EECS E6893 Big Data Analytics
HW2: Classification and Twitter data analysis with Spark Streaming

Yunhang Lin, yl4860@columbia.edu
Rui Chu, rc3414@columbia.edu

Agenda

- Binary classification with Spark MLlib
  - Logistic Regression

- Twitter data analysis with Spark Streaming
  - LDA

Logistic Regression

- Logistic Function:

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}} \qquad \log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X.$$

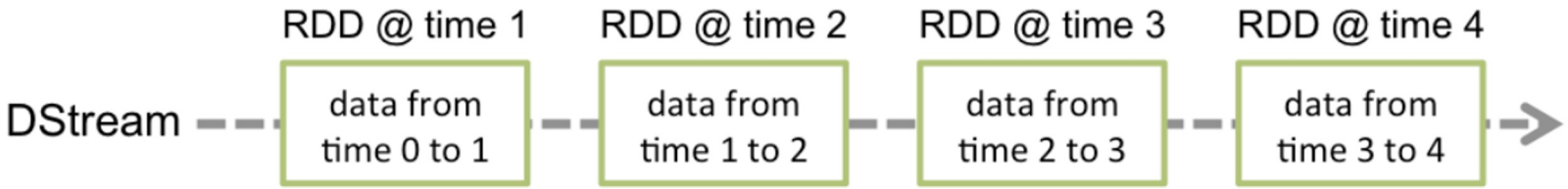
- Likelihood Function:

$$\ell(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_{i'}=0} (1-p(x_{i'})).$$

Spark Streaming

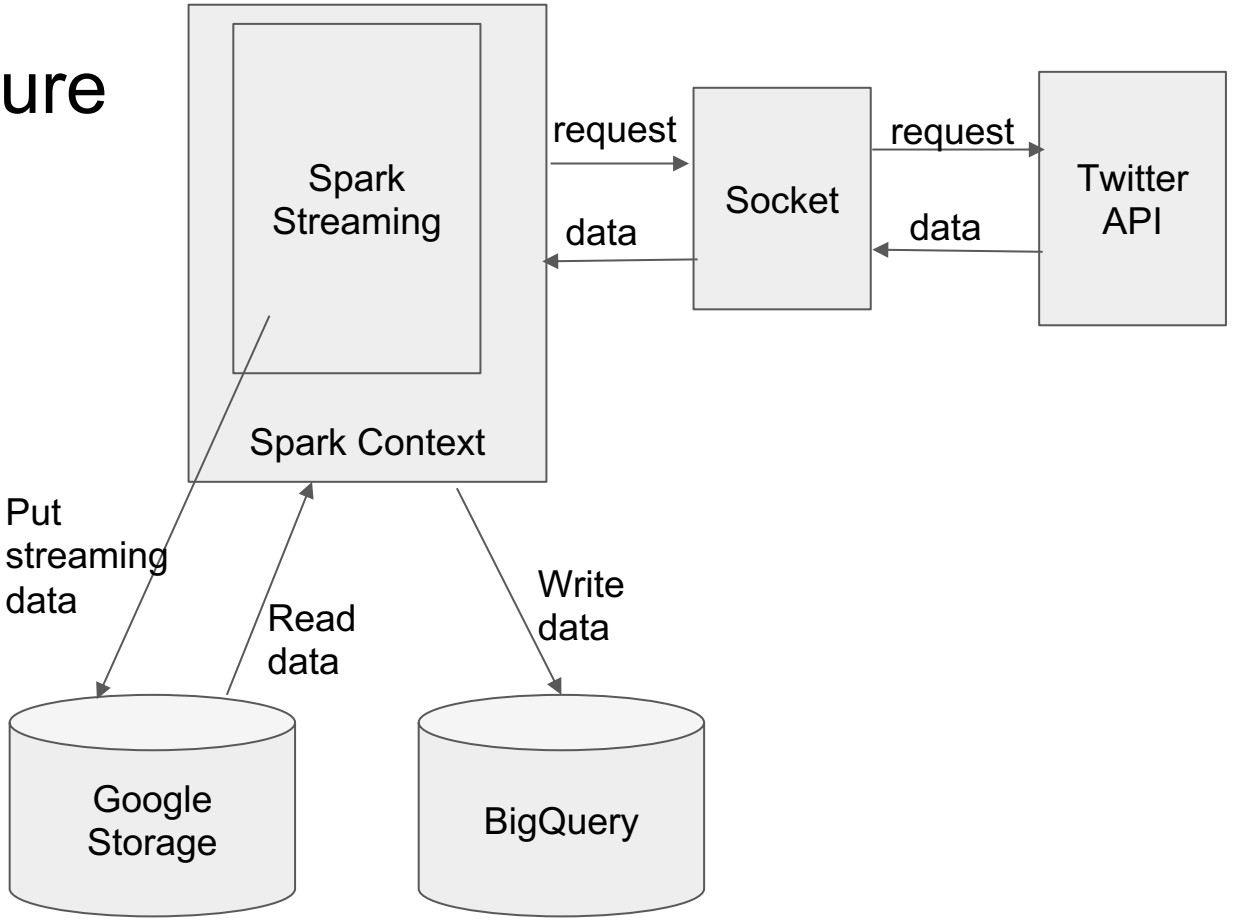
https://spark.apache.org/docs/latest/streaming-programming-guide.html

