Massive Data Analysis

Instructor: Dr.Gholampour

Fall 2024

Course Project

Arman Yazdani - 400102255

Contents

1 Pixie **3**

1.1 Graph Construction Stage 3

1.2 User Scores 4

1.3 User Similarity Detection via Pixie Algorithm 5

2 TrustRank-SpamMass Detection System **6**

2.1 Methodology 7

2.2 Critical Implementation 7

2.3 Scoring 8

2.4 Post-Filtering 8

2.5 Output Results 8

3 Stream Data **9**

3.1 Execution Parameters 10

3.2 Output Structure 10

1 Pixie

1.1 Graph Construction Stage

This stage constructs a directed multigraph $\mathcal{G} = (V, E)$ where:

- V: Set of vertices representing Twitter users
- E: Set of edges representing different types of interactions

Interaction Types

Four types of social media interactions are modeled as directed edges:

- **Reply Edges** (E_{reply}):

$$E_{\text{reply}} = \{(u, v) \mid u \text{ replies to } v's \text{ tweet}\}$$

  Extracted from `in_reply_to_screen_name` field

- **Retweet Edges** (E_{retweet}):

$$E_{\text{retweet}} = \{(u, v) \mid u \text{ retweets } v's \text{ content}\}$$

  Derived from `retweeted_status.user.screen_name`

- **Quote Edges** (E_{quote}):

$$E_{\text{quote}} = \{(u, v) \mid u \text{ quotes } v's \text{ tweet}\}$$

  Obtained from `quoted_status.user.screen_name`

- **Mention Edges** (E_{mention}):

$$E_{\text{mention}} = \{(u, v) \mid u \text{ mentions } v \text{ in tweet}\}$$

  Generated by exploding `entities.user_mentions` array

Edge Combination

The complete edge set is constructed through:

$$E = \bigcup_{i \in \{\text{reply,retweet,quote,mention}\}} E_i \setminus \{\text{duplicates}\}$$

Implemented via PySpark's `union()` followed by `dropDuplicates()` to maintain unique interactions while preserving edge multiplicity from different interaction types.

1.2 User Scores

Develop a quantitative measure of user influence based on interaction patterns in the dataset, then visualize relationships between top users.

Methodology

- **Interaction Weighting**:
  - Replies: 2.0 (highest weight, indicates direct engagement)
  - Retweets: 1.5 (medium weight, shows content amplification)
  - Quotes: 1.0 (basic weight, partial endorsement)
  - Others: 1.0 (fallback weight)
- **Score Calculation**:

$$\text{UserScore}_u = \sum_{\substack{\text{all outgoing} \\ \text{interactions } i \text{ from } u}} w_i$$

  Where w_i is the weight of interaction type i
- **Top User Selection**:
  - Sort users by descending score
  - Select top 200 users: $\text{TopUsers} = \{u_1, u_2, ..., u_{200}\}$

Graph Construction

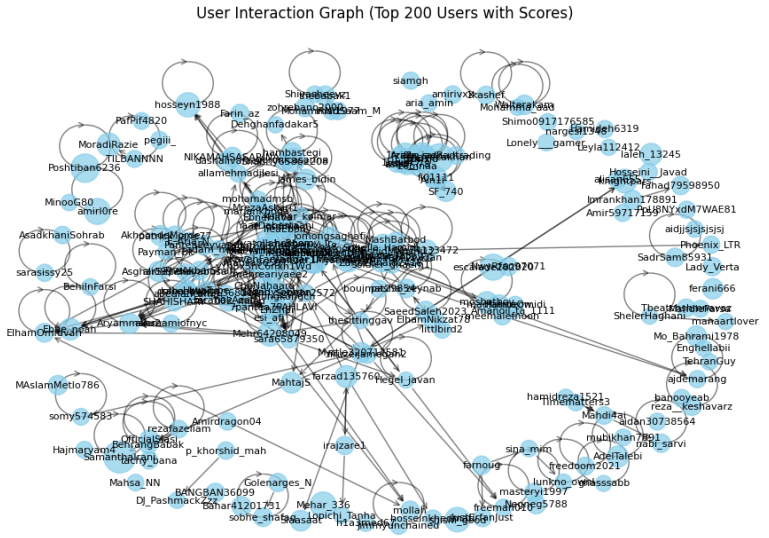
- **Node Representation**:
  - Each node corresponds to a user $u \in \text{TopUsers}$
  - Node size $\propto \text{UserScore}_u$ (visual proportionality)
  - Baseline size multiplier: $5\times$ for visibility
- **Edge Filtering**:

$$E_{\text{viz}} = \{(s, d) \in E \mid s \in \text{TopUsers} \wedge d \in \text{TopUsers}\}$$

Visualization Parameters

- **Layout**: Force-directed (spring layout) with fixed seed for reproducibility
- **Styling**:
  - Node transparency: $\alpha = 0.7$
  - Edge arrows: 10pt size with 0.5 alpha

– Color scheme: Cold tones for nodes



1.3 User Similarity Detection via Pixie Algorithm

Identify users with similar interaction patterns through weighted random walks on the directed graph G, leveraging edge weights as transition probabilities.

