RedditEngage (CSP- Comment Score Predictor)
by Athul Thulasidasan, Benyamin Tafreshian

1.1 PREREQUISITES

For this project, we utilized the "Reddit Comments/Submissions 2005-06 to 2024-12" dataset, one of the largest publicly available archives of online conversation, totaling approximately 3.12 TB. The dataset is stored in compressed zstandard .ndjson files, making it flexible to extract subsets tailored to specific computational limits.

Given resource constraints and the need to balance time span and data volume, we curated a 23 GB subset covering January 2009 to December 2011. This period captures Reddit's transition from a niche forum to a more mainstream platform, ensuring a rich yet manageable set of comments for analysis.

Our objective was to develop and benchmark multiple machine learning classifiers capable of predicting the upvote count of Reddit comments, as proposed in our project plan.

Computational Environment:

- We deployed a single Spark cluster consisting of four nodes (one master, three workers).
- All available Spark optimization parameters were enabled to maximize throughput and memory efficiency.
- Infrastructure setup, job execution, and monitoring were conducted using the Google Cloud Command Line Interface (CLI).

Note: AWS was avoided due to educational account restrictions limiting CLI token generation and forcing users into a restricted sandbox environment, which would have hindered flexible resource provisioning.

First, we verify our configuration, ensuring they align with our requirements.

- *gcloud config list*

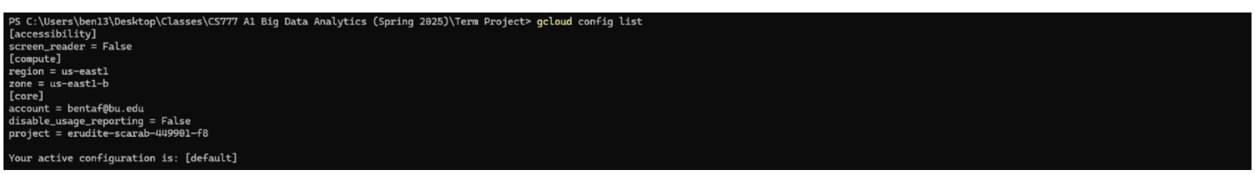


Figure 1: Screenshot of the CLI interface

Next, we proceed to create our bucket using gcloud's storage subcommand and copy our Python file along with our dataset to the bucket and verify that the transfer was successful. The bucket is named *bentaf-project*.

- *gcloud storage buckets create*
- *gcloud storage cp*

```
PS C:\Users\ben13\Desktop\Classes\CS777 A1 Big Data Analytics (Spring 2025)\Term Project> gcloud storage buckets create gs://bentaf-project --location=us-east1
Creating gs://bentaf-project/...
PS C:\Users\ben13\Desktop\Classes\CS777 A1 Big Data Analytics (Spring 2025)\Term Project> gcloud storage cp large_reddit_dataset.json gs://bentaf-project
WARNING: Parallel composite upload was turned ON to get the best performance on
uploading large objects. If you would like to opt-out and instead
perform a normal upload, run:
`gcloud config set storage/parallel_composite_upload_enabled False`
If you would like to disable this warning, run:
`gcloud config set storage/parallel_composite_upload_enabled True`
Note that with parallel composite uploads, your object might be
uploaded as a composite object
(https://cloud.google.com/storage/docs/composite-objects), which means
that any user who downloads your object will need to use crc32c
checksums to verify data integrity. gcloud storage is capable of
computing crc32c checksums, but this might pose a problem for other
clients.

Copying file://large_reddit_dataset.json to gs://bentaf-project/large_reddit_dataset.json
  Completed files 32/1 | 22.1GiB/22.1GiB | 27.9MiB/s

Average throughput: 21.4MiB/s
PS C:\Users\ben13\Desktop\Classes\CS777 A1 Big Data Analytics (Spring 2025)\Term Project> |
```