Dstream



- A basic abstraction provided by Spark Streaming
- Represents a continuous stream of data
- Contains a continuous series of RDDs at different time

Architecture



LDA (Latent Dirichlet allocation)

- A topic model.
- A three-layer Bayesian probability model, including a three-layer structure of words, topics, and documents.
- It can be used to generate a document, and identify themes in a large-scale document.

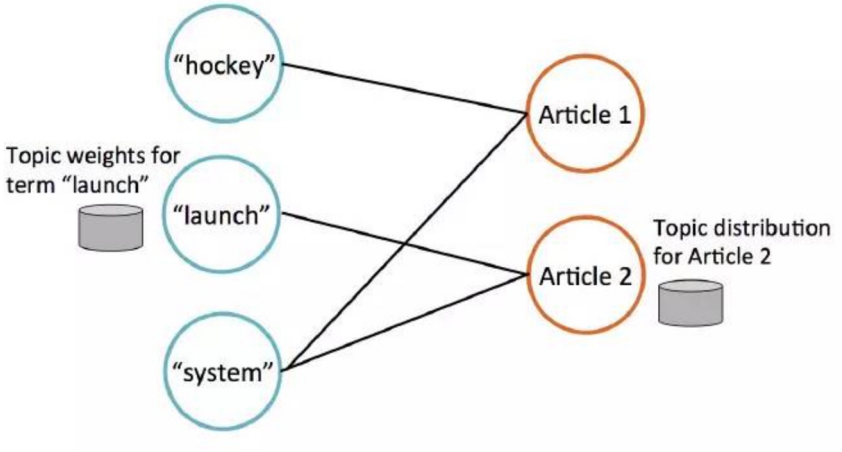
LDA (Latent Dirichlet allocation)

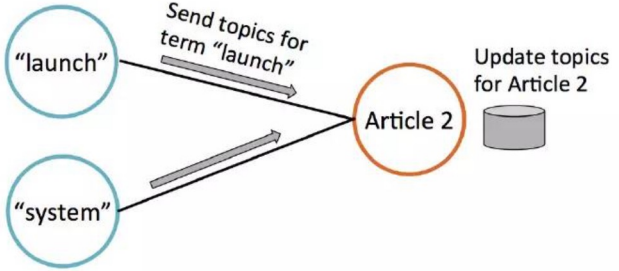
"Arts"	"Budgets"	"Children"	"Education"
NEW	MILLION	CHILDREN	SCHOOL
FILM	TAX	WOMEN	STUDENTS
SHOW	PROGRAM	PEOPLE	SCHOOLS
MUSIC	BUDGET	CHILD	EDUCATION
MOVIE	BILLION	YEARS	TEACHERS
PLAY	FEDERAL	FAMILIES	HIGH
MUSICAL	YEAR	WORK	PUBLIC
BEST	SPENDING	PARENTS	TEACHER
ACTOR	NEW	SAYS	BENNETT
FIRST	STATE	FAMILY	MANIGAT
YORK	PLAN	WELFARE	NAMPHY
OPERA	MONEY	MEN	STATE
THEATER	PROGRAMS	PERCENT	PRESIDENT
ACTRESS	GOVERNMENT	CARE	ELEMENTARY
LOVE	CONGRESS	LIFE	HAITI

The William Randolph Hearst Foundation will give $1.25 million to Lincoln Center, Metropolitan Opera Co., New York Philharmonic and Juilliard School. "Our board felt that we had a real opportunity to make a mark on the future of the performing arts with these grants an act every bit as important as our traditional areas of support in health, medical research, education and the social services," Hearst Foundation President Randolph A. Hearst said Monday in announcing the grants. Lincoln Center's share will be $200,000 for its new building, which will house young artists and provide new public facilities. The Metropolitan Opera Co. and New York Philharmonic will receive $400,000 each. The Juilliard School, where music and the performing arts are taught, will get $250,000. The Hearst Foundation, a leading supporter of the Lincoln Center Consolidated Corporate Fund, will make its usual annual $100,000 donation, too.

LDA (Latent Dirichlet allocation)

- The left side is the word node, and the right side is the document node. Each word node stores some weight values to indicate which topic the word is related to; similarly, each article node stores an estimate of the topic discussed in the current article.





$$p(w|d) = \sum_{i=1}^{k} p(w|z_k) * p(z_k|d)$$

d is the document, w is the word, z is the topic, and k is the number of topics.

