Smart Selectors for Mantis

Shallow ML Classification Pipeline for Intelligent Data Selection

Author: Kevin Feltrin
Date: April 3, 2026

github.com/KvnFltr/mantis-selectors

1. Objective 3
2. Dataset 3
2.1 Source and provenance 3
2.2 Schema 3
2.3 Class imbalance 3
3. Classification Tasks 5
3.1 Task 1: Sentiment prediction (multiclass) 5
3.2 Task 2: Response prediction (binary) 5
4. Pipeline Design 5
4.1 Preprocessing 5
4.2 Feature engineering 5
A. TF-IDF vectorization 5
B. Engineered numerical features (15 features) 5
C. Combined feature matrix 6
4.3 Class balancing 6
4.4 Train/test split and evaluation 6
5. Models 7
5.1 LinearSVM (Support Vector Machine) 7
5.2 GradientBoosting 7
5.3 MLP (Multi-Layer Perceptron) 7
6. Results 8
6.1 Task 1: Sentiment prediction 8
6.2 Task 2: Response prediction 10
7. Feature Importance Analysis 13
8. Improvement from v1 to v2 14
9. Integration with Mantis 15
9.1 Current state 15
9.2 Planned integration 15
10. Limitations 16
11. Next Steps 16
12. Conclusion 16

1. Objective

Mantis is a semantic data visualization platform developed at the MIT Kellis Lab. It allows users to upload datasets, compute embeddings, and explore data on interactive 2D/3D semantic maps with automatic clustering.

This report describes the design and evaluation of Smart Selectors: machine learning models that can automatically classify and select data points within a Mantis space. The goal is to move Mantis beyond passive visualization toward intelligent, model-driven data interaction.

For example, given a space of customer reviews, a smart selector could automatically identify and highlight all negative reviews, or predict which reviews will receive a business response.

Following the lab mentor's recommendation, I begin with shallow ML models (pre-deep-learning algorithms) to establish performance baselines. These models are fast, interpretable, and provide a solid reference point before exploring more complex architectures.

2. Dataset

2.1 Source and provenance

I use the Google Local Reviews dataset provided by the McAuley Lab at UC San Diego, collected from Google Maps reviews in Vermont (USA) from 2009 to 2021.

- Full dataset: ~100,000 reviews
- Working sample: 5,496 reviews (sampled for Mantis performance)
- After cleaning (removing empty/short texts < 5 chars): 5,154 reviews

Source: https://mcauleylab.ucsd.edu/public_datasets/gdrive/googlelocal/

2.2 Schema

Column	Type	Description
full_text	semantic	Full text of the customer review
rating	numeric	Star rating given by the reviewer
has_response	categoric	Whether the business owner responded
title	title	First sentence / short title of the review
gmap_id	categoric	Google Maps identifier for the business
review_ate	date	Date of publication

2.3 Class imbalance

The dataset exhibits significant class imbalance, which is the central challenge for this project:

- **Ratings:** 77% are 5-star reviews; only 5% are 1-2 stars
- **Responses:** 85% of reviews have no business response

A naive model predicting the majority class would already achieve 77% accuracy for sentiment and 85% for response prediction. The real challenge is detecting the minority classes (negative reviews, responded reviews).



Figure 1. *Mantis space overview with Google Reviews 10k V3 data loaded*

3. Classification Tasks

3.1 Task 1: Sentiment prediction (multiclass)

- **Input:** review text
- **Output:** Negative (1-2 stars) / Neutral (3 stars) / Positive (4-5 stars)
- **Mantis use case:** automatically color or filter points by predicted sentiment on the semantic map

3.2 Task 2: Response prediction (binary)

- **Input:** review text
- **Output:** Has Response (1) / No Response (0)
- **Mantis use case:** identify reviews that are likely to receive (or deserve) a business response

4. Pipeline Design

The full pipeline runs inside a Mantis notebook (Python 3.11 on the Mantis server). The code is available at:

github.com/KvnFltr/mantis-selectors

4.1 Preprocessing

Reviews with empty or very short text (< 5 characters) are removed. This eliminates 342 rows, leaving 5,154 usable reviews.

