Reddit SaaS Post Scraper Documentation

Optimized for Supervised Machine Learning - Real Data Only

Version 2.0 - Comment Filtering Approach

Table of Contents

1. Overview

Purpose

This scraper collects Reddit posts from entrepreneurship and startup-related subreddits, performs sentiment analysis on posts and comments, and creates a high-quality labeled dataset optimized for supervised machine learning models.

Critical Innovation (v2.0)

Comment Filtering Strategy - Only collects posts with minimum 3 comments to ensure ALL posts have valid comment sentiment scores. This eliminates the NaN problem and provides natural variance in comment sentiment without artificial noise.

Key Guarantee

No Zero Weights - All validation parameters (post sentiment, comment sentiment, upvote ratio, post recency) are guaranteed to contribute to the final validation score with real data variance.

Target Use Case

Training supervised ML models to classify SaaS ideas as "good", "neutral", or "bad" based on:

- Post content sentiment
- Public feedback (comment sentiment) - **Always present**
- Community engagement (upvote ratio)
- Temporal relevance (post recency)

2. System Architecture

Components

Core Modules

1. **reddit_scraper.py** - Main scraping engine with comment filtering
2. helpers.py - Utility functions for data processing
3. **weight_validator.py** - Automated weight optimization and labeling
4. **config.yaml** - Configuration file for all parameters

External Dependencies

- **PRAW** - Reddit API wrapper
- **Transformers** - Hugging Face sentiment analysis model
- **Pandas** - Data manipulation
- **NumPy** - Numerical operations

Data Flow

```
Reddit API → Post Filtering (min comments) → Sentiment Analysis →
NaN Handling (minimal) → Weight Optimization → Labeling → CSV Export
```

3. Key Features

3.1 Multi-Source Data Collection

Rotates through 7 different Reddit post sources:

1. New Posts
2. Hot/Trending Posts
3. Top Posts (Week)
4. Top Posts (Month)
5. Top Posts (Year)
6. Controversial Posts (Week)
7. Controversial Posts (Month)

Benefit: Maximum diversity in dataset, prevents sampling bias

3.2 Dual Sentiment Analysis

- **Post Sentiment**: Analyzes title + body text
- **Comment Sentiment**: Analyzes up to 15 newest comments (public feedback)

Model: DistilBERT (fine-tuned for sentiment analysis)

3.3 Smart Comment Filtering

- **Skips posts with < 3 comments** (configurable)
- Ensures all collected posts have valid comment sentiment
- Natural variance in comment sentiment (no artificial noise needed)

This is the key to solving the zero-weight problem

3.4 Smart Deduplication

- Global tracking of collected post IDs across all batches
- Prevents duplicate posts in dataset
- Automatic retry with fallback sources

3.5 Minimal NaN Handling

- Since posts are filtered, very few NaN values
- Simple median replacement (no noise needed)
- Enforces minimum value of 0.01

3.6 Minimum Weight Enforcement

- All parameters guaranteed to have weight >= 0.01
- Prevents unused features in validation
- Better for supervised ML training

3.7 Automated Per-Batch Weight Optimization

- Tests 286 weight combinations per batch
- Finds optimal weights based on label distribution entropy
- Adapts to data characteristics of each batch

4. Critical Innovation: Comment Filtering

The Problem (v1.0)

```
Collected 100 posts:
- 55 posts with comments → real comment sentiment
- 45 posts with NO comments → NaN comment sentiment

After NaN replacement with mean/median:
- 45 identical values → LOW variance
- Optimizer says "comment sentiment is useless"
- Weight forced to minimum 0.01

Result: Comment sentiment not actually used!
```

The Solution (v2.0)

```
Filter during collection:
- Check if post has &gt;= 3 comments
- If yes → collect it
- If no → skip it and fetch next post

After collection:
- ALL 100 posts have real comment sentiments
- Natural variance (different posts, different comments)
- NO NaN problem!

Result: Comment sentiment weight = 0.15-0.35 naturally!
```