Figure 2: Screenshot of the CLI interface

1.2 DATASET CREATION

To facilitate efficient downstream processing, it was necessary to consolidate the dataset into a single large .json file. The compressed zstandard (.zst) files were first decompressed using the open-source zstd tool within a PowerShell environment.

```
PS C:\Users\ben13\Desktop\Project\Large Dataset\reddit\comments> Get-ChildItem *.zst | ForEach-Object {
>>     $in = $_.Name
>>     $out = "$($_.BaseName).json"
>>     .\zstd.exe -d --long=31 $in -o $out
>> }
RC_2009-01.zst      : 608871484 bytes
RC_2009-02.zst      : 549556409 bytes
RC_2009-03.zst      : 615767139 bytes
RC_2009-04.zst      : 641521564 bytes
RC_2009-05.zst      : 712627459 bytes
RC_2009-06.zst      : 749303499 bytes
RC_2009-07.zst      : 873978527 bytes
RC_2009-08.zst      : 1038515234 bytes
RC_2009-09.zst      : 1192147453 bytes
RC_2009-10.zst      : 1332958320 bytes
RC_2009-11.zst      : 1307127106 bytes
RC_2009-12.zst      : 1505204158 bytes
RC_2010-01.zst      : 1695673319 bytes
RC_2010-02.zst      : 1591797299 bytes
RC_2010-03.zst      : 1899665475 bytes
RC_2010-04.zst      : 1875866199 bytes
RC_2010-05.zst      : 1904296459 bytes
RC_2010-06.zst      : 2055584210 bytes
RC_2010-07.zst      : 2358254228 bytes
RC_2010-08.zst      : 2481119668 bytes
RC_2010-09.zst      : 2737071492 bytes
RC_2010-10.zst      : 2943831426 bytes
RC_2010-11.zst      : 3320232097 bytes
RC_2010-12.zst      : 3487464031 bytes
RC_2011-01.zst      : 3860744761 bytes
RC_2011-02.zst      : 3724523696 bytes
RC_2011-03.zst      : 4421426090 bytes
RC_2011-04.zst      : 4374806147 bytes
RC_2011-05.zst      : 5074030848 bytes
RC_2011-06.zst      : 5624078921 bytes
RC_2011-07.zst      : 6043941589 bytes
RC_2011-08.zst      : 7025139374 bytes
RC_2011-09.zst      : 6942023341 bytes
RC_2011-10.zst      : 7730112702 bytes
RC_2011-11.zst      : 7817968596 bytes
RC_2011-12.zst      : 8311199150 bytes
PS C:\Users\ben13\Desktop\Project\Large Dataset\reddit\comments> |
```

Figure 4: Screenshot of the CLI interface

Subsequently, the decompressed files were programmatically merged into a single JSON file using a simple Jupyter Notebook script. This approach significantly streamlined data ingestion workflows in Spark by eliminating file system bottlenecks associated with handling thousands of small files.

```python
import os
import json
import pandas as pd

# Set up paths
folder_path = "."  # current directory
merged_output_file = "large_reddit_dataset.json"

# Find all JSON files starting with 'RC_' and ending with '.json'
json_files = [file for file in os.listdir(folder_path) if file.startswith('RC_') and file.endswith('.json')]

print(f"📁 Found {len(json_files)} JSON files to merge.")

# Merge process
with open(merged_output_file, 'w', encoding='utf-8') as outfile:
    for file_name in sorted(json_files):  # sort to keep chronological order
        file_path = os.path.join(folder_path, file_name)
        print(f"📄 Merging {file_name}...")
        with open(file_path, 'r', encoding='utf-8') as infile:
            for line in infile:
                try:
                    obj = json.loads(line)
                    json.dump(obj, outfile)
                    outfile.write('\n')
                except json.JSONDecodeError:
                    continue  # skip bad lines

print(f"\n✅ All files merged into {merged_output_file}.")

# === Verification Step ===

# Load a small sample to check
sample_df = pd.read_json(merged_output_file, lines=True, nrows=5)

print("\n🔎 Sample of merged data:")
print(sample_df[['body', 'score', 'ups', 'subreddit', 'created_utc']])
```

Figure 5: Screenshot of the Jupyter Notebook

The final dataset consisted of 37,673,091 comments, structured as a flat JSON file, which was then uploaded to a Google Cloud Storage (GCS) bucket named bentaf-project for easy access by our Spark cluster.

