Reddit Comment Analysis

Project Final

Table of Contents

Introduction

Background

Reddit is a massive online community wherein users can post content into content-specific sub-communities called ‘subreddits’ to be viewed by others within and without the subreddit. Users can ‘upvote’ or ‘downvote’ posts to shift their rankings on the website and in this way as a community dictate which posts are more easily seen by the wider community and which are not.

Users can also comment on and discuss posted content under a chosen pseudonym in each post’s ‘thread’ and comments can also be upvoted or downvoted as well as ‘gilded’ to show the community’s reaction to them. Reddit also uses other metrics to score comments such as ‘controversiality’ and ‘distinguished’.

A dataset containing all comments made on Reddit in January 2015 was released by a Reddit user, detailing for each comment; the sub-reddit it belongs to, the user who posted it, the number of upvotes and downvotes it received as well as other information as will be shown in the MongoDB section of this report.

Project Objectives

Given this dataset, a lot can be learned about the nature of online communication and various data handling tools can be used to process the data. This project will focus on the use of databases, specifically the MongoDB noSQL query language, and word processing to come to an understanding of the dynamics of the Reddit online community and how the way people communicate online leads to different reactions from the community.
Specific tasks have been selected to implement various tools and are as follows:

Data Exploration:

- Set up a MongoDB database for the dataset
- Use Spark to extract information about the dataset
- Use SQL and Databricks to create visualizations

Machine Learning and Featurization Tasks:

- Compare Feature Hashing vs Bag of Words representation
- Determine if words are a good predictor of the score of the comment
- Determine if score is a good predictor of gilded comments
- Train a classifier to match comments to certain subreddits

Cluster Tasks:

- Create a multi node cluster on Databricks / Amazon EC2 to run Spark tasks
- Perform computations and machine learning tasks on all the data efficiently

Description of Data

A brief overview of the file if given below.

Filename	rc_2015_1.bz2	
Total Comments	53,851,542	
Compression Type	bzip2	
Byte Size	Compressed	5,452,413,560
	Uncompressed	31,648,374,104

The following is an explanation of each key in the database.

gilded - Number of 'reddit golds' a person was given, costs the donor money
author_flair_text - Gender
author_flair_css_class - Styling used for author's username
retrieved_on - Date+time
ups- Number of upvotes
subreddit_id- Subreddit ID linked to a different table/database
edited-Boolean; if the comment was edited or not
controversiality- Related having a high number of upvotes and downvotes. Indicated the community having a mixed reaction to the comment
parent_id - Unique identifier of the comment's parent
subreddit - Name of subreddit comment is a part of
body - Text of comment
created_utc - Time and date of comments creation
downs - Number of downvotes
score - Overall score of comment
author - Author's name of the comment
archived - Identifies if the comment has been archived
distinguished - Identify if a subreddit moderators has marked the comment as 'distinguished'
id - Unique identifier for the comment
score_hidden - Boolean; score shown on webpage or not
name - Username of the commenter
link_id - Link ID of the comment

Example of a comment from the dataset.


```
{
  "gilded": 0,
  "author_flair_text": "Male",
  "author_flair_css_class": "male",
  "retrieved_on": 1425124228,
  "ups": 3,
  "subreddit_id": "t5_2s30g",
  "edited": false,
  "controversiality": 0,
  "parent_id": "t1_cnapn0k",
  "subreddit": "AskMen",
  "body": "I can't agree with passing the blame, but I'm glad to hear it's at least helping you with the anxiety. I went the other direction and started taking responsibility for everything. I had to realize that people make mistakes including myself and it's gonna be alright. I don't have to be shackled to my mistakes and I don't have to be afraid of making them. ",
  "created_utc": "1420070668",
  "downs": 0,
  "score": 3,
  "author": "TheDukeofEtown",
  "archived": false,
  "distinguished": null,
  "id": "cnasd6x",
  "score_hidden": false,
  "name": "t1_cnasd6x",
  "link_id": "t3_2qyhmp"
}
```

Development Environment

In this section we will investigate multiple different solutions to perform analysis on our dataset and determine which solution is most optimal to operate on 30GB of Reddit comments.

Data Storage

The data is currently stored on AWS S3. EC2, EMR and Databricks all provide methods to retrieve data from S3. The data can also be accessed using the python interface **boto**, using the **AWS cli** or using **wget**.

Name	Storage Class	Size	Last Modified
RC_2015-01.bz2	Standard	5 GB	Sat Nov 19 13:46:20 GMT+100 2016
output	--	--	--
rc_minimized_100.bz2	Standard	16.7 MB	Sun Dec 04 02:40:11 GMT+100 2016
rc_minimized_1000.bz2	Standard	170.4 MB	Sun Dec 04 03:03:49 GMT+100 2016
scripts	--	--	--

Figure 1: Compressed dataset uploaded to S3, includes minimized datasets.

AWS EC2

First, we wanted to install Spark on an Amazon Elastic Compute Cloud (EC2) instance and determine its capabilities. The initial idea was to create a cluster of EC2 instances that would perform Spark tasks. The following steps below outline hown

Creating EC2 Instance

First, a security group is created to expose the EC2 instance illustrated in Figure 2.

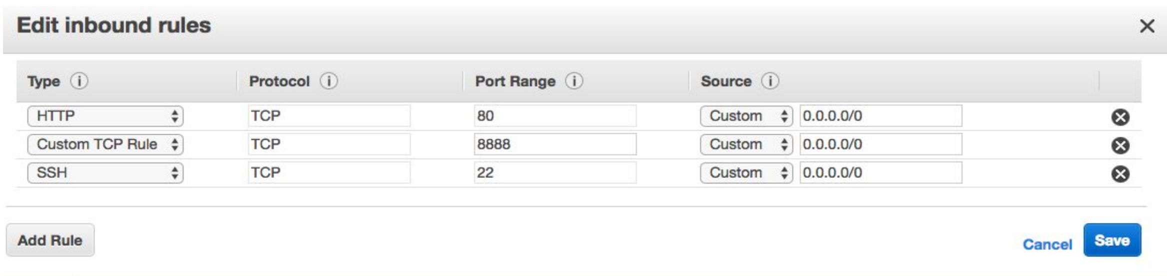

Figure 2: The inbound rules set for the EC2 instance

Setting the rule for SSH will allow us to connect to the EC2 instance from our local machine using SSH. In order to access the jupyter notebook we must expose HTTP port 80 and 8888. We can then connect to **ec2-XX-XXX-XXX-XXX.compute-1.amazonaws.com:8888** to view the jupyter notebook running Pyspark once these components are installed.

Following the steps on the Amazon EC2 dashboard, we can create an instance and assign it the security group created above. We also created a keypair and downloaded the corresponding pem file, **mypem.pem**. If the instance was created and launched successfully you should be able to see something similar to Figure 3 in the dashboard.

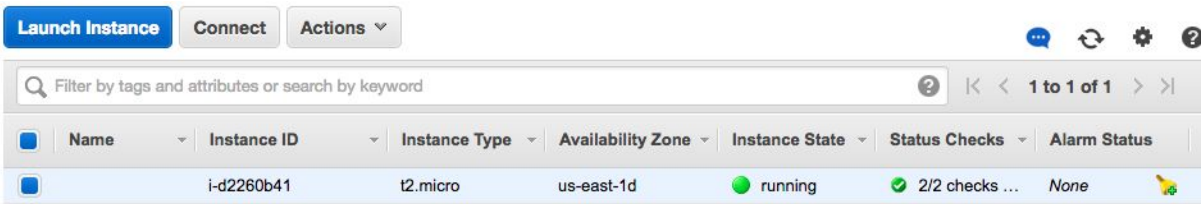

Figure 3: Dashboard containing EC2 instances

Once the instance was launched, the following commands were ran to connect to the instance.

```
chmod 400 mypem.pem
ssh -i "mypem.pem" ec2-user@ec2-54-XXX-XXX-XXX.compute-1.amazonaws.com
```

The first command will provide the proper access to the pem file and the second command will SSH into our remote EC2 instance. Once connected to the remote host, packages can be installed.

Installing and Running Pyspark on EC2

Before installing Spark packages, we must update the current packages and install git. Git will be used to clone packages.

```
sudo yum update
sudo yum install git
```

Many packages require AWS_KEY_ACCESS_ID and AWS_SECRET_ACCESS_KEY environment variables be set. Running **aws configure** will also us to set these variables.

