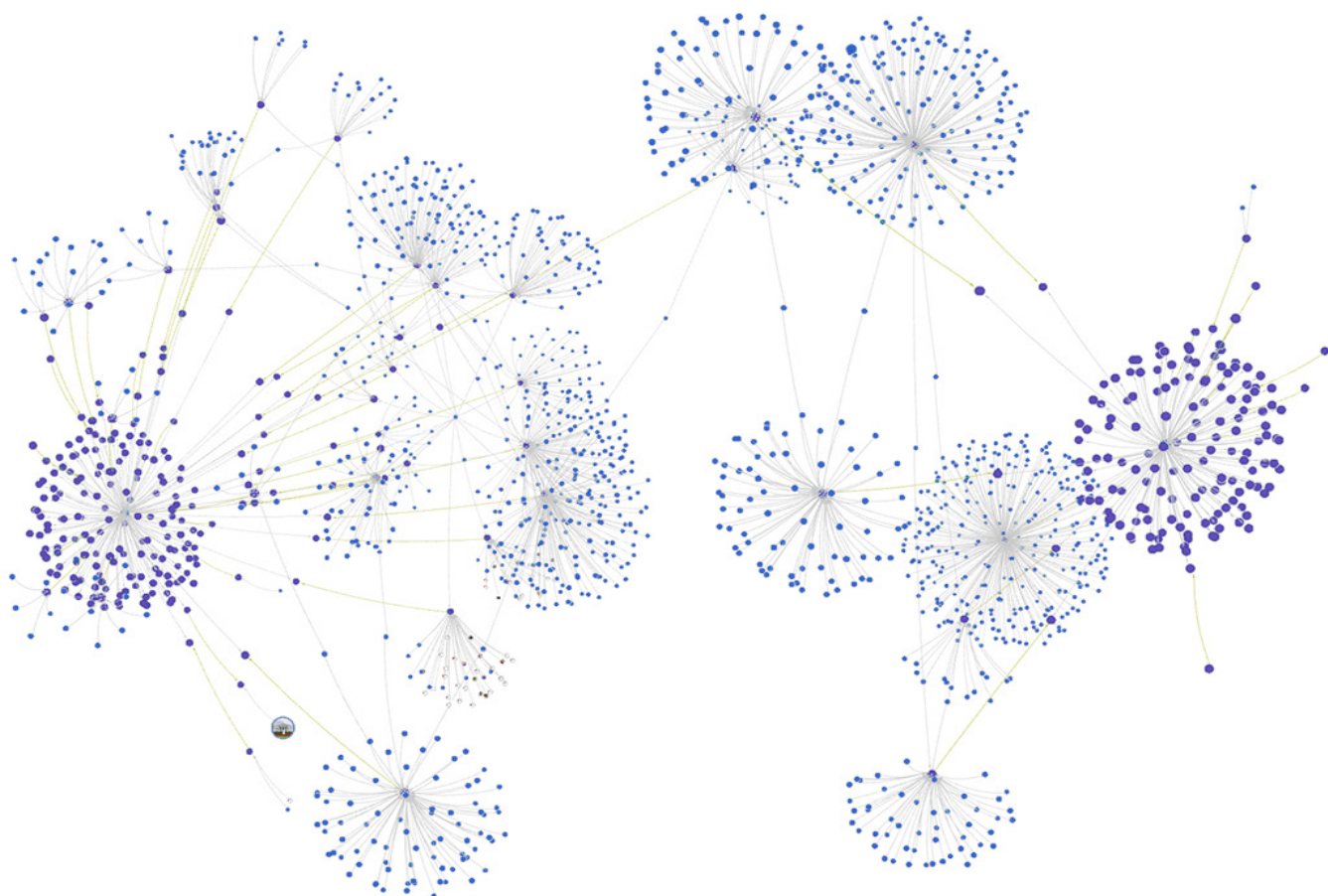


Provisioning Twitter Data for Exploratory Analysis

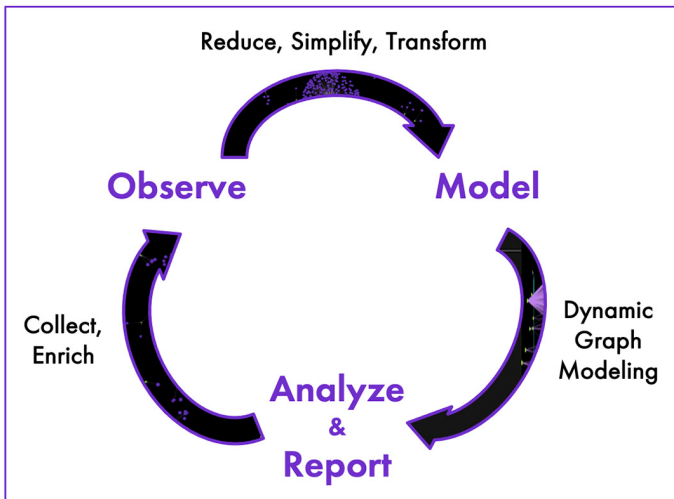


Contents

Overview	3
The Election 2020 Project	4
Project Requirements	4
Cloud Services for Provisioning	4
Technical Framework	5
Twitter Developer APIs and Twitter Tracker	5
Amazon Web Services (AWS)	6
Initial Graph Data Model	7
Data Enrichment	7
Machine Learning and Natural Language Processing	7
Machine Learning and Natural Language Processing	8
ML and NLP Process and Performance	9
Demographic Identity with PyTorch and FairFace	9
Human or Bot Likelihood with Botometer	10
Named Entity Recognition (NER) with spaCy	10
Sentiment Labeling with BERT	11
Data Transformation in Neo4j and Kineviz	12
User Influence	12
Group Affiliation with Graph Algorithms	13
User Identity with OSINT (Open Source Intelligence) APIs	13
Iterative Visualization and Analysis	14
Conclusions	15
References and Further Reading	15

Overview

We developed a technical framework to supply Twitter data to a graph data environment for exploration of user identity and sentiment during the 2020 US election cycle. The provisioning workflow supports an iterative cycle of rapid observation, modeling, and analysis.



With serverless cloud provisioning, automated processes can be set up to deliver data streams, either on demand or as batch processes, to a graph database or to third party ML and NLP applications. By avoiding the need to set up owned IT infrastructure, enhancing and exploring data in near real time becomes much easier and more cost-effective. We were able to integrate and automate:

- Cloud provisioning through Twitter APIs and Amazon Web Services (AWS).
- Machine Learning (ML) and Natural Language Processing (NLP) services.
- Exploration and analysis in a Neo4j and Kineviz graph data environment.