Implementation

```
# In scraper
MIN_COMMENTS_REQUIRED = 3  # From config.yaml

# Skip posts with too few comments
if num_comments &lt; MIN_COMMENTS_REQUIRED:
    total_no_comments += 1
    continue  # Don't collect this post

# If comment sentiment still can't be calculated, skip
if len(comment_sentiments) == 0:
    total_no_comments += 1
    continue
```

Trade-offs

Pros:

- Real data only (no artificial noise)
- Natural variance in comment sentiment
- Comment sentiment actually used in validation
- Cleaner dataset

Cons:

- Takes longer to collect 100 posts (may need to check 120-150)
- Misses posts with no comments (but those aren't useful for comment sentiment anyway)

Verdict: The pros far outweigh the cons. This is the correct approach.

5. Installation & Setup

Prerequisites

```
Python 3.8+
pip (Python package manager)
Reddit API credentials
```

Step 1: Install Dependencies

```
pip install praw pandas numpy transformers python-dotenv pyyaml torch
```

Step 2: Reddit API Setup

1. Go to https://www.reddit.com/prefs/apps
2. Create a new application (script type)
3. Note down:
    - Client ID
    - Client Secret
    - User Agent

Step 3: Environment Variables

Create `.env` file in project root:

```
REDDIT_CLIENT_ID=your_client_id_here
REDDIT_SECRET=your_client_secret_here
REDDIT_USER_AGENT=your_user_agent_here
```

Step 4: Directory Structure

```
project_root/
├── src/
│   └── data_collection/
│       └── reddit_scraper.py
├── utils/
│   ├── helpers.py
│   └── weight_validator.py
├── config.yaml
├── .env
└── data/
    └── raw/
        ├── raw_batch/          # Output CSV files
        └── raw_batch_report/   # Optimization reports
```

6. Configuration Guide

config.yaml Structure

Scraper Settings

```
scraper:
  subreddits:
    - Entrepreneur
    - startups
    - indiehackers
  batch_size: 100          # Posts per batch
  max_batches: 10          # Total batches
  max_comments_per_post: 15 # For sentiment analysis

  # CRITICAL: Minimum comments required
  min_comments_required: 3  # Skip posts with &lt; 3 comments

  delay_min: 1.0           # Min delay between requests
  delay_max: 3.0           # Max delay between requests
```

Key Parameter: `min_comments_required`

- Default: 3
- Recommended: 3-5
- Higher = more selective, longer collection time
- Lower = faster collection, potentially more NaN

NaN Handling

```
nan_handling:
  method: 'median'  # Simple median (no noise needed!)
  min_value: 0.01   # Minimum value enforcement
```

Note: With comment filtering, NaN is rare, so simple median works fine.

Validation Thresholds

```
validation_thresholds:
  good: 70      # Score &gt;= 70 → "good"
  neutral: 40   # 40 &lt;= Score &lt; 70 → "neutral"
                # Score &lt; 40 → "bad"
```

Weight Validator

```
weight_validator:
  step_size: 0.1     # Weight granularity
  min_weight: 0.01   # Minimum weight per parameter
```

7. Data Collection Process

Phase 1: Post Extraction with Filtering

For each batch:

1. **Select Source**: Rotates through 7 sources based on batch number
2. **Fetch Posts**: Starts fetching from selected source
3. **Age Filter**: Checks if post age <= 180 days

4. **Deduplication Check**: Checks if post ID already collected
5. **Comment Filter**: **NEW** - Checks if post has >= min_comments_required
6. **Extract Metadata**: If all filters pass

Key Difference from v1.0: Step 5 is new and critical

Phase 2: Sentiment Analysis

Post Sentiment

```
full_text = title + " " + body
post_sentiment = sentiment_model(full_text)
# Returns: 0.0 (negative) to 1.0 (positive)
# Returns: NaN if text too short
```

Comment Sentiment

```
# Fetch newest 15 comments (already filtered for min count)
comments = sorted_by_newest[:15]

# Analyze each comment
comment_sentiments = [sentiment_model(c.body) for c in comments]

# If no valid sentiments after analysis, SKIP THIS POST
if len(comment_sentiments) == 0:
    skip_post()
    continue

# Calculate average (guaranteed to have values)
avg_comment_sentiment = mean(comment_sentiments)
# Returns: Real value, very rarely NaN
```