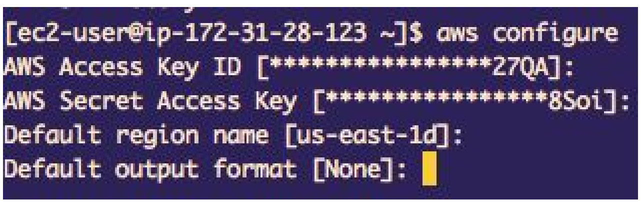

Figure 4: Setting AWS keys, keys already entered.

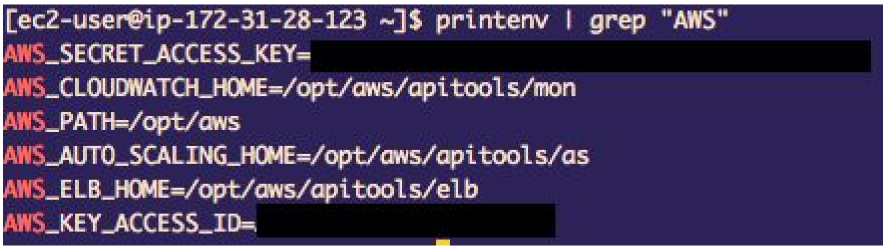

Figure 5: Checking current environment variables

Next we need to install Spark. The package is remotely installed on the instance using **wget** and unpackaged using **tar**.

```
wget http://mirrors.dotsrc.org/apache/spark/spark-2.0.2/spark-2.0.2-bin-hadoop2.7.tgz
tar -zxvf spark-2.0.2-bin-hadoop2.7.tgz
```

Next we need to set pyspark to the path. We open **.bashrc** with **vi** to edit the file and add the following alias. This will connect pyspark to our jupyter notebook. The terminal is then restarted.

```
function snotebook ()
{
    #Spark path (based on your computer)
    SPARK_PATH=~/spark-2.0.2-bin-hadoop2.7
    export PYSPARK_DRIVER_PYTHON="jupyter"
    export PYSPARK_DRIVER_PYTHON_OPTS="notebook"
    $SPARK_PATH/bin/pyspark --master local[2]
}
```

Anaconda is then installed so we can use Pyspark in a jupyter notebook. Anaconda is downloaded remotely using **wget** and then installed by running the **bash** script.

```
wget https://repo.continuum.io/archive/Anaconda2-4.2.0-Linux-x86_64.sh
bash Anaconda2-4.2.0-Linux-x86_64.sh
```

Finally we need to make sure that we can access the notebook on my local machine and that it's protected. In the ipython terminal we run the following commands and are then prompted to set the password.

```
$ source .bashrc
$ ipython
[1]: from IPython.lib import passwd
[2]: passwd()
```

Next we create a certificate for HTTPS.

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout other.key -out
other.pem
```

Lastly, we open the jupyter config file with the following commands.

```
jupyter notebook --generate-config
cd ~/.jupyter/
vi jupyter_notebook_config.py
```

And then add the text from Figure 6 to the jupyter_notebook_config.py file.

```
# Configuration file for jupyter-notebook.

c = get_config()

# Kernel config
c.IPKernelApp.pylab = 'inline'  # if you want plotting support always in your notebook

# Notebook config
c.NotebookApp.certfile = u'/home/ec2-user/certificates/other.pem' #location of your certificate file
c.NotebookApp.keyfile = u'/home/ec2-user/certificates/other.key' #location of your certificate key
c.NotebookApp.ip = '*'
c.NotebookApp.open_browser = False  #so that the ipython notebook does not opens up a browser by default
c.NotebookApp.password = u'sha1:ceb9999bfba0:a010f143fa2d99779088fe998924dde5021b38f2'
# It is a good idea to put it on a known, fixed port
c.NotebookApp.port = 8888
```

Figure 6: Configurations to password protect Jupyter Notebook

We test that everything has been installed correctly by opening the Spark Test Python notebook provided in class. The notebook can be uploaded using the Jupyter Notebook GUI.

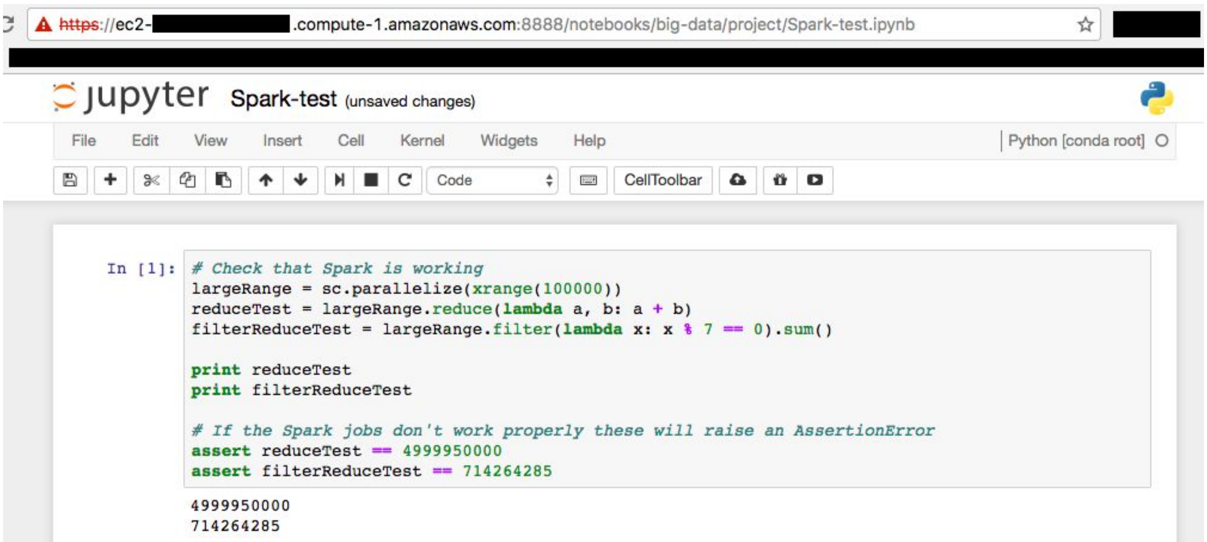

Figure 7: Testing pyspark on EC2 instance using Spark Test.ipynb

AWS EMR

AWS Elastic Map Reduce is a service built on top of EC2 to quickly create a MapReduce cluster, this includes Apache Spark. Amazon EMR Steps are used to submit work to the

Spark framework installed on an EMR cluster. Figure 8 illustrates how an EMR cluster is created.

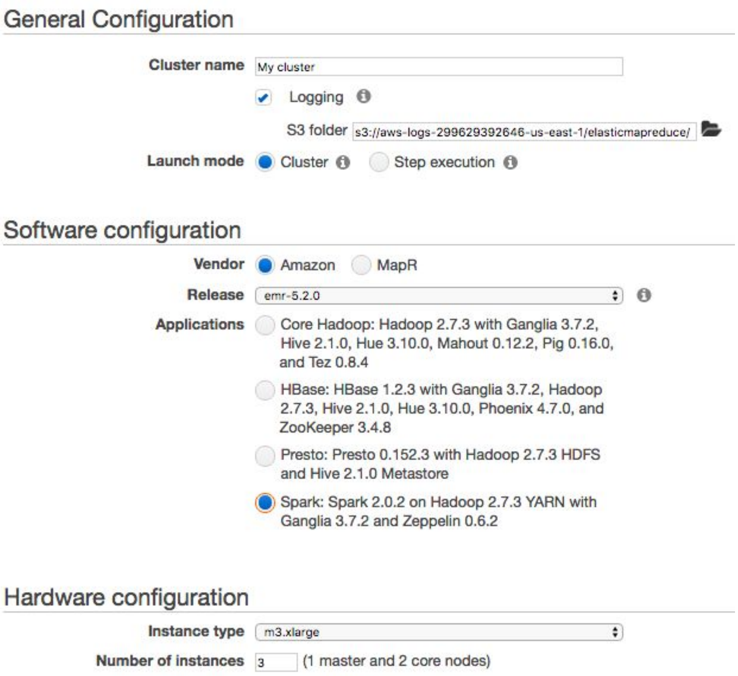

Figure 8: Creating a cluster with EMR

The script in Figure 9 is used to test using pyspark on EMR. In order to run the pyspark file, the script must first be uploaded to S3 then a **step** must be created. A step can be created using the AWS cli.

