Hands-on lab: Security analytics
ENEE 657

Octavian Suciu
osuciu@umiacs.umd.edu
University of Maryland, College Park

http://ter.ps/enee657

Today’s Lecture

• Where we’ve been
  – OS protection mechanisms

• Where we’re going today
  – Intro to supervised learning
  – Intro to Apache Spark
  – Document Similarity
  – Hands-on: Spark

• Where we’re going next
  – Homework 2 out today, due next Wednesday!
  – First paper critiques due next Monday!
  – Network security fundamentals
Homework & Paper Critique Submissions

- Use the submit command on GRACE
  - SSH into grace.umd.edu
  - `submit <year> <semester> <college> <course> <section> <assignment> <filename>`
    - Example: `submit 2017 fall enee 657 0101 1 exploit_1.c`
  - Wrapper that performs some checks on the submission
    /afs/glue.umd.edu/class/fall2017/enee/657/0101/bin/submit
  - For more information on GRACE: [http://www.grace.umd.edu/](http://www.grace.umd.edu/)

- For critiques, submit BibTeX files in plain text
  - No Word DOC, no RTF, no HTML!
  - Do not remove BibTeX syntax (e.g. the @ sign before entries)
    - This confuses my parser and I may think that you did not submit the homework if I don’t catch the error!
  - Submission deadline: at noon one week before class
    - Example: critiques for Mon 09/25 papers due Mon 09/18

Predicting which papayas are tasty

- You arrive in a small Pacific island. Papayas are an important ingredient here.
- You don’t know how papayas taste like, but you want to be able to pick tasty papayas from the market.
- You taste a lot of papayas and record a part of their features: softness and color.
- Based on these features, you want to predict which (new) papayas from the market are tasty.
- Supervised learning aims to solve this!
Introduction to Supervised Learning

- Components of a classification system:
  - Training set \((X, y)\)
  - Prediction: \(y' = f(X;w)\)
  - Cost function: \(c(y', y)\)
  - Goal: find \(w\) that minimizes \(c\) w.r.t. to \((X,y)\) on \(f\)

Supervised Learning in Context

- Training set \((X, y)\)
  - If \(y\) is unknown \(\rightarrow\) unsupervised learning
  - If \(y\) is categorical \(\rightarrow\) classification
    - If \(y\) is binary \(\rightarrow\) detection
    - If \(y\) is continuous \(\rightarrow\) regression
Supervised Learning

- Training: a learning algorithm reads in training data and computes $f$
- Testing: $f$ can then automatically label future text examples.

Example of Available Tools

- Libraries
  - Scikit-learn
  - Spark MLlib
  - R
  - Weka

- Specific
  - OpenCV
  - LibSVM
  - TensorFlow, Theano, Keras, …
Popular Classification Techniques

- Logistic regression
- Naïve Bayes
- SVM
- Decision trees

SVM

- Support Vector Machine
- Intuition
  - Training instances
    - Points in feature space
  - Classifier
    - Hyperplane that maximizes separation
Artificial Example

- Two groups (randomly generated)
  - $X_1$
    - $y=0$: $N(0, 0.5)$
    - $y=1$: $N(1, 0.5)$
  - $X_2$
    - $U(0, 1)$

Artificial Example (2)

- Separation hyperplane:
  - Ideal:
    - $X_1 = 0.5$
  - Estimated:
    - Slope = 10
- Finding the ideal classifier is hard!
Implementation

- Steps
  1. Extract features
  2. Select model and classifier
  3. Select features
  4. Train the model
  5. Evaluate the performance
  6. Test on unlabeled examples

Feature Extraction

- Detecting malicious Android apps
**Model Selection**

- Fitting a 2D set of points
  - Linear hypothesis
  - Higher order polynomial

**Classifier Selection**

- SVM
  ```python
  >>> from sklearn import svm
  >>> X = [[0, 0], [1, 1]]
  >>> y = [0, 1]
  >>> clf = svm.SVC()
  >>> clf.fit(X, y)
  ```

- Naïve Bayes
  ```python
  >>> from sklearn.naive_bayes import GaussianNB
  >>> clf = GaussianNB()
  >>> clf.fit(X, Y)
  ```

- Decision Tree
  ```python
  >>> from sklearn import tree
  >>> X = [[0, 0], [1, 1]]
  >>> y = [0, 1]
  >>> clf = tree.DecisionTreeClassifier()
  >>> clf = clf.fit(X, Y)
  ```
Feature Selection

- Remove features with low variance

```python
>>> from sklearn.feature_selection import VarianceThreshold
>>> X = [[0, 0, 1], [0, 1, 1], [1, 0, 1], [0, 1, 0], [0, 1, 1]]
>>> sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
>>> sel.fit_transform(X)

```

Performance Evaluation

- Popular metrics
  - Precision $\frac{TP}{TP+FP}$
    - Fraction of detected samples that are malicious
  - Recall (True positive rate) $\frac{TP}{TP+FN}$
    - Fraction of malicious samples that are detected
  - False positive rate $\frac{FP}{FP+TN}$

<table>
<thead>
<tr>
<th></th>
<th>TRUE (Truth)</th>
<th>FALSE (Truth)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE (Detector)</td>
<td>True Positive (TP)</td>
<td>False Positive (FP)</td>
</tr>
<tr>
<td>FALSE (Detector)</td>
<td>False Negative (FN)</td>
<td>True Negative (TN)</td>
</tr>
</tbody>
</table>
Performance Evaluation (2)