Phase 3: Feature Calculation

```
features = {
    'post_sentiment': 0.0 - 1.0,
    'avg_comment_sentiment': 0.0 - 1.0,  # Always valid!
    'upvote_ratio': 0.0 - 1.0,
    'post_recency': exp(-days_old / 30)
}
```

Collection Statistics

Expected:

- Posts examined: ~120-150 per batch
- Posts collected: 100 per batch

- Posts skipped (no comments): ~20-50 per batch
- Collection time: 8-12 minutes per batch (slightly longer)

8. NaN Handling Strategy

Why NaN Values Are Minimal

With comment filtering:

- **Post Sentiment NaN**: Rare (only if text too short)
- **Comment Sentiment NaN**: Very rare (posts filtered, comments analyzed)
- **Upvote Ratio NaN**: Rare (Reddit API issue)

Handling Method: Simple Median

```
# For the few NaN that remain
median_value = df['feature'].median()
df['feature'].fillna(median_value)
```

Why median instead of mean+noise:

- NaN count is very low (< 5%)
- Natural variance already high
- No need for artificial noise
- Simpler and more honest

Comparison

Method	NaN Count	Approach	Comment Weight
v1.0 (no filter)	~45%	mean+noise	0.01 (failed)
v2.0 (filter)	**~2%**	**simple median**	**0.15-0.35**

9. Weight Validation System

Weight Components

```
weights = [w1, w2, w3, w4]  # Sum = 1.0

where:
  w1 = post_sentiment weight
  w2 = avg_comment_sentiment weight
  w3 = upvote_ratio weight
  w4 = post_recency weight
```

Validation Score Calculation

```
score = (
    w1 * post_sentiment +
    w2 * avg_comment_sentiment +
    w3 * upvote_ratio +
    w4 * post_recency
) * 100

# Range: 0 - 100
```

Label Assignment

```
if score &gt;= 70:
    label = "good"
elif score &gt;= 40:
    label = "neutral"
else:
    label = "bad"
```

Expected Weight Distribution (v2.0)

With real comment sentiment variance:

```
Typical weights:
  - Post Sentiment:      0.25 - 0.40
  - Comment Sentiment:   0.15 - 0.35  ← Actually used!
  - Upvote Ratio:        0.15 - 0.30
  - Post Recency:        0.15 - 0.30
```

Example Output

```
Best Weights (All Parameters Used):
  - Post Sentiment:           0.32 [USED]
  - Comment Sentiment (avg):  0.24 [USED]  ← Real weight!
  - Upvote Ratio:             0.22 [USED]
  - Post Recency:             0.22 [USED]
  - Total: 1.00

Comment Sentiment Variance: 0.084523  ← High variance!
```

10. Output Files & Structure

10.1 CSV Files

Location: `data/raw/raw_batch/`

Filename: `reddit_data_batch_{N}.csv`

Columns:

```
post_id                 - Unique Reddit post ID
subreddit              - Source subreddit
title                  - Post title
text                   - Title + body combined
author                 - Reddit username
created_utc            - Timestamp (ISO format)
num_comments           - Number of comments (always &gt;= min_required)
upvotes                - Score from Reddit
upvote_ratio           - Ratio of upvotes (0-1)
post_age_days          - Age in days
post_sentiment         - Sentiment score (0-1)
avg_comment_sentiment  - Avg comment sentiment (0-1, rarely NaN)
post_recency           - Recency weight (0-1)
validation_score       - Final score (0-100)
label                  - good/neutral/bad
source_url             - Reddit permalink
source_type            - Data source
```

10.2 Report Files

Location: `data/raw/raw_batch_report/`

Filename: `reddit_data_batch_{N}_report.txt`

Contents (v2.0 format):

```
BATCH 1 - WEIGHT OPTIMIZATION RESULTS
==================================================

Source: NEW_POSTS
Posts skipped (no comments): 38

Optimization Accuracy: 98.45

Best Weights:
  - Post Sentiment:           0.32
  - Comment Sentiment (avg):  0.24  ← Real weight!
  - Upvote Ratio:             0.22
  - Post Recency:             0.22

Weights String: 0.32,0.24,0.22,0.22

Comment Sentiment Variance: 0.084523  ← High!
```

```
==================================================
```