HW2

HW2 Part I Binary classification with Spark MLlib

- Adult dataset from UCI Machine Learning Repository
- Given information of a person, predict if the person could earn > 50k per year

Index of /ml/

- Parent Directory
- Index
- adult.data
- adult.names
- adult.test
- old.adult.names

Apache/2.4.6 (CentOS) Open

HW2 Part I Binary classification with Spark MLlib

- Workflow
  - Data loading: load data into Dataframe

    Spark Dataframe Guide: https://spark.apache.org/docs/latest/sql-getting-started.html

```
#============================
data.show(3)
```

```
+---+----------------+--------+----------+----+------------------+-----------------+-------------+------+-----+------+----+----+--------
------+------+
|_c0|             _c1|     _c2|       _c3| _c4|               _c5|              _c6|          _c7|   _c8|  _c9|  _c10|_c11|_c12|
_c13|  _c14|
+---+----------------+--------+----------+----+------------------+-----------------+-------------+------+-----+------+----+----+--------
------+------+
| 39|       State-gov| 77516.0| Bachelors|13.0|     Never-married|     Adm-clerical| Not-in-family| White| Male|2174.0| 0.0|40.0| United-
States| <=50K|
| 50| Self-emp-not-inc| 83311.0| Bachelors|13.0| Married-civ-spouse|   Exec-managerial|       Husband| White| Male|   0.0| 0.0|13.0| United-
States| <=50K|
| 38|         Private|215646.0|   HS-grad| 9.0|          Divorced| Handlers-cleaners| Not-in-family| White| Male|   0.0| 0.0|40.0| United-
States| <=50K|
+---+----------------+--------+----------+----+------------------+-----------------+-------------+------+-----+------+----+----+--------
------+------+
only showing top 3 rows
```

HW2 Part I Binary classification with Spark MLlib

- Workflow
  - Data preprocessing: Convert the categorical variables into numeric variables with ML Pipelines and Feature Transformers

```
dataset = df
```

```
root
 |-- age: integer (nullable = true)
 |-- workclass: string (nullable = true)
 |-- fnlwgt: double (nullable = true)
 |-- education: string (nullable = true)
 |-- education_num: double (nullable = true)
 |-- marital_status: string (nullable = true)
 |-- occupation: string (nullable = true)
 |-- relationship: string (nullable = true)
 |-- race: string (nullable = true)
 |-- sex: string (nullable = true)
 |-- capital_gain: double (nullable = true)
 |-- capital_loss: double (nullable = true)
 |-- hours_per_week: double (nullable = true)
 |-- native_country: string (nullable = true)
 |-- income: string (nullable = true)
```

```
+---+-----------------+------+---------+-------------+------------------+----------------+-------------+-----+----+------------+------------+--------------+--------------+------+
|age|        workclass|fnlwgt|education|education_num|    marital_status|      occupation|relationship| race| sex|capital_gain|capital_loss|hours_per_week|native_country|income|
+---+-----------------+------+---------+-------------+------------------+----------------+------------+-----+----+------------+------------+--------------+--------------+------+
| 39|        State-gov|77516.0|Bachelors|         13.0|     Never-married|    Adm-clerical|Not-in-family|White|Male|      2174.0|         0.0|          40.0| United-States| <=50K|
| 50| Self-emp-not-inc|83311.0|Bachelors|         13.0|Married-civ-spouse|Exec-managerial|     Husband|White|Male|         0.0|         0.0|          13.0| United-States| <=50K|
+---+-----------------+------+---------+-------------+------------------+----------------+------------+-----+----+------------+------------+--------------+--------------+------+
only showing top 2 rows
```

```
In [43]: preppedDataDF.take(3)
```

```
Out[43]: [Row(age=39, workclass=' State-gov', fnlwgt=77516.0, education=' Bachelors', education_num=13.0, marital_status=' Never-married', occupation=' Adm-clerical', relationship=' Not-in-family', race=' White', sex=' Male', capital_gain=2174.0, capital_loss=0.0, hours_per_week=40.0, native_country=' United-States', income=' <=50K', workclassIndex=4.0, workclassclassVec=SparseVector(8, {4: 1.0}), educationIndex=2.0, educationclassVec=SparseVector(15, {2: 1.0}), marital_statusIndex=1.0, marital_statusclassVec=SparseVector(6, {1: 1.0}), occupationIndex=3.0, occupationclassVec=SparseVector(14, {3: 1.0}), relationshipIndex=1.0, relationshipclassVec=SparseVector(5, {1: 1.0}), raceIndex=0.0, raceclassVec=SparseVector(4, {0: 1.0}), sexIndex=0.0, sexclassVec=SparseVector(1, {0: 1.0}), native_countryIndex=0.0, native_countryclassVec=SparseVector(41, {0: 1.0}), label=0.0, features=SparseVector(100, {4: 1.0, 10: 1.0, 24: 1.0, 32: 1.0, 44: 1.0, 48: 1.0, 52: 1.0, 53: 1.0, 94: 39.0, 95: 77516.0, 96: 13.0, 97: 2174.0, 99: 40.0})),
 Row(age=50, workclass=' Self-emp-not-inc', fnlwgt=83311.0, education=' Bachelors', education_num=13.0, marital_status=' Married-civ-spouse', occupation=' Exec-managerial', relationship=' Husband', race=' White', sex=' Male', capital_gain=0.0, capital_loss=0.0, hours_per_week=13.0, native_country=' United-States', income=' <=50K', workclassIndex=1.0, workclassclassVec=SparseVector(8, {1: 1.0}), educationIndex=2.0, educationclassVec=SparseVector(15, {2: 1.0}), marital_statusIndex=0.0, marital_statusclassVec=SparseVector(6, {0: 1.0}), occupationIndex=2.0, occupationclassVec=SparseVector(14, {2: 1.0}), relationshipIndex=0.0, relationshipclassVec=SparseVector(5, {0: 1.0}), raceIndex=0.0, raceclassVec=SparseVector(4, {0: 1.0}), sexIndex=0.0, sexclassVec=SparseVector(1, {0: 1.0}), native_countryIndex=0.0, native_countryclassVec=SparseVector(41, {0: 1.0}), label=0.0, features=SparseVector(100, {1: 1.0, 10: 1.0, 23: 1.0, 31: 1.0, 43: 1.0, 48: 1.0, 52: 1.0, 53: 1.0, 94: 50.0, 95: 83311.0, 96: 13.0, 99: 13.0})),
 Row(age=38, workclass=' Private', fnlwgt=215646.0, education=' HS-grad', education_num=9.0, marital_status=' Divorced', occupation=' Handlers-cleaners', relationship=' Not-in-family', race=' White', sex=' Male', capital_gain=0.0, capital_loss=0.0, hours_per_week=40.0, native_country=' United-States', income=' <=50K', workclassIndex=0.0, workclassclassVec=SparseVector(8, {0: 1.0}), educationIndex=0.0, educationclassVec=SparseVector(15, {0: 1.0}), marital_statusIndex=2.0, marital_statusclassVec=SparseVector(6, {2: 1.0}), occupationIndex=9.0, occupationclassVec=SparseVector(14, {9: 1.0}), relationshipIndex=1.0, relationshipclassVec=SparseVector(5, {1: 1.0}), raceIndex=0.0, raceclassVec=SparseVector(4, {0: 1.0}), sexIndex=0.0, sexclassVec=SparseVector(1, {0: 1.0}), native_countryIndex=0.0, native_countryclassVec=SparseVector(41, {0: 1.0}), label=0.0, features=SparseVector(100, {0: 1.0, 8: 1.0, 25: 1.0, 38: 1.0, 44: 1.0, 48: 1.0, 52: 1.0, 53: 1.0, 94: 38.0, 95: 215646.0, 96: 9.0, 99: 40.0}))]
```