```
aws emr add-steps --cluster-id <cluster_id> --steps Type=Spark,Name="Spark Program",
Args=[--class,org.apache.spark.examples.SparkPi,~/big-data/project/test_spark.py]
```

```
import sys
from random import random
from operator import add

from pyspark import SparkContext


if __name__ == "__main__":
    """
        Usage: pi [partitions]
    """
    sc = SparkContext(appName="PythonPi")
    partitions = int(sys.argv[1]) if len(sys.argv) > 1 else 2
    n = 100000 * partitions

    def f(_):
        x = random() * 2 - 1
        y = random() * 2 - 1
        return 1 if x ** 2 + y ** 2 < 1 else 0

    count = sc.parallelize(xrange(1, n + 1), partitions).map(f).reduce(add)
    print "Pi is roughly %f" % (4.0 * count / n)

    sc.stop()
```

Figure 9: Script to test creating a step on EMR (reference: http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-spark-application.html)

Databricks

Databricks is a cloud-based solution for data processing using Apache Spark, it was also created by the founders of Apache Spark. Databrick provides an easy way to setup a Spark cluster. IPython notebooks can be created to run Spark scripts. The notebooks also support SQL, R and Scala.

Figure 10 demonstrates how a cluster is created on Databricks. Further options below specify which type of AWS EC2 instance is to be selected. The cheapest option was chosen at US$2.00 an hour. Once completed, the cluster should appear in the list of active clusters.

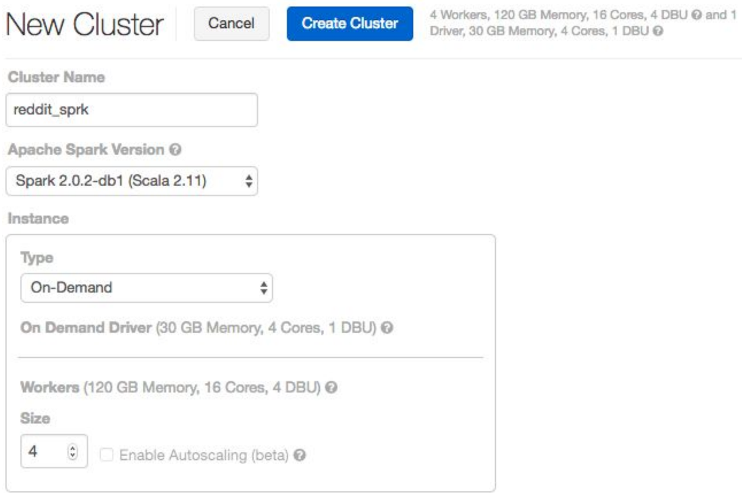

Figure 10: Create a cluster on Databricks

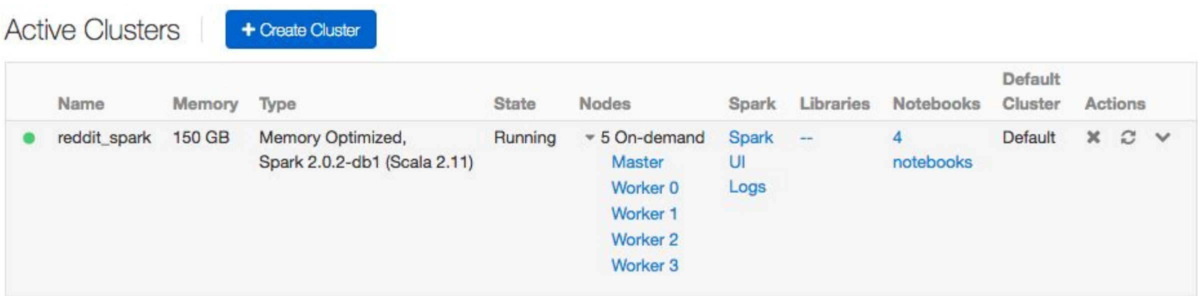

Figure 11: Statistics about active cluster

Loading Data into Databricks

The reddit comment dataset can be loaded directly from S3. The following code in Figure 12 mounts the S3 bucket to the databricks filesystem. The dataset is currently compressed using the bz2 format, the uncompressed the file size is approximately 30GB. The dataset will be converted to a Spark DataFrame. A DataFrame is essentially an organized dataset, the dataset is organized into rows and columns. By using a DataFrame we can perform computations on large datasets while not overloading the RAM and causing the program to crash. When creating the test DataFrame we must first load a list as an RDD then create the DataFrame using the RDD.

```
import urllib
# Access keys for AWS
ACCESS_KEY = [redacted]
SECRET_KEY = [redacted]
ENCODED_SECRET_KEY = urllib.quote(SECRET_KEY, '')
AWS_BUCKET_NAME = 'big-data-project-[redacted]
MOUNT_NAME = 'reddit_data'

# Mount S3 bucket to databricks filesystem
try:
  dbutils.fs.mount("s3n://%s:%s@%s" % (ACCESS_KEY, ENCODED_SECRET_KEY, AWS_BUCKET_NAME),
"/mnt/%s" % MOUNT_NAME)
except Exception as e:
  print "Already mounted."

display(dbutils.fs.ls("/mnt/"+MOUNT_NAME))
```

path	name	size
dbfs:/mnt/reddit_data/RC_2015-01.bz2	RC_2015-01.bz2	5452413560
dbfs:/mnt/reddit_data/output/	output/	0
dbfs:/mnt/reddit_data/rc_minimized_100.bz2	rc_minimized_100.bz2	17604206
dbfs:/mnt/reddit_data/rc_minimized_1000.bz2	rc_minimized_1000.bz2	178731611
dbfs:/mnt/reddit_data/scripts/	scripts/	0

Figure 12: Mounting dataset saved in S3 to Databricks filesystem.

The dataset is loaded into the notebook in Figure 13 using the load_data function found in Figure 12. Deleted comments are removed and there is a total of 50580999 comments and the command took 8.14 minutes.

```
def load_data(filename, limit=None):
    '''
    Returns a DataFrame containing all the comments.

    @params:
        filename
    '''

    if limit:
      df = spark.read.json(filename).limit(limit)
    else:
      df = spark.read.json(filename)

    # return a new DataFrame that doesn't contain deleted comments
    return df.filter("body != '[deleted]'")
```

Figure 12: Function to load data, can specify a subset of the data using limit.

```
> filename = "/mnt/" + MOUNT_NAME + "/RC_2015-01.bz2"

  # load the comments into a DataFrame
  commentDF = load_data(filename)

  # Display comments and information
  print "Snippet of Comment DataFrame:"
  commentDF.select('id', 'body', 'ups', 'downs', 'gilded', 'subreddit').show(5)
  print "Column names of comment DataFrame:"
  print commentDF.columns
  print "\nThe total number of comments: %s (deleted comments removed)" % commentDF.count()
```

```
Snippet of Comment DataFrame:
+-------+--------------------+---+-----+------+-------------+
|     id|                body|ups|downs|gilded|    subreddit|
+-------+--------------------+---+-----+------+-------------+
|cnas8zv|Most of us have s...| 14|    0|     0|     exmormon|
|cnas8zw|But Mill's career...|  3|    0|     0|CanadaPolitics|
|cnas8zx|Mine uses a strai...|  1|    0|     0| AdviceAnimals|
|cnas8zz|Very fast, thank ...|  2|    0|     0|    freedonuts|
|cnas900|The guy is a prof...|  6|    0|     0|           WTF|
+-------+--------------------+---+-----+------+-------------+
only showing top 5 rows

Column names of comment DataFrame:
['archived', 'author', 'author_flair_css_class', 'author_flair_text', 'body', 'controversial
ity', 'created_utc', 'distinguished', 'downs', 'edited', 'gilded', 'id', 'link_id', 'name',
 'parent_id', 'retrieved_on', 'score', 'score_hidden', 'subreddit', 'subreddit_id', 'ups']