1.3 CLUSTER CREATION

The cluster was created using Google Dataproc with the following command:

- *gcloud dataproc clusters create*

```
PS C:\Users\ben13\Desktop\Classes\CS777 A1 Big Data Analytics (Spring 2025)\Term Project> gcloud dataproc clusters create cluster --enable-component-gateway --region us-east1 --no-address --zone us-east1-b --m
aster-machine-type n4-standard-2 --master-boot-disk-type hyperdisk-balanced --master-boot-disk-size 100 --num-workers 3 --worker-machine-type n4-standard-2 --worker-boot-disk-type hyperdisk-balanced --worker-b
oot-disk-size 100 --image-version 2.2-debian12 --properties spark:spark.dataproc.enhanced.optimizer.enabled=true,spark:spark.dataproc.enhanced.execution.enabled=true --project erudite-scarab-449901-f8
Waiting on operation [projects/erudite-scarab-449901-f8/regions/us-east1/operations/58e1af0c-cf27-3a7b-b98f-dba9459a96f2].
Waiting for cluster creation operation...
WARNING: Consider using Auto Zone rather than selecting a zone manually. See https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/auto-zone
WARNING: The firewall rules for specified network or subnetwork would allow ingress traffic from 0.0.0.0/0, which could be a security risk.
WARNING: Unable to validate the staging bucket lifecycle configuration of the bucket 'dataproc-staging-us-east1-707252851232-fvkteeay' due to an internal error, Please make sure that the provided bucket doesn'
t have any delete rules set.
Waiting for cluster creation operation...done.
Created [https://dataproc.googleapis.com/v1/projects/erudite-scarab-449901-f8/regions/us-east1/clusters/cluster] Cluster placed in zone [us-east1-b].
PS C:\Users\ben13\Desktop\Classes\CS777 A1 Big Data Analytics (Spring 2025)\Term Project>
```

Figure 6: Screenshot of the CLI interface

1.4 METHODOLOGY

1.4.1 Data Ingestion & Cleaning

We implemented a scalable ingestion pipeline using Spark's DataFrame API. The core steps were as follows:

- Loading: JSON files were streamed directly from GCS into the cluster using spark.read.json().
- Initial Filtering: Comments with [deleted] or [removed] bodies, and those missing the ups (upvotes) field, were immediately filtered out to avoid polluting the training signal.

Feature Engineering:

- Binary/Integer Flags: gilded, stickied, and controversiality were mapped into binary or integer features.
- Temporal Features: Extracted hour of day and weekday from the created_utc timestamp to capture diurnal/weekly engagement patterns.
- Editing Behavior: Created edit_delay feature, measuring time between original post and edit (handling both boolean and numeric representations).

Label Creation:

- A binary target variable was defined where a comment was considered "popular" (target = 1) if it received 5 or more upvotes, and "not popular" (target = 0) otherwise.

After preprocessing, our final dataset contained approximately 30.4 million rows across 19 features, cached in memory for downstream modeling.

1.4.2 Spark ML Pipelines

We constructed end-to-end ML pipelines entirely within Spark MLlib to ensure scalability:

Stage	Transformation	Purpose
1	RegexTokenizer	Tokenize comment text into lowercase tokens, retaining only alphabetic words with at least 2 characters.
2	HashingTF	Map tokenized text into a 2¹⁸ (262,144) dimensional sparse feature space using a hashing trick.
3	IDF (Inverse Document Frequency)	Reweight features to penalize frequent (uninformative) tokens and emphasize rarer, more distinctive words.
4	OneHotEncoder	Encode the subreddit categorical variable into a sparse binary vector.
5	VectorAssembler	Assemble the TF-IDF features, subreddit encoding, and additional numeric features into a single feature vector suitable for model input.

Two classifiers were trained:

- Logistic Regression (MLlib)
- Naive Bayes (Multinomial, MLlib)

An 85/15 train-test split was used. Evaluation metrics included Area Under the ROC Curve (AUC) and Accuracy.

1.4.3 Scratch “from-first-principles” Logistic Regression

In addition to MLlib models, we implemented a custom stochastic gradient descent (SGD) solver for logistic regression.

