IMDB Analysis.

Team «rotten tomatoes»

BagOfWords-Model.

1) Building a **vocabulary** of most frequent words / n-grams

['the film' = 1234, 'I liked' = 989, 'bad' = 892, ...]

2) **Vectorize** reviews (count occurrences of vocabulary entries)

Review: 'the film was bad' → Vector: [1,0,1]

3) Use word count as feature

@attribute 'the film' numeric
@attribute 'I liked' numeric
@attribute bad numeric

4) Use wekas attributeSelection algorithm to rank features according to their correlation, entropy etc: **CorrelationAttributeEval**

Our approach.

- We didn't know about wekas StringToWordVector()

- Idea: Use only words with a sentiment score > |0.25|

SentiWordNet ('good', 'A') → 0.5

Resulting vocabulary: **[masterpiece, bad, great, stupid, 'the best', poor, ridiculous, terrible, awful, dull, wasted, terrific, poorly, wonderfully, masterpiece, 'the only thing', 'of the worst', hilarious, ...]**

- Better score with 1000 prefiltered words than with 20k unfiltered words

Additional features.

- **Average review polarity (with SentiWordNet): ~67%**

$$f(review) = \frac{\sum_{j=1..T} pol(w_j)}{T(review)}$$

T = total number of words in review

```
Features.addFeature(new NummericFeature(„avgReviewPolarity",
    (review) -> {
        PipelineFactory
            .start(Preprocessing.negationFilter)
            .append(Preprocessing.luceneTokenizer)
            .append(SentiAnalysis.sentiWordNet)
            .append(SentiAnalysis.avgTextPolarity);
            .run(review);
      }
));
```

Additional features.

- **Average review purity**

$$f(\text{review}) = \frac{\sum_{j=1..T} \text{pol}(w_j)}{\left|\sum_{j=1..T} \text{pol}(w_j)\right|}$$

T = total number of words in review

- **Review length**
- **Count negative words / Count positive words**
- **Average first sentence polarity,?/! Count, etc. → removed**

Conclusion.

Overall score: 88.25%

- Extra work, no real profit

Lessons learned / Pitfalls:

- Exclude test-dataset from generation of the BagOfWords vocabulary.
- Build separate vocabularies for every sentiment class and merge them.
- Performance issues: parallelStream(), caches for pre-processing steps etc.

```
reviews.parallelStream().forEach(
  (review) -> {
      List<String> tokens = _cache.getOrDefault(review, null);
      if(tokens == null) {
          tokens = PipelineFactory
              .start(Preprocessing.negationFilter)
              .append(Preprocessing.nGramTokenizer)
              .append(Preprocessing.stopwordFilter)
              .run(review);
      }

      HashMap<String, Integer> wordVector = new HashMap();
      Map<String, Long> wordCount = tokens.stream().collect(
             Collectors.groupingBy(Function.identity(),
             Collectors.counting())
      );

      Iterator<Map.Entry<String, Integer> = vocab.keySet().iterator();
      while (it.hasNext()) {
          String word = it.next();
          vec.put(word, wordCount.getOrDefault(word, 0L).intValue());
      }
      return this.vec;
      //…
```

Questions?