The total number of comments: 50580999 (deleted comments removed)
```

Figure 13: Loading the full dataset and displaying statistics.

Discussion

Databricks is the preferred environment to perform operations on. Databricks is the easiest to setup and also the easiest to write scripts for. A huge benefit of using Databricks is the option write jobs using notebooks, which allows for easy visualizations of data which is extremely important. EMR is also easy to setup up but the lack of notebooks makes it difficult to debug any errors. EC2 lacks the power the other clusters have and it is initially not set up as a cluster. The extra work to create a cluster of EC2 instances is not desired. For all of the reasons Databricks was chosen as the primary computing method. This is further outlined in the table below.

Table 1: Pros and Cons of each Development Environment

	Pros	**Cons**
EC2	- Free (if using t2.micro) - Can work with notebooks so it's very easy to visualize the code	- Very hard to cluster - Not much power (only 1GB memory) - Long setup time
EMR	- Easy to setup a cluster - High flexibility of cluster type	- Hard to debug software - UI is confusing and hard to use - Can't use Notebooks (potentially but not trivial to setup
Databricks	- Can use notebooks - Can write in native SQL - Easy to setup up a cluster - Easy to connect to S3	- Most expensive option (cheapest option is around 15kr an hour)

Data Exploration and Visualization

In this section we experiment with multiple different ways to extract information and visualize our data. Each section was performed on different amounts of data, notably the MongoDB section was performed on 10GB while the Spark SQL section was performed on the entire dataset

Our data was analyzed on two platforms:

- Mongo Database
- Databricks (Spark and SQL)

MongoDB

MongoDB is a NoSQL database and was chosen since it matches the format of our dataset perfectly. Since our data contains only comment objects that link to other comments it did not make sense to create a relational database with these objects.

We wanted to know what were the top 5 subreddits and users as well as bottom 5 subreddits and users based on upvotes. We also wanted to see what were the highest rated comments and most negative comments based on other users voting on the comment. We used the first 10gb of data (first 17million lines) when processing these queries.

First we downloaded the reddit comment database file, torrented it, then decompressed the result using the following command in the terminal:

```
bzip2 -d reddit_database.bz2
```

Then we ran the following Python code to give us the 10gb file that we will use to upload into our mongo database. This code will write the first 17,000,000 lines from the decompressed ‘reddit_database’ file into the ‘new_database’ file.

```
#Get first 10gb of data
N=17000000
f=open("reddit_database")
f2=open("new_database", "w+")
for i in range(N):
    line=f.next().strip()
    f2.write(line)
f.close()
f2.close()
```

We then imported the database into MongoDB with the following command:

```
mongoimport --db test -c reddit_comments --file ~/Dropbox/Big\ Data/new_database
```

To connect to the MongoDB database and print out the keys we used the following:

```
#Connect to DataBase
from pymongo  import MongoClient
connection = MongoClient()
db = connection.test


# Find all Keys in MongoDB reddit_comments table
cursor = db.reddit_comments.find()
print "Column names:"
print [key for key in cursor[0]]
```

Giving us the output:

```
Column names:
[u'subreddit_id', u'subreddit', u'id', u'gilded', u'archived', u'author', u'parent_id',
u'score', u'retrieved_on', u'controversiality', u'body', u'edited',
u'author_flair_css_class', u'downs', u'link_id', u'score_hidden', u'name',
u'author_flair_text', u'created_utc', u'distinguished', u'_id', u'ups']
```

We then found the following:

Most upvoted comment

Code:

```
# Find Most upvoted comment by sorting in descending order
print "Most upvoted comment: "
data = db.reddit_comments.find_one(sort=[('ups',-1)])

print "Data:"
print data

print "\nComment:"
print data['body']
```

Output:

```
Data:
{u'subreddit_id': u't5_2qh1e', u'link_id': u't3_2r9055', u'id': u'cndtg46', u'gilded':
1, u'archived': False, u'author': u'parkedcar', u'parent_id': u't3_2r9055', u'score':
5831, u'retrieved_on': 1425070578, u'controversiality': 0, u'body': u"This same Gaston
refused to sign my bicep 2 days ago at Disney because he couldn't find it.\nEdit: It was
actually yesterday not 2 days ago. ", u'edited': 1420351940, u'author_flair_css_class':
None, u'downs': 0, u'subreddit': u'videos', u'score_hidden': False, u'name':
u't1_cndtg46', u'author_flair_text': None, u'created_utc': u'1420346568', u'ups': 5831,
u'_id': ObjectId('5829db109e80f70cd70c7426'), u'distinguished': None}

"Comment:"
This same Gaston refused to sign my bicep 2 days ago at Disney because he couldn't find
it.
Edit: It was actually yesterday not 2 days ago.
```

Least upvoted comment

Code:

```
# Find Most upvoted comment by sorting in ascending order
print "Least upvoted comment: "
```

```
data = db.reddit_comments.find_one(sort=[('ups',1)])
print "Data:"
print data
print "\nComment:"
print data['body']
```

Output:

```
Least upvoted comment:
Data:
{u'subreddit_id': u't5_2qh61', u'link_id': u't3_2r4ko4', u'id': u'cncg14f', u'gilded':
0, u'archived': False, u'author': u'[deleted]', u'parent_id': u't1_cncfpxv', u'score':
-875, u'retrieved_on': 1425095736, u'controversiality': 0, u'body': u'[deleted]',
u'edited': False, u'author_flair_css_class': None, u'downs': 0, u'subreddit': u'WTF',
u'score_hidden': False, u'name': u't1_cncg14f', u'author_flair_text': None,
u'created_utc': u'1420230503', u'ups': -875, u'_id':
ObjectId('5829da4c9e80f70cd7e9a022'), u'distinguished': None}

Comment:
[deleted]
```

Number of positive comments

Code:

```
# Find Number of positive comments by finding all comments with upvotes greater than 0
print "Number of positive comments: "
print db.reddit_comments.find({
    "ups": { "$gt" : 0 }
}).count()
```

Output:

```
Number of positive comments:
15442083
```

Number of negative comments

Code:

```
# Find Number of negative comments by finding all comments with upvotes less than 0
print "Number of negative comments: "
print db.reddit_comments.find({
    "ups": { "$lt" : 0 }
}).count()
```

Output:

```
Number of negative comments:
756094
```

Top 5 Highest Rated comments

Code:

```
# Find Top 5 Highest Rated comments in the subreddit AMA by finding all comments with
the column name ‘subreddit’ with a value of “AMA” and sort by descending order limiting
the output by 5
```

```
print "Top 5 Highest rated comments in subreddit AMA : "
temp_data =  db.reddit_comments.find(
    {"subreddit" : "AMA"},
    sort=[('ups',-1)]
).limit(5)
for data in temp_data:
    print "------------------------"
    print data["body"]
```

Output:

```
Top 5 Highest rated comments in subreddit AMA :
------------------------
4'9" is average sized?

You two are only 1'2" different and that's not bad. I've dated a girl 5'0" and I'm 6'2",
so I know the feeling too, although backwards.

How awesome is slow dancing when you are only chest high or is it annoying not seeing
anything but her chest?
------------------------
My minds nose smells bullshit.
------------------------
I worked there once.  It's nothing [redacted] or [redacted]. We [redacted] and
[redacted], one time [redacted] bolted [redacted] and [redacted].

The coolest thing was [redacted]!

Not sure if I'm allowed to [redacted], but it's [redacted].


------------------------
Sooo like.. are you going to answer questions?
------------------------
( ͡° ͜ʖ ͡°)
------------------------
```

Top 5 Subreddits with most upvotes

Code:

```
# Find the Top 5 Subreddits with most upvotes by grouping all the comments by the
'subreddit' column and adding up the upvotes into 'total' then sorting by descending
order and limiting by 5
print "Top 5 Subreddits with most upvotes"
pipe = [{'$group': {'_id': "$subreddit", 'total': {'$sum': '$ups'}}},
        {'$sort': {'total': -1}},
        {'$limit': 5}
        ]
data = db.reddit_comments.aggregate(pipeline=pipe, allowDiskUse=True)
for line in data: print line
```

Output:

```
Top 5 Subreddits with most upvotes
{u'total': 15993623, u'_id': u'AskReddit'}
{u'total': 2981146, u'_id': u'funny'}
{u'total': 2923738, u'_id': u'pics'}
```

```
{u'total': 2412113, u'_id': u'nfl'}
{u'total': 2087586, u'_id': u'videos'}
```

Top 5 Subreddits with least upvotes

Code:

```
# Find the Top 5 Subreddits with least upvotes by grouping all the comments by the
‘subreddit’ column and adding up the upvotes into ‘total’ then sorting by ascending
order and limiting by 5
print "Top 5 Subreddits with least upvotes"
pipe = [{'$group': {'_id': "$subreddit", 'total': {'$sum': '$ups'}}},
        {'$sort': {'total': 1}},
        {'$limit': 5}
        ]
data = db.reddit_comments.aggregate(pipeline=pipe, allowDiskUse=True)
for line in data: print line
```

Output:

```
Top 5 Subreddits with least upvotes
{u'total': -115, u'_id': u'GallowBoob'}
{u'total': -87, u'_id': u'endracism'}
{u'total': -41, u'_id': u'TruePPD'}
{u'total': -32, u'_id': u'JapaneseASMR'}
{u'total': -29, u'_id': u'redditroommates'}
```

Top 5 Users with most upvotes

Code:

