MLscan

Machine Learning Mailing List Scanner

Greg Padgett
March 29, 2016

Motivation

- A lot of traffic on the ovirt-users list.
  - A lot of emails for the qe contact to scan.

- What if we could flag these messages automatically?
  - Even send an email to the current qe contact?

- Create a program that can scan email and identify these messages based on some properties of the message.
  - What properties?

Some Ideas…

- Don’t worry about mails sent from @redhat.com addresses.
- Only look at the root message in email threads.
- Does “storage” appear in the mail?

Some data:

```
[~]$ wc -l /tmp/mlscan.Interesting
188 /tmp/mlscan.Interesting
[~]$ grep -ic storage /tmp/mlscan.Interesting
104
[~]$ wc -l /tmp/mlscan.NotInteresting
188 /tmp/mlscan.NotInteresting
[~]$ grep -ic storage /tmp/mlscan.NotInteresting
48
```

“Storage” appears in 55% of messages we replied to, and 25% of the messages we didn’t reply to.

→ Maybe we don’t know what features to look for after all?

Machine Learning

- What if we could have the program figure out what helps identify the interesting emails by itself?
  - Sounds like Machine Learning

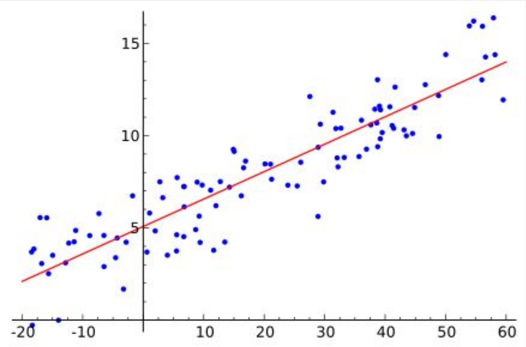
- Machine Learning:
  - "Field of study that gives computers the ability to learn without being explicitly programmed" - Arthur Samuel, 1959
  - “[...] explores the study and construction of algorithms that can learn from and make predictions on data.” - wikipedia

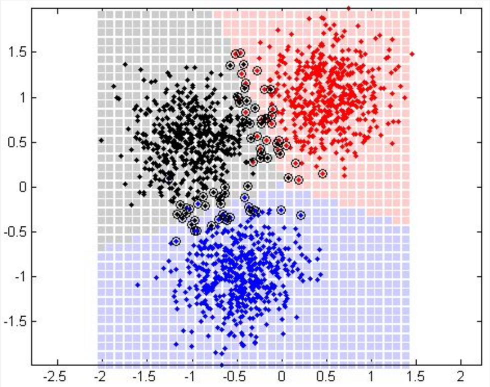
- Glorified statistics

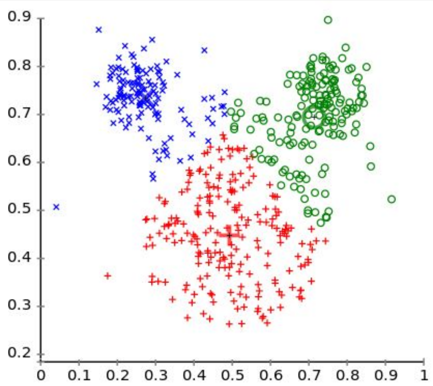
Machine Learning: More Detail

- Learning
  - Supervised - give the algorithm the right answer
  - Unsupervised - learns the answer itself
  - Reinforcement - on-line exploration of environment to maximize reward

- Categories:







Regression

- Continuous-valued data.
  - Linear Regression
  - Polynomial, etc.

Classification

- Divide into known groups.
  - Naive Bayes
  - Support Vector Machines
  - Neural Networks

Clustering

- Divide into unknown groups.
  - k-means

Naive Bayes

- Family of supervised classification algorithms:
  - Bernoulli - binary occurrence of features
  - Multinomial - counts occurrence of features
  - Gaussian - continuous data

- Often used for spam filtering

- What's naive about it?
  - Assumes feature probabilities are independent.
  - Not always true (e.g. language processing) but works well anyway…

- Relatively simple
  - Basically, the classification is chosen based on the probability of the input's features occurring in that class: highest wins.

The Heuristic

“Interesting Senders” Heuristic:

- Consider a thread interesting if one of us (storage) replied in the thread.
- Don’t ever consider an email from @redhat.com interesting
- Replies within a thread aren’t interesting, only the first message.
    - We can always monitor a thread that’s already interesting.

The Program

1. Parse mailing list archives
   - Text mbox format; “\nFrom” delineates records
2. Parse messages
   - Apache mime4j
3. Collate email threads
   - Simple tree structure, using Message-Id, In-Reply-To headers
4. Label training examples
   - “Interesting Senders” heuristic
5. Train the classifier
   - Naive Bayes using Datumbox Machine Learning Framework (Java)
6. Classify new inputs
   - Not yet on-line

Features

- Chosen by the classifier based on the input
  - Email subject and body text
  - Chi Square - keep most statistically significant features
- N-grams
  - Word sequences
  - In our case, actually uni-grams (possibly due to small training set size)
  - Tri-grams also work pretty well (and are commonly used)

- Garbage In, Garbage Out

```
public static String getClassificationDataFromMail(EmailThreadItem message) {
    return removeStopWords(removeSingleCharWords(removePunctuation(
            message.getEmail().getSubject().toLowerCase()
            + " "
            + removeLogs(removeAttachmentText(
                    message.getEmail().getBody().toLowerCase())))));
}
```

- Remove “stop words”, too.

```
    private TextClassifier getClassifier(Map<String, File> trainingSetFiles) {
        RandomGenerator.setGlobalSeed(randomSeed);
        Configuration conf = Configuration.getConfiguration();

        Map<Object, URI> dataset = new HashMap<>();
        for (String classification : trainingSetFiles.keySet()) {
            dataset.put(classification, trainingSetFiles.get(classification).toURI());
        }

        TextClassifier.TrainingParameters trainingParameters = new TextClassifier.TrainingParameters();

        trainingParameters.setModelerClass(MultinomialNaiveBayes.class);
        trainingParameters.setModelerTrainingParameters(new MultinomialNaiveBayes.TrainingParameters());

        trainingParameters.setDataTransformerClass(null);
        trainingParameters.setDataTransformerTrainingParameters(null);

        trainingParameters.setFeatureSelectorClass(ChisquareSelect.class);
        ChisquareSelect.TrainingParameters chisquareParameters = new ChisquareSelect.TrainingParameters();
        chisquareParameters.setALevel(0.10);  // p-ratio; the default, made explicit
        trainingParameters.setFeatureSelectorTrainingParameters(chisquareParameters);

        trainingParameters.setTextExtractorClass(NgramsExtractor.class);
        NgramsExtractor.Parameters ngramParameters = new NgramsExtractor.Parameters();
        ngramParameters.setMaxCombinations(1);  // The default, made explicit
        trainingParameters.setTextExtractorParameters(ngramParameters);

        TextClassifier classifier = new TextClassifier("InterestingSendersClassifier", conf);
        classifier.fit(dataset, trainingParameters);

        ValidationMetrics vm = classifier.validate(dataset);
        classifier.setValidationMetrics(vm); //store them in the model for future reference

        return classifier;
    }
```

Demo

Results

Test run:

- 92 messages total, 62 predicted successfully (67.39%)
- 72 uninteresting messages, 44 predicted successfully (61.11%)
- 20 interesting messages, 18 predicted successfully (90.00%)

Takeaway:

- We can skip 46 of 92 threads (50%) and still find 90% of the "interesting" messages.
- Should get even better with more training data.