HW2 Part I Binary classification with Spark MLlib

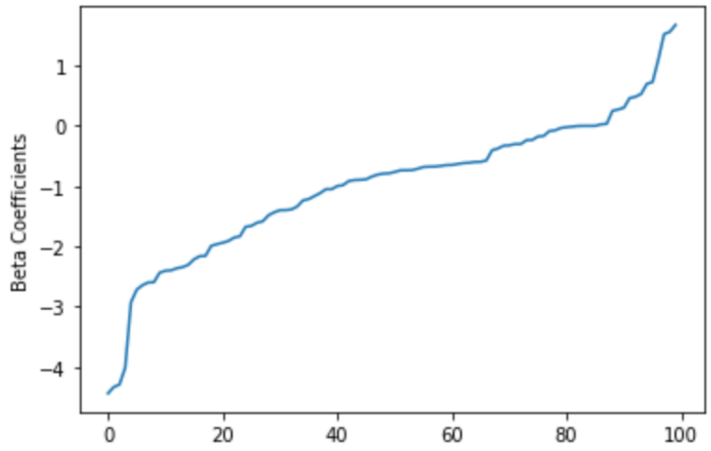
- Workflow
  - Modelling：
    Logistic Regression
    KNN
    Random Forest
    Naive Bayes
    Decision Tree
    Gradient Boosting Trees
    Multi-layer perceptron
    Linear Support Vector Machine
    One-vs-Rest