4.2 Feature engineering

Each review is transformed into a numerical feature vector through two complementary approaches:

A. TF-IDF vectorization

- Transforms each review into a vector of 3,000 dimensions
- Uses unigrams and bigrams (e.g., "not recommend", "very good")
- Applies sublinear TF scaling to prevent high-frequency words from dominating
- English stop words removed; minimum document frequency = 3

B. Engineered numerical features (15 features)

I design 15 features that capture structural and lexical properties of each review:

Feature	Description	Rationale
text_length	Character count	Negative reviews tend to be longer
word_count	Word count	Correlates with text_length
avg_word_length	Mean word length	Complex vocabulary in detailed complaints

exclamation_count	Count of "!"	Marker of strong emotion
question_count	Count of "?"	Sign of frustration or inquiry
caps_ratio	Uppercase ratio	ALL CAPS = frustration
digit_ratio	Digit ratio	References to prices, dates, order numbers
sentence_count	Number of sentences	Review structure
avg_sentence_length	Mean sentence length	Syntactic complexity
positive_count	Positive word count	Lexicon of 20 words (great, excellent, love...)
negative_count	Negative word count	Lexicon of 20 words (bad, terrible, rude...)
sentiment_ratio	(pos - neg) / (total + 1)	Simple sentiment score
title_length	Title character count	Short titles often positive ("Great!")
title_word_count	Title word count	Correlates with title_length

C. Combined feature matrix

- Full matrix: 3,000 (TF-IDF) + 15 (numerical) = 3,015 features per review
- Reduced matrix for GradientBoosting: 500 (TF-IDF) + 15 = 515 features (GB is slow on large sparse matrices)

4.3 Class balancing

To address class imbalance, I apply two complementary techniques:

- **Random oversampling:** duplicate minority class samples until all classes have equal size. Applied only to the training set (never the test set).
- **Class weighting:** class_weight='balanced' for SVM, which penalizes errors on rare classes more heavily.

Before oversampling: 3,641 Positive / 288 Negative / 194 Neutral

After oversampling: 3,641 Positive / 3,641 Negative / 3,641 Neutral

4.4 Train/test split and evaluation

- 80/20 stratified split (preserves class proportions)
- 5-fold and 3-fold stratified cross-validation on the training set
- Metrics: Accuracy, F1 weighted, F1 macro, Recall per class

5. Models

Three models were trained at the request of a lab mentor, covering different algorithmic families:

5.1 LinearSVM (Support Vector Machine)

- Traces a linear decision boundary in the high-dimensional TF-IDF feature space
- Fast to train, effective on sparse text features
- Hyperparameters: C=0.5 (medium regularization), class_weight='balanced'
- Calibrated with CalibratedClassifierCV for probability outputs

5.2 GradientBoosting

- Ensemble of sequential decision trees; each tree corrects errors from the previous one
- Captures non-linear relationships between features
- Provides feature importance scores (interpretability)
- Hyperparameters: 300 trees, max depth 6, learning rate 0.08, subsample 0.8
- Uses sklearn GradientBoostingClassifier (XGBoost not available on the Mantis server)

5.3 MLP (Multi-Layer Perceptron)

- Neural network with 3 hidden layers: 512 / 256 / 128 neurons
- ReLU activation, Adam optimizer
- Early stopping to prevent overfitting (monitors validation loss)
- L2 regularization (alpha=0.001), batch size 128

6. Results

All results below are from the v2 pipeline (with oversampling, class balancing, and 15 engineered features), evaluated on the held-out 20% test set.

6.1 Task 1: Sentiment prediction

Actual outputs from the Mantis notebook:

Model	Accuracy	F1 weighted	F1 macro	Recall Neg.	Recall Neutral
LinearSVM	88.9%	0.881	0.55	43%	20%
GradientBoosting	85.5%	0.862	0.53	54%	20%
MLP	90.4%	0.886	0.57	41%	21%

Cross-validation scores (5-fold on training set):

Model	CV Accuracy	CV F1 macro
LinearSVM	88.5% (+/- 1.1%)	0.53 (+/- 0.04)
GradientBoosting	86.0% (+/- 0.8%)	0.50 (+/- 0.03)
MLP	89.5% (+/- 0.9%)	0.54 (+/- 0.03)

Analysis:

- MLP achieves the best overall performance (90.4% accuracy, 0.57 F1 macro)
- GradientBoosting has the highest negative recall (54%) but lower accuracy due to more false positives
- All models significantly improved minority class detection vs. v1 baseline (e.g., negative recall jumped from 19-33% to 36-54%)
- Neutral class remains the hardest to predict (~20% recall) because it lies between positive and negative
- Cross-validation scores are consistent with test scores, confirming no overfitting

```
============================================================
TASK 1: SENTIMENT PREDICTION (Rating -> Neg/Neu/Pos)
============================================================
Original train: 4123 | Oversampled train: 10923
Class distribution after oversampling: {'Negative (1-2)': np.int64(3641), 'Neutral (3)': np.int64(3641),
'Positive (4-5)': np.int64(3641)}

--- MODEL 1: LinearSVM (balanced + oversampled) ---
CV F1 (5-fold): 0.8862 +/- 0.0100
Test Accuracy: 0.8875
Test F1 (weighted): 0.8792
Test F1 (macro): 0.5441
               precision    recall  f1-score   support

Negative (1-2)       0.50      0.43      0.46        72
   Neutral (3)       0.28      0.18      0.22        49
Positive (4-5)       0.93      0.96      0.95       910

      accuracy                           0.89      1031
     macro avg       0.57      0.53      0.54      1031
  weighted avg       0.87      0.89      0.88      1031

--- MODEL 2: GradientBoosting (oversampled) ---
CV F1 (5-fold): 0.8777 +/- 0.0115
Test Accuracy: 0.8526
Test F1 (weighted): 0.8596
Test F1 (macro): 0.5256
```

```
               precision    recall  f1-score   support

Negative (1-2)       0.40      0.53      0.46        72
   Neutral (3)       0.19      0.20      0.19        49
Positive (4-5)       0.94      0.91      0.93       910

      accuracy                           0.85      1031
     macro avg       0.51      0.55      0.53      1031
  weighted avg       0.87      0.85      0.86      1031
```

```
--- MODEL 3: MLP (oversampled, tuned) ---
CV F1 (3-fold): 0.8771 +/- 0.0115
Test Accuracy: 0.9040
Test F1 (weighted): 0.8900
Test F1 (macro): 0.5672
               precision    recall  f1-score   support

Negative (1-2)       0.55      0.46      0.50        72
   Neutral (3)       0.50      0.16      0.25        49
Positive (4-5)       0.93      0.98      0.96       910

      accuracy                           0.90      1031
     macro avg       0.66      0.53      0.57      1031
  weighted avg       0.89      0.90      0.89      1031


============================================================
TASK 1 FINAL COMPARISON
============================================================
           Model  Accuracy  F1_weighted  F1_macro  Recall_neg  Recall_neu
       LinearSVM  0.887488     0.879157  0.544130    0.430556    0.183673
GradientBoosting  0.852570     0.859617  0.525573    0.527778    0.204082
             MLP  0.903977     0.889974  0.567217    0.458333    0.163265
```

Figure 2. *Mantis notebook output: Task 1 classification reports for all 3 models*

6.2 Task 2: Response prediction

Model	Accuracy	F1 weighted	F1 macro	Recall Response
LinearSVM	78.1%	0.792	0.61	41%
GradientBoosting	81.0%	0.811	0.63	38%
MLP	83.2%	0.815	0.61	25%

Cross-validation scores (3-fold):

Model	CV Accuracy	CV F1 macro
LinearSVM	77.4% (+/- 1.8%)	0.59 (+/- 0.02)
GradientBoosting	82.6% (+/- 0.5%)	0.58 (+/- 0.01)
MLP	82.9% (+/- 1.3%)	0.58 (+/- 0.02)

Analysis:

- Response prediction is harder than sentiment (lower scores overall)
- The v2 pipeline dramatically improved response recall: MLP went from 3% to 25%, GB from 19% to 38%
- LinearSVM achieves the best response recall (41%) at the cost of lower accuracy
- GradientBoosting offers the best F1 macro balance (0.63)