This ability to stream data and explore relationships quickly and iteratively is essential for evaluating the sentiment of influential Twitter users over time. Although developed for an open-source investigation of social media sentiment, the workflows apply equally well to many other use cases that require timely exploration and analysis of big data in a graph environment.

The Election 2020 Project

Election 2020 is an ongoing open-source project to explore possible insights available in social media during the 2020 US election cycle. A graph data environment is ideal for exploring the behavior of Twitter users and visualizing their connections, influence, and sentiments. An automated streaming and processing pipeline that can deliver large volumes of data is one of the key requirements for the iterative exploratory analysis that we envisioned.

Project Requirements

Data provisioning workflows must address the following requirements:

- Collect a large enough sample of publicly available Twitter data relating to specific elections and candidates, at specified time intervals, to provide credible and comparable insights over the many months of the 2020 election cycle.
- Enrich data reproducibly using third party Natural Language Processing (NLP) and machine learning (ML) applications, to label user identity and evaluate sentiment in tweet texts.
- Provide data to the graph environment in near-real time, to engage with fast breaking election events.
- Automatically inject data to the graph environment for further enrichment, exploration, and analysis.

Cloud Services for Provisioning

Twitter itself provides Developer APIs to collect streams of public data. Additional cloud services can provide the IT infrastructure to store and route data, as well as a variety of processing micro-services. Most providers charge only for space and services that are actually used. This means that instead of managing IT infrastructure as a project matures, one can focus on exploring the data itself.