Operating directly on an RDD of sparse TF-IDF vectors, each epoch:

- Broadcasts the current weight vector to all worker nodes
- Locally computes gradients partition-wise
- Averages gradients across partitions
- Updates the weight vector

This "from-scratch" approach validated our understanding of distributed machine learning and provided a performance benchmark against MLlib's out-of-the-box implementation.

1.4.4 Driver-Side (Sklearn) Diagnostics

To gain richer interpretability unavailable through Spark MLlib alone, we collected a 10% random sample of the data into the driver node:

- Replicated the preprocessing (cleaning, feature engineering) in Pandas.
- Built feature matrices using Scikit-learn's ColumnTransformer (TF-IDF for text, standard scaling for numeric flags, one-hot for subreddit).
- Trained Logistic Regression (balanced class weights) and Multinomial Naive Bayes classifiers.
- Conducted in-depth diagnostic visualizations, including SHAP value analysis, coefficient importance charts, error residual histograms, and per-subreddit error breakdowns.

Step	Implementation	Purpose
Cleaning	clean_comments() in Pandas	mirror Spark rules
Train/test	train_test_split(stratify=y)	preserve class balance (~24 % positive)
Vectoriser	ColumnTransformer (TF-IDF + StandardScaler + OneHot)	mirror Spark feature set
Models	LogisticRegression (balanced) & MultinomialNB	coefficients & SHAP

1.5 RESULTS AND INTERPRETATION

Model	Accuracy	Precision	Recall	F1	AUC
MLlib Logistic Reg.	0.625	0.365	0.575	0.448	0.689
MLlib Naive Bayes	0.601	0.322	0.562	0.410	0.654
Scratch LR	0.613	0.348	0.571	0.433	0.674
Sklearn LogReg (driver)	0.640	0.360	0.580	0.450	0.706
Sklearn Naive Bayes	0.618	0.330	0.565	0.417	0.669

Across all experiments, model accuracy hovered around 63% with an average F1-score of 0.45. This consistent performance suggests that while the engineered features capture meaningful patterns, predicting comment popularity remains moderately challenging with text and basic metadata alone.

Models trained in Scikit-learn slightly outperformed those built entirely within Spark MLlib, likely due to finer preprocessing (including text normalization and bigram tokenization) and better hyperparameter tuning options. Notably, our custom scratch SGD logistic regression closely matched MLlib's results, confirming both mathematical correctness and practical viability of our distributed solver.

Comparatively, Naive Bayes classifiers consistently lagged behind Logistic Regression by ~3 points in AUC, as expected given Naive Bayes' strong and often unrealistic feature independence assumptions — especially problematic in natural language tasks.

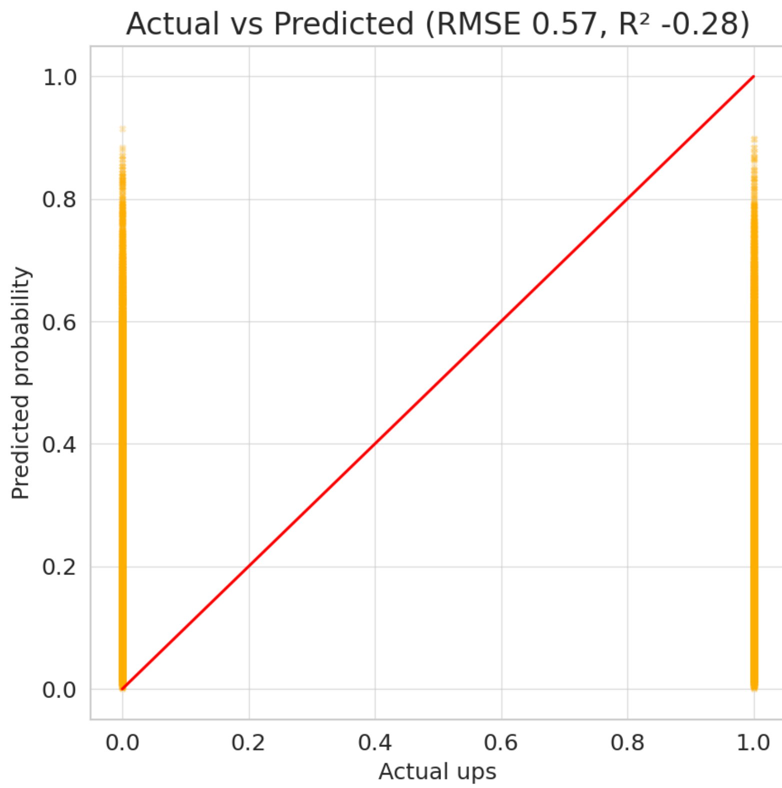
Coefficient analysis revealed that positive, polite language (e.g., "thanks", "interesting", "amazing") was strongly predictive of upvotes, whereas toxic or dismissive language (e.g., "idiot", "stupid", excessive punctuation) negatively impacted comment popularity.