Methodology

- **Graph Structure**: Operate on $G = (V, E)$ where Directed edges with weight $w_{s \to d}$ representing interaction strength.
- **Walk Dynamics**:

$$P(s \to d) = \frac{w_{s \to d}}{\sum_{n \in \mathcal{N}(s)} w_{s \to n}}$$

  where $\mathcal{N}(s)$ denotes neighbors of node s
- **Similarity Metric**:
  – Visit count c_u to node u across all walks
  – Exclude seed node from counts to prevent self-similarity

Algorithm Parameters

- `num_walks`=200: Balances computation vs. statistical significance
- `walk_length`=3: Limits walk depth to local neighborhood
- Weight normalization: Default $w = 1.0$ if missing (safety mechanism)

Critical Implementation Details

```
def pixie_networkx_weighted(G, seed, num_walks=200,
```

```
                        walk_length=3):
    visit_counts = {}
    for _ in range(num_walks):
        current = seed
        for _ in range(walk_length):
            # Weighted neighbor selection
            weights = [G[current][nbr].get('weight', 1.0)
                    for nbr in neighbors]
            current = random.choices(neighbors,
                                weights=weights, k=1)[0]
        if current != seed:
            visit_counts[current] = visit_counts.get(current, 0) + 1
```

Result Interpretation

- Output ordering reflects similarity strength
- Visit counts c_u represent relative similarity scores
- Higher c_u indicates stronger connection patterns

Output Results

```
Pixie Similarity Results (based on random walks starting from 'Armanjasoor'):
User: adam_hesabi, Visits: 19
Pixie Similarity Results (based on random walks starting from 'Armanjasoor'):
User: Ivar_lathbrug2, Visits: 15
...
```

Theoretical Guarantees

- Convergence to stationary distribution with $\lim_{\text{walks}\to\infty}$
- Edge weights act as similarity reinforcement mechanism
- Directed graph preserves interaction causality

2 TrustRank-SpamMass Detection System

Identify spam tweets through hybrid analysis combining:

- TrustRank: Authority propagation from verified seed accounts
- SpamMass: Content/behavioral features indicating suspicious patterns

2.1 Methodology

- **Graph Construction**:
  - Build a directed graph based on user mentions to analyze interactions.
  - Seed users are selected based on engagement metrics and trust indicators.
- **Seed Selection**:

$$\text{SeedUsers} = \left\{ u \;\middle|\; \begin{array}{l} u_{\text{verified}} = \text{True}, \\ u_{\text{followers}} > 10^4, \\ \dfrac{u_{\text{friends}}}{u_{\text{followers}}} < 0.1, \\ \text{engagement\_rate} > 0.5 \end{array} \right\}$$

- **Trust Propagation**:

$$\text{Trust}_v^{(k+1)} = \underbrace{(1-\alpha)}_{\text{decay}} \cdot \text{Trust}_v^{(k)} + \alpha \sum_{u \to v} \frac{\text{Trust}_u^{(k)} \cdot w_{u \to v}}{d_{\text{out}}(u)}$$

  Where $\alpha = 0.85$ (damping), $k \leq 10$ iterations

- **Spam Features**:
  - Engagement risk: $1 - \text{engagement\_rate}$
  - URL density: $\min(\frac{\#\text{urls}}{5}, 1)$
  - Retweet ratio: $\min\left(\frac{\text{RTs}}{\text{followers}+1}, 5\right)$

2.2 Critical Implementation

```
# 1. Build Link Graph from User Mentions
mention_edges = df.select(
    F.col("user.id").alias("source_user"),
    F.explode("entities.user_mentions").alias("mention")
).select(
    "source_user",
    F.col("mention.id").alias("target_user")
)
# Trust propagation with decay
new_trust = mention_edges.join(trust_rank)
.withColumn("propagated_trust",
    col("trust_score") * DAMPING_FACTOR/(col("out_links")+1e-5)
```

```
).groupBy("target_user_id").agg(sum("propagated_trust"))

# Spam score calculation
spam_mass = spam_features.withColumn("spam_score",
exp(-(2.5*engagement_risk + 1.8*url_density + ...))
)

# Adaptive threshold
quantiles = final_scores.approxQuantile("spam_value", [0.75], 0.05)
dynamic_threshold = quantiles[0] * 1.5
```