Amazon Web Services, Microsoft Azure, Google Cloud, and others provide a range of services including data storage, processing capacity, and on-demand processing. Lambdas (micro-service programs, typically in Java or Python), SQL query, and increasingly, machine learning and NLP offerings are all available. Cloud infrastructure includes replicated data stores, failover, secure role-based access, and various levels of guaranteed availability or retrieval speed.

Technical Framework

The technical stack and workflow for Twitter data streams (Figure 1) is designed to:

- Automate delivery of Twitter data to a cloud archive and to the Neo4j - Kineviz graph environment. For initial selection and enhancement, data can be injected directly into a Neo4j graph database, sent to an AWS S3 bucket, or both. AWS Lambdas in Python or Javascript are used to batch process data as needed.
- Connect with third-party ML and NLP APIs to label data with respect to user identity and sentiment. Due to processing constraints, these applications are typically run as batch processes pulling data from the graph environment or S3.
- Provide data to the graph environment for additional selection, aggregation, and analysis needed to measure user influence and to label users by group affiliation.

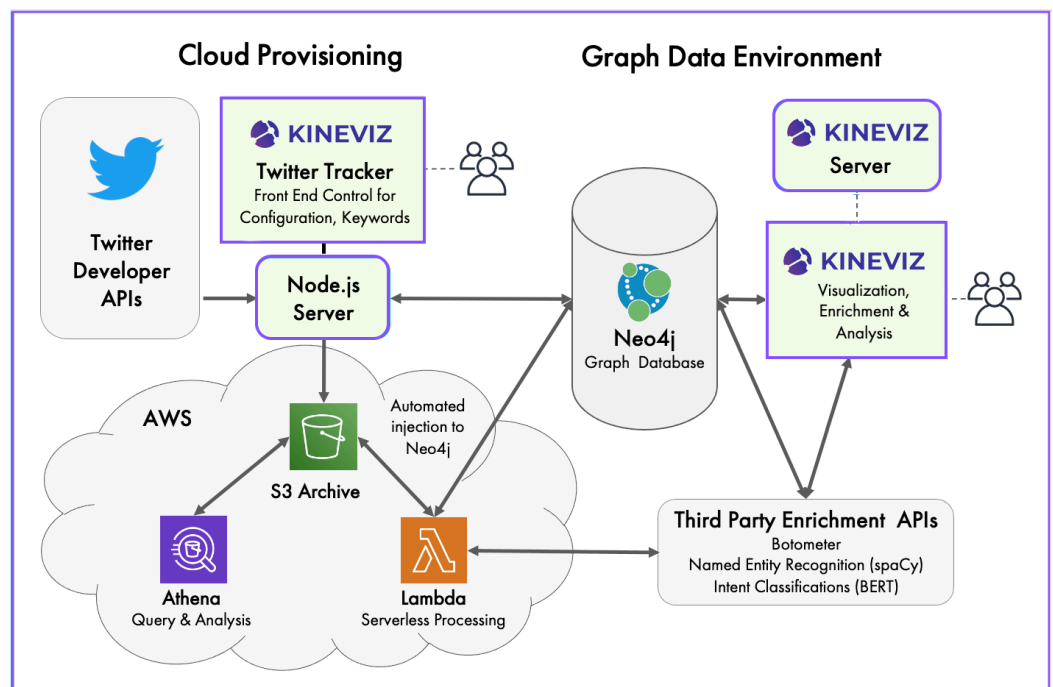


Figure 1. Technical framework for provisioning Twitter data.

Twitter Developer APIs and Twitter Tracker

Twitter's Developer APIs let you stream or sample the very large data stream of publicly shared tweets. We sampled about 3 million tweets per day related to elections by:

- Specifying the keywords Election2020, JoeBiden, andrealDonaldTrump.
- Limiting sampling to one minute every hour.

Provisioning Twitter Data for Exploratory Analysis

To automate sampling and routing of the data, we developed Twitter Tracker, a custom front end control for Twitter's Node.js API. It enables a user to enter and log selection, routing, and other configuration details for the Twitter data stream. This establishes the pipeline between Twitter and Neo4j and/or the AWS S3 archive and processing services.

