</td>
</tr>
<tr>
<td>EuclideanDistanceSimilarity</td>
<td>0.76</td>
</tr>
<tr>
<td>EuclideanDistanceSimilarity + weighting</td>
<td>0.78</td>
</tr>
<tr>
<td>TanimotoCoefficientSimilarity</td>
<td>0.77</td>
</tr>
<tr>
<td>LogLikelihoodSimilarity</td>
<td>0.77</td>
</tr>
</tbody>
</table>

One thing you may notice is that this recommender setup runs significantly faster. That’s not surprising, given that the data set has about 70,000 users and 10,000 items. Item-based recommenders are generally faster when there are fewer items than users.
Clustering a collection involves three things:

- *An algorithm*—This is the method used to group the books together.
- *A notion of both similarity and dissimilarity*—In the previous discussion, we relied on your assessment of which books belonged in an existing stack and which should start a new one.
- *A stopping condition*—In the library example, this might be the point beyond which books can’t be stacked anymore, or when the stacks are already quite dissimilar.
Clustering — on feature plane
Clustering example

(1, 1)
(2, 1)
(1, 2)
(2, 2)
(3, 3)
(8, 8)
(8, 9)
(9, 8)
(9, 9)
Steps on clustering

1. Generate Vectors from input data
2. Write Vectors to input directory
3. Write initial cluster centers
4. Run clustering job
5. Read clusters from output directory
K-means clustering in action. Starting with three random points as centroids (top left), the map stage (top right) assigns each point to the cluster nearest to it. In the reduce stage (bottom left), the associated points are averaged out to produce the new location of the centroid, leaving you with the final configuration (bottom right). After each iteration, the final configuration is fed back into the same loop until the centroids come to rest at their final positions.
Making initial cluster centers
Hadoop k-means clustering jobs

In Mahout, the MapReduce version of the k-means algorithm is instantiated using the \texttt{KMeansDriver} class. The class has just a single entry point—the \texttt{runJob} method.

- The Hadoop configuration.
- The \texttt{SequenceFile} containing the input Vectors.
- The \texttt{SequenceFile} containing the initial Cluster centers.
- The similarity measure to be used. We’ll use \texttt{EuclideanDistanceMeasure} as the measure of similarity and experiment with the others later.
- The convergenceThreshold. If in an iteration, the centroids don’t move more than this distance, no further iterations are done and clustering stops.
- The number of iterations to be done. This is a hard limit; the clustering stops if this threshold is reached.
K-means clustering running as MapReduce job

SequenceFile containing Vectors

chunk-0

chunk-0

...

chunk-n

Parallel Mappers

Parallel Reducers

Each Mapper reads in the centroid vector at startup

Each Reducer gets the partial sums of all points of a cluster from each Mapper and recomputes the centroids

Each Mapper computes nearest centroid for a vector

Repeat until converged
public static final double[][] points = {
    {1, 1}, {2, 1}, {1, 2},
    {2, 2}, {3, 3}, {8, 8},
    {9, 8}, {8, 9}, {9, 9}};

public static void writePointsToFile(List<Vector> points,
    String fileName,
    FileSystem fs,
    Configuration conf) throws IOException {
    Path path = new Path(fileName);
    SequenceFile.Writer writer = new SequenceFile.Writer(fs, conf,
        path, LongWritable.class, VectorWritable.class);
    long recNum = 0;
    VectorWritable vec = new VectorWritable();
    for (Vector point : points) {
        vec.set(point);
        writer.append(new LongWritable(recNum++), vec);
    }
    writer.close();
}

public static List<Vector> getPoints(double[][] raw) {
    List<Vector> points = new ArrayList<Vector>();
    for (int i = 0; i < raw.length; i++) {
        double[] fr = raw[i];
        Vector vec = new RandomAccessSparseVector(fr.length);
        vec.assign(fr);
        points.add(vec);
    }
    return points;
}
public static void main(String args[]) throws Exception {
    int k = 2;

    List<Vector> vectors = getPoints(points);

    File testData = new File("testdata");
    if (!testData.exists()) {
        testData.mkdir();
    }
    testData = new File("testdata/points");
    if (!testData.exists()) {
        testData.mkdir();
    }

    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(conf);
    writePointsToFile(vectors,
        "testdata/points/file1", fs, conf);