https://spark.apache.org/docs/latest/ml-classification-regression.html

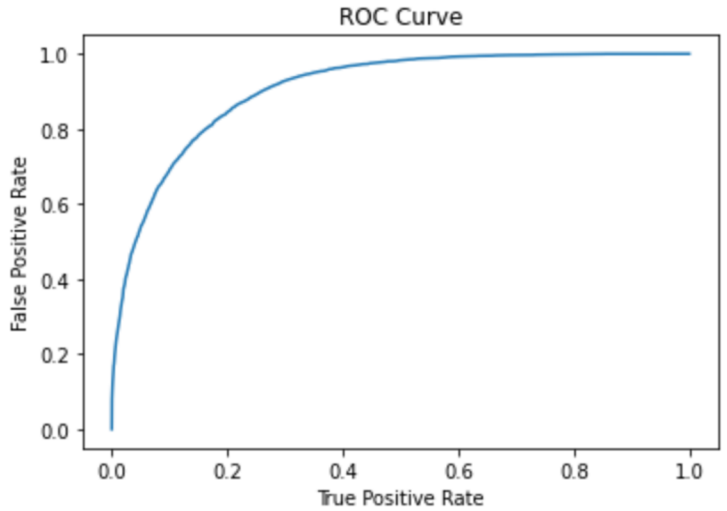
HW2 Part I Binary classification with Spark MLlib

- Workflow
  - Evaluation (Logistic Regression)

```
plt.ylabel('Beta Coefficients')
plt.show()
```



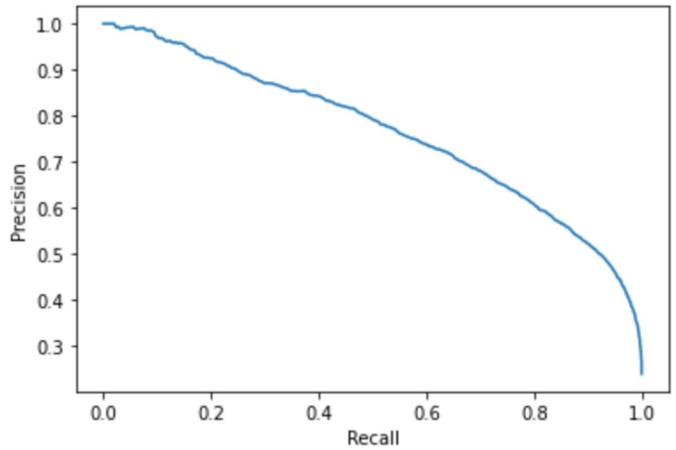

```
print('Training set areaUnderROC: ' + str(trainingSummary.areaUnderROC))
```



HW2 Part I Binary classification with Spark MLlib

- Workflow
  - Evaluation (Logistic Regression)

```
plt.ylabel('Precision')
plt.xlabel('Recall')
plt.show()
```




```
print("Accuracy: %s\nFPR: %s\nTPR: %s\nF-measure: %s\nPrecision: %s\nRecall: %s"
      % (accuracy, falsePositiveRate, truePositiveRate, fMeasure, precision, recall))
```

```
Accuracy: 0.8526629292221444
FPR: 0.3172889625959339
TPR: 0.8526629292221444
F-measure: 0.8475083896808702
Precision: 0.8464035949642436
Recall: 0.8526629292221444
```

```
evaluator.evaluate(predictions)
```

Out[54]: 0.8993574699928725

In [55]:
```
# accuracy
correct = float(predictions.filter(predi
total = float(predictions.count())
print(correct, total, correct/total)
```