- K-fold Cross validation
  - Partition data randomly to \( k \) subsamples
  - Training data: \( k-1 \) subsamples
  - Testing data: 1 subsample

---

Performance Evaluation (3)

```python
>>> from sklearn import metrics
>>> scores = cross_validation.cross_val_score(clf, iris.data, iris.target, ... cv=5, scoring='f1_weighted')
```

<table>
<thead>
<tr>
<th>Scoring</th>
<th>Function</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classification</td>
<td></td>
<td></td>
</tr>
<tr>
<td>'accuracy'</td>
<td>metrics.accuracy_score</td>
<td></td>
</tr>
<tr>
<td>'average_precision'</td>
<td>metrics.average_precision_score</td>
<td></td>
</tr>
<tr>
<td>'f1'</td>
<td>metrics.f1_score</td>
<td>for binary targets</td>
</tr>
<tr>
<td>'f1_micro'</td>
<td>metrics.f1_score</td>
<td>micro-averaged</td>
</tr>
<tr>
<td>'f1_macro'</td>
<td>metrics.f1_score</td>
<td>macro-averaged</td>
</tr>
<tr>
<td>'f1_weighted'</td>
<td>metrics.f1_score</td>
<td>weighted average</td>
</tr>
<tr>
<td>'f1_samples'</td>
<td>metrics.f1_score</td>
<td>by multilabel sample</td>
</tr>
<tr>
<td>'log_loss'</td>
<td>metrics.log_loss</td>
<td>requires predict_proba support</td>
</tr>
<tr>
<td>'precision'</td>
<td>metrics.precision_score</td>
<td>suffixes apply as with 'f1'</td>
</tr>
<tr>
<td>'recall'</td>
<td>metrics.recall_score</td>
<td>suffixes apply as with 'f1'</td>
</tr>
<tr>
<td>'roc_auc'</td>
<td>metrics.roc_auc_score</td>
<td></td>
</tr>
</tbody>
</table>
Apache Spark

- Framework for processing large volumes of data
- Based on the Map/Reduce paradigm
- Architecture based on Driver & Workers
  - Driver sends computation to the workers
  - Workers compute & report to the Driver for synchronization
- Primitive: Resilient DistributedDatasets - RDDs
- All workers execute the same task

Spark architecture

- Driver spawns & assigns tasks to workers
RDDs

- Distributed array, evenly split across workers
- Enable operation pipelining & fault tolerance

Broadcast Variables

- Immutable values cached by all workers
Operations on RDDs

Example Operation: Map()

- Each element of the RDD is transformed using \( f() \)

```
// transformed RDDs
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t")).map(x => x(1))
messages.cache()

// action 1
messages.filter(_.contains("mysql")).count()
```
Example Operation: Reduce()

- Merges elements using a commutative & associative function \( f() \)

Example Operation: Filter()

- Retains elements matching a condition
Document Similarity

- Textual similarity between millions of documents
- All-pairs similarity is not feasible
- Example applications:
  - Plagiarism detection
  - Exploit code reuse

Document Similarity Approach
Jaccard Similarity

- Popular metric where documents are represented as sets

\[ SIM(S, T) = \frac{|S \cap T|}{|S \cup T|} \]

SIM(S, T) = 3/8

Computing the Jaccard Similarity Efficiently

- Document
  - Shingling
    - The set of strings of length k that appear in the document
  - Min Hashing
  - Locality-Sensitive Hashing
  - Signatures: short integer vectors that represent the sets, and reflect their similarity
- Candidate pairs: those pairs of signatures that we need to test for similarity
Computing the Jaccard Similarity Efficiently

- Split document in sequences of tokens
- Tokens are words/characters etc
- Sequence of k tokens = k-shingle (k-gram)
- Example:
  - D = abcab
  - k = 2 chars
  - S(D) = { ab, bc, ca}
Minhashing

- Represent large sets of tokens through smaller signatures
- Preserves the original Jaccard similarity when compared
Computing the minhash (1)

• Characteristic matrix:

<table>
<thead>
<tr>
<th>Element</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$b$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$c$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$d$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$e$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Computing the minhash (2)

• Random permutation of rows
• MinHash = first row in which a document has ‘1’

<table>
<thead>
<tr>
<th>$h(S1)$=a</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$d$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$c$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
**Minhash Example**

<table>
<thead>
<tr>
<th>Documents</th>
<th>Input matrix</th>
<th>Signature matrix $M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 0 1 0</td>
<td>1 2 1 2</td>
</tr>
<tr>
<td>2</td>
<td>1 0 0 1</td>
<td>1 2 4 1</td>
</tr>
<tr>
<td>3</td>
<td>0 1 0 1</td>
<td>2 1 4 1</td>
</tr>
<tr>
<td>4</td>
<td>0 1 0 1</td>
<td>1 2 1 2</td>
</tr>
</tbody>
</table>

**Minhash Property**

- Probability that $h(D_1) = h(D_2) \sim \text{SIM}(D_1, D_2)$
Computing the Jaccard Similarity Efficiently

Locality Sensitive Hashing (LSH)

- Generate small list of candidate pairs from collection of signatures
- Idea:
  - Hash signatures to many buckets
  - Elements in the same bucket are candidate pairs
- Documents are split in bands (chunks) then hashed independently
Property of LSH

- Probability that two similar signatures will agree on at least one of the bands is high

« And that is exactly when they become candidates!
Sources


• A Course in Machine Learning by Hal Daumé III, http://ciml.info/


• Ziyun Zhu

• Radu Marginean