    Path path = new Path("testdata/clusters/part-00000");
    SequenceFile.Writer writer
        = new SequenceFile.Writer(
            fs, conf, path, Text.class, Cluster.class);

    for (int i = 0; i < k; i++) {
        Vector vec = vectors.get(i);
        Cluster cluster = new Cluster(
            vec, i, new EuclideanDistanceMeasure());
        writer.append(new Text(cluster.getIdentifier()), cluster);
    }

    writer.close();
KMeansDriver.run(conf, new Path("testdata/points"),
    new Path("testdata/clusters"),
    new Path("output"), new EuclideanDistanceMeasure(),
    0.001, 10, true, false);

SequenceFile.Reader reader
    = new SequenceFile.Reader(fs,
            new Path("output/" + Cluster.CLUSTERED_POINTS_DIR
                + "/part-m-00000"), conf);

IntWritable key = new IntWritable();
WeightedVectorWritable value = new WeightedVectorWritable();
while (reader.next(key, value)) {
    System.out.println(
        value.toString() + " belongs to cluster ")
        + key.toString());
}
reader.close();
HelloWorld clustering scenario result

1.0: [1.000, 1.000] belongs to cluster 0
1.0: [2.000, 1.000] belongs to cluster 0
1.0: [1.000, 2.000] belongs to cluster 0
1.0: [2.000, 2.000] belongs to cluster 0
1.0: [3.000, 3.000] belongs to cluster 0
1.0: [8.000, 8.000] belongs to cluster 1
1.0: [9.000, 8.000] belongs to cluster 1
1.0: [8.000, 9.000] belongs to cluster 1
1.0: [9.000, 9.000] belongs to cluster 1
Euclidean distance measure

\[ d = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_n - b_n)^2} \]

Squared Euclidean distance measure

\[ d = (a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_n - b_n)^2 \]

Manhattan distance measure

\[ d = |a_1 - b_1| + |a_2 - b_2| + \ldots + |a_n - b_n| \]
Manhattan and Cosine distances

**Manhattan distance measure**

\[ d = |a_1 - b_1| + |a_2 - b_2| + \ldots + |a_n - b_n| \]

**Cosine distance measure**

\[ d = 1 - \frac{(a_1 b_1 + a_2 b_2 + \ldots + a_n b_n)}{\left(\sqrt{a_1^2 + a_2^2 + \ldots + a_n^2}\right)\left(\sqrt{b_1^2 + b_2^2 + \ldots + b_n^2}\right)} \]
Tanimoto distance and weighted distance

**Tanimoto distance measure**

\[
d = 1 - \frac{(a_1 b_1 + a_2 b_2 + \ldots + a_n b_n)}{\sqrt{(a_1^2 + a_2^2 + \ldots + a_n^2)} + \sqrt{(b_1^2 + b_2^2 + \ldots + b_n^2)} - (a_1 b_1 + a_2 b_2 + \ldots + a_n b_n)}
\]

**Weighted distance measure**

Mahout also provides a `WeightedDistanceMeasure` class, and implementations of Euclidean and Manhattan distance measures that use it. A weighted distance measure is an advanced feature in Mahout that allows you to give weights to different dimensions in order to either increase or decrease the effect of a dimension.
## Results comparison

<table>
<thead>
<tr>
<th>Distance measure</th>
<th>Number of iterations</th>
<th>Vectors in cluster 0</th>
<th>Vectors in cluster 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>EuclideanDistanceMeasure</td>
<td>3</td>
<td>0, 1, 2, 3, 4</td>
<td>5, 6, 7, 8</td>
</tr>
<tr>
<td>SquaredEuclideanDistanceMeasure</td>
<td>5</td>
<td>0, 1, 2, 3, 4</td>
<td>5, 6, 7, 8</td>
</tr>
<tr>
<td>ManhattanDistanceMeasure</td>
<td>3</td>
<td>0, 1, 2, 3, 4</td>
<td>5, 6, 7, 8</td>
</tr>
<tr>
<td>CosineDistanceMeasure</td>
<td>1</td>
<td>1</td>
<td>0, 2, 3, 4, 5, 6, 7, 8</td>
</tr>
<tr>
<td>TanimotoDistanceMeasure</td>
<td>3</td>
<td>0, 1, 2, 3, 4</td>
<td>5, 6, 7, 8</td>
</tr>
</tbody>
</table>

![Diagram showing initial centres of clusters 0 and 1 and their respective points](image)
In Mahout, vectors are implemented as three different classes, each of which is optimized for different scenarios: DenseVector, RandomAccessSparseVector, and SequentialAccessSparseVector.

- **DenseVector** can be thought of as an array of doubles, whose size is the number of features in the data. Because all the entries in the array are preallocated regardless of whether the value is 0 or not, we call it *dense*.

- **RandomAccessSparseVector** is implemented as a HashMap between an integer and a double, where only nonzero valued features are allocated. Hence, they’re called as SparseVectors.

- **SequentialAccessSparseVector** is implemented as two parallel arrays, one of integers and the other of doubles. Only nonzero valued entries are kept in it. Unlike the **RandomAccessSparseVector**, which is optimized for random access, this one is optimized for linear reading.
vectorization example

<table>
<thead>
<tr>
<th>Apple</th>
<th>Weight (kg) (0)</th>
<th>Color (1)</th>
<th>Size (2)</th>
<th>Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small, round, green</td>
<td>0.11</td>
<td>510</td>
<td>1</td>
<td>[0.11, 510, 1]</td>
</tr>
<tr>
<td>Large, oval, red</td>
<td>0.23</td>
<td>650</td>
<td>3</td>
<td>[0.23, 650, 3]</td>
</tr>
<tr>
<td>Small, elongated, red</td>
<td>0.09</td>
<td>630</td>
<td>1</td>
<td>[0.09, 630, 1]</td>
</tr>
<tr>
<td>Large, round, yellow</td>
<td>0.25</td>
<td>590</td>
<td>3</td>
<td>[0.25, 590, 3]</td>
</tr>
<tr>
<td>Medium, oval, green</td>
<td>0.18</td>
<td>520</td>
<td>2</td>
<td>[0.18, 520, 2]</td>
</tr>
</tbody>
</table>

[0 => 100 gram, 1 => red, 2 => small]
Hadoop k-means clustering code

KmeansDriver.runJob(hadoopConf,
inputVectorFilesDirPath, clusterCenterFilesDirPath,
outputDir, new EuclideanDistanceMeasure(),
convergenceThreshold, numIterations, true, false);

Mahout reads and writes data using the Hadoop FileSystem class. This provides seamless access to both the local filesystem (via java.io) and distributed filesystems like HDFS and S3FS (using internal Hadoop classes). This way, the same code that works on the local system will also work on the Hadoop filesystem on the cluster, provided the paths to the Hadoop configuration files are correctly set in the environment variables. In Mahout, the bin/mahout shell script finds the Hadoop configuration files automatically from the $HADOOP_CONF environment variable.

$ bin/mahout kmeans -i reuters-vectors/tfidf-vectors/ \
-c reuters-initial-clusters \ 
-o reuters-kmeans-clusters \ 
-dm org.apache.mahout.common.distance.SquaredEuclideanDistanceMeasure \ 
-cd 1.0 -k 20 -x 20 -cl
Canopy clustering to estimate the number of clusters

Tell what size clusters to look for. The algorithm will find the number of clusters that have approximately that size. The algorithm uses two distance thresholds. This method prevents all points close to an already existing canopy from being the center of a new canopy.

Figure 9.3  Canopy clustering: if you start with a point (top left) and mark it as part of a canopy, all the points within distance $T_2$ (top right) are removed from the data set and prevented from becoming new canopies. The points within the outer circle (bottom-right) are also put in the same canopy, but they’re allowed to be part of other canopies. This assignment process is done in a single pass on a mapper. The reducer computes the average of the centroid (bottom right) and merges close canopies.
Running canopy clustering

To run canopy generation over the Reuters data set, execute the canopy program using the Mahout launcher as follows:

```bash
$ bin/mahout canopy -i reuters-vectors/tfidf-vectors \
-o reuters-canopy-centroids \
-dm org.apache.mahout.common.distance.EuclideanDistanceMeasure \
-tl 1500 -t2 2000
```

Within a minute, CanopyDriver will generate the centroids in the output folder.

```bash
$ bin/mahout kmeans -i reuters-vectors/tfidf-vectors \
-o reuters-kmeans-clusters \
-dm org.apache.mahout.common.distance.TanimotoDistanceMeasure \ 
-c reuters-canopy-centroids/clusters-0 -cd 0.1 -ow -x 20 -c1
```

After the clustering is done, use ClusterDumper to inspect the clusters. Some of them are listed here:

- **Id: 21523**: name:
  - Top Terms:
    - tones, wheat, grain, said, usda, corn, us, sugar, export, agriculture

- **Id: 21409**: name:
  - Top Terms:
    - stock, share, shares, shareholders, dividend, said, its, common, board, company

- **Id: 21155**: name:
  - Top Terms:
    - oil, effective, crude, raises, prices, barrel, price, cts, said, dhrs

- **Id: 19658**: name:
  - Top Terms:
    - drug, said, aids, inc, company, its, patent, test, products, food

- **Id: 21323**: name:
  - Top Terms:
    - 7-apr-1987, 11, 10, 12, 07, 09, 15, 16, 02, 17
Other clustering algorithms
Different clustering approaches

**FIXED NUMBER OF CENTERS**

**BOTTOM-UP APPROACH: FROM POINTS TO CLUSTERS VIA GROUPING**

Initial datapoints

Clusters formed after 1st iteration

**TOP-DOWN APPROACH: SPLITTING THE GIANT CLUSTER**
**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.
How does a classification system work?
### When to use Mahout for classification?

<table>
<thead>
<tr>
<th>System size in number of examples</th>
<th>Choice of classification approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 100,000</td>
<td>Traditional, non-Mahout approaches should work very well. Mahout may even be slower for training.</td>
</tr>
<tr>
<td>100,000 to 1 million</td>
<td>Mahout begins to be a good choice. The flexible API may make Mahout a preferred choice, even though there is no performance advantage.</td>
</tr>
<tr>
<td>1 million to 10 million</td>
<td>Mahout is an excellent choice in this range.</td>
</tr>
<tr>
<td>&gt; 10 million</td>
<td>Mahout excels where others fail.</td>
</tr>
</tbody>
</table>
The advantage of using Mahout for classification
# Key terminology for classification

<table>
<thead>
<tr>
<th>Key Idea</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>A computer program that makes decisions; in classification, the output of the training algorithm is a model.</td>
</tr>
<tr>
<td>Training data</td>
<td>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.</td>
</tr>
<tr>
<td>Test data</td>
<td>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.</td>
</tr>
<tr>
<td>Training</td>
<td>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.</td>
</tr>
<tr>
<td>Training example</td>
<td>An entity with features that will be used as input for learning algorithm.</td>
</tr>
<tr>
<td>Feature</td>
<td>A known characteristic of a training or a new example; a feature is equivalent to a characteristic.</td>
</tr>
<tr>
<td>Variable</td>
<td>In this context, the value of a feature or a function of several features. This usage is somewhat different from the use of variable in a computer program.</td>
</tr>
<tr>
<td>Record</td>
<td>A container where an example is stored; such a record is composed of fields.</td>
</tr>
<tr>
<td>Field</td>
<td>Part of a record that contains the value of a feature (a variable).</td>
</tr>
<tr>
<td>Predictor variable</td>
<td>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.</td>
</tr>
<tr>
<td>Target variable</td>
<td>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.</td>
</tr>
</tbody>
</table>
Input and Output of a classification model
## Four types of values for predictor variables

<table>
<thead>
<tr>
<th>Type of value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuous</td>
<td>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.</td>
</tr>
<tr>
<td>Categorical</td>
<td>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.</td>
</tr>
<tr>
<td>Word-like</td>
<td>A word-like value is like a categorical value, but it has an open-ended set of possible values.</td>
</tr>
<tr>
<td>Text-like</td>
<td>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.</td>
</tr>
</tbody>
</table>
Sample data that illustrates all four value types

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>from-address</td>
<td>Word-like</td>
<td>George <a href="mailto:george@fumble-tech.com">george@fumble-tech.com</a></td>
</tr>
<tr>
<td>in-address-book?</td>
<td>Categorical (TRUE, FALSE)</td>
<td>TRUE</td>
</tr>
<tr>
<td>non-spam-words</td>
<td>Text-like</td>
<td>Ted, Mahout, User, lunch</td>
</tr>
<tr>
<td>spam-words</td>
<td>Text-like</td>
<td>available</td>
</tr>
<tr>
<td>unknown-words</td>
<td>Continuous</td>
<td>0</td>
</tr>
<tr>
<td>message-length</td>
<td>Continuous</td>
<td>31</td>
</tr>
</tbody>
</table>
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.
# Workflow in a typical classification project

<table>
<thead>
<tr>
<th>Stage</th>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Training the model</td>
<td>Define target variable. Collect historical data.</td>
</tr>
<tr>
<td>2. Evaluating the model</td>
<td>Run test data. Adjust the input (use different predictor variables, different algorithms, or both).</td>
</tr>
<tr>
<td>3. Using the model in production</td>
<td>Input new examples to estimate unknown target values. Retrain the model as needed.</td>
</tr>
</tbody>
</table>
Mahout classification algorithms

Mahout classification algorithms include:

- Naive Bayesian
- Complementary Naive Bayesian
- Stochastic Gradient Descent (SDG)
- Random Forest
Comparing two types of Mahout Scalable algorithms

- Sequential algorithm
- Parallel algorithm

[Graph showing the comparison of wall clock time with number of training examples, indicating when sequential or parallel algorithms are preferred.]
### Choose algorithm via Mahout

<table>
<thead>
<tr>
<th>Size of data set</th>
<th>Mahout algorithm</th>
<th>Execution model</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small to medium (less than tens of</td>
<td>Stochastic gradient descent (SGD) family:</td>
<td>Sequential, online,</td>
<td>Uses all types of predictor variables; sleek and efficient over the appropriate</td>
</tr>
<tr>
<td>millions of training examples)</td>
<td>OnlineLogisticRegression, CrossFoldLearner,</td>
<td>incremental</td>
<td>data range (up to millions of training examples)</td>
</tr>
<tr>
<td></td>
<td>AdaptiveLogisticRegression</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Support vector machine (SVM)</td>
<td>Sequential</td>
<td>Experimental still; sleek and efficient over the appropriate data range</td>
</tr>
<tr>
<td></td>
<td>Naive Bayes</td>
<td>Parallel</td>
<td>Strongly prefers text-like data; medium to high overhead for training; effective</td>
</tr>
<tr>
<td></td>
<td>Complementary naive Bayes</td>
<td>Parallel</td>
<td>and useful for data sets too large for SGD or SVM</td>
</tr>
<tr>
<td>Small to medium (less than tens of</td>
<td>Random forests</td>
<td>Parallel</td>
<td>Somewhat more expensive to train than naive Bayes; effective and useful</td>
</tr>
<tr>
<td>millions of training examples)</td>
<td></td>
<td></td>
<td>for data sets too large for SGD, but has similar limitations to naive Bayes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Uses all types of predictor variables; high overhead for training; not widely</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>used (yet); costly but offers complex and interesting classifications, handles</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>nonlinear and conditional relationships in data better than other techniques</td>
</tr>
</tbody>
</table>
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 $\alpha$.
- Randomly shuffle examples in the training set.
- Repeat until an approximate minimum is obtained:
  - For $i = 1, 2, \ldots, n$, 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:

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

nonlinear kernels maximize boundary distances; remembering “support vectors”
**Naive Bayes**

<table>
<thead>
<tr>
<th>sex</th>
<th>height (feet)</th>
<th>weight (lbs)</th>
<th>foot size(inches)</th>
</tr>
</thead>
<tbody>
<tr>
<td>male</td>
<td>6</td>
<td>180</td>
<td>12</td>
</tr>
<tr>
<td>male</td>
<td>5.92 (5'11&quot;)</td>
<td>190</td>
<td>11</td>
</tr>
<tr>
<td>male</td>
<td>5.88 (5'7&quot;)</td>
<td>170</td>
<td>12</td>
</tr>
<tr>
<td>male</td>
<td>5.92 (5'11&quot;)</td>
<td>165</td>
<td>10</td>
</tr>
<tr>
<td>female</td>
<td>5</td>
<td>100</td>
<td>6</td>
</tr>
<tr>
<td>female</td>
<td>5.5 (5'6&quot;)</td>
<td>150</td>
<td>8</td>
</tr>
<tr>
<td>female</td>
<td>5.42 (5'5&quot;)</td>
<td>130</td>
<td>7</td>
</tr>
<tr>
<td>female</td>
<td>5.75 (5'9&quot;)</td>
<td>150</td>
<td>9</td>
</tr>
</tbody>
</table>

**Training set:**
- Classifier using Gaussian assumptions:

**Test Set:**
- female

<table>
<thead>
<tr>
<th>sex</th>
<th>mean (height)</th>
<th>variance (height)</th>
<th>mean (weight)</th>
<th>variance (weight)</th>
<th>mean (foot size)</th>
<th>variance (foot size)</th>
</tr>
</thead>
<tbody>
<tr>
<td>male</td>
<td>5.855</td>
<td>3.5033e-02</td>
<td>176.25</td>
<td>1.2292e+02</td>
<td>11.25</td>
<td>9.1667e-01</td>
</tr>
<tr>
<td>female</td>
<td>5.4175</td>
<td>9.7225e-02</td>
<td>132.5</td>
<td>5.5833e+02</td>
<td>7.5</td>
<td>1.6667e+00</td>
</tr>
</tbody>
</table>

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

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

\[
P(\text{male}) = 0.5
\]

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

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

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

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

\[
P(\text{female}) = 0.5
\]

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

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

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

posterior numerator (female) = their product = $5.3778 \cdot 10^{-4}$
Example — classifying 20 newsgroups data set (~ 20K docs)

**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/

<table>
<thead>
<tr>
<th>Comp.graphics</th>
<th>Rec.autos</th>
<th>Sci.crypt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comp.os.ms-windows.mic</td>
<td>Rec.motorcycles</td>
<td>Sci.electronics</td>
</tr>
<tr>
<td>Comp.sys.ibm.pc.hardwa</td>
<td>Rec.sport.baseball</td>
<td>Sci.med</td>
</tr>
<tr>
<td>Comp.sys.mac.hardware</td>
<td>Rec.sport.hockey</td>
<td>Sci.space</td>
</tr>
<tr>
<td>Comp.windows.x</td>
<td>Misc.forsale</td>
<td>Talk.politics.misc</td>
</tr>
<tr>
<td>Talk.politics.guns</td>
<td>Talk.religion.misc</td>
<td>Alt.atheism</td>
</tr>
<tr>
<td>Talk.politics.mideast</td>
<td></td>
<td>Soc.religion.christian</td>
</tr>
</tbody>
</table>

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@qualcomm.qualcomm.com
Nntp-Posting-Host: qualcomm.qualcomm.com
Organization: Qualcomm, Inc., San Diego, CA
Lines: 12

ted@mmsu.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
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.
bin/mahout testclassifier -d 20news-train -m 20news-model\n    -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
388 386 347 377 12   304 18
281 14 43 21
313 22 16
26  69 83 41
45  225 13 11
334 32 367
19  23 15 307 13
16  335 371
393
12  364
11  22
102 160
362
```

Default Category: unknown: 20

<table>
<thead>
<tr>
<th>a = rec.sport.baseball</th>
<th>b = sci.crypt</th>
</tr>
</thead>
<tbody>
<tr>
<td>c = rec.sport.hockey</td>
<td>d = talk.politics.guns</td>
</tr>
<tr>
<td>e = soc.religion.christian</td>
<td>f = sci.electronics</td>
</tr>
<tr>
<td>g = comp.os.ms-windows.misc</td>
<td>h = misc.forsale</td>
</tr>
<tr>
<td>i = talk.religion.misc</td>
<td>j = alt.atheism</td>
</tr>
<tr>
<td>k = comp.windows.x</td>
<td>l = talk.politics.mideast</td>
</tr>
<tr>
<td>m = comp.sys.ibm.pc.hardware</td>
<td>n = comp.sys.mac.hardware</td>
</tr>
<tr>
<td>o = sci.space</td>
<td>p = rec.motorcycles</td>
</tr>
<tr>
<td>q = rec.autos</td>
<td>r = comp.graphics</td>
</tr>
<tr>
<td>s = talk.politics.misc</td>
<td>t = sci.med</td>
</tr>
</tbody>
</table>

Correctly Classified Instances : 6398 84.9442%

Incorrectly Classified Instances : 1134 15.0558%

Total Classified Instances : 7532

=================================================================
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