11. Troubleshooting

Issue 1: Collection Takes Too Long

Symptom: Taking > 15 minutes per batch

Cause: High `min_comments_required` or subreddits with few commented posts

Solution:

- Lower `min_comments_required` to 2-3
- Add more active subreddits to the list
- Increase `batch_size` slightly to compensate

Issue 2: Comment Sentiment Weight Still Low

Symptom: Weight = 0.01-0.05 even with filtering

Cause: Comments are too similar in sentiment (all positive or all negative)

Solution:

- Add controversial subreddits (more varied opinions)
- Increase `max_comments_per_post` to 20
- Check if sentiment model is working correctly

Issue 3: Too Many Posts Skipped

Symptom: "Posts skipped (no comments): 80+"

Cause: Source has many low-engagement posts

Solution:

- Use "Hot" or "Top" sources instead of "New"
- Lower `min_comments_required` to 2
- Focus on more active subreddits

Issue 4: Comment Sentiment Variance Check Shows Low Value

Symptom: Variance < 0.02

Solution:

```
# In scraper output, look for:
[VARIANCE CHECK]
  avg_comment_sentiment: 0.015000  ← Too low!

# Then:
1. Increase min_comments_required to 5-7
2. Use more diverse subreddits
3. Check sentiment model is analyzing correctly
```

Issue 5: NaN in Comment Sentiment

Symptom: Despite filtering, some posts still have NaN

Cause: Comments exist but all fail sentiment analysis (too short, special chars, etc.)

Solution: Already handled - scraper skips these posts

```
if len(comment_sentiments) == 0:
    total_no_comments += 1
    continue
```

12. Best Practices

Data Quality

1. Comment Filtering Configuration

```
# Recommended settings
min_comments_required: 3-5

# Too low (1-2): Some posts may still lack diverse comments
# Recommended (3-5): Good balance
# Too high (&gt; 7): Very slow collection, may skip quality posts
```

2. Batch Size

- **Recommended**: 100 posts per batch
- Actual posts examined: ~120-150 with filtering
- Collection time: ~10 minutes per batch

3. Number of Batches

- **Recommended**: 10 batches minimum
- With filtering: 1000 high-quality posts
- Total time: ~2 hours for full dataset

4. Subreddit Selection

Choose active subreddits with engaged communities:

- Entrepreneur (high volume, many comments)
- startups (quality discussions)
- indiehackers (active builder community)
- microsaas (niche but engaged)

Comment Filtering

Optimal Settings

For Maximum Quality (slower):

```
min_comments_required: 5
max_comments_per_post: 20
```

For Balanced Approach (recommended):

```
min_comments_required: 3
max_comments_per_post: 15
```

For Faster Collection:

```
min_comments_required: 2
max_comments_per_post: 10
```

Monitoring Collection

Watch for these metrics:

```
-&gt; Collecting: 67/100 | No comments skipped: 42

If "No comments skipped" is very high:
- Lower min_comments_required
- Change to more active source (Hot instead of New)
```

Variance Verification

After each batch, check:

```
[VARIANCE CHECK]
  post_sentiment: 0.142000  ← Good
  avg_comment_sentiment: 0.078000  ← Good (should be &gt; 0.05)
  upvote_ratio: 0.039000  ← OK
```

If comment sentiment variance < 0.05:

- Increase min_comments_required
- Add more diverse subreddits
- Check sentiment model

Supervised ML Training

Feature Usage

With v2.0, all features have real contribution:

- Post Sentiment: Always used (0.25-0.40)
- **Comment Sentiment: Now properly used (0.15-0.35)**
- Upvote Ratio: Always used (0.15-0.30)
- Post Recency: Always used (0.15-0.30)

Model Training

```
# All 4 features are meaningful
X = df[['post_sentiment', 'avg_comment_sentiment', 'upvote_ratio', 'post_recency']]
y = df['label']

# No need to drop any features!
model.fit(X, y)
```