```
# Find the Top 5 Users with most upvotes by grouping all the comments by the ‘author’
column and adding up the upvotes into ‘total’ then sorting by descending order and
limiting by 5
print "Top 5 Users with most upvotes"
pipe = [{'$group': {'_id': "$author", 'total': {'$sum': '$ups'}}},
        {'$sort': {'total': -1}},
        {'$limit': 5}
        ]
data = db.reddit_comments.aggregate(pipeline=pipe, allowDiskUse=True)
for line in data: print line
```

Output:

```
Top 5 Users with most upvotes
{u'total': 3304366, u'_id': u'[deleted]'}
{u'total': 78184, u'_id': u'ElonMuskOfficial'}
{u'total': 77276, u'_id': u'donald_faison'}
{u'total': 72170, u'_id': u'AutoModerator'}
{u'total': 58964, u'_id': u'PainMatrix'}
```

Top 5 Users with Least Upvotes
Code:

```
# Find the Top 5 Users with least upvotes by grouping all the comments by the 'author'
column and adding up the upvotes into 'total' then sorting by ascending order and
limiting by 5
print "Top 5 Users with least upvotes"
pipe = [{'$group': {'_id': "$author", 'total': {'$sum': '$ups'}}},
        {'$sort': {'total': 1}},
        {'$limit': 5}
        ]
data = db.reddit_comments.aggregate(pipeline=pipe, allowDiskUse=True)
for line in data: print line
```

```
Top 5 Users with least upvotes
{u'total': -4645, u'_id': u'wutshappening'}
{u'total': -3294, u'_id': u'salmonhelmet'}
{u'total': -2686, u'_id': u'dwimback'}
{u'total': -1891, u'_id': u'ur_mom_was_a_hamster'}
{u'total': -1836, u'_id': u'damipereira'}
```

Then we close the connection to the database:

```
connection.close()
```

Spark

In order to extract interesting information from the database, it was decided to use Apache Spark's mapReduce functionality as a tool for processing and comparing different types of subreddits as represented by the dataset.

Three key questions asked were:

- "What subreddits are most likely to produce highly rated users?"
- "How does a user's individual score relate to the popularity of the subreddits they frequent?"
- "Does the popularity of a subreddit have an effect on how well comments are received by its users?".
-

In order to read the dataset straight from the file, due to incompatible formatting, the lines needed to be mapped by the following function:

```
map(lambda a: eval(a, {'false': False, 'true': True, 'null': None}))
```

This used python's eval() function to take the string as a literal, replacing 'false' and 'true' as were used in the document, to 'False' and 'True' which are python compatible.

To answer the first of the questions asked, "What subreddits are most likely to produce highly rated users?", mapReduce was used to first rank users by their individual score, following which the highest rated usernames were one by one used to filter the the database for only their comments which were then tallied by subreddit they were posted to. To rank the users, the following code was used:

```
#Top Users by Score
highScorers = paraData.map(lambda a: eval(a, {'false': False, 'true': True, 'null': None})).map(lambda a: [a['author'],a['score']]).filter(lambda a: a[1]>10).reduceByKey(lambda u, v: u+v).map(lambda a: [a[1],a[0]]).sortByKey(False).collect()
```

This function creates a key value pair for each comment consisting of its author as the key and score as the value using `map(lambda a: [a['author']`. The pairs were then filtered to only include authors with a score of greater than 10 using `filter(lambda a: a[1]>10)`, following which the filtered scores were reduced by their score using `.reduceByKey(lambda u, v: u+v)` and sorted by swapping the key and the value, then sorting in descending order by the new key, now the author name, using `map(lambda a: [a[1],a[0]]).sortByKey(False)`.

To make use of this data, the following code was applied:

```
#Top User Favourite Subreddits
for user in highScorers[0:5]:
    print user[1]
    print paraData.map(lambda a: eval(a, {'false': False, 'true': True, 'null': None})).filter(lambda a: user[1] in a['author']).map(lambda a: [a['subreddit'], 1]).reduceByKey(lambda a,b: a+b).map(lambda a: [a[1],a[0]]).sortByKey(False).take(10)
```

This looped through the top five users found by the previous function, and used `.filter(lambda a: user[1] in a['author'])` to reduce the dataset to just comments posted by the user in question. `map(lambda a: [a['subreddit'], 1])` was then used to turn these comments into key value pairs with subreddit name as the key and 1 as the value, so that the pairs could be reduced and sorted similarly to how they were in the previous function. The results of this came out as follows:

```
[deleted]
[(402288, 'AskReddit'), (81150, 'worldnews'), (76324, 'funny'), (67047, 'nfl'), (66470, 'pics'), (61546, 'news'), (53898, 'leagueoflegends'), (51023, 'videos'), (47091, 'todayilearned'), (44573, 'AdviceAnimals')]
PainMatrix
[(493, 'AskReddit'), (306, 'funny'), (235, 'pics'), (108, 'todayilearned'), (50, 'gifs'), (36, 'videos'), (30, 'Showerthoughts'), (26, 'worldnews'), (18, 'movies'), (13, 'news')]
maisiewilliamsAMA
[(77, 'IAmA')]
Smeeee
[(100, 'AskDoctorSmeeee'), (57, 'funny'), (48, 'WTF'), (42, 'AskReddit'), (35, 'pics'), (33, 'AdviceAnimals'), (18, 'gifs'), (13, 'todayilearned'), (8, 'mildlyinteresting'), (2, 'nfl')]
IranianGenius
[(121, 'AskReddit'), (59, 'nfl'), (52, 'funny'), (21, 'roosterteeth'), (20, 'needamod'), (20, 'pics'), (18, 'AdviceAnimals'), (12, 'IranianGenius'), (12, 'Tinder'), (11, 'oddlysatisfying')]
```

From these results, it can be seen that three of the top four commenters in the dataset, not including deleted comments, were active in the “AskReddit”, “funny” and “pics” subreddits and 2 of the top four were active in the “todayilearned” and “gifs” subreddits. As can be seen from these results, “AskReddit”, “funny” and “pics” are subreddits that are the most popular users in the dataset tend to frequent, and it may be inferred from this that posting in these subreddits may make a user more likely to have a higher overall score.

To answer the second question asked, “How does a user’s individual score relate to the popularity of the subreddits they frequent?”, the comments were first organised by subreddit and reduced by score, following which the most active users by number of comments was found and their overall scores were found. This was done for top and bottom three subreddits as ranked by score.
Subreddits were first ranked using the following function:

```
#Subreddits ordered by total score
subScores = paraData.map(lambda a: eval(a, {'false': False, 'true': True, 'null': None})).map(lambda a:
[a['subreddit'],a['score']]).reduceByKey(lambda u, v: u+v).map(lambda a: [a[1],a[0]]).sortByKey(False).collect()
```

which used the same key value pairing method that was used when ranking users by score, but using the subreddit name as the key instead of the author name.
Following this, the ten most active users in the top three subreddits were found by using the following function:

```
#10 most active users in 3 most popular subs
topSubsUsers = []
for sub in subScores[0:3]:
    topSubsUsers.append([sub[1], paraData.map(lambda a: eval(a, {'false': False, 'true': True, 'null': None})).filter(lambda a: sub[1] in
a['subreddit']).map(lambda a: [a['author'], 1]).reduceByKey(lambda a,b: a+b).map(lambda a: [a[1],a[0]]).sortByKey(False).take(10)])
print topSubsUsers
```

The function loops through the first three subreddits found in the subreddit ordering function, and for each subreddit name filters the comments to only include ones posted into that subreddit. The result of this was then mapped to a key value pair consisting of author and the value one and reduced by key in order to count the number of comments made to that subreddit by each user. The values were then sorted by user name and the top ten were taken, using `take(10)]`.
The results were found to be:

```
[['AskReddit', [(14133, '[deleted]'), (2151, 'AutoModerator'), (375, 'IUsePretzelLogic'), (340, 'Late_Night_Grumbler'), (303, 'Mrs_Holman_7'), (270,
 'OuttaSightVegemite'), (161, 'luckjes112'), (131, 'youremomisweird'), (126, 'save_the_pigs'), (120, 'PapaDeltaYankee')]], ['CFB', [(6998, '[delete
d]'), (407, 'nittanylionstorm07'), (316, 'OnthefarWind'), (308, 'AJinxyCat'), (288, '740Buckeye'), (275, 'HardKnockRiffe'), (269, 'Mufro'), (269, 'd
elatriangle'), (261, 'mreatsum'), (251, 'Spiffstronaut')]], ['pics', [(3359, '[deleted]'), (158, 'FaceBadger'), (90, 'neondeon25'), (90, 'picsonlybo
t'), (72, 'IRateBoobies'), (67, 'nicholasmoegly'), (62, 'TheSpanishImposition'), (59, 'noahbradley'), (53, 'funkalunatic'), (53, 'PavelSokov')]]]
```

Having the users who made the most comments on the top three subreddits, their individual overall scores were then found using the following function:

```
#Overall scores of 3 most active users in top 3 subs
for sub in topSubsUsers:
    print sub[0]
    for subUser in sub[1][0:3]:
        print subUser[1], paraData.map(lambda a: eval(a, {'false': False, 'true': True, 'null': None})).filter(lambda a: subUser[1] in
a['author']).map(lambda a: a['score']).reduce(lambda a,b: a+b)
    print "-------"
```

This function loops through the three most active users in the top three subreddits one by one, using their user names to filter the full dataset and sum the scores of all of their comments. The scores were found and reduced using `.map(lambda a: a['score']).reduce(lambda a,b: a+b)` after having filtered by user name similar to how it was done before.
The results came out as:

```
AskReddit
[deleted] 358528
AutoModerator 8703
IUsePretzelLogic 693
-------
CFB
[deleted] 358528
nittanylionstorm07 1107
OnthefarWind 2260
-------
pics
[deleted] 358528
FaceBadger 1364
neondeon25 5870
-------
```

The same method was applied to the lower scoring subreddits using:

```
#10 most active users in 3 least popular subs
botSubsUsers = []
for sub in subScores[-4:-1]:
    botSubsUsers.append([sub[1], paraData.map(lambda a: eval(a, {'false': False, 'true': True, 'null': None})).filter(lambda a: sub[1] in
a['subreddit']).map(lambda a: [a['author'], 1]).reduceByKey(lambda a,b: a+b).map(lambda a: [a[1],a[0]]).sortByKey(False).take(10)])
print botSubsUsers
```

To find the most active users, which resulted in:

```
[['askaconservative', [(75, '[deleted]'), (7, 'EatSleepDanceRepeat'), (4, 'glennflynn'), (3, 'szymanovikich'), (3, 'redditcons'), (2, 'SchwarzeSonne
_'), (2, 'pumpyourstillskin'), (2, 'diversity_is_racism'), (1, 'IMULTRAHARDCORE'), (1, 'student_of_yoshi')]], ['endracism', [(25, '[deleted]'), (17,
 'Hatesracists'), (14, 'BurnStabKillNazis'), (14, 'NS_white'), (4, 'SkaterMan'), (3, 'TheEmbernova'), (3, 'Too_Many-Chimps'), (3, 'Andrea--Dworki
n'), (3, 'ExposeRacists'), (3, 'GasTheRacists')]], ['KarmaConspiracy', [(5, 'iDownvoteGallowBoob'), (4, 'GallowBoob'), (3, 'DarthTauri'), (3, 'Im_Br
uce_Wayne_AMA'), (2, '[deleted]'), (2, 'KarmaConspiracy_Bot'), (2, 'Obama4presidentJAJA'), (1, 'kongomueller'), (1, 'Suffercure'), (1, 'Nytra')]]]
```

The most active users' scores were then found using:

```
#Overall scores of 3 most active users in bottom 3 subs
for sub in botSubsUsers:
    print sub[0]
    for subUser in sub[1][0:3]:
        print subUser[1], paraData.map(lambda a: eval(a, {'false': False, 'true': True, 'null': None})).filter(lambda a: subUser[1] in
a['author']).map(lambda a: a['score']).reduce(lambda a,b: a+b)
    print "-------"
```

And found to be:

```
askaconservative
[deleted] 358528
EatSleepDanceRepeat 38
glennflynn 0
-------
endracism
[deleted] 358528
Hatesracists -72
BurnStabKillNazis -29
-------
KarmaConspiracy
iDownvoteGallowBoob 458
GallowBoob 5100
DarthTauri 73
-------
```

To synthesize these results, it is required to compare the scores of the most active users in the highest scoring subreddits with those of the lowest scoring. The three highest scoring subreddits were found to be “AskReddit”, “CFB” and “pics”, the two most active users of which, not including deleted comments, averaged scores of 4698 points, 6183.5 and 3617 respectively.
To compare with this, the three lowest scoring subreddits were found to be “askaconservative”, “endracism” and “KarmaConspiracy” whose two most active users averaged 19, -50.5 and 2779 points respectively.
Comparing these two results, it appears that higher scoring users do not tend to be very active in lower scoring subreddits and users with a low score are not likely to be the most active in more popular subreddits. One possible conclusion from this is that being active on popular subreddits is more likely to produce a high scoring user than being active on less popular subreddits.

To answer the final question, “Does the popularity of a subreddit have an effect on how well comments are received by its users?”, the three most popular subreddits had their number of positive and negative comments tallied using the function:

```
#Number of positive and negative scoring comments in top 3 subs
topSubComs = []
for sub in subScores[0:3]:
    d = paraData.map(lambda a: eval(a, {'false': False, 'true': True, 'null': None})).filter(lambda a: sub[1] in a['subreddit'])
    a = d.filter(lambda a: a['score'] > 0).count()
    b = d.filter(lambda a: a['score'] < 0).count()
    topSubComs.append([sub[1],a,b])
print topSubComs
```

This uses the names of the top three subreddits as found previously, and
Which gave the the results:

```
[['AskReddit', 147724, 4195], ['CFB', 100918, 2592], ['pics', 30080, 2787]]
```

For the bottom subs, the same code was used:

```
#Number of positive and negative scoring comments in bottom 3 subs
botSubComs = []
for sub in subScores[-4:-1]:
    d = paraData.map(lambda a: eval(a, {'false': False, 'true': True, 'null': None})).filter(lambda a: sub[1] in a['subreddit'])
    a = d.filter(lambda a: a['score'] > 0).count()
    b = d.filter(lambda a: a['score'] < 0).count()4
    botSubComs.append([sub[1],a,b])
print botSubComs
```

And the results were:

```
[['askaconservative', 60, 38], ['endracism', 73, 37], ['KarmaConspiracy', 23, 5]]
```

Using these results and taking the average proportion of positive to negative scoring comments on the most and least popular subreddits, it was found that the most popular subreddits, “AskReddit”, “CFB” and “pics” had an average ratio of 28.31:1 for positive to negative comments and the least popular had 2.72:1. Using ratios in this case accounts for more popular subreddits receiving many more comments than less popular ones by giving a proportion, and it is clear from the results that less popular subreddit communities tend more to score comments negatively than more popular subreddits do.

Spark SQL

Databricks provides the ability to perform SQL queries on the dataset. The SQL is technically optimized for working across clusters as it still uses MapReduce when these queries are performed. The dataset must first be uploaded as a table, this can be done through the Databricks UI. The full 30GB dataset was used for this section. Databricks also provides functionality for displaying the results of a query as a plot.

Retrieving the total number of comments. Computation time = 4.86 minutes.

```
%sql
SELECT COUNT(*) FROM rc_2015_01_bz2
```

(1) Spark Jobs

count(1)
53851542

Subreddits

The following code will count the number of comments of each subreddit, then order the number by the total number of comments and limit the output to the first ten results. The result can be visualized in the bar plot in Figure 14. Computation time = 4.32 minutes.