Amazon Web Services (AWS)

We chose Amazon Web Services (AWS) to provide the required cost-effective IT infrastructure. Data collection and processing services used include:

- S3 bucket to store data for further processing, and to archive saved data.
- EC2 instances for processing.
- Lambda services (Python or JS) to process data on demand or in batch mode. For example, a Python lambda can filter and parse data, and inject it into Neo4j, either directly as it is streamed or as batches from the S3 bucket.
- Athena to explore and re-cast S3 data directly using SQL. This service lets us explore and query the S3 archive directly. to verify that we have sampled the data as specified. It also provides additional options for selection or filtering archived data.

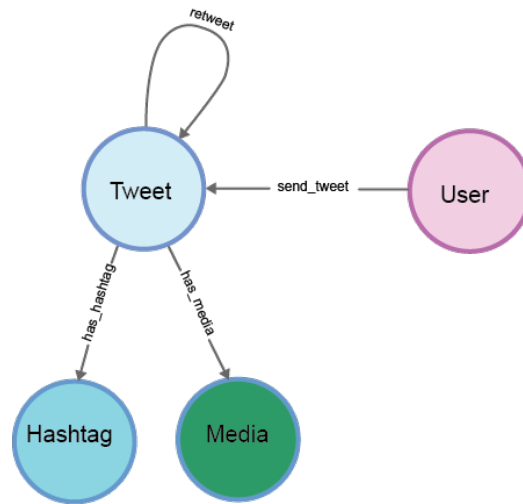
We can choose to inject data directly into Neo4j, route it to the S3 archive for further batch processing, or to send it to both places. For smaller data volumes, we send to both. At times traffic can increase to over 100 tweets/second, and even more than 10 tweets/second (a sample of roughly 1 million tweets/day) would overwhelm the Neo4j server. So when traffic is too heavy, we send only to S3, and resume sending to Neo4j only for a short period of time.

For the data routed to S3, we can run batch queries to load about 50,000 tweets at a time to Neo4j, and to speed up loading, processes can be set up to run in parallel.

Provisioning Twitter Data for Exploratory Analysis

Initial Graph Data Model

An initial schema is applied to data injected into Neo4j. It defines how the data streamed from Twitter will be assigned to categories, relationships, and properties in the graph data.



This initial model is flexible and extensible, meaning that the data model can be re-defined as needed. As data is added through enrichment processes, new categories, properties, and relationships will be created in the graph data.

Table 1. Initial categories and their associated properties.

Category	User	Tweet	Media	Hashtag
Properties	<i>createdAt</i> <i>followersCount</i> <i>friendsCount</i> <i>name</i> <i>profileImageUrl</i> <i>screenname</i> <i>userId</i>	<i>createTime</i> <i>text</i> <i>timestamp</i> <i>tweetId</i>	<i>mediaId</i> <i>type</i> <i>url</i>	<i>text</i>

Table 2. Initial relationships.

Relationship	User=>Tweet	Tweet=>Tweet	Tweet=>Media	Hashtag
	<i>SEND_TWEET</i>	<i>RETWEET</i>	<i>HAS_MEDIA</i>	<i>HAS_HASHTAG</i>

Provisioning Twitter Data for Exploratory Analysis

Data Enrichment

Although Twitter datasets are a rich source of raw data, additional labels are needed to categorize user identity and sentiment reliably. This can be done automatically and at scale using Machine Learning (ML) and Natural Language Processing (NLP) applications.

Measures of user influence and group affiliation are also needed, and these can be added within the graph environment using Cypher queries and graph-aware algorithms.

Machine Learning and Natural Language Processing

To flag the categories and sentiment implied in the Twitter data, we identified key open-source ML and NLP applications available through third party APIs (Table 3). The applications can label users with respect to demographics and whether the user is likely a human or a bot, and tweet texts with respect to group affiliation and various types of sentiment expressed.

Table 3. ML and NLP applications used to enrich Twitter data.