Appendix A: v1.0 vs v2.0 Comparison

Key Differences

Aspect	v1.0	v2.0
Collection	All posts	Filtered (min comments)
Comment NaN	~45%	~2%
NaN Handling	mean + noise	simple median

Aspect	v1.0	v2.0
Comment Variance	Low (0.02)	High (0.08)
Comment Weight	0.01 (min)	0.15-0.35 (real)
Collection Time	5-7 min	8-12 min
Data Quality	Mixed	High
Approach	Artificial fix	Natural solution

Why v2.0 is Better

1. **Real Data**: No artificial noise, just actual comment sentiments
2. **Natural Variance**: Different posts genuinely have different comment tones
3. **All Features Used**: Comment sentiment actually contributes
4. **Cleaner Dataset**: All posts have the features we care about
5. **ML-Ready**: Better for training supervised models

When to Use Each

v1.0 (with noise):

- When you need to collect ALL posts regardless of comments
- When dataset size is more important than quality
- Research/analysis where comment sentiment is not critical

v2.0 (with filtering):

- **For supervised ML training** (recommended)
- When comment sentiment is important
- When you want clean, high-quality data
- Production use cases

Appendix B: Complete Workflow

Step-by-Step Process

1. **Setup** (One-time)
   ```
   pip install requirements
   Configure Reddit API in .env
   Set min_comments_required in config.yaml
   ```
2. **Run Scraper**

```
python src/data_collection/reddit_scraper.py
```

3. **Monitor Progress**
   Watch for:
   - Posts skipped (no comments)
   - Variance check values
   - Weight distribution
4. **Verify Quality**

```
import pandas as pd
df = pd.read_csv('data/raw/raw_batch/reddit_data_batch_1.csv')

# Check NaN count
print(df['avg_comment_sentiment'].isna().sum())  # Should be &lt; 5

# Check variance
print(df['avg_comment_sentiment'].var())  # Should be &gt; 0.05
```

5. **Combine Batches**

```
import glob
csv_files = glob.glob('data/raw/raw_batch/*.csv')
df_combined = pd.concat([pd.read_csv(f) for f in csv_files])
df_combined.to_csv('combined_dataset.csv', index=False)
```

6. **Train ML Model**

```
from sklearn.ensemble import RandomForestClassifier

X = df_combined[['post_sentiment', 'avg_comment_sentiment',
                  'upvote_ratio', 'post_recency']]
y = df_combined['label']

model = RandomForestClassifier()
model.fit(X, y)
```

Appendix C: Configuration Examples

Example 1: High Quality, Slower

```
scraper:
  batch_size: 100
  max_batches: 10
  min_comments_required: 5
  max_comments_per_post: 20

# Expected: 1000 posts in ~2.5 hours
# Quality: Excellent
# Comment weight: 0.25-0.40
```

Example 2: Balanced (Recommended)

```
scraper:
  batch_size: 100
  max_batches: 10
  min_comments_required: 3
  max_comments_per_post: 15

# Expected: 1000 posts in ~2 hours
# Quality: Very good
# Comment weight: 0.15-0.35
```

Example 3: Quick Test

```
scraper:
  batch_size: 50
  max_batches: 2
  min_comments_required: 2
  max_comments_per_post: 10

# Expected: 100 posts in ~15 minutes
# Quality: Good
# For testing only
```

Conclusion

Version 2.0 solves the critical comment sentiment weight problem through a simple but effective approach: **only collect posts that have comments**. This ensures natural variance in comment sentiment without resorting to artificial noise, resulting in a cleaner, higher-quality dataset perfect for supervised ML training.

Key Improvements in v2.0:

- Comment filtering ensures all posts have valid comment sentiment
- Natural variance (no artificial noise needed)
- Comment sentiment weight 15-35% (vs 1% in v1.0)
- All 4 validation parameters properly used
- Better dataset quality for ML

Recommendation: Use v2.0 for all supervised ML applications.

Document Version: 2.0
Last Updated: October 29, 2025
Critical Change: Comment filtering strategy implemented