Illinois Institute of Technology

# Distributed Classifiers

Fall 2010

# Distributed Naïve Bayes

Illinois Institute of Technology

- Step 1: Use Map-Reduce to obtain basic counts
  - Number of distinct terms
  - Number of documents that contain a term for a given class
  - Weight of occurrences of a term in a given class

    <put in math here>

  - Number of occurrences of a document in a class
  - Frequency of a term across the collection
- Step 2: Now compute weight (tf * idf) for each term
- Step 3:

# Distributed Naïve Bayes – Preprocessing Step 1

Illinois Institute of Technology

- Hadoop reads the entire data set into <class, document> pairs, where
  - document is a list of words in the document being read
  - Class is the class label or category of document. With the newsgroup example, this would be the newsgroup the document was posted in.

# Distributed Naïve Bayes – Preprocessing Step 1

Illinois Institute of Technology

The newsgroup posting data set is read into <*class*, *document*> pairs, where *class* is the newsgroup *document* was posted in

<misc.forsale, from rupin.dang@dartmouth.edu rupin dang subject nikon fm2 lens forsale nikon fm-2n 50 mm nikkor accessories sale.i bought camera hong kong two years ago everything has been looked after very well i'm now selling some more gear finance my next big film project asking 350 package bargains>

<rec.autos, from bw662@cleveland.freenet.edu bill cray subject re thinking about buying intrepid good bad idea i bought intrepid about two months ago am very happy lots room inside even smaller engine has enough power me only problem i found small selection dealer lots hot sellers around here>

…

# Distributed Naïve Bayes – Preprocessing Step 1

Illinois Institute of Technology

- The mapper outputs tuples of the following types
  - WEIGHT: a normalized term frequency (term count) with a <class, term> key
  - DF: the value 1 with a <class, term> key
    - Later we calculate each term's DF by summing these
  - FEATURE_COUNT: the value 1 with a <term> key
  - FEATURE_TF: a term key with the term's TF as the value
  - LABEL_COUNT: the value 1 with a <class> key

# Distributed Naïve Bayes – Preprocessing Step 1

Illinois Institute of Technology

The mapper receives the document as input and outputs tuples of five different types. Each 2-tuple's key is itself a 2- or 3-tuple.

<<*WEIGHT, class, term*>, $tf_{term}$>
<<WEIGHT, misc.forsale, bargains>, $tf_{bargains}$>
<<WEIGHT, rec.autos, intrepid>, $tf_{intrepid}$>
…
<<*DF, class, term*>, *1*>
<<DF, misc.forsale, bargains>, 1>
<<DF, rec.autos, intrepid>, 1>
…

# Distributed Naïve Bayes – Preprocessing Step 1

Illinois Institute of Technology

The mapper receives the document as input and outputs tuples of five different types. Each 2-tuple's key is itself a 2- or 3-tuple.

*<<FEATURE_COUNT, term>, 1>*
<<FEATURE_COUNT, bargains>, 1>
<<FEATURE_COUNT, intrepid>, 1>
…
*<<FEATURE_TF, term>, term count>*
<<FEATURE_TF, bargains, 1>
<<FEATURE_TF, intrepid, 1>
…
*<<LABEL_COUNT, class>, 1>*
<<LABEL_COUNT, misc.forsale>, 1>
<<LABEL_COUNT, rec.autos>, 1>
…

# Distributed Naïve Bayes – Preprocessing Step 1

Illinois Institute of Technology

- The step 1 reducer sums the values for each unique key and returns the input key and summation as its key-value pair
  - Each key is a tuple consisting of a static object indicating the tuple's type and the actual keys
- FEATURE_TF tuples are summed to calculate the term count of each term and written to disk
- LABEL_COUNT tuples are summed to calculate the total number of instances for each class and written to disk
- The remaining tuples are summed and passed on to step 2

# Distributed Naïve Bayes – Preprocessing Step 1

Illinois Institute of Technology

The step 1 reducer sums the values for each unique key and returns the input key and summation as its key-value pair

<<*WEIGHT*, misc.forsale, bargains>, $tf_1 + tf_2 + \ldots + tf_n$>
<<*WEIGHT*, rec.autos, intrepid>, $tf_1 + tf_2 + \ldots + tf_n$>
…

<<*DF*, misc.forsale, bargains>, *1 + 1 + … + 1*>
…

<<*FEATURE_COUNT*, intrepid>, *1 + 1*>
…

<<*FEATURE_TF*, bargains>, *term count*>
…

<<*LABEL_COUNT*, misc.forsale>, *1 + 1 + … + 1*>
<<*LABEL_COUNT*, rec.autos>, *1 + 1 + … + 1*>