Application	Data Added	Training Datasets
PyTorch & FairFace	Labels users with demographic labels including race, gender, and age, using profile images.	FairFace
BotoMeter	Estimates a user's bot-like activity on a scale of 0-1.	Botometer Datasets
SpaCy	Named Entity Recognition (NER) Labels text with defined entities (organization, person, place, etc.)	spaCy NER models
BERT	Natural Language Processing (NLP) Labels text as Disinformation, Hate speech, and Sexist language	Hate Speech and Offensive Language Tweets flagged under Twitter's Hateful Conduct policy Fake News

Automatically deriving scores, categorical labels, or meaning from text or images involves the use of sophisticated models coupled with large archives of pre-labeled data. Some applications, such as Botometer, provide the necessary training datasets. Other applications enable you to use publicly available training datasets designed for particular tasks, or to build your own training data.

Provisioning Twitter Data for Exploratory Analysis

ML and NLP Process and Performance

Although ML and NLP applications could be run on the live data, in practice batch processing from the Neo4j graph database or the S3 archive is usually necessary. Tweet traffic is often so heavy that it would overwhelm the Neo4j server. In addition, ML and NLP applications are computationally time consuming for large amounts of data (Table 4). However, performance can be dramatically improved by running in parallel in the cloud. It is also possible to reduce or simplify data in the graph environment and then select only the data that requires enrichment.

Table 4. Batch processing times for third-party ML and NLP applications.

Application	Processing Time (minutes per 50,000 tweets)	Potential for faster processing
SpaCy (NER)	20	Can run in parallel; 10x increase easily possible.
BERT	5	Can run in parallel with a large enough GPU, or on multiple GPUs.

Demographic Identity with PyTorch and FairFace

PyTorch is an open source Machine Learning library based on the Torch library which is widely used for computer vision and NLP (Natural Language Processing). Its torchvision package can provide popular datasets and model architectures. However, for labeling demographic identity, we chose FairFace, a face attribute dataset balanced across race, gender, and age. The FairFace model is a neural network that has been pre-trained on millions of images and then fine-tuned using the FairFace dataset. The package can be used to label a user's profile image by gender (Male or Female), race either as one of four races (White, Black, Asian, Indian) or one of seven races (White, Black, Latino_Hispanic, East, Southeast Asian, Indian, Middle Eastern), and age range (0-2, 3-9, 10-19, 20-29, 30-39, 40-49, 50-59, 60-69, 70+).

We pull data for demographic enrichment from the Neo4j graph database. The new properties added to User nodes are *fairfaceGender*, *fairfaceRace4*, and *fairfaceRace*, which return text values, and *fairfaceAge*, which returns an age range as shown above. For racial analysis, we found that the seven categories returned in the *fairfaceRace* property were the most useful.

Provisioning Twitter Data for Exploratory Analysis

Human or Bot Likelihood with Botometer

Botometer is a machine learning algorithm trained to check the activity of Twitter accounts and evaluate the likelihood that a user account is operated by a human or a bot. A project of the [Observatory on Social Media \(OSoMe\)](#) at Indiana University, it is well-documented and supported. Its peer-reviewed [publications](#) provide technical details on features, training, machine learning models, and accuracy. Connection to Neo4j and Kineviz is through [Botometer APIs](#).

To check an account, its public profile and hundreds of its public tweets and mentions are fetched using the Twitter API. This data is passed to the Botometer API, which extracts over a thousand features to characterize the account's profile, friends, social network structure, temporal activity patterns, language, and sentiment. These features are used by various machine learning models to compute the bot scores. Botometer training data includes tens of thousands of labeled examples, and is regularly updated to maintain effectiveness as the Twitter ecosystem evolves. We used Version 4, released in September 2020. Botometer returns an overall bot score, as well as separate scores for six basic types of bot:

- **Astroturf:** manually labeled political bots and accounts involved in follow trains that systematically delete content.
- **Fake follower:** bots purchased to increase follower counts.
- **Financial:** bots that post using cashtags.
- **Self declared:** bots from botwiki.org.
- **Spammer:** accounts labeled as spambots from several datasets.
- **Other:** miscellaneous other bots from manual annotation, user feedback, etc.

The score is between 0 and 1, with 0 the most human-like, and 1 the most bot-like. The type of bot and the natural language used (e.g. English, French, etc.) is also returned. The results are written to User nodes as new properties *botoProb*, *botoType*, and *botoLang*.