2.3 Scoring

$$\text{SpamValue} = 0.6 \cdot (1 - \text{TrustScore}) + 0.4 \cdot \text{SpamScore}$$

2.4 Post-Filtering

- Remove tweets from users with $\text{TrustScore}_{\text{avg}} > 0.4$
- Keep top-5 spammy tweets per user (prevents over-flagging)

2.5 Output Results

text	spam_value
...دوستانی که ریموت	2.3017633100286403
...به عنوان پزشک، دا	6.385299525535547
...بنا به نظر #وزیر_	11920.432214382174
...ظاهراً سازمان اطل	3126.438134169463
...اگر صحبت‌های آ.عج	797.080963052442
...در پرونده مرحوم #	541.3463808223413

Why so much political tweets?

- Political tweets exhibit:

$$\#\text{Hashtags} \uparrow \Rightarrow f_3 \uparrow$$

$$\text{Retweet Ratio} \uparrow \Rightarrow f_5 \uparrow$$

$$\text{Negative Sentiment} \uparrow \Rightarrow f_4 \uparrow$$

$$\varepsilon_{\text{engagement}} \downarrow \Rightarrow f_1 \uparrow$$

Where features f_i penalize:

$$\begin{cases} f_1 = 1 - \varepsilon_{\text{engagement}} & \text{(Low engagement risk)} \\ f_2 = \frac{\#\text{URLs}}{5} & \text{(Link density)} \\ f_3 = \frac{\#\text{Hashtags}}{10} & \text{(Hashtag spam)} \\ f_4 = \mathbb{I}_{\text{negative}} & \text{(Controversial sentiment)} \\ f_5 = \frac{\text{retweets}}{\text{followers}+1} \cdot f_2 & \text{(Virality boost)} \end{cases}$$

- Political accounts often violate seed criteria:

$$\text{Seed Condition} = \underbrace{\text{verified}}_{P_1} \wedge \underbrace{\frac{\text{friends}}{\text{followers}} < 0.1}_{P_2} \wedge \underbrace{\varepsilon_{\text{engagement}} > 0.5}_{P_3}$$

  Many political accounts fail P_2 (high follow/follower ratio) and P_3 (engagement polarization).

- Political communities often form:

$$\frac{\partial T(u)}{\partial k} \propto \frac{1}{\text{CC}(G_{\text{pol}})}$$

  Where clustering coefficient $\text{CC}(G_{\text{pol}})$ reduces trust propagation between polarized groups.

3 Stream Data

Architecture Overview

The pipeline consists of 2 main stages:

- **Batch Simulation**: Convert static JSON data to CSV micro-batches
- **Stream Processing**: Two parallel analyses on the stream:
  - Real-time hashtag counts (2-second window)
  - Rolling sentiment average per hashtag

Core Processing Logic

1. Temporal Windowing for Hashtag Counts

Listing 1: Windowed Aggregation

```
hashtag_counts = tweets.withWatermark("event_time", "2s")
    .groupBy(
        window(col("event_time"), "2s"),
        col("hashtag")
    ).agg(count("*").alias("count"))
```

- **Watermarking**: Sets 2s threshold for late data
- **Tumbling Window**: Fixed 2-second intervals
- **Output Mode**: Update (only changes emitted)

2. Sentiment Scoring

Listing 2: Sentiment Mapping

```
sentiment_value = when(col("sentiment") == "Neg", -1)
                  .when(col("sentiment") == "Pos", 1)
                  .otherwise(0)
```

3. Stateful Sentiment Averaging

Listing 3: Sentiment Aggregation

```
hashtag_sentiment_avg = tweets.groupBy("hashtag")
    .agg(avg("sentiment_value").alias("avg_sentiment"))
```

- **Complete Mode**: Full recalc on each trigger
- **State Management**: Spark maintains hashtag aggregates

3.1 Execution Parameters

Parameter	Value
Trigger Interval	1 second
Watermark Delay	2 seconds
Micro-batch Size	10 records
Total Runtime	30 seconds

3.2 Output Structure

window	hashtag	hashtag_count
{2023-12-01 06:14...}	پرستو_معینی	1
{2023-11-22 11:18...}	دلیران_میدان	1
{2023-12-01 06:14...}	زهرا_صفایی	1
{2023-11-10 07:10...}	درمان_سرطان	1
{2023-11-10 07:10...}	سرطان_پروستات	1

hashtag	avg_sentiment
حسن_روحانی	0.0
قیام_سراسری	0.0
یمن	0.0
آرمیتا_گراوند	0.0
KingRezaPahlavi	0.0