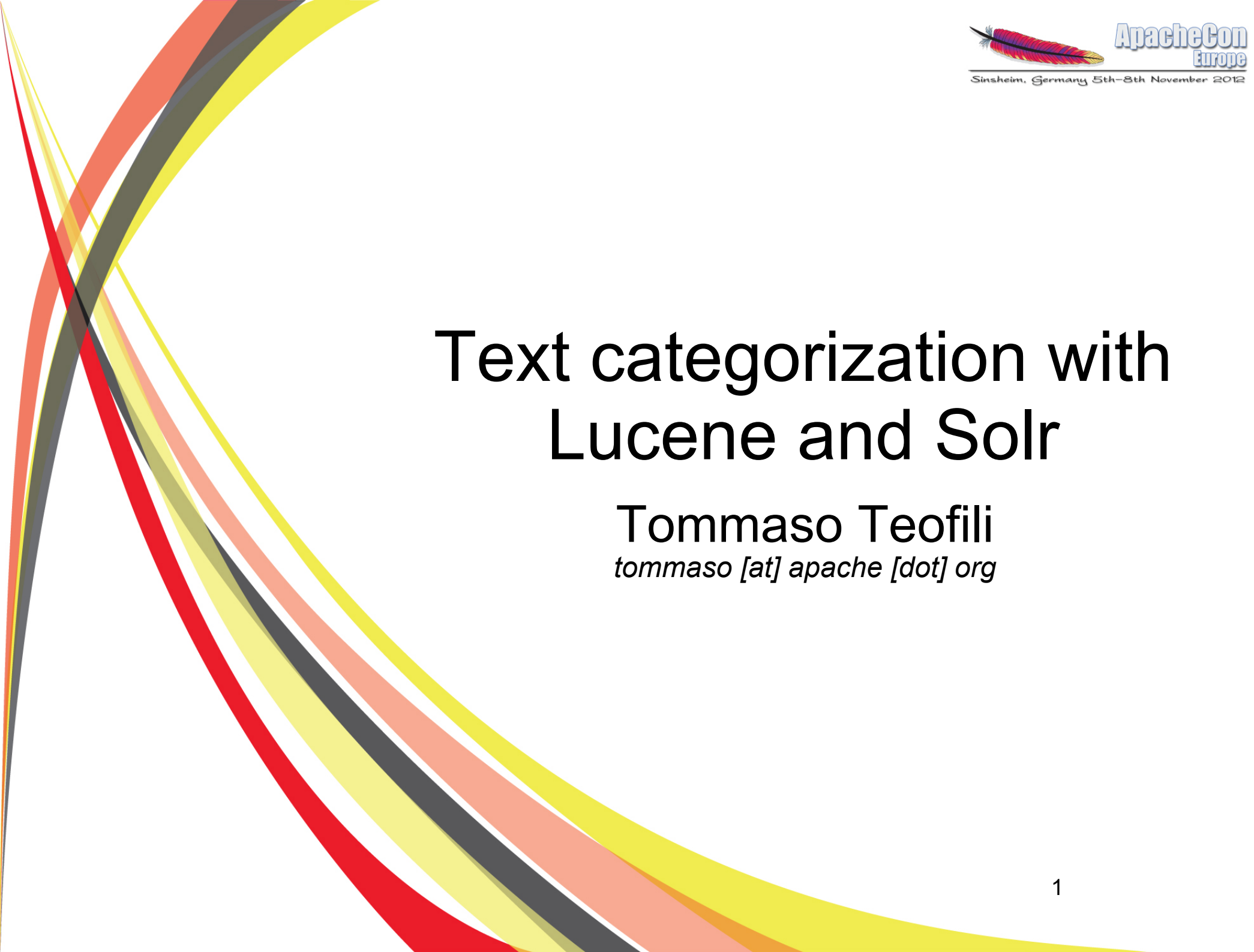

Text categorization with Lucene and Solr

Tommaso Teofili
tommaso [at] apache [dot] org

About me

- ASF member having fun with:
  - Lucene / Solr
  - Hama
  - UIMA
  - Stanbol
  - … some others
- SW engineer @ Adobe R&D

Agenda

- Classification
- Lucene classification module
- Solr text categorization services
- Conclusions

Classification

- Let the algorithm assign one or more labels (classes) to some item given some previous knowledge
  - Spam filter
  - Tagging system
  - Digit recognition system
  - Text categorization
  - etc.

Classification?
why with Lucene?

The short story

- Lucene already has a lot of features for common information retrieval needs
  - Postings
  - Term vectors
  - Statistics
  - Positions
  - TF / IDF
  - maybe Payloads
  - etc.
- We may avoid bringing in new components to do classification just leveraging what we get for free from Lucene

The (slightly) longer story #1

- While playing with NLP stuff
- Need to implement a naïve bayes classifier
  - Not possible to plug in stuff requiring touching the architecture
  - Not really interested in (near) real time performance
- Iteration 1
  - Plain in memory Java stuff
- Iteration 2
  - Same stuff but using Lucene instead of loading things into memory
    - Too much faster ☺

The (slightly) longer story #2

- So I realized
  - Lucene has so many *features* stored you can take advantage of **for free**
  - Therefore writing the classification algorithm is relatively simple
  - In many cases you're just not adding anything to the architecture
    - Your Lucene index was already there for searching
  - Lucene index is, to some extent, already a *model* which we just need to "query" with the proper algorithm
  - And it is fast enough

Lucene classification module

- Work in progress on trunk
- LUCENE-4345
  - Establishing classification API
  - With currently two implementations
    - Naïve bayes
    - K nearest neighbor