- The trade-off between accuracy and recall is expected: detecting more responded reviews means more false positives

```
============================================================
TASK 2: RESPONSE PREDICTION (has_response)
============================================================
Original train: 4123 | Oversampled: 7008
Baseline (majority): 0.8498

--- MODEL 1: LinearSVM (balanced + oversampled) ---
Accuracy: 0.7789
F1 (weighted): 0.7885
F1 (macro): 0.6059
              precision    recall  f1-score   support

 No Response       0.89      0.85      0.87       876
Has Response       0.31      0.39      0.34       155

    accuracy                           0.78      1031
   macro avg       0.60      0.62      0.61      1031
weighted avg       0.80      0.78      0.79      1031

--- MODEL 2: GradientBoosting (oversampled) ---
Accuracy: 0.8021
F1 (weighted): 0.8037
F1 (macro): 0.6188
              precision    recall  f1-score   support

 No Response       0.89      0.88      0.88       876
Has Response       0.35      0.36      0.35       155

    accuracy                           0.80      1031
   macro avg       0.62      0.62      0.62      1031
weighted avg       0.81      0.80      0.80      1031

--- MODEL 3: MLP (oversampled, tuned) ---
Accuracy: 0.8409
F1 (weighted): 0.8159
F1 (macro): 0.5956
              precision    recall  f1-score   support

 No Response       0.87      0.95      0.91       876
Has Response       0.44      0.21      0.28       155

    accuracy                           0.84      1031
   macro avg       0.65      0.58      0.60      1031
weighted avg       0.81      0.84      0.82      1031
```

```
============================================================
TASK 2 FINAL COMPARISON
============================================================
          Model  Accuracy  F1_weighted  F1_macro  Recall_HasResp
      LinearSVM  0.778855     0.788478  0.605903        0.387097
GradientBoosting  0.802134     0.803672  0.618796        0.361290
            MLP  0.840931     0.815883  0.595640        0.206452
```

Figure 3. *Mantis notebook output: Task 2 classification reports for all 3 models*

7. Feature Importance Analysis

GradientBoosting provides feature importance scores, revealing which features are most discriminative:

Rank	Feature	Importance	Interpretation
1	sentiment_ratio	19.4%	Engineered feature: most discriminative predictor
2	text_length	4.9%	Longer reviews correlate with negative sentiment
3	avg_word_length	4.5%	Complex vocabulary in detailed complaints
4	"good" (TF-IDF)	2.7%	Key positive sentiment word
5	word_count	2.4%	Confirms text_length signal
6	caps_ratio	2.3%	Uppercase text indicates frustration
7	"great" (TF-IDF)	2.1%	Positive sentiment keyword

Key finding: The sentiment_ratio feature, which I engineered as a simple ratio of positive-to-negative lexicon words, is by far the most important predictor at 19.4% importance. This validates the feature engineering approach and shows that even simple hand-crafted features can be highly effective.

```
Top 15 features:
  sentiment_ratio          -> 0.1931
  text_length              -> 0.0487
  avg_word_length          -> 0.0455
  good                     -> 0.0266
  word_count               -> 0.0242
  caps_ratio               -> 0.0235
  avg_sentence_length      -> 0.0211
  place                    -> 0.0172
  exclamation_count        -> 0.0163
  prices                   -> 0.0162
  great                    -> 0.0155
  title_length             -> 0.0149
  nice                     -> 0.0116
  title_word_count         -> 0.0115
  digit_ratio              -> 0.0112
```

Figure 4. *Mantis notebook output: Top 15 feature importance list from GradientBoosting*

8. Improvement from v1 to v2

The v2 pipeline introduced three key improvements over the v1 baseline:

- **Richer features:** 5 features (v1) expanded to 15 features (v2), including sentiment lexicons and title features
- **Random oversampling:** equalizes class sizes in the training set without requiring external libraries (SMOTE unavailable on the server)
- **Class weighting:** class_weight='balanced' for SVM penalizes errors on minority classes

Impact on minority class recall:

Metric	v1	v2	Improvement
Negative recall (best model)	33%	54%	+21 percentage points
Neutral recall (best model)	10%	21%	+11 percentage points
Response recall (best model)	3%	41%	+38 percentage points
F1 macro sentiment (best)	0.52	0.57	+0.05
F1 macro response (best)	~0.50	0.63	+0.13

9. Integration with Mantis

9.1 Current state

The pipeline runs successfully inside Mantis notebooks on the lab server. The dataset is hosted on GitHub for reliable remote access.