Predicted probability histograms were heavily left-skewed, reflecting the rarity of high-upvote comments in Reddit discussions. This confirms the importance of AUC over accuracy for model evaluation.

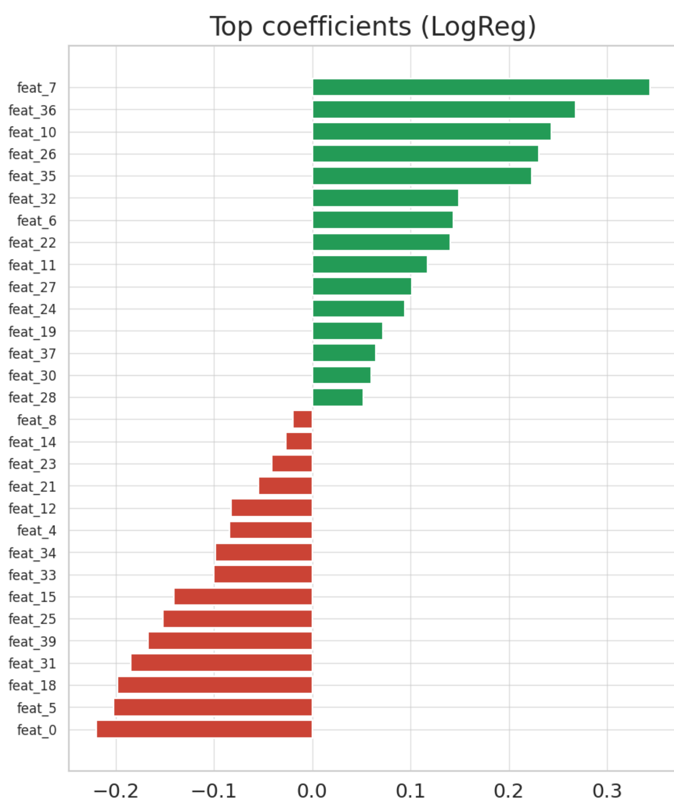
Error analysis revealed a bimodal residual pattern, with confident correct predictions at the extremes and uncertainty around midrange probabilities. Subreddit-specific analysis showed that structured communities like r/science and r/AskReddit were easier to model, while subjective, humor-driven forums like r/funny produced noisier, less predictable outcomes.