Lucene classification module

- Classifier API
  - Training
  - *void **train**(atomicReader, contentField, classField, analyzer) throws IOException*
    - *atomicReader* : the reader on the Lucene index to use for classification
      - still unsure if IR'd be better
    - *textFieldName* : the name of the field which contains documents' texts
    - *classFieldName* : the name of the field which contains the class assigned to existing documents
    - *analyzer* : the item used for analyzing the unseen texts

Lucene classification module

- Classifier API
  - Classifying
  - *ClassificationResult* ***assignClass****(String text) throws IOException*
    - *text*: the unseen text of the document to classify
    - *ClassificationResult* : the object containing the assigned class along with the related score

K Nearest neighbor classifier

- Fairly simple classification algorithm
- Given some new unseen item
- I search in my knowledge base the k items which are nearer to the new one
- I get the k classes assigned to the k nearest items
- I assign to the new item the class that is most frequent in the k returned items

K Nearest neighbor classifier

- How can we do this in Lucene?
  - We have VSM for representing documents as vectors and eventually find distances
  - Lucene MoreLikeThis module can do a lot for it
  - Given a new document
    - It's represented as a MoreLikeThisQuery which filters out too frequent words and helps on keeping only the relevant tokens for finding the neighbors
    - The query is executed returning only the first k results
    - The result is then browsed in order to find the most frequent class and that is then assigned with a score of classFreq / k

Naïve Bayes classifier

- Slightly more complicated
- Based on probabilities
- C = argmax(P(d|c) * P(c))
  - P(d|c) : likelihood
  - P(c) : prior
  - With some assumptions:
    - bag of words assumption: positions don't matter
    - conditional independence: the feature probabilities are independent given a class

Naïve Bayes classifier

- Prior calculation is easy
  - It's the relative frequency of each class #docsWithClassC / #docs
- Likelihood is easy too because of the bag of words assumption
  - P(d|c) := P(x1,..,xn|c) == P(x1|c)*...P(xn|c)
  - So we just need probabilities of single terms
    - P(x|c) := (tf of x in documents with class c + 1)/ (#terms in docs with class c + #docs)

Naïve Bayes classifier

- Does the bag of words assumption affect the classifier's precision?
  - Yes in theory
    - in text documents (nearby) words are strictly correlated
  - Not always in practice
    - depending on your index data it may or not have an impact

Using different indexes

- The Classifier API makes usage of an AtomicReader to get the data for the training
- It must not be the very same index used for every day index / search
  - For performance reasons
  - For enhancing classifier effectiveness
    - Using more specific analyzers
    - Indexing data in a different way
      - e.g. one big document for each class and use kNN (with a small k) or TF-IDF similarity

Things to consider - bootstrapping

- How are your first documents classified?
  - Manually
    - Categories are already there in the documents
    - Someone is explicitly charged to do that (e.g. article authors) at some point in time
  - (semi) automatically
    - Using some existing service / library
      - With or without human supervision
- In either case the classifier needs something to be fed with to be effective

Things to consider – tokenizing

- How are your content field tokenized?
  - Whitespace
    - It doesn't work for each language
  - Standard
  - Sentence
  - What about using N-Grams?
  - What about using Shingles?

Things to consider - filtering

- Some words may / should be filtered while
  - Training
  - Classifying
- Often
  - Stopwords
  - Punctuation
  - Not relevant PoS tagged tokens

Raw benchmarking

- Tried both algorithms on ~1M docs index
  - Naïve bayes is affected by the # of classes
  - kNN is affected by k being large
- None of them took more than 1-2m to train even with great number of classes or large k values

From Lucene to Solr

- The Lucene classifiers can be easily used in Solr
  - As specific search services
    - A classification based more like this
  - While indexing
    - For automatic text categorization

Classification based MLT

- Use case:
  - “give me all the documents that belong to the same category of a new not indexed document”
  - Slightly different from basic MLT since it does not return the nearest docs
  - That is useful if the user doesn’t want / need to index the document and still want to find all the other documents of the same category, whatever this category means

Classification based MLT

- ClassificationMLTHandler
  - String d = req.getParams().get(DOC);
  - ClassificationResult r = classifier.assignClass(d);
  - String c = r.getAssignedClass();
  - req.getSearcher().search(new TermQuery(new Term(classFieldName, c)), rows);

Automatic text categorization

- Once a doc reaches Solr
- We can use the Lucene classifiers to automate assigning document's category
- We can leverage existing Solr facilites for enhancing the indexing pipeline
  - An UpdateChain can be decorated with one or more UpdateRequestProcessors

Automatic text categorization

- Configuration
  - <updateRequestProcessorChain name=“ctgr">
  - <processor class=”solr.CategorizationUpdateRequestProcessorFactory">
  - <processor class="solr.RunUpdateProcessorFactory" />
  - </updateRequestProcessorChain>

Automatic text categorization

- CategorizationUpdateRequestProcessor
- *void **processAdd**(AddUpdateCommand cmd) throws IOException*
  - String text = solrInputDocument.getFieldValue(“text”);
  - String class = classifier.assignClass(text);
  - solrInputDocument.addField(“cat”, class);
- Every now and then need to retrain to get latest stuff in the current index, but that can be done in the background without affecting performances

Automatic text categorization

- CategorizationUpdateRequestProcessor
- Finer grained control
  - Use automatic text categorization only if a value does not exist for the “cat” field
  - Add the classifier output class to the “cat” field only if it’s above a certain score

Wrap up

- Simple classifiers with no or little effort
- No architecture change
- Both available to Lucene and Solr
- Still reasonably fast
- A lot more can be done
  - Implement a MaxEnt Lucene based classifier
    - which takes into account words correlation

Thanks!