Named Entity Recognition (NER) with spaCy

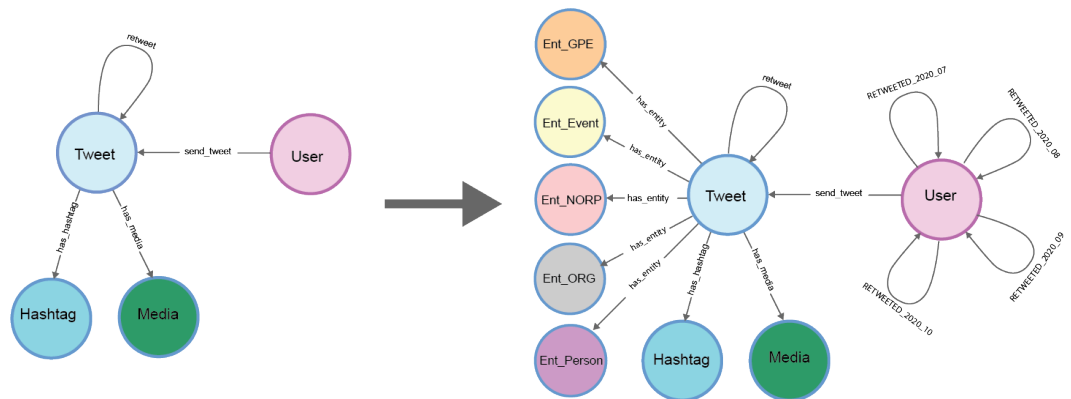
Named entity recognition (NER) can automatically label parts of a text with entities such as Person, Event, Organization, and so on. [spaCy](#) is an open-source Python library for advanced natural language processing. It provides [pre-trained NER models](#) as well as options for tuning and customization. Although the pre-trained models can be useful, custom models may return more satisfactory results. To set up and test the NER pipeline, an existing spaCy model was used to label tweets and extract the labels as separate categories in the graph data (Table 5).

Provisioning Twitter Data for Exploratory Analysis

Table 5. Named Entities added as categories to graph data.

NER Entity Label	Category (Node Label)
Event	Ent_Event
GPE (Geopolitical Entities)	Ent_GPE
ORG (Organizations)	Ent_ORG
NORP (Nationalities or Religious or Political) Groups	Ent_NORP
Person	Ent_Person

Using an AWS Lambda that runs daily, we automatically select tweets yet to be processed, run the appropriate NER, and add the new entity nodes and *has_entity* relationships to the database. The *has_entity* relationship connects the node to its corresponding Tweet node.



Sentiment Labeling with BERT

Bi-directional Encoder Representations from Transformers, or BERT, is a Natural Language Processing (NLP) application developed by Google. It was the first unsupervised, deeply bidirectional system for pre-training NLP, and as such outperforms previous unidirectional methods.

Using BERT involves a pre-training and a fine-tuning stage. The application must first pre-train language representations on a large body of plain text, then use that model for downstream NLP tasks. Since BERT is contextual and bi-directional, it generates a representation of each word based on the other words in the sentence. The process is both complex and iterative, and can be computationally expensive.

Provisioning Twitter Data for Exploratory Analysis

The base BERT model used was pre-trained on the Wikipedia English corpus and Toronto Book corpus. Pre-training on shorter tweet texts could improve performance. For fine-tuning, open-source training datasets were available for different types of text, as follows:

- Hate speech: [Hate Speech and Offensive Language](#)
- Sexism : Tweets flagged under Twitter's Hateful Conduct policy
- Disinformation: [Kaggle Fake News Dataset](#). This dataset includes about 20,000 real news articles and 17,000 fake news articles. Since the maximum length of a tweet is only 280 characters, the model was trained using only the article titles, not the full text.

The BERT process returns a label of 0 or 1 for a given type of text, which is written to Tweet nodes as the new properties *hateLabel*, *isFake*, or *isSexism*.

Data Transformation in Neo4j and Kineviz

Once in the graph environment, graph-aware transformations and algorithms let us calculate data such as the number of tweets and retweets sent by each user, or to assign users to custom-defined partisan groups such as Right and Left camps.