# Distributed Naïve Bayes – Preprocessing Step 2 (compute IDF)

Illinois Institute of Technology

- The next mapper receives tuples from the previous step
  - WEIGHT tuples are passed on unchanged
  - DF tuples are changed to type WEIGHT and their values changed to IDFs
  - Now there are two WEIGHT tuples for each unique keys
  - FEATURE_COUNT tuples are all returned as a string

# Distributed Naïve Bayes – Preprocessing Step 2

Illinois Institute of Technology

The next mapper receives WEIGHT, DF, and FEATURE_COUNT tuples from the previous step. WEIGHT tuples are unchanged.

<<*WEIGHT,* misc.forsale, bargains>, $tf_{bargains}$>
<<*WEIGHT,* rec.autos, intrepid>, $tf_{intrepid}$>
*…*

DF tuples are changed to the WEIGHT type and their value is changed to the idf of their term, calculated using our results.
<<*WEIGHT,* misc.forsale, bargains>, $idf_{bargains}$>
*…*

FEATURE_COUNT tuples lose their class and change type.
*<VOCAB_COUNT, 1>*
*…*

# Distributed Naïve Bayes – Preprocessing Step 2

Illinois Institute of Technology

- The reducer receives these tuples and processes them further
  - The total number of FEATURE_COUNT tuples is summed, representing the total number of terms in the data set
  - WEIGHT tuples' TF and IDF components are multiplied to calculate and return the key's TF-IDF
    - This works because there is one WEIGHT TF tuple and one WEIGHT IDF tuple for each key
    - This happened in the map step when the DF tuple was changed to a WEIGHT tuple

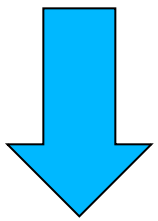
# Distributed Naïve Bayes – Preprocessing Step 2

Illinois Institute of Technology

The reducer receives VOCAB_COUNT and WEIGHT tuples.

It sums VOCAB_COUNT tuples and stores the result as the number of features in the data set
Features = <VOCAB_COUNT, 1 + 1 + … + 1 + 1>

Each term's two weight tuples are multiplied to calculate TF-IDF
<<*WEIGHT*, misc.forsale, bargains>, $tf_{bargains}$>
<<*WEIGHT*, misc.forsale, bargains>, $idf_{bargains}$>



<<*WEIGHT*, misc.forsale, bargains>, $tf_{bargains} * idf_{bargains}$>
…

# Distributed Naïve Bayes – Preprocessing Step 3

Illinois Institute of Technology

- The value of the one FEATURE_COUNT tuple is written to disk as the total number of features (terms)
- The WEIGHT tuples containing TF-IDFs are also stored
- Mapper 3 receives WEIGHT tuples from the previous step
  - For each input tuple it outputs three new types of tuples with <class, term> keys and TF-IDF values
  - featureSum tuple with a <term> key and TF-IDF value
  - labelSum tuple with a <class> key and TF-IDF value
  - totalSum tuple with no key (other than the tuple's totalSum type) and TF-IDF value
- The reducer sums each of these fields and writes each key-sum pair to disk

# Distributed Naïve Bayes – Preprocessing Step 3

Illinois Institute of Technology

Mapper 3 receives WEIGHT tuples from the previous step and outputs three new types of tuple

<<featureSum, term>, $TF\text{-}IDF_{term}$>
<<featureSum, intrepid>, $TF\text{-}IDF_{intrepid}$>
<<featureSum, bargains>, $TF\text{-}IDF_{bargains}$>
…
<<labelSum, class>, $TF\text{-}IDF_{term}$
<<labelSum, rec.auto, $TF\text{-}IDF_{intrepid}$ >
<<labelSum, misc.forsale, $TF\text{-}IDF_{bargains}$ >
…
<totalSum, $TF\text{-}IDF_{term}$>
<totalSum, $TF\text{-}IDF_{intrepid}$>
<totalSum, $TF\text{-}IDF_{bargains}$>
…

The final reducer simply sums the values for each unique key and stores the final values.

# Distributed Naïve Bayes – Performance

Illinois Institute of Technology

- Run distributed Naïve Bayes on Hadoop against the 25mb 20-newsgroups data set
- Break each term into 3-grams (increases workload)
- Compare with Hadoop clusters of various sizes running on identical hardware

| | 1 node | 4 nodes |
|---|---|---|
| Time | 794 seconds | 396 seconds |