8218.0 9729.0 0.8446911296124987

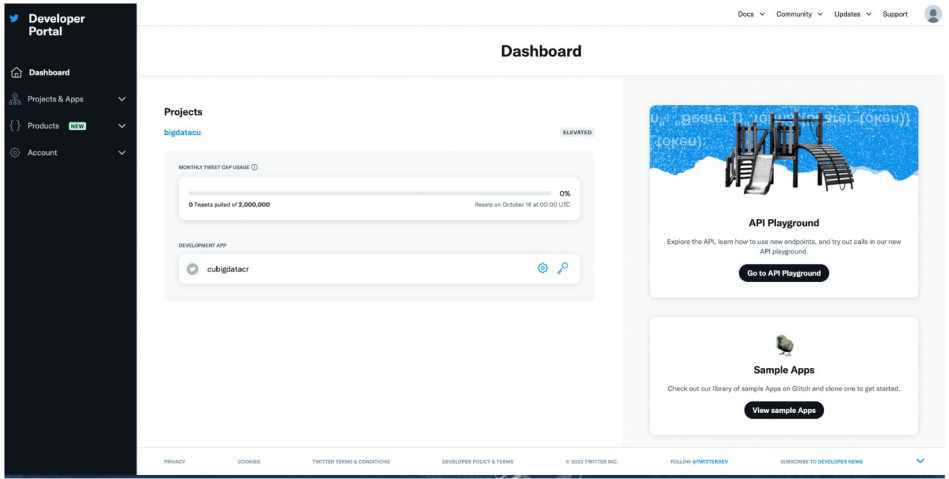
HW2 Part II Twitter Data Analysis

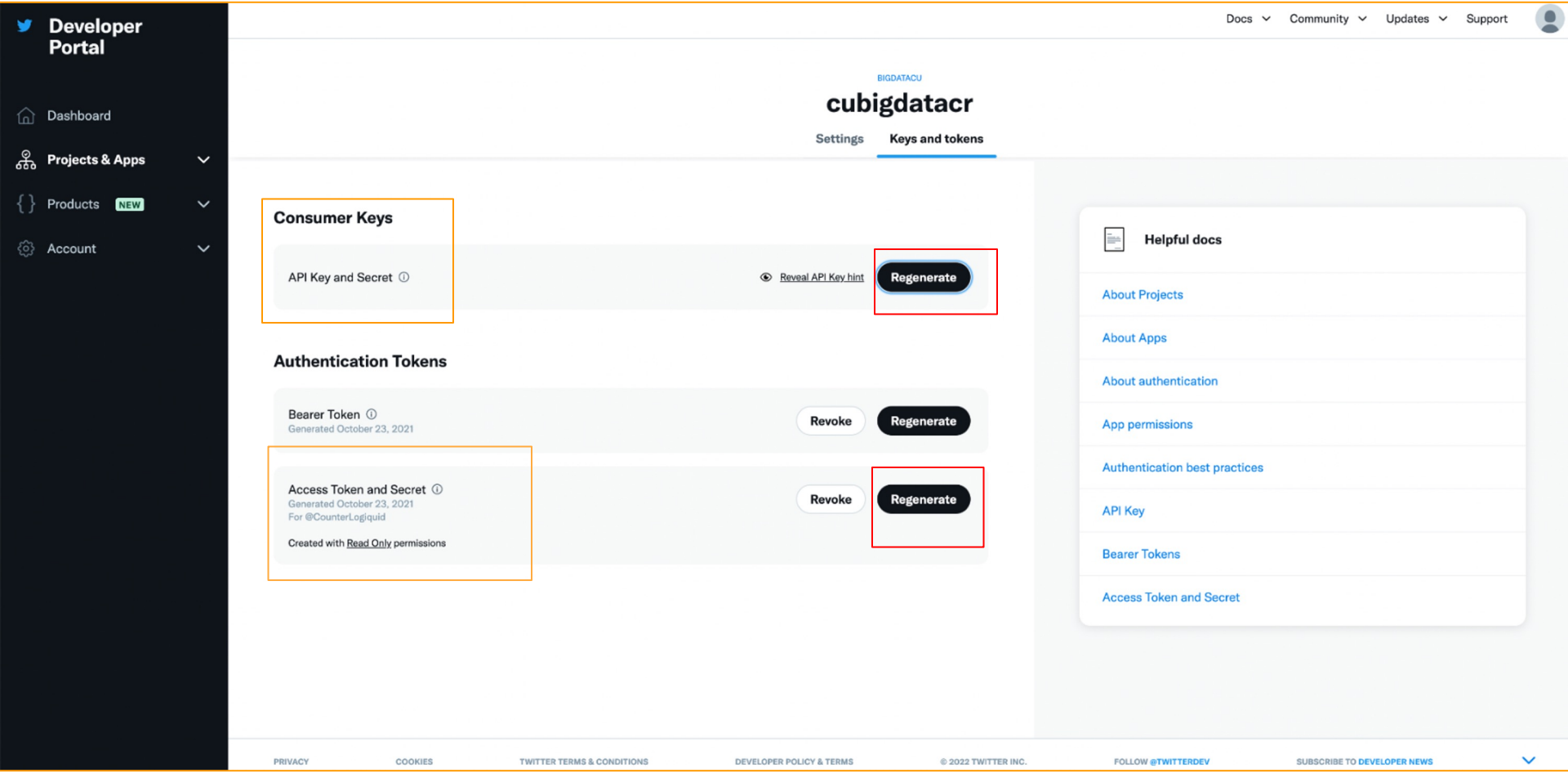
- Calculate the accumulated hashtags count sum for 600 seconds and sort it by descending order of the count.
- Filter the chosen 5 words and calculate the appearance frequency of them in 60 seconds for every 60 seconds (no overlap).
- Save results to google BigQuery.
- Use LDA to do Clustering to your streaming, see the topic distribution.

Register on Twitter Apps (Do this ASAP)

```
# credentials
# TODO: replace with your own credentials
ACCESS_TOKEN = ''      # your access token
ACCESS_SECRET = ''     # your access token secret
CONSUMER_KEY = ''      # your API key
CONSUMER_SECRET = ''   # your API secret key
```

https://developer.twitter.com/en/apply-for-access.html




Developer Portal
Dashboard
Projects & Apps
Products NEW
Account
Docs
Community
Updates
Support
BIGDATACU
cubigdatacr
Settings
Keys and tokens
Consumer Keys
API Key and Secret
Reveal API Key hint
Regenerate
Authentication Tokens
Bearer Token
Generated October 23, 2021
Revoke
Regenerate
Access Token and Secret
Generated October 23, 2021
For @CounterLogiquid
Created with Read Only permissions
Revoke
Regenerate
Helpful docs
About Projects
About Apps
About authentication
App permissions
Authentication best practices
API Key
Bearer Tokens
Access Token and Secret
PRIVACY
COOKIES
TWITTER TERMS & CONDITIONS
DEVELOPER POLICY & TERMS
© 2022 TWITTER INC.
FOLLOW @TWITTERDEV
SUBSCRIBE TO DEVELOPER NEWS