Most enrichment is done routinely through Cypher queries on the Neo4j graph database, ideally after each injection of new data. At a minimum, queries are run daily. This ensures that the enriched data is available for exploration and analysis in near real time.

When a query cannot be run in the database as a single query because it would run out of memory before completion, batch processing can be set up using Neo4j's periodic iteration routine *apoc.periodic.iterate*.

User Influence

Twitter data includes many properties that can be used to evaluate user influence. For the elections project, the number of tweets and retweets seemed appropriate measures, in that retweets can represent a vote of confidence in a user's expressed views. User properties such as the number of friends and followers, while also available, seemed of lesser value.

In the data stream, a retweet is defined as a relationship between tweets, and is not explicitly connected to users. However, when injected as graph data, Cypher queries can be used to flag and count the retweet relationship and add the result as a User property. Cypher queries can also count a user's tweets or retweets which contain labels added by ML or NLP applications.

Provisioning Twitter Data for Exploratory Analysis

Newly injected data is queried at least daily to add the following properties:

- *isRT* (1 if retweeted, 0 if original, added to Tweet nodes)
- *tweetRetweetedCnt* (number of retweets, added to User nodes) (Batch only, in batches of 1000)
- *hateTweetRetweetedCnt* (number of retweets with hateLabel = 1, added to User nodes)

To aid in analysis of monthly trends, a new retweeted relationship is added between User nodes:

- *RETWEETED_MONTH* (e.g. *RETWEETED_2020_10*, for retweets within a specific month)

Group Affiliation with Graph Algorithms

In addition to the group affiliations labeled using spaCy NER, graph-aware algorithms can be used to categorize users into groups that may not be evident from NER text labels. Both Neo4j and Kineviz provide pre-defined community detection algorithms, but a custom algorithm that performs more reliably can also be implemented.

For example, to assign users to Right or Left camps, we created a custom community detection algorithm which iterates on a population of highly followed users interconnected via their retweets. To be categorized in one or the other camp, a user must be mutually connected to at least 3 other users in the same camp. If connected to more than three other users, the group with greatest connectivity is favored.

We also used a pre-defined centrality algorithm to flag central users and hierarchies in selected data. Unlike Right and Left camps, these can be one way connections, and are not necessarily mutual relationships.

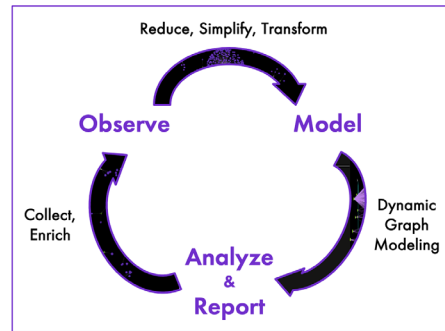
User Identity with OSINT (Open Source Intelligence) APIs

A large number of Open Source Intelligence applications have been developed, particularly for use in cybersecurity and investigation of fraud or other crimes. A recent [OSINT Handbook](#) lists APIs for a panoply of services that can gather additional information about a user account based on known data such as IP address, email, social media links, and more.

A few useful OSINT APIs have been made available within the graph environment (either Neo4j or Kineviz). For example, in Kineviz, the **Transform** panel's **Connector** tab links to an external application that can return a geographic location in lat-long coordinates from a user's IP address.

Iterative Visualization and Analysis

The Neo4j and Kineviz visualization and analytic framework works hand-in-glove with provisioning to support the highly iterative exploratory process that's required.



It becomes straightforward to:

- Reduce, Simplify, and Transform. Remove data such as Tweets with nonsense text or Users with no or few followers. Transform data to simplify or clarify connections. Both Neo4j and Kineviz enable data transformation of many kinds. In Kineviz such functions are available in **f(x)** (formulas), **Extract**, **Aggregate**, **Merge**, **Link**, and **ShortCut** tabs. Categories and relationships can be remodeled, for example to merge or aggregate data by category, and to create new relationships. In addition, data formats can be converted to those required for specific types of analysis or visualization and the results written to new properties. For example, strings can be converted to dates or numbers, or addresses to latitude/longitude coordinates.
- Model and analyze graph patterns dynamically. For example, we can select users with a large enough number of followers for further detailed analysis, and use graph algorithms to discover group affiliation. Or use the number of retweets per month as a measure of User influence over time.
- Answer questions of interest. For example, how does group sentiment differ by type or intensity? Does group sentiment differ by gender, race, or age? The results inevitably lead us to ask new questions and create new stories.
- Acquire new data to investigate new insights. Modify keywords in Twitter Tracker, and use additional ML or NLP applications and models to enrich data.
- Manage data persistence and archival to support comparative analysis and visualization over time. For the Elections project, graph data are stored in Neo4j. Data analyzed and transformed in Kineviz can be written back to Neo4j, saved as Kineviz data views and snapshots, or saved in CSV and JSON formats.