```
%sql
SELECT COUNT(id), subreddit FROM rc_2015_01_bz2
GROUP BY subreddit
ORDER BY COUNT(id) DESC
LIMIT 10
```

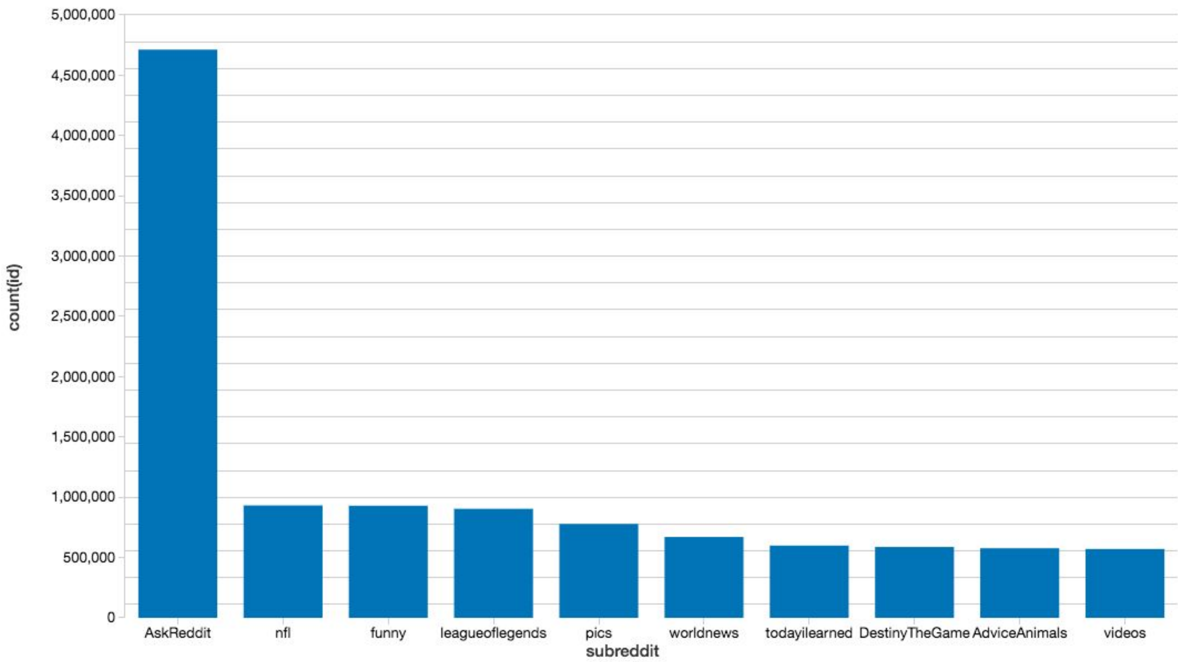

Figure 14: Number of comments for the top ten subreddits.

The next query will find the highest upvoted comment and order by the score. Computation time = 2.89 minutes.

```
%sql
SELECT score, author FROM rc_2015_01_bz2
WHERE subreddit = "AskReddit"
ORDER BY score DESC
```

(1) Spark Jobs

score	author
6597	a1988eli
5835	rugtoad
5734	pyromaster55
5714	toojer
5704	theottomaddox
5703	Cidochrone
5694	uglyhag
5689	lockd0wn

Showing the first 1000 rows.

Gilded

Count the number of comments that received Reddit gold. Computation time = 2.91 minutes.

```
%sql
SELECT COUNT(id) FROM rc_2015_01_bz2
WHERE gilded > 0
```

(1) Spark Jobs

count(id)
19688

Order by the column gilded and in descending fashion. Only include comments that were actually gilded. The score is displayed to illustrate that higher gilded comments also are highly upvoted. Computation Time = 3.20 minutes.

```
%sql
SELECT gilded, ups, downs, score FROM rc_2015_01_bz2
WHERE gilded > 0
ORDER BY gilded DESC
```

(1) Spark Jobs

gilded	ups	downs	score
16	4797	0	4797
14	5835	0	5835
14	2207	0	2207
14	2054	0	2054
14	3197	0	3197
12	966	0	966
11	5260	0	5260
10	3101	0	3101
10	2329	0	2329
9	2956	0	2956
9	6597	0	6597
9	4077	0	4077

Showing the first 1000 rows.

To further illustrate this point a scatterplot (Figure 15) was made to show the correlation between gilded and score. We can see that it is not uncommon for gilded posts to have very high upvote scores.

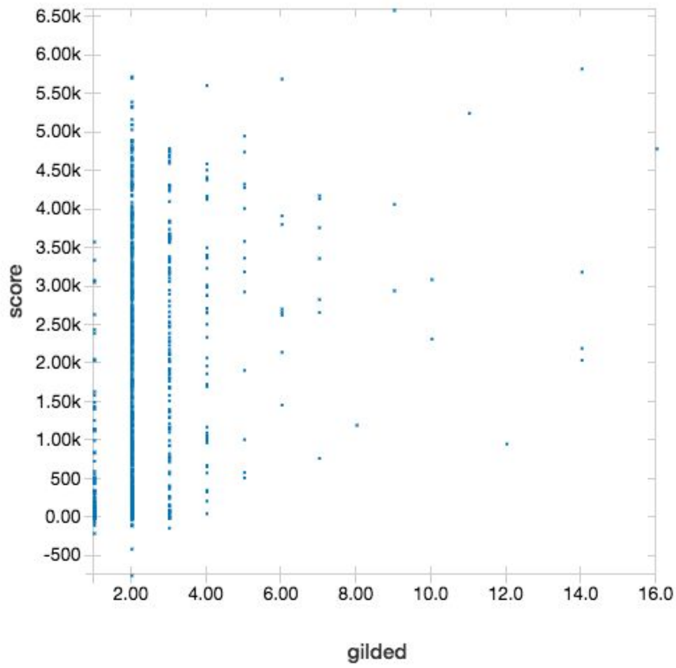

Figure 15: Scatter plot of gilded versus score. Limited to first 1000 points, order by descending values

Machine Learning

Three Machine learning tasks will be performed:

- Regression:
    - Predict the score of a comment using the text
    - Predict the number of golds a comment has based on the upvotes
- Classification:
    - Predict the subreddit based on the comment text

It is important to note that the main task of this machine learning is to not generate excellent models but to **perform computationally difficult tasks and finding ways to optimize them**.

All machine learning tasks will use the Random Forest model. This model was chosen as the other models available were too computationally demanding or were prone to large overfitting errors (decision trees for example).

Predict the Score Using Comment Text (Regression)

The first machine learning task is to predict the score based on the comment text. What we are trying to find is a correlation between some words and higher scores.

Note, a helper function called timeit was created. The timeit function will be used to measured the time it takes for functions to run. We can use this to determine the efficiency with a smaller dataset and see how this will translate to the full dataset.

Featurization

First we will create a subset of the comment DataFrame only containing the id, upvotes and body since we will not be required to use the other columns for our first regression task. We will start featurization by first tokenizing the body of the comment. We will use MLib to do the following:

- Use the tokenizer to convert the comment bodies to arrays
- Remove stop words from words column

In the last two weeks we will update the regex expression to better handle punctuation, reddit mentions and emoticons in the comments.

```
# Remove all unnessacary fields from comment DataFrame
sentenceDF = commentDF.select('id','ups','body')

# use pyspark tokenizer object to split words in array
tokenizer = RegexTokenizer(inputCol='body', outputCol='tokens', pattern='\W',
minTokenLength=2)
wordsDF = tokenizer.transform(sentenceDF)

# Remove stop words
remover = StopWordsRemover(inputCol="words", outputCol="filtered_words")
wordsFilteredDF = remover.transform(wordsDF)

# Remove body and words since they will no longer be used
wordsFilteredDF = wordsFilteredDF.select('id','ups','filtered_words')
```

We will be comparing two methods of featurization, CountVectorizer and HashingTF. CountVectorizer will create a bag of words representation of the words found in the body of the comment. HashingTF is a Feature Hasher and will also create a bag of words representation but will place similar words into the same bucket to limit the size of the matrix.

```
@timeit
def term_frequency(df, inputCol, outputCol, hashFeatures=None):
    '''
    Returns a DataFrame object containing a new row with the extracted features.
    Passing hashed=True will return a Featured Hashed matrix.

    @params:
        df - DataFrame
        inputCol - name of input column from DataFrame to find features
        outputCol - name of the column to save the features
        hashFeatures - number of features for HashingTF, if None will perform
            CountVectorization
    '''

    # since the number of features was not passed perform standard CountVectorization
    if hashFeatures is None:
        cv = CountVectorizer(inputCol=inputCol, outputCol=outputCol)
        feature_extractor = cv.fit(wordsFilteredDF)
    # otherwise perform a feature extractor with
    else:
        feature_extractor = HashingTF(\
                                inputCol=inputCol, outputCol=outputCol,
numFeatures=hashFeatures)

    # create a new DataFrame using either feature extraction method
    return feature_extractor.transform(df)
```

Next we compute both the CountVectorizer DataFrame and the HashingTF and compare the result and computation time. As we can see the HashingTF method is much more efficient.

```
# Find the occurence of each word in the comment content
cvDF = term_frequency(\
    df=wordsFilteredDF, inputCol="filtered_words", outputCol="features")
```

'term_frequency' took 599.58 sec

```
# Feature Hash the comment content
# number of features for Feature Hash matrix, reccomended too use power of 2
hashDF = term_frequency(\
    df=wordsFilteredDF, inputCol="filtered_words", outputCol="features",
hashFeatures=1024)
```

'term_frequency' took 0.03 sec

Results

From testing on a smaller dataset we were able to show that the error is comparable but the feature hashed matrix was much more efficient. On a 1MB dataset, when performing the Regression task the Feature Hashed matrix took 7.43 seconds to complete while the Count Vectorized matrix took 25.62 seconds. When performing the task on the 30GB dataset the Feature Hashed matrix took 1.27 hours. As a result the regression task was not performed on the Count Vectorized matrix using 30GB.

First we will create a function that will return the predicted DataFrame with the timeit decorator to keep track of run time.