Socket

Use TCP, need to provide IP and Port for client to connect

```
class twitter_client:
    def __init__(self, TCP_IP, TCP_PORT):
      self.s = s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
      self.s.bind((TCP_IP, TCP_PORT))

    def run_client(self, tags):
      try:
        self.s.listen(1)
        while True:
          print("Waiting for TCP connection...")
          conn, addr = self.s.accept()
          print("Connected... Starting getting tweets.")
          sendData(conn,tags)
          conn.close()
      except KeyboardInterrupt:
        exit
```

Spark Streaming

```
if __name__ == '__main__':
    # Spark settings
    conf = SparkConf()
    conf.setMaster('local[2]')
    conf.setAppName("TwitterStreamApp")

    # create spark context with the above configuration
    sc = SparkContext(conf=conf)
    sc.setLogLevel("ERROR")

    # create sql context, used for saving rdd
    sql_context = SQLContext(sc)

    # create the Streaming Context from the above spark context with batch interval size 5 seconds
    ssc = StreamingContext(sc, 5)
    # setting a checkpoint to allow RDD recovery
    ssc.checkpoint("~/checkpoint_TwitterApp")

    # read data from port 9001
    dataStream = ssc.socketTextStream("localhost",9001)
    dataStream.pprint()
```

Create a local StreamingContext with two working thread and batch interval of 5 second.

Create stream from TCP socket IP localhost and Port 9001

Spark Streaming

```
# Start streaming process, wait for 600s and then stop.
ssc.start()
time.sleep(STREAMTIME)
ssc.stop(stopSparkContext=False, stopGraceFully=True)
```

```
STREAMTIME = 600          # time that the streaming process runs
```

Start streaming context

Stop after 600 seconds (You can set STREAMTIME to a smaller value at first)

Save results to BigQuery

Start streaming

1. Run twitterHTTPClient.py
2. Run sparkStreaming.py
3. You can test sparkStreaming.py multiple times and leave twitterHTTPClient.py running
4. Stop twitterHTTPClient.py (on job page of the cluster or use gcloud command)

Task1: hashtagCount

```
def hashtagCount(words):
    """
    Calculate the accumulated hashtags count sum from the beginning of the stream
    and sort it by descending order of the count.
    Ignore case sensitivity when counting the hashtags:
        "#Ab" and "#ab" is considered to be a same hashtag
    You have to:
    1. Filter out the word that is hashtags.
       Hashtag usually start with "#" and followed by a serious of alphanumeric
    2. map (hashtag) to (hashtag, 1)
    3. sum the count of current DStream state and previous state
    4. transform unordered DStream to a ordered Dstream
    Hints:
        you may use regular expression to filter the words
        You can take a look at updateStateByKey and transform transformations
    Args:
        dstream(DStream): stream of real time tweets
    Returns:
        DStream Object with inner structure (hashtag, count)
    """

    # TODO: insert your code here
    pass
```

Task2: wordCount

```
WORD = ['data', 'spark', 'ai', 'movie', 'good']     #the words you should filter and do word count

# Helper functions
```

```
def wordCount(words):
    """
    Calculte the count of 5 sepcial words for every 60 seconds (window no overlap)
    You can choose your own words.
    Your should:
    1. filter the words
    2. count the word during a special window size
    3. add a time related mark to the output of each window, ex: a datetime type
    Hints:
        You can take a look at reduceByKeyAndWindow transformation
        Dstream is a serious of rdd, each RDD in a DStream contains data from a certain interval
        You may want to take a look of transform transformation of DStream when trying to add a time
    Args:
        dstream(DStream): stream of real time tweets
    Returns:
        DStream Object with inner structure (word, (count, time))
    """

    # TODO: insert your code here
    pass
```

Task3: Save results

Create a dataset:

bq mk <Dataset name>

Replace with your own bucket and dataset name:

```
# global variables
bucket = ""    # TODO : replace with your own bucket name
output_directory_hashtags = 'gs://{}/hadoop/tmp/bigquery/pyspark_output/hashtagsCount'.format(bucket)
output_directory_wordcount = 'gs://{}/hadoop/tmp/bigquery/pyspark_output/wordcount'.format(bucket)

# output table and columns name
output_dataset = ''                    #the name of your dataset in BigQuery
output_table_hashtags = 'hashtags'
columns_name_hashtags = ['hashtags', 'count']
output_table_wordcount = 'wordcount'
columns_name_wordcount = ['word', 'count', 'timestamp']
```

Task3: Save results

```
# save hashtags count and word count to google storage
# used to save to google BigQuery
# You should:
#   1. topTags: only save the last DStream
#   2. wordCount: save each DStream
# Hints:
#   1. You can take a look at foreachRDD transformation
#   2. You may want to use helper function saveToStorage
# TODO: insert your code here
```

Sample Results

hashtags

Schema Details Preview

Row	count	hashtags
1	25	#valimai
2	14	#ai
3	11	#isapafeelgoodweekend
4	8	#isapawithfeelings
5	7	#วาร์ปสิวะ
6	7	#thalaajith
7	6	#warpsiwax
8	5	#mainemendoza
9	5	#jojorabbit
10	4	#bigdata
11	4	#uncen

wordcount

Schema Details Preview

Row	time	count	word
1	2019-10-18 23:30:10 UTC	2	ai
2	2019-10-18 23:30:50 UTC	2	ai
3	2019-10-18 23:31:30 UTC	3	ai
4	2019-10-18 23:31:50 UTC	5	ai
5	2019-10-18 23:31:10 UTC	8	ai
6	2019-10-18 23:30:30 UTC	10	ai
7	2019-10-18 23:30:10 UTC	1	data
8	2019-10-18 23:31:50 UTC	1	data
9	2019-10-18 23:30:50 UTC	1	data
10	2019-10-18 23:31:10 UTC	2	data
11	2019-10-18 23:31:30 UTC	2	good

Task4: LDA Clustering

- Load your streaming

```
spark_df.show()
```

```
+--------------------+
|               words|
+--------------------+
|[Hi, I, heard, ab...|
|[I, wish, Java, c...|
|[Logistic, regres...|
+--------------------+
```

Task4: LDA Clustering

- Do Clustering
- Check the weight of every topic distribution

```
transformed.show(truncate=False)
```

```
+------------------------------------------------------------------------------------------------------------------------
|topicDistribution
+------------------------------------------------------------------------------------------------------------------------
|[0.015507417459822004,0.01550728393967869,0.8580192339534809,0.015507283495435582,0.01550734753977979,0.015507454092087107,(
|[0.011611617277430928,0.011611687576324213,0.014384356731200081,0.011611693199529186,0.011611469213723304,0.01161200370440549
|[0.015507709935122908,0.015507792523275011,0.019206149229381127,0.015507722462775812,0.01550796508759411,0.01550781094983775
+------------------------------------------------------------------------------------------------------------------------
```

Task4: LDA Clustering

- Output topic and vocabulary distribution

```
print(topics)
```

```
Learned topics (as distributions over vocab of 16 words):
DenseMatrix([[0.69497654, 0.67450655, 1.46407035, 0.71606946, 0.68823238,
              0.83995925, 0.71790871, 1.13603183, 0.86361908, 0.88846945],
             [0.8478173 , 0.61279355, 1.17889067, 0.73649848, 0.64281227,
              0.65817783, 0.79481916, 0.8035318 , 0.76980272, 0.92066953],
             [0.78510867, 0.77847208, 0.86279637, 0.81701682, 0.58656788,
              0.79704796, 0.77827958, 1.22829593, 0.73297109, 0.76163522],
             [0.78489061, 0.77776822, 1.33782363, 0.74432285, 0.76384188,
              0.81880924, 0.78365265, 0.81179619, 0.97346241, 0.86002109],
             [0.70132889, 0.81153272, 0.70012903, 0.81389106, 0.71879648,
              0.80034944, 0.61262942, 1.06997399, 0.72904109, 0.6892238 ],
             [0.72429644, 0.94308907, 0.73973033, 0.74222637, 0.73192789,
              0.72408692, 1.0964796 , 0.66223273, 0.6901715 , 0.78957828],
             [0.75369384, 0.90705663, 0.5635939 , 0.66292681, 0.60902154,
              0.76171012, 0.58526171, 1.15064891, 0.8222317 , 0.66119302],
             [0.72804759, 0.88032802, 1.3391255 , 0.63925354, 0.79383212,
              0.90687066, 0.69586907, 0.68069846, 0.69006699, 0.94959428],
             [0.75946987, 0.78727924, 0.56509596, 0.79689776, 0.80397701,
              0.73442704, 0.9370068 , 0.76574326, 0.66565683, 0.67089845],
             [0.76248368, 0.58984395, 0.69364698, 0.71571205, 0.92289406,
              0.89340239, 1.06406147, 0.81177578, 0.7188162 , 0.78183528],
             [0.64441378, 0.68018412, 1.4021663 , 0.74996672, 0.74679628,
```

```
Topic0:
a
documents
of
in
vectors
to
and
feature
correspond
are
```

```
Topic1:
as
follows:
feature
and
vectors
estimating
function
words).
using
Rather
```

```
Topic2:
a
of
as
be
follows:
topics
infers
algorithm
documents.
thought
```