Conclusions

The graph environment that includes Neo4j and Kineviz provides the workspace and tools necessary for rapid exploratory data analysis, transformation, modeling, visualization, and archival as graph data. The processes and workflows are fundamentally iterative, because we expect analysis to spark further questions and new stories that call for additional cycles of data collection, transformation, and graph-aware modeling and analysis.

Automated provisioning is designed to support this iterative cycle by providing new enriched data either on demand or as batch processes. For the Election 2020 project, data must be provided reproducibly over time, but flexibility to select different data or to enrich it in different ways is also critically important. With this project we demonstrated highly responsive and flexible automated provisioning that delivered the data we needed over the election cycle. These are requirements that also exist for many other use cases in the graph space.

References and Further Reading

Twitter Developer APIs

Twitter Developer APIs. [Twitter Streaming API](#)

Twitter [NLP API](#).

Cloud Computing Services

Amazon Web Services (AWS)

AWS Overview.	Amazon Web Services
EC2	AWS EC2: Computing Service
S3	AWS S3. Storage Services
Glue	AWS Glue: Serverless ETL (Extract, Transform,, Load)
Lambda	AWS Lambda: Processing Microservices
Athena	AWS Athena: On-Demand SQL Query and Analysis
	AWS Athena & Quicksight
	Ippon: Using Athena and Glue
AWS Toolchains	AWS Toolchain: S3, Athena, Kinesis, SES, etc.
	Ippon: AWS Athena and Lambda for Twitter Trending Analysis
AWS ML/NLP	Amazon Comprehend (Natural Language processing service)

Google Cloud

[Google Cloud](#)
[Google Cloud Products](#)
[Google Cloud Services](#)

Microsoft Azure

[Microsoft Azure](#)
[Microsoft Azure Solutions](#)

Machine Learning and Natural Language Processing Applications

BERT

BERT NLP [BERT Explained](#)
[BERT Tutorial](#)

Hate Speech [Kaggle Hate Speech and Offensive Language Dataset](#)
[Pitsilis et. al., 2018. Detecting Offensive Language in Tweets Using Deep Learning](#)
[Davidson et. al. 2019. Racial Bias in Hate Speech...Detection Datasets.](#)
[Automated Hate Speech Detection and the Problem of Offensive Language](#)
[Hate Speech Twitter Annotations](#)
[Hate Speech Dataset Catalogue](#)
[Hate and Abusive Speech on Twitter](#)

Fake News [Fake and real news dataset](#)

Gender Bias [Voigt, R. et. al. 2018. RtGender: ...Studying Differential Responses to Gender](#)
[Dinan, E. et. al. 2020. Multi-Dimensional Gender Bias Classification](#)
[Hitti, Y. et. al. 2019. Dataset for Tracking Gender bias in Text](#)

Botometer

[BotoMeter by OsOMe](#)
[BotoMeter FAQ](#)

OSINT APIs and Toolkits

[OSINT Handbook](#)
[The OSINT Toolkit](#)
[Top 9 OSINT Tools](#)

PyTorch & FairFace

[PyTorch Classification](#)
[PyTorch Custom Image Classifier](#)
[Github--predict.py](#)
[Fairface Attribute Dataset for Balanced Race, Gender, and Age](#)
[Kärkkäinen, K., & Joo, J., 2019. Face Attribute Dataset for Balanced Race, Gender, and Age.](#)

spaCy Named Entity Recognition (NER)

[spaCy](#)
[Entity Extraction with spaCy](#)