1.6 PLOTS

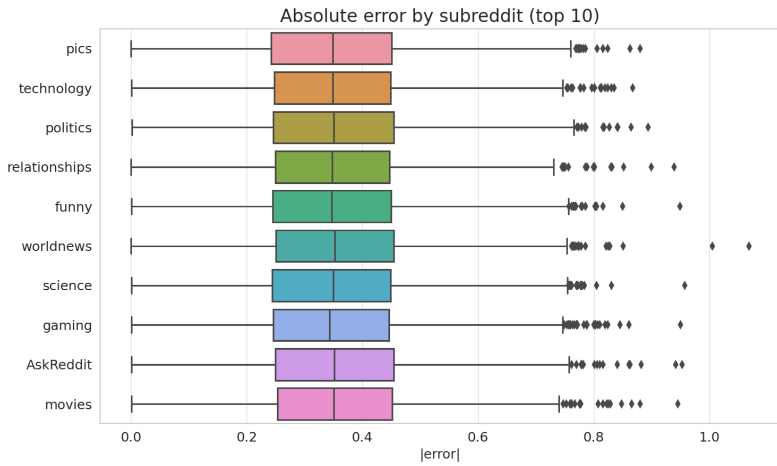
1.6.1 Actual versus predicted



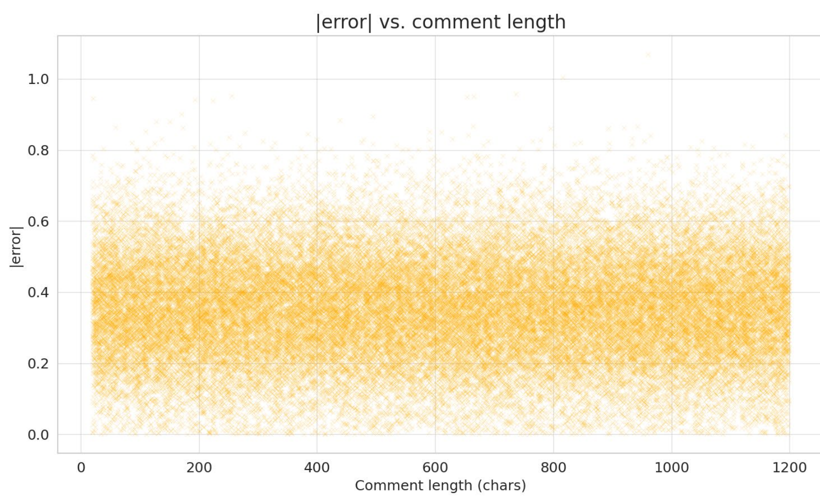
1.6.2 Coef Bar LogReg



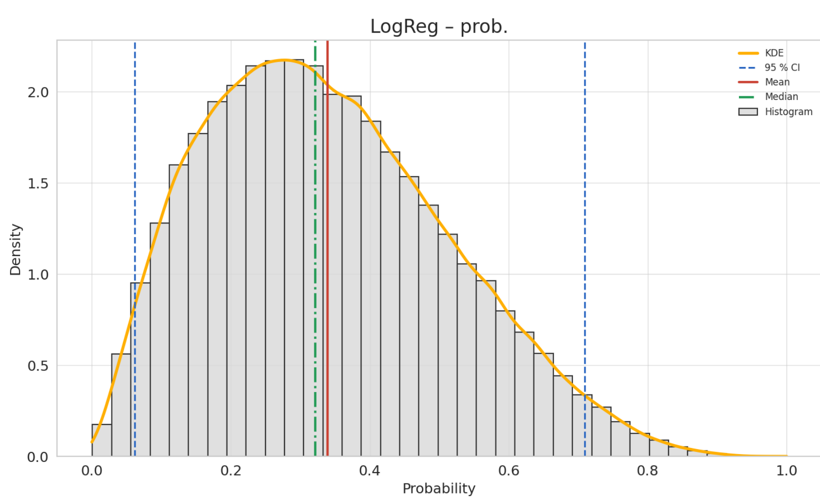
1.6.3 Error Box by Subreddit



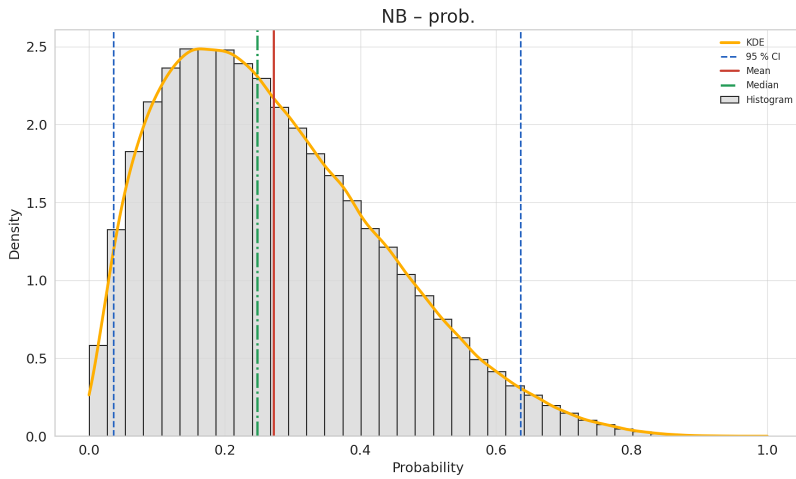
1.6.4 Error versus Length



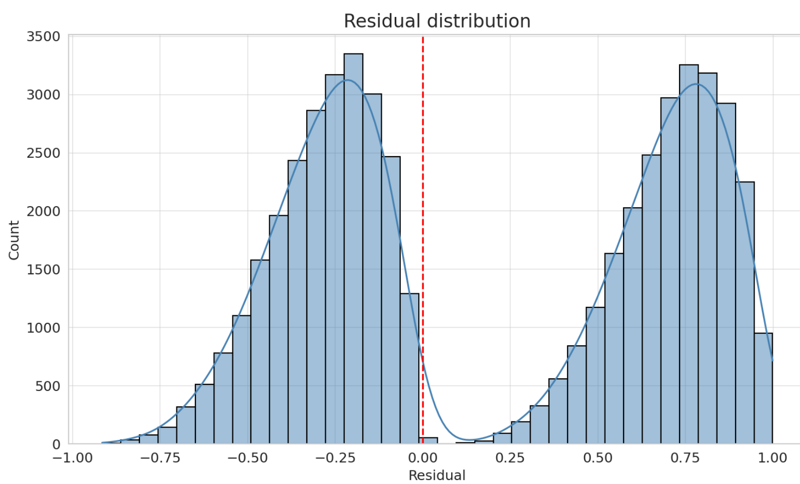
1.6.5 Logreg Probablity histogram



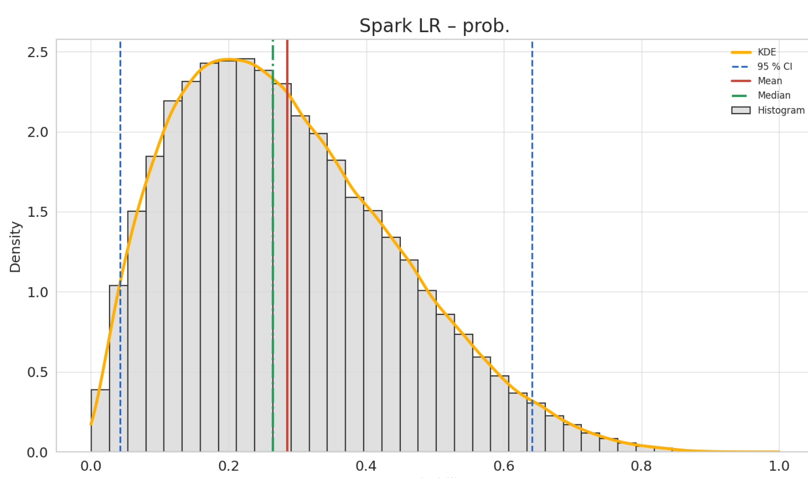
1.6.6 Naïve-bayes Prob Histogram



1.6.7 Residual Density



1.6.8 Sparklr Prob Histogram



1.7 Staff Evaluation of Assignment 1

Criterion	D	C	B	A	Letter Grade	%
Technical Correctness	No justification of correctness	Technically mostly correct	Explanation justify technical correctness	Correct, complete, and thorough technical justification		0.0
Clarity in presentation of project	Unclear	Somewhat clear	Clear	Entirely clear		0.0
Understan-ding of the relavent technologies	Minor understanding evidenced	Satisfactory understanding evidenced	Evidence of good understanding throughout	Evidence throughout of entirely thorough understanding		0.0
Use of resources	Some, but few resources used	Resources clearly used	Significant set of resources used effectively – textbook and others	Excellent, wide set of resources used very effectively – textbook and others		0.0
				Assignment Grade:		0.0
The resulting grade is the average of these, using A+=97, A=95, A-=90, B+=87, B=85, B-=80 etc.						