```
@timeit
def random_forest_regression(df, featuresCol, labelCol):
    '''
    Returns a DataFrame containing a column of predicted values of the labelCol.
    Predict the output of labelCol using values in featuresCol y = rf(x).

    @params:
        df - DataFrame
        featuresCol - input features, x
        labelCol - output variable, y
    '''
    # split the training and test data using the holdout method
    (trainingData, testData) = df.randomSplit([0.8, 0.2])

    # create the random forest regressor, limit number of trees to ten
    dtr = RandomForestRegressor(\
       featuresCol=featuresCol, labelCol=labelCol)

    # fit the training data to the regressor to create the model
    model = dtr.fit(trainingData)

    # create a DataFrame contained a column with predicted values of the labelCol
    predictions = model.transform(testData)

    return predictions
```

The results of the regression are calculated below using the RegressionEvalulator.

```
# train random forest regression
predictions = random_forest_regression(df=hashDF,featuresCol="features",labelCol="ups")

# compute the error
evaluator = RegressionEvaluator(labelCol="ups", predictionCol="prediction",
metricName="rmse")
rmse = evaluator.evaluate(predictions)
print "Root Mean Squared Error (RMSE) on test data = %g" % rmse
```

(10) Spark Jobs

```
'random_forest_regression' took 4075.14 sec

Root Mean Squared Error (RMSE) on test data = 47.5471
```

We can see that the error is relatively high for this machine learning task and our model is not complex enough to properly determine which words lead to higher upvotes.

```
> predictions.show(10)
```

▸ (1) Spark Jobs

```
+--------+---+--------------------+--------------------+------------------+
|      id|ups|      filtered_words|            features|        prediction|
+--------+---+--------------------+--------------------+------------------+
|cnas905|  2|[don't, know, des...|(1024,[47,57,304,...|5.446674009053898|
|cnas90i|  1|       [wheredugit?]|  (1024,[817],[1.0])|5.377043200885622|
|cnas90u|  1|           [goonies]|  (1024,[932],[1.0])|5.377043200885622|
|cnas90x|  1|[finish, masters,...|(1024,[125,232,25...|5.377043200885622|
|cnas914|  1|[like, idea,, tho...|(1024,[0,27,56,88...|5.446674009053898|
|cnas917|  2|[[](/hellohuman)m...|(1024,[71,90,139,...|5.377043200885622|
|cnas91h|  2|[people, stop, sa...|(1024,[28,39,40,6...|5.446674009053898|
|cnas91j|  1|[totally, agree,,...|(1024,[0,25,26,39...|4.488249188193347|
|cnas91o|  4|[would, say, coin...|(1024,[117,131,25...|9.601278434518049|
|cnas91w| 14|[hear, sdss, j122...|(1024,[340,493,54...|5.377043200885622|
+--------+---+--------------------+--------------------+------------------+
only showing top 10 rows
```

Figure X: Results of the Random Forest Regression on the Feature Hashed matrix

Predict Number of Gilded based on Score (Regression)

The text task is to predict how many times a comment was gilded based on its score. This idea arose from plotting the scatter plot in the Data Exploration section and noticing a correlation between upvotes and gilded comments.

For this task we have simplified the machine learning workflow using an MLib Pipeline. A pipeline will allow us to chain tasks together. Previously, after even step (Tokenization, Removing stop words, …) we were required to create a new dataframe for each step. Using a pipeline we no longer have to do this as can be seen in the code below.

Note, the VectorAssembler allows use to convert the score column into a vector so it can be used as a feature column when fitting the regressor.

```
df = commentDF

# Transform score column into a vector
assembler = VectorAssembler(
    inputCols=['score'],
    outputCol='features')

# create a random forest regressor to predict the value of the gilded column
rf = RandomForestRegressor(featuresCol='features',labelCol='gilded')

# combine the assebmler and random forest regressor into a machine learning pipeline
pipeline = Pipeline(stages=[assembler, rf])

# split data into training and test set
(trainingData, testData) = df.randomSplit([0.8, 0.2])

# Train model, this also runs the assembler
model = pipeline.fit(trainingData)

# Make predictions.
predictions = model.transform(testData)

# Select (prediction, true label) and compute test error
evaluator = RegressionEvaluator(
    labelCol="gilded", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

(10) Spark Jobs

```
Root Mean Squared Error (RMSE) on test data = 0.0233083
```

The total operation of this task was 30.04 minutes and the RMSE was 2.33%. From these results we can see that score is a good predictor of measuring how many times a comment was gilded. There is a large error with this assumption, there is an overwhelming large amount of comments that were not gilded.

By specifying to only look at columns that were actually gilded we can get a better idea of how accurate our prediction is for higher gilded comments.

```
df = commentDF.where(col('gilded') > 0)
```

The results of using this regression model was RMSE = ~40%. This confirms the hypothesis that the reason our regressor was so accurate previously was because there was such a small number of gilded comments and most were zero.

Predict Subreddit based on Comment Text (Classification)

In this task we will try to predict which subreddit a comment came from. To simplify this we are using comments only from the top five ‘themed’ subreddits. A themed subreddit is a subreddit that falls in a specific category like sports or a video game. This was chosen since we believe it will make the classification simpler.

```
# Found using SQL queries
top_five_themed_subreddits = {'nfl', 'leagueoflegends', 'worldnews', 'DestinyTheGame',
'AdviceAnimals'}

df = commentDF.select('id', 'body',
'subreddit').where(col('subreddit').isin(top_five_themed_subreddits))
print "\nThe total number of comments in the top ten subreddits: %s (deleted comments
removed)" % df.count()
```

▸ (1) Spark Jobs

```
The total number of comments in the top ten subreddits: 3443915 (deleted comments removed)
```

Similar to the previous machine learning task we also create a pipeline but have more steps. The purpose of each step of the pipeline is outlined briefly below:

StringIndexer	Convert the classes (subreddits) into numbers. Requirement of the RandomForestClassifier.
RegexTokenizer	Tokenize the body of the comment and remove all non-word characters and words less the size 2.
StopWordsRemover	Remove all stop words from tokens. Ex. the, this, or, ...
HashingTF	Feature Hash the tokens.
IDF	Calculate the TF_IDF of the tokens, this will put less emphasis on words that appear more frequently. The objective of this is to create a better model.
RandomForestClass	Create a Random Forest Classifier using the features created from the steps above.

```
# Convert subreddit string to number
labelIndexer = StringIndexer(inputCol="subreddit", outputCol="subreddit_num")
labelCol = labelIndexer.getOutputCol()

# Convert body of comment to tokens
tokenizer = RegexTokenizer(inputCol='body', outputCol='tokens', pattern='\W',
minTokenLength=2)

# Remove Stop words from tokens
remover = StopWordsRemover(inputCol=tokenizer.getOutputCol(),
outputCol="tokens_filtered")

# Hash tokens into a feature hashed matrix
hashingTF = HashingTF(inputCol=remover.getOutputCol(), outputCol='features',
numFeatures=1024)

# Caluclate the TF_IDF of the words, this will put let emphasis on words that appear more
often
idf = IDF(inputCol=hashingTF.getOutputCol(), outputCol="tf_idf_features")

# Train a random forest classifier
rf = RandomForestClassifier(labelCol=labelCol, featuresCol=idf.getOutputCol(),
numTrees=3)
# Steps in pipeline, tokenize, hash then model
pipeline = Pipeline(stages=[labelIndexer, tokenizer, remover, hashingTF, idf, rf])

# split data into training and test set
(trainingData, testData) = df.randomSplit([0.8, 0.2])

# Fit model to trainingData
model = pipeline.fit(trainingData)

prediction = model.transform(testData)
```

The computation time of the task above was 33.05 minutes.

```
evaluator = MulticlassClassificationEvaluator(
    labelCol=labelCol, predictionCol="prediction", metricName='accuracy')

accuracy = evaluator.evaluate(prediction)

print("Test Error = %g" % (1.0 - accuracy))
```

(2) Spark Jobs

```
Test Error = 0.700086
```

The accuracy of the classifier was found to be only around 30%. The reason for this error is probably due to the fact that our model is not complex enough. Our hashing matrix has a dimension of 1024, a lot of dimensions to create a classifier for. Using artificial neural networks would allow for a more complex model at the sake of computation time.