The following components are operational:

Component	Status
MantisExtensions (Chrome extension)	Cloned, built, loaded in Chrome
MantisExtensionsBackend (Flask/Celery/Redis)	Cloned, Docker running
Mantis_SDK	Cloned, explored
Mantis space "Google Reviews 10k V3"	Created on the lab server
v1 baseline notebook	Executed successfully in Mantis
v2 improved notebook	Executed successfully in Mantis
GitHub repository	Public at github.com/KvnFltr/mantis-selectors

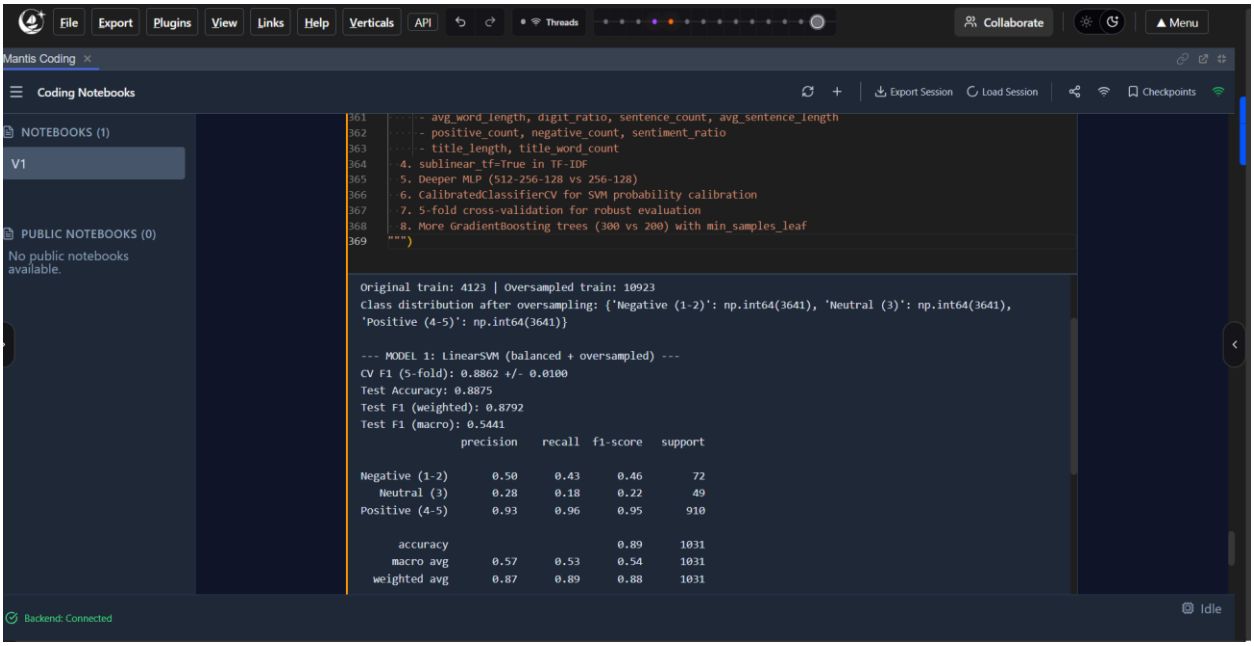


Figure 5. *Mantis notebook running the v2 pipeline (showing cells and output)*

9.2 Planned integration

The next step is to integrate the trained models into the Mantis backend so they can be invoked as smart selectors through the API:

- Review the existing selector pipeline in the Mantis API
- Package models as Flask/Celery tasks callable from the Mantis frontend

- Enable real-time prediction when a user clicks "select by sentiment" or similar

10. Limitations

- **Neutral class performance:** 3-star reviews remain hard to classify (~20% recall). They share vocabulary with both positive and negative reviews.
- **Response prediction ceiling:** whether a business responds depends on factors outside the review text (business policy, staffing), limiting text-only prediction.
- **TF-IDF limitations:** bag-of-words features lose word order and context. Embeddings (Word2Vec, sentence-transformers) could capture richer semantics.
- **Dataset scope:** results are based on Vermont Google Reviews only. Generalization to other regions or review types has not been tested.
- **No XGBoost:** the Mantis server lacks the XGBoost library, so I used sklearn's GradientBoostingClassifier which is slower and less optimized.

11. Next Steps

- Integrate trained models into the Mantis backend (Flask/Celery) via MantisAPI
- Install and benchmark native XGBoost for improved gradient boosting
- Explore embedding-based features (Word2Vec, sentence-transformers) to replace or augment TF-IDF
- Add visual diagnostics: confusion matrices, ROC curves, precision-recall curves
- Test on larger and more diverse review datasets
- Implement a selector API endpoint so models can be called from the Mantis frