Evan Sparks and Ameet Talwalkar
UC Berkeley

Collaborators: Tim Kraska\textsuperscript{2}, Virginia Smith\textsuperscript{1}, Xinghao Pan\textsuperscript{1}, Shivaram Venkataraman\textsuperscript{1}, Matei Zaharia\textsuperscript{1}, Rean Griffith\textsuperscript{3}, John Duchi\textsuperscript{1}, Joseph Gonzalez\textsuperscript{1}, Michael Franklin\textsuperscript{1}, Michael I. Jordan\textsuperscript{1}

\textsuperscript{1}UC Berkeley \quad \textsuperscript{2}Brown \quad \textsuperscript{3}VMware
**Problem:** Scalable implementations are difficult for ML Developers...
Problem: Scalable implementations difficult for ML Developers...
Problem: Scalable implementations difficult for ML Developers...
Problem: ML is difficult for End Users...

Too many algorithms...
Problem: ML is difficult for End Users...

Too many algorithms...

Too many knobs...
Problem: ML is difficult for End Users...

- Too many algorithms...
- Too many knobs...

Difficult to debug...
**Problem:** ML is difficult for End Users...

- Too many algorithms...
- Too many knobs...
- Difficult to debug...
- Doesn’t scale...
Problem: ML is difficult for End Users...

- Too many algorithms...
- Too many knobs...
- Difficult to debug...
- Doesn’t scale...
- Doesn’t scale...
- Fast
- Reliable
- Provable
- Accurate
1. Easy scalable ML development (*ML Developers*)
2. User-friendly ML at scale (*End Users*)
1. Easy scalable ML development (*ML Developers*)
2. User-friendly ML at scale (*End Users*)

Along the way, we gain insight into data intensive computing
Vision
MLI Details
Current Status
ML Workflow
Matlab Stack
Matlab Stack

Single Machine
Matlab Stack

- Lapack: low-level Fortran linear algebra library
Matlab Stack

- **Lapack**: low-level Fortran linear algebra library
- **Matlab Interface**
  - Higher-level abstractions for data access / processing
  - More extensive functionality than Lapack
  - Leverages Lapack whenever possible
Matlab Stack

- Lapack: low-level Fortran linear algebra library
- Matlab Interface
  - Higher-level abstractions for data access / processing
  - More extensive functionality than Lapack
  - Leverages Lapack whenever possible
- Similar stories for R and Python
MLbase Stack

Matlab Interface
Lapack
Single Machine
MLbase Stack

Matlab Interface
Lapack
Single Machine

Runtime(s)
MLbase Stack

Spark: cluster computing system designed for iterative computation
Spark: cluster computing system designed for iterative computation
MLlib: low-level ML library in Spark
MLbase Stack

Spark: cluster computing system designed for iterative computation

MLlib: low-level ML library in Spark

MLI: API / platform for feature extraction and algorithm development
  ✦ Platform independent
Spark: cluster computing system designed for iterative computation

MLlib: low-level ML library in Spark

MLI: API / platform for feature extraction and algorithm development
  - Platform independent

ML Optimizer: automates model selection
  - Solves a search problem over feature extractors and algorithms in MLI
Example: MLLib
Example: MLlib

- Goal: Classification of text file
Example: MLlib

Goal:
- Classification of text file
- Featurize data manually

```scala
def main(args: Array[String]) {
  val sc = new SparkContext("local", "SparkLR")

  // Load data from HDFS
  val data = sc.textFile(args(0)) // RDD[String]

  // User is responsible for formatting/featurizing/normalizing their RDD!
  val featurizedData: RDD[(Double, Array[Double])] = processData(data)
```
Example: MLlib

def main(args: Array[String]) {
    val sc = new SparkContext("local", "SparkLR")

    //Load data from HDFS
    val data = sc.textFile(args(0)) //RDD[String]

    //User is responsible for formatting/featurizing/normalizing their RDD!
    val featurizedData: RDD[(Double,Array[Double])] = processData(data)

    //Train the model using MLlib.
    val model = new LogisticRegressionLocalRandomSGD()
        .setStepSize(0.1)
        .setNumIterations(50)
        .train(featurizedData)
}

- Goal: Classification of text file
- Featurize data manually
- Calls MLlib’s LR function
Example: MLI
Example: MLI

```scala
def main(args: Array[String]) {
    val mc = new MLContext("local", "MLILR")

    // Read in file from HDFS
    val rawTextTable = mc.csvFile(args(0), Seq("class","text"))

    // Run feature extraction
    val classes = rawTextTable(??, "class")
    val ngrams = tfIdf(nGrams(rawTextTable(??, "text"), n=2, top=30000))
    val featureizedTable = classes.zip(ngrams)
}
```

✧ Use built-in feature extraction functionality
Example: MLI

```scala
def main(args: Array[String]) {
  val mc = new MLContext("local", "MLILR")

  // Read in file from HDFS
  val rawTextTable = mc.csvFile(args(0), Seq("class","text"))

  // Run feature extraction
  val classes = rawTextTable(??, "class")
  val ngrams = tfIdf(nGrams(rawTextTable(??, "text"), n=2, top=30000))
  val featureizedTable = classes.zip(ngrams)

  // Classify the data using Logistic Regression.
  val lrModel = LogisticRegression(featurizedTable, stepSize=0.1, numIter=12)
}
```

- Use built-in feature extraction functionality
- MLI Logistic Regression leverages MLlib
Example: MLI

```scala
def main(args: Array[String]) {
  val mc = new MLContext("local", "MLILR")

  //Read in file from HDFS
  val rawTextTable = mc.csvFile(args(0), Seq("class","text"))

  //Run feature extraction
  val classes = rawTextTable(??, "class")
  val ngrams = tfIdf(nGrams(rawTextTable(??, "text"), n=2, top=30000))
  val featureizedTable = classes.zip(ngrams)

  //Classify the data using Logistic Regression.
  val lrModel = LogisticRegression(featureizedTable, stepSize=0.1, numIter=12)
}
```

- Use built-in feature extraction functionality
- MLI Logistic Regression leverages MLlib
- Extensions:
  - Embed in cross-validation routine
  - Use different feature extractors / algorithms
  - Write new ones

Fig. 15: Matrix Factorization via ALS code in MATLAB (top) and MLI (bottom).
Example: ML Optimizer

```javascript
var X = load("text_file", 2 to 10)
var y = load("text_file", 1)
var (fn-model, summary) = doClassify(X, y)
```

- User declaratively specifies task
- ML Optimizer searches through MLI
Example: ML Optimizer

- User declaratively specifies task
- ML Optimizer searches through MLI

Diagram:
- SQL → Result
- MQL → Model
Lay of the Land

Ease of use

Performance, Scalability
Lay of the Land

Ease of use

Performance, Scalability

Matlab, R

+ Easy (Resembles math, limited set up)
+ Sufficient for prototyping / writing papers
  - Ad-hoc, non-scalable scripts
  - Loss of translation upon re-implementation
Lay of the Land

- **Matlab, R**
  - Easy (Resembles math, limited set up)
  - Sufficient for prototyping / writing papers
    - Ad-hoc, non-scalable scripts
    - Loss of translation upon re-implementation

- **Mahout**, **GraphLab, VW**
  - Scalable and (sometimes) fast
  - Existing open-source libraries
    - Difficult to set up, extend

Ease of use vs. Performance, Scalability
Examples
Examples

‘Distributed’ Divide-Factor-Combine (DFC)

- Initial studies in MATLAB (Not distributed)
- Distributed prototype involving compiled MATLAB
Examples

‘Distributed’ Divide-Factor-Combine (DFC)
- Initial studies in MATLAB (Not distributed)
- Distributed prototype involving compiled MATLAB

Mahout ALS with Early Stopping
- Theory: simple if-statement (3 lines of code)
Examples

‘Distributed’ Divide-Factor-Combine (DFC)
♦ Initial studies in MATLAB (Not distributed)
♦ Distributed prototype involving compiled MATLAB

Mahout ALS with Early Stopping
♦ Theory: simple if-statement (3 lines of code)
♦ Practice: sift through 7 files, nearly 1K lines of code
Lay of the Land

Ease of use

<table>
<thead>
<tr>
<th>Performance, Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Matlab, R</strong></td>
</tr>
<tr>
<td>X</td>
</tr>
</tbody>
</table>

+ Easy (Resembles math, limited set up)
+ Sufficient for prototyping / writing papers
  - Ad-hoc, non-scalable scripts
  - Loss of translation upon re-implementation

+ Scalable and (sometimes) fast
+ Existing open-source libraries
  - Difficult to set up, extend
Lay of the Land

<table>
<thead>
<tr>
<th>Ease of use</th>
<th>Performance, Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Matlab, R</strong></td>
<td>x</td>
</tr>
<tr>
<td><strong>MLlib</strong></td>
<td>x</td>
</tr>
<tr>
<td><strong>Mahout</strong></td>
<td>x</td>
</tr>
<tr>
<td><strong>GraphLab, VW</strong></td>
<td>x</td>
</tr>
<tr>
<td><strong>VOWPAL WABBIT</strong></td>
<td></td>
</tr>
<tr>
<td><strong>GraphLab</strong></td>
<td></td>
</tr>
</tbody>
</table>

- Easy (Resembles math, limited set up)
- Sufficient for prototyping / writing papers
  - Ad-hoc, non-scalable scripts
  - Loss of translation upon re-implementation
- Scalable and (sometimes) fast
- Existing open-source libraries
  - Difficult to set up, extend
Lay of the Land

- **Matlab, R**: Easy (Resembles math, limited set up)  
  - Ad-hoc, non-scalable scripts  
  - Loss of translation upon re-implementation

- **MLI**: Sufficient for prototyping / writing papers  
  - Ad-hoc, non-scalable scripts

- **MLlib** and **Mahout**: Scalable and (sometimes) fast  
  - Existing open-source libraries  
  - Difficult to set up, extend

**Performance, Scalability**

**Ease of use**
ML Developer API (MLI)
ML Developer API (MLI)

OLD
val x: RDD[Array[Double]]
ML Developer API (MLI)

OLD
val x: RDD[Array[Double]]
val x: RDD[spark.util.Vector]
ML Developer API (MLI)

OLD
val x: RDD[Array[Double]]
val x: RDD[spark.util.Vector]
val x: RDD[breeze.linalg.Vector]
ML Developer API (MLI)

OLD
val x: RDD[Array[Double]]
val x: RDD[spark.util.Vector]
val x: RDD[breeze.linalg.Vector]
val x: RDD[BIDMat.SMat]
ML Developer API (MLI)

OLD
val x: RDD[Array[Double]]
val x: RDD[spark.util.Vector]
val x: RDD[breeze.linalg.Vector]
val x: RDD[BIDMat.SMat]
ML Developer API (MLI)

OLD
val x: RDD[Array[Double]]
val x: RDD[spark.ml.Vector]
val x: RDD[breeze.linalg.Vector]
val x: RDD[BIDMat.SMat]

NEW
val x: MLTable
ML Developer API (MLI)

- Abstract interface for arbitrary backend
- Common interface to support an optimizer
ML Developer API (MLI)
ML Developer API (MLI)

- Shield ML Developers from low-details
  - provide familiar mathematical operators in distributed setting
ML Developer API (MLI)

✧ **Shield ML Developers from low-details**
  ✧ provide familiar mathematical operators in distributed setting

✧ **Table Computation (MLTable)**
  ✧ Flexibility when loading data (heterogenous, missing)
  ✧ Common interface for feature extraction / algorithms
  ✧ Supports MapReduce and relational operators
ML Developer API (MLI)

- **Shield ML Developers from low-details**
  - provide familiar mathematical operators in distributed setting

- **Table Computation** (*MLTable*)
  - Flexibility when loading data (heterogenous, missing)
  - Common interface for feature extraction / algorithms
  - Supports MapReduce and relational operators

- **Linear Algebra** (*MLSubMatrix*)
  - Linear algebra on *local* partitions
  - Sparse and Dense matrix support
ML Developer API (MLI)

- **Shield ML Developers from low-details**
  - provide familiar mathematical operators in distributed setting

- **Table Computation** (*MLTable*)
  - Flexibility when loading data (heterogenous, missing)
  - Common interface for feature extraction / algorithms
  - Supports MapReduce and relational operators

- **Linear Algebra** (*MLSubMatrix*)
  - Linear algebra on *local* partitions
  - Sparse and Dense matrix support

- **Optimization Primitives** (*MLSolve*)
  - Distributed implementations of common patterns
ML Developer API (MLI)

- **Shield ML Developers from low-details**
  - provide familiar mathematical operators in distributed setting

- **Table Computation** (*MLTable*)
  - Flexibility when loading data (heterogenous, missing)
  - Common interface for feature extraction / algorithms
  - Supports MapReduce and relational operators

- **Linear Algebra** (*MLSubMatrix*)
  - Linear algebra on *local* partitions
  - Sparse and Dense matrix support

- **Optimization Primitives** (*MLSolve*)
  - Distributed implementations of common patterns

- **DFC**: \(~50\) lines of code
- **ALS**: early stopping in 3 lines; \(<40\) lines total
MLI Ease of Use
MLI Ease of Use

### Logistic Regression

<table>
<thead>
<tr>
<th>System</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab</td>
<td>11</td>
</tr>
</tbody>
</table>

### Alternating Least Squares

<table>
<thead>
<tr>
<th>System</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab</td>
<td>20</td>
</tr>
</tbody>
</table>
# MLI Ease of Use

## Logistic Regression

<table>
<thead>
<tr>
<th>System</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab</td>
<td>11</td>
</tr>
<tr>
<td>Vowpal Wabbit</td>
<td>721</td>
</tr>
</tbody>
</table>

## Alternating Least Squares

<table>
<thead>
<tr>
<th>System</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab</td>
<td>20</td>
</tr>
<tr>
<td>Mahout</td>
<td>865</td>
</tr>
<tr>
<td>GraphLab</td>
<td>383</td>
</tr>
</tbody>
</table>
# MLI Ease of Use

## Logistic Regression

<table>
<thead>
<tr>
<th>System</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab</td>
<td>11</td>
</tr>
<tr>
<td>Vowpal Wabbit</td>
<td>721</td>
</tr>
<tr>
<td>MLI</td>
<td>55</td>
</tr>
</tbody>
</table>

## Alternating Least Squares

<table>
<thead>
<tr>
<th>System</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab</td>
<td>20</td>
</tr>
<tr>
<td>Mahout</td>
<td>865</td>
</tr>
<tr>
<td>GraphLab</td>
<td>383</td>
</tr>
<tr>
<td>MLI</td>
<td>32</td>
</tr>
</tbody>
</table>
MLI/Spark Performance
MLI/Spark Performance

- **Walltime**: elapsed time to execute task
MLI/Spark Performance

- **Walltime**: elapsed time to execute task
- **Weak scaling**
  - fix problem size *per processor*
  - ideally: constant walltime as we grow cluster
MLI/Spark Performance

- **Walltime**: elapsed time to execute task

- **Weak scaling**
  - fix problem size *per processor*
  - ideally: constant walltime as we grow cluster

- **Strong scaling**
  - fix total problem size
  - ideally: linear speed up as we grow cluster
MLI/Spark Performance

- **Walltime**: elapsed time to execute task

- **Weak scaling**
  - fix problem size *per processor*
  - ideally: constant walltime as we grow cluster

- **Strong scaling**
  - fix total problem size
  - ideally: linear speed up as we grow cluster

- **EC2 Experiments**
  - m2.4xlarge instances, up to 32 machine clusters
Logistic Regression - Weak Scaling
Logistic Regression - Weak Scaling

- Full dataset: 200K images, 160K dense features
Logistic Regression - Weak Scaling

- Full dataset: 200K images, 160K dense features
- Similar weak scaling
Logistic Regression - Weak Scaling

- Full dataset: 200K images, 160K dense features
- Similar weak scaling
- MLI/Spark within a factor of 2 of VW’s walltime
Logistic Regression - Strong Scaling
Logistic Regression - Strong Scaling

- Fixed Dataset: 50K images, 160K dense features
Logistic Regression - Strong Scaling

- Fixed Dataset: 50K images, 160K dense features
- MLI/Spark exhibits better scaling properties
Logistic Regression - Strong Scaling

- Fixed Dataset: 50K images, 160K dense features
- MLI/Spark exhibits better scaling properties
- MLI/Spark faster than VW with 16 and 32 machines
ALS - Walltime
ALS - Walltime

- Dataset: Scaled version of Netflix data (9X in size)
- Cluster: 9 machines
ALS - Walltime

<table>
<thead>
<tr>
<th>System</th>
<th>Walltime (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab</td>
<td>15443</td>
</tr>
</tbody>
</table>

- Dataset: Scaled version of Netflix data (9X in size)
- Cluster: 9 machines
## ALS - Walltime

<table>
<thead>
<tr>
<th>System</th>
<th>Walltime (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab</td>
<td>15443</td>
</tr>
<tr>
<td>Mahout</td>
<td>4206</td>
</tr>
</tbody>
</table>

- Dataset: Scaled version of Netflix data (9X in size)
- Cluster: 9 machines
## ALS - Walltime

<table>
<thead>
<tr>
<th>System</th>
<th>Walltime (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab</td>
<td>15443</td>
</tr>
<tr>
<td>Mahout</td>
<td>4206</td>
</tr>
<tr>
<td>GraphLab</td>
<td>291</td>
</tr>
<tr>
<td>MLI/Spark</td>
<td>481</td>
</tr>
</tbody>
</table>

- Dataset: Scaled version of Netflix data (9X in size)
- Cluster: 9 machines
- **MLI/Spark** an order of magnitude faster than Mahout
- **MLI/Spark** within factor of 2 of GraphLab
Vision
MLI Details
**Current Status**
ML Workflow
MLI Functionality

**Regression:** Linear Regression (+Lasso, Ridge)

**Collaborative Filtering:** Alternating Least Squares

**Clustering:** K-Means

**Classification:** Logistic Regression, Linear SVM (+L1, L2)

**Optimization Primitives:** Parallel Gradient
MLI Functionality

**Regression:** Linear Regression (+Lasso, Ridge)

**Collaborative Filtering:** Alternating Least Squares, DFC

**Clustering:** K-Means, DP-Means

**Classification:** Logistic Regression, Linear SVM (+L1, L2), Multinomial Regression, Naive Bayes, Decision Trees

**Optimization Primitives:** Parallel Gradient, Local SGD, L-BFGS, ADMM, Adagrad

**Feature Extraction:** Principal Component Analysis (PCA), N-grams, feature normalization

**ML Tools:** Cross Validation, Evaluation Metrics
MLbase Stack Status

Goal 1: Summer Release

ML Developer

ML Optimizer

MLI

MLlib

Spark

End User
Goal 1: Summer Release

ML Developer

Goal 2: Winter Release

End User

MLbase Stack Status

ML Optimizer

MLI

MLlib

Spark

(ML base stack status diagram with layers of technology and release goals.)
Future Directions

✦ **Identify minimal set of ML operators**
  ✦ Expose internals of ML algorithms to optimizer

✦ **Plug-ins to Python, R**

✦ **Visualization** for unsupervised learning and exploration

✦ **Advanced ML capabilities**
  ✦ Time-series algorithms
  ✦ Graphical models
  ✦ Advanced Optimization (e.g., asynchronous computation)
  ✦ Online updates
  ✦ Sampling for efficiency
Vision
MLI Details
Current Status
ML Workflow
Typical Data Analysis Workflow
Typical Data Analysis Workflow

*Spark, MLI*  
**Obtain / Load Raw Data**
Typical Data Analysis Workflow

Spark, MLI

Obtain / Load Raw Data

Spark, [MLI]

Data Exploration

Adapted from slides by Ariel Kleiner
Typical Data Analysis Workflow

1. **Obtain / Load Raw Data**
   - Spark, MLI

2. **Data Exploration**
   - Spark, [MLI]

3. **Feature Extraction**
   - MLI

Adapted from slides by Ariel Kleiner
Typical Data Analysis Workflow

- **Spark, MLI**: Obtain / Load Raw Data
- **Spark, [MLI]**: Data Exploration
- **MLI**: Feature Extraction
- **MLI, MLlib**: Learning

Adapted from slides by Ariel Kleiner
Typical Data Analysis Workflow

1. Obtain / Load Raw Data
2. Data Exploration
3. Feature Extraction
4. Learning
5. Evaluation

Adapted from slides by Ariel Kleiner
Typical Data Analysis Workflow

1. **Spark, MLI**
   - Obtain / Load Raw Data

2. **Spark, [MLI]**
   - Data Exploration

3. **MLI**
   - Feature Extraction

4. **MLI, MLlib**
   - Learning

5. **MLI**
   - Evaluation

6. **Scala**
   - Deployment

Adapted from slides by Ariel Kleiner
Typical Data Analysis Workflow

Spark, MLI
Obtain / Load Raw Data

Spark, [MLI]
Data Exploration

MLI
Feature Extraction

MLI, MLlib
Learning

MLI
Evaluation

Scala
Deployment

Adapted from slides by Ariel Kleiner
Binary Classification

1.2 Definitions and terminology

We will use the canonical problem of spam detection as a running example to illustrate some basic definitions and to describe the use and evaluation of machine learning algorithms in practice. Spam detection is the problem of learning to automatically classify email messages as either spam or non-spam.

Examples: Items or instances of data used for learning or evaluation. In our spam problem, these examples correspond to the collection of email messages we will use for learning and testing.

Features: The set of attributes, often represented as a vector, associated to an example. In the case of email messages, some relevant features may include the length of the message, the name of the sender, various characteristics of the header, the presence of certain keywords in the body of the message, and so on.

Labels: Values or categories assigned to examples. In classification problems, examples are assigned specific categories, for instance, the spam and non-spam categories in our binary classification problem. In regression, items are assigned real-valued labels.

Training sample: Examples used to train a learning algorithm. In our spam problem, the training sample consists of a set of email examples along with their associated labels. The training sample varies for different learning scenarios, as described in section 1.4.

Validation sample: Examples used to tune the parameters of a learning algorithm.
Goal: Learn a mapping from entities to discrete labels
**Binary Classification**

**Goal:** Learn a mapping from entities to discrete labels

**Example:** Spam Classification
- Entities are emails
- Labels are \{spam, not-spam\}

Adapted from slides by Ariel Kleiner
**Goal**: Learn a mapping from entities to discrete labels

**Example**: Spam Classification

- Entities are emails
- Labels are \{*spam*, *not-spam*\}
- Given past labeled emails, we want to predict whether a new email is *spam* or *not-spam*
Goal: Learn a mapping from entities to discrete labels

Other Examples:
- Click (and clickthrough rate) prediction
- Fraud detection
- Face detection
- Exercise: “ARTS” vs “LIFE” on Wikipedia
- Real data
Classification Pipeline

full dataset
1. Randomly split full data into disjoint subsets
1. Randomly split full data into disjoint subsets
2. Featurize the data
1. Randomly split full data into disjoint subsets
2. Featurize the data
3. Use training set to learn a classifier

Adapted from slides by Ariel Kleiner
1. Randomly split full data into disjoint subsets
2. Featurize the data
3. Use training set to learn a classifier
4. Evaluate classifier on test set (avoid overfitting)

Adapted from slides by Ariel Kleiner
1. Randomly split full data into disjoint subsets
2. Featurize the data
3. Use training set to learn a classifier
4. Evaluate classifier on test set (avoid overfitting)
5. Use classifier to predict in the wild

Adapted from slides by Ariel Kleiner
E.g., Spam Classification

From: illegitimate@bad.com

"Eliminate your debt by giving us your money..."

From: bob@good.com

"Hi, it's been a while! How are you? ..."

spam

not-spam

Adapted from slides by Ariel Kleiner
Featurization

Adapted from slides by Ariel Kleiner
Featurization

- Most classifiers require numeric descriptions of entities

Adapted from slides by Ariel Kleiner
Featurization

- Most classifiers require numeric descriptions of entities

- **Featurization**: Transform each entity into a vector of real numbers

Adapted from slides by Ariel Kleiner
Featurization

✦ Most classifiers require numeric descriptions of entities

✦ **Featurization**: Transform each entity into a vector of real numbers
  ✦ Opportunity to incorporate domain knowledge
  ✦ Useful even when original data is already numeric
E.g., “Bag of Words”

From: illegitimate@bad.com

"Eliminate your debt by giving us your money..."

From: bob@good.com

"Hi, it's been a while! How are you? ..."

Adapted from slides by Ariel Kleiner
E.g., “Bag of Words”

♦ Entities are documents

From: illegitimate@bad.com

"Eliminate your debt by giving us your money..."

From: bob@good.com

"Hi, it's been a while! How are you? ..."
E.g., “Bag of Words”

- Entities are documents
- Build Vocabulary

From: illegitimate@bad.com
"Eliminate your debt by giving us your money..."

From: bob@good.com
"Hi, it's been a while! How are you? ..."

Vocabulary

been
debt
eliminate
giving
how
it's
money
while

Adapted from slides by Ariel Kleiner
E.g., "Bag of Words"

- Entities are documents
- Build Vocabulary
- Derive feature vectors from Vocabulary
  - Exercise: we’ll use bigrams

From: illegitimate@bad.com

"Eliminate your debt by giving us your money..."
Support Vector Machines (SVMs)

In this section, we assume that the training sample $S$ can be linearly separated, that is, we assume the existence of a hyperplane that perfectly separates the training sample into two populations of positively and negatively labeled points, as illustrated by the left panel of figure 4.1. But there are then infinitely many such separating hyperplanes. Which hyperplane should a learning algorithm select?

The solution returned by the SVM algorithm is the hyperplane with the maximum margin, or distance to the closest points, and is thus known as the maximum-margin hyperplane. The right panel of figure 4.1 illustrates that choice.

We will present later in this chapter a margin theory that provides a strong justification for this solution. We can observe already, however, that the SVM solution can also be viewed as the “safest” choice in the following sense: a test point is classified correctly by a separating hyperplane with margin $\varepsilon$ even when it falls within a distance $\varepsilon$ of the training samples sharing the same label; for the SVM solution, $\varepsilon$ is the maximum margin and thus the “safest” value.

4.2.1 Primal optimization problem

We now derive the equations and optimization problem that define the SVM solution. The general equation of a hyperplane in $\mathbb{R}^N$ is

$$w \cdot x + b = 0,$$

where $w \in \mathbb{R}^N$ is a non-zero vector normal to the hyperplane and $b \in \mathbb{R}$ is a scalar. Note that this definition of a hyperplane is invariant to non-zero scalar multiplication. Hence, for a hyperplane that does not pass through any sample point, we can scale $w$ and $b$ appropriately such that

$$\min_{(x,y) \in S} |w \cdot x + b| = 1.$$
Support Vector Machines (SVMs)

Figure 4.1
Two possible separating hyperplanes. The right-hand side figure shows a hyperplane that maximizes the margin.

4.2 SVMs — separable case
In this section, we assume that the training sample $S$ can be linearly separated, that is, we assume the existence of a hyperplane that perfectly separates the training sample into two populations of positively and negatively labeled points, as illustrated by the left panel of figure 4.1. But there are then infinitely many such separating hyperplanes. Which hyperplane should a learning algorithm select? The solution returned by the SVM algorithm is the hyperplane with the maximum margin, or distance to the closest points, and is thus known as the maximum-margin hyperplane. The right panel of figure 4.1 illustrates that choice.

We will present later in this chapter a margin theory that provides a strong justification for this solution. We can observe already, however, that the SVM solution can also be viewed as the “safest” choice in the following sense: a test point is classified correctly by a separating hyperplane with margin $\varepsilon$ even when it falls within a distance $\varepsilon$ of the training samples sharing the same label; for the SVM solution, $\varepsilon$ is the maximum margin and thus the “safest” value.

4.2.1 Primal optimization problem
We now derive the equations and optimization problem that define the SVM solution. The general equation of a hyperplane in $\mathbb{R}^N$ is $w \cdot x + b = 0$, (4.3) where $w \in \mathbb{R}^N$ is a non-zero vector normal to the hyperplane and $b \in \mathbb{R}$ a scalar. Note that this definition of a hyperplane is invariant to non-zero scalar multiplication. Hence, for a hyperplane that does not pass through any sample point, we can scale $w$ and $b$ appropriately such that $\min_{(x, y) \in S} |w \cdot x + b| = 1$. 

Credit: Foundations of Machine Learning
Mohri, Rostamizadeh, Talwalkar
Support Vector Machines (SVMs)

4.2 SVMs — separable case

In this section, we assume that the training sample $S$ can be linearly separated, that is, we assume the existence of a hyperplane that perfectly separates the training sample into two populations of positively and negatively labeled points, as illustrated by the left panel of figure 4.1. But there are then infinitely many such separating hyperplanes. Which hyperplane should a learning algorithm select? The solution returned by the SVM algorithm is the hyperplane with the maximum margin, or distance to the closest points, and is thus known as the maximum-margin hyperplane. The right panel of figure 4.1 illustrates that choice.

We will present later in this chapter a margin theory that provides a strong justification for this solution. We can observe already, however, that the SVM solution can also be viewed as the “safest” choice in the following sense: a test point is classified correctly by a separating hyperplane with margin $\epsilon$ even when it falls within a distance $\epsilon$ of the training samples sharing the same label; for the SVM solution, $\epsilon$ is the maximum margin and thus the “safest” value.

4.2.1 Primal optimization problem

We now derive the equations and optimization problem that define the SVM solution. The general equation of a hyperplane in $\mathbb{R}^N$ is $w \cdot x + b = 0$, (4.3) where $w \in \mathbb{R}^N$ is a non-zero vector normal to the hyperplane and $b \in \mathbb{R}$ is a scalar. Note that this definition of a hyperplane is invariant to non-zero scalar multiplication. Hence, for a hyperplane that does not pass through any sample point, we can scale $w$ and $b$ appropriately such that $\min_{(x,y) \in S} |w \cdot x + b| = 1.$

Credit: Foundations of Machine Learning
Mohri, Rostamizadeh, Talwalkar
Support Vector Machines (SVMs)

“Max-Margin”: find linear separator with the largest separation between the two classes.
Support Vector Machines (SVMs)

- "Max-Margin": find linear separator with the largest separation between the two classes
- Extensions:
  - non-separable setting
  - non-linear classifiers (kernels)

Figure 4.1

Two possible separating hyperplanes. The right-hand side figure shows a hyperplane that maximizes the margin.

4.2 SVMs — separable case

In this section, we assume that the training sample \( S \) can be linearly separated, that is, we assume the existence of a hyperplane that perfectly separates the training sample into two populations of positively and negatively labeled points, as illustrated by the left panel of figure 4.1. But there are then infinitely many such separating hyperplanes. Which hyperplane should a learning algorithm select?

The solution returned by the SVM algorithm is the hyperplane with the maximum margin, or distance to the closest points, and is thus known as the maximum-margin hyperplane. The right panel of figure 4.1 illustrates that choice.

We will present later in this chapter a margin theory that provides a strong justification for this solution. We can observe already, however, that the SVM solution can also be viewed as the "safest" choice in the following sense: a test point is classified correctly by a separating hyperplane with margin \( \varepsilon \) even when it falls within a distance \( \varepsilon \) of the training samples sharing the same label; for the SVM solution, \( \varepsilon \) is the maximum margin and thus the "safest" value.

4.2.1 Primal optimization problem

We now derive the equations and optimization problem that define the SVM solution. The general equation of a hyperplane in \( \mathbb{R}^N \) is

\[
w \cdot x + b = 0,
\]

where \( w \in \mathbb{R}^N \) is a non-zero vector normal to the hyperplane and \( b \in \mathbb{R} \) is a scalar. Note that this definition of a hyperplane is invariant to non-zero scalar multiplication. Hence, for a hyperplane that does not pass through any sample point, we can scale \( w \) and \( b \) appropriately such that

\[
\min_{(x,y) \in S} |w \cdot x + b| = 1.
\]

Credit: Foundations of Machine Learning
Mohri, Rostamizadeh, Talwalkar
Model Evaluation

Adapted from slides by Ariel Kleiner
Model Evaluation

- Test set simulates performance on new entity
- Performance on training data overly optimistic!
- “Overfitting”; “Generalization”
Model Evaluation

- Test set simulates performance on new entity
- Performance on training data overly optimistic!
- “Overfitting”; “Generalization”
- Various metrics for quality; accuracy is most common
Model Evaluation

- Test set simulates performance on new entity
- Performance on training data overly optimistic!
- “Overfitting”; “Generalization”
- Various metrics for quality; accuracy is most common
- Evaluation process

Adapted from slides by Ariel Kleiner
Model Evaluation

- Test set simulates performance on new entity
- Performance on training data overly optimistic!
- “Overfitting”; “Generalization”
- Various metrics for quality; accuracy is most common
- Evaluation process
  - Train on training set (don’t expose test set to classifier)

Adapted from slides by Ariel Kleiner
Model Evaluation

- Test set simulates performance on new entity
  - Performance on training data overly optimistic!
  - “Overfitting”; “Generalization”
- Various metrics for quality; accuracy is most common
- Evaluation process
  - Train on training set (don’t expose test set to classifier)
  - Make predictions using test set (ignoring test labels)

Adapted from slides by Ariel Kleiner
Model Evaluation

- Test set simulates performance on new entity
  - Performance on training data overly optimistic!
  - “Overfitting”; “Generalization”

- Various metrics for quality; accuracy is most common

- Evaluation process
  - Train on training set (don’t expose test set to classifier)
  - Make predictions using test set (ignoring test labels)
  - Compute fraction of correct predictions on test set

Adapted from slides by Ariel Kleiner
Model Evaluation

- Test set simulates performance on new entity
- Performance on training data overly optimistic!
- “Overfitting”; “Generalization”
- Various metrics for quality; accuracy is most common
- Evaluation process
  - Train on training set (don’t expose test set to classifier)
  - Make predictions using test set (ignoring test labels)
  - Compute fraction of correct predictions on test set
- Other more sophisticated evaluation methods, e.g., cross-validation

Adapted from slides by Ariel Kleiner
Contributions encouraged!

www.mlbase.org