# CS6240: Project Presentation

ANISH ALATE

ANUP PATIL

# A bit something about the Data

- Where is the data from?
- What is the format of Data?
- How much data are we talking about?

->**1.6M** reviews and **500K** tips by **366K** users for **61K** businesses

# Project Tasks

- TASK 1: Classification and prediction of User Reviews using Data Mining libraries based on the star Rating.

- TASK 2: Decision trees to Predict of success of restaurant business.

# Task1: Classification and Prediction of User Reviews

- What is the purpose of the task?

- How did we solve this task?
  - Naïve Bayes Classifier
  - Spell Checker
  - Positive & Negative sentiment Dictionaries
  - And of course Hadoop's MapReduce

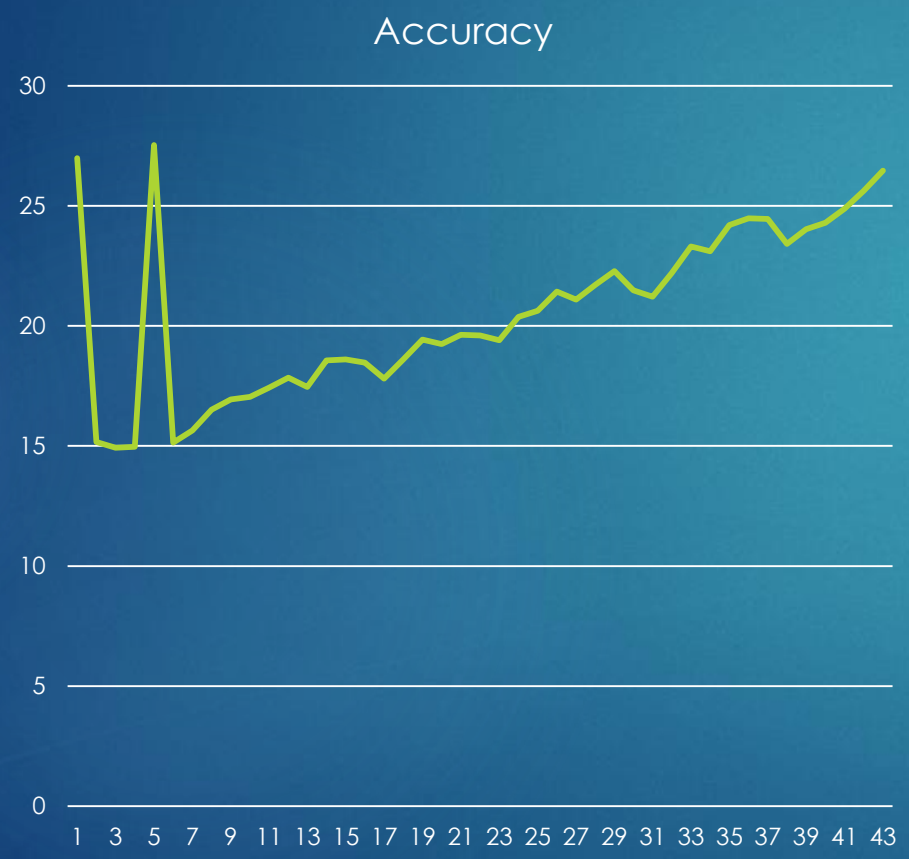
# The MapReduce aspect! Take1: Monogram Model

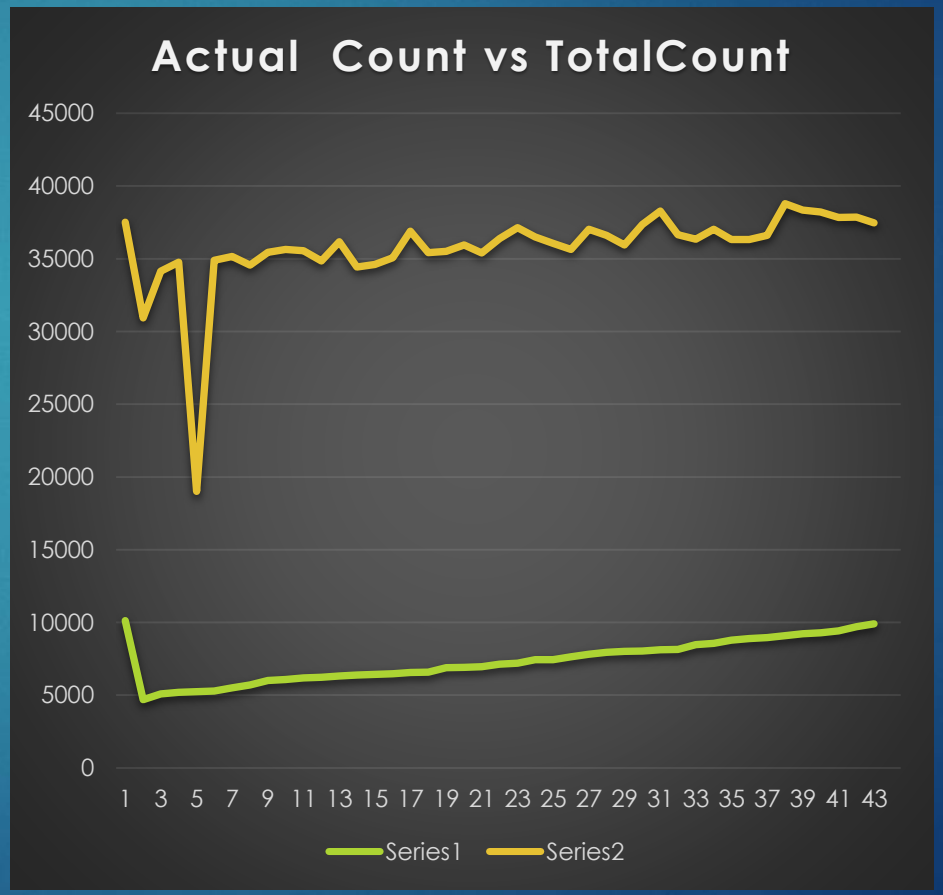
- A Map only task

```
Main(args)
{
        Initialize PositiveSentimentHashmap and populate it
        Initialize NegativeSentimentHashmap and populate it
}
Setup()
{
        Tcount =0
        ActCount =0
}

Map(key,value)

{

        JsonParser p = parseLinefromReviewText;

        Star = getstar from p

        ReviewText = getReviewtext from p.

        negCount = posCount = tCount =0

        guessedStar =0

For each word in ReviewText:
                If posHash.lookup(word)
                        posCount ++
                if negHash.lookup(word)
                        negCount ++
        get the percentage occurrence of positive  words
and negative words.
        doConditionChecking and predict the star
rating.(guessedStar = val)
        Tcount++
        if(guessedStar == Star)
                actCount ++
}
Cleanup()
{
        Write(tCount,actCount)
}
```

# Results?

Average Accuracy: 22.54%



# Other (Not so successful)Takes

- Take 2: N-gram Model
  - Used an online API http://sentiment.vivekn.com/docs/api/
  - Implements a Naïve Bayes Classifier with Laplacian Smoothing
  - But it was trained on movie reviews!
  - Network was a bottleneck.
  - Accuracy reported = 39.24%
- Take 3: Weka
  - Wrote a simple MR program to create the training data required for Weka.
  - Takes almost 2/3rd data as training data
  - Faced challenges setting it up
  - Couldn't customize certain aspects and we didn't have fine grain control.

# The Final take(Take something from all)

```
Main(args)
{
        Initialise the bayes classifier
        TrainBayes(bayes)
        Initialize PositiveSentimentHashmap
        Initialize NegativeSentimentHashmap
}

TrainBayes(bayes)
{
        Read from review training file
        If star ==1
                Bayes.add(1, review)
        If star ==2
                Bayes.add(2, review)
        If star ==3
                Bayes.add(3, review)
        If star ==4
                Bayes.add(4, review)
        If star ==5
                Bayes.add(5, review)
}
```

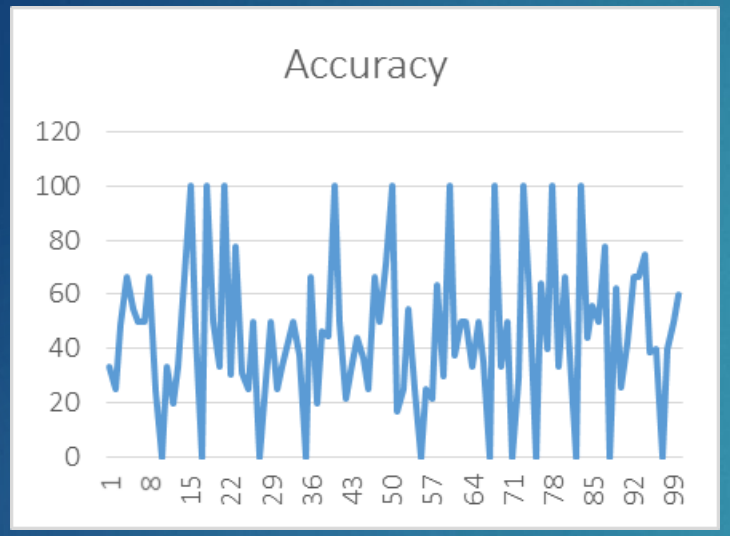
```
ReviewMapper(key, value)

{

        JsonParser p = parseLinefromReviewText;

        Star = getstar from p

        ReviewText = getReviewtext from p.

        SpellCheckText(ReviewText) //calling spell check
api and getting corrected sentence

        Business id = getbusinessid from p

        Ourstar = bayes.predict(ReviewText)

        If(ourStar != star)

                ourStar =useDictionarylogic(reviewtext)


        emit(businessid , Star-Ourtar)

 }
```

```
useDictionarylogic(reviewText)

{
        negCount = posCount = tCount =0
        guessedStar =0
        For each word in ReviewText:
                If posHash.lookup(word)
                        posCount ++
                if negHash.lookup(word)
                        negCount ++
        get the percentage occurrence of positive words
and negative words.
        doConditionChecking and predictthe star
rating.(guessedStar = val)
return guessedStar

}


BusinessMapper(key,value)
{
        JsonParser p = parseLinefromReviewText;
        Star = getstar from p
        Business id = getbusinessid from p
        Emit(businessid, star)
}
```

```
ReviewReducer(key, val1,val2.....)
{
        totCount=actCount=0
        For val in value_iterable
        {
                If(val.lenght ==1)
                {
                        Bstar = val
                }
                Break;
        }
        For val in value_iterable
        {
                If(val.length >1)
                {
                        Star = val[0]
                        ourStar =val[1]
                        totCount ++
                        if(star ==ourStar)
                                actCount++
                        else
                        {
                                ourStar = bstar
                                if(star == ourStar)
                                        actCount ++
                        }
                }
        Emit(actCount,totCount)
}
```

# Results

- Variation 1(Without Spell checker)

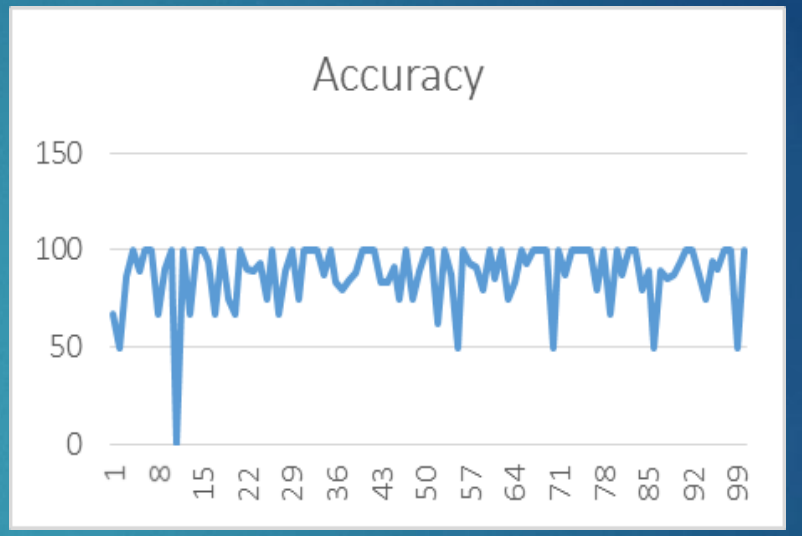



Average Accuracy:48.44%

- Variation 2(With Spell Checker)

Average Accuracy 57.21%

- Variation 3(Binary Classification)




Average Accuracy: 86.03%

# Is the program Scalable?

| | 1 master 5 workers (all small) | 1 master 10 workers(all small) | 1 master 15 workers(all small) |
|---|---|---|---|
| **MapReduce Task Run time** | 245 seconds | 175 seconds | 151 seconds |
| **Total Run Time** | 329 seconds | 291 seconds | 246 seconds |

# What can be done to better this?

- Write another Map only job  to train the classifier
- Use n-gram model for better accuracy
- Use negation handling of text data
- Try and compare results using different classifiers

# Task 2: Using Decision Tree for Prediction of Business Success

- Data provided in form of JSON objects.
- Business facilities provided in form of attributes.
- Nature of provided Business Attributes:
- "attributes": {"Alcohol": "full_bar", "Noise Level": "average", "Has TV": true, "Attire": "casual", "Good for Kids": true, "Price Range": 1, "Good For Dancing": false, "Delivery": false, "Coat Check": false, "Smoking": "no", "Accepts Credit Cards": true, "Take-out": true, "Happy Hour": false, "Outdoor Seating": false, "Takes Reservations": false, "Waiter Service": true, "Wi-Fi": "no", "Good For": {"dessert": false, "latenight": false, "lunch": false, "dinner": false, "breakfast": false, "brunch": false}, "Parking": {"garage": false, "street": false, "validated": false, "lot": false, "valet": false}, "Good For Groups": true}

# 3 Phases of Task

- Phase 1 : Filtering the restaurant records from all business records
  Learning attribute domain ranges
- Phase 2 : Parallel decision tree Induction (forming tree and training)
  1. Iterative Driver Program Using Output of Last iteration as input to next iteration
  2. Using CustomFileInputFormat to read all records relevant for mapper instead of a line at a time.
  3. Writing in different output files
  4. Creation of on disk decision tree
- Phase 3 : Prediction success for test records using Decision Tree
  1. Load on disk decision tree in Distributed Cache
  2. Create in memory Decision Tree
  3. Traverse tree to appropriate branch using test record attribute values

# Parallel Decision Tree Induction

```
// For simplicity, this program assumes that each node
// performs a binary split into "left" and "right" partition
map( nodeID, listOfRecords ) {
  leftPartition = {}
  rightPartition = {}

  // Find best split predicate for the node
  for each attribute A
    for each possible split point S
      computeScore( listOfRecords, A, S )
      Keep track of the (A, S) pair with the highest score

  // Let (bestA, bestS) be the winning split predicate
  // Partition the data according to the split predicate
  for each record r in listOfRecords
    if satisfiesPredicate(r , (bestA, bestS) )
      leftPartition.add( r )
    else
      rightPartition.add( r )

  // Write the node information containing nodeID, split predicate,
  // and child node IDs to a file storing the tree structure
  treeFile.write( nodeID, (bestA, bestS), newID(nodeID, left), newID(nodeID, right) )

  // Emit the two partitions for further splitting in the next iteration
  if not stoppingConditionMet( leftPartition )
    emit( newID(nodeID, left), leftPartition )

  if not stoppingConditionMet( rightPartition )
    emit( newID(nodeID, right), rightPartition )
}
```

# More about Decision Tree Algorithm:

- Instead of just left child and right child, the branching was dependent of domain range of attribute.
- Use of Entropy / Information Gain to decide best splitting attribute.
- Entropy: Entropy  is a measure of the amount of uncertainty in the data set . Entropy is calculated for each remaining attribute.
- Information gain IG(A)  is the measure of the difference in entropy from before to after the set  is split on an attribute A. In other words, how much uncertainty in  was reduced after splitting set  on attribute A.
- Attribute with best gain is used for splitting.

# Algorithm used for prediction

```
// The file with the decision tree is made available
// through the distributed file cache
Class Mapper {
  tree

  setup()
    tree = load tree from file cache

  map( …, test record r ) {
    output = tree.makePrediction( r )
    emit( r, output)
  }
}
```

# Results:

- Using 25 different attributes, 22 level tree was generated in breadth first manner
- Accuracy ranged from 71% to 84%:
- When success is considered for starRating > 2.5 Success was 81%
- When success is considered for starRating > 3 Success was 77%
- When success is considered for starRating >= 3 Success was 75%
- Attributes dominated at top level of Decision Tree Noise Level, Parking, Takeout, Alcohol, Good for Lunch/Dinner
- The increased percentage between 2.5 and 3 star rating shows that there are many records in the range of 2.5 to 3 star rating which can be classified on either side to predict the success.

# Running time and Scalability

| | 1 master 5 workers (all large) | 1 master 10 workers (all large) | 1 master 14 workers (all large) |
|---|---|---|---|
| Filtering Task | 4 min | 3 min | 1 min |
| Induction Task | 110 min | 68 min | 61 min |
| Predication Task | 60 sec | 50 sec | 50 sec |

- Since tree induction driver program iteratively sets the map reduce jobs, delay gets introduced between two successive jobs which increased overall running time.
- As branching increases, the number of record files available for mappers also increase. In initial phases there is limit on map tasks, which restricts parallelism, but since there are 25+ attributes available for partitioning with millions of records, branching increases rapidly and more and more data files are available for subsequent mappers.

# Improvements

- Learning business attributes for different business categories, rather than pre deciding them.
- Having one decision tree for each business category, this can utilize parallelism in more effective manner.
- Reducing HDFS I/O as each iteration performs read and write to HDFS.

# References


1. Philipp Nolte: Java based implementation of Naïve Bayes Classifier
   https://github.com/ptnplanet/Java-Naive-Bayes-Classifier
2. Jeffrey Breen : For the dictionaries of positive and negative words
   https://github.com/jeffreybreen/twitter-sentiment-analysis-tutorial-201107/tree/master/data/opinion-lexicon-English
3. Jazzy : The Java open source spell checker
   http://jazzy.sourceforge.net/
4. Jazzy tutorial based code for the spell checker implementation
   http://moderntone.blogspot.com/2013/02/tutorial-on-jazzy-spell-checker.html
5. English Dictionary for the spellchecker
   http://introcs.cs.princeton.edu/java/data/words.utf-8.txt
6. Yelp DataSet challenge for the Dataset
   http://www.yelp.com/dataset_challenge/
7. Building custom delimiter (custom input format class)
   https://hadoopi.wordpress.com/2013/05/31/custom-recordreader-processing-string-pattern-delimited-records/
   https://github.com/aamend/hadoop-recordreader/tree/master/src/main/java/com/aamend/hadoop/recordreader
8. Formula for entropy and Information gain:
   http://en.wikipedia.org/wiki/ID3_algorithm
9. json Parser
   https://code.google.com/p/json-simple/
10. Generic Queue data Structure
    http://www.tutorialspoint.com/javaexamples/data_queue.htm

# Thank You !

Planning to Work Ahead with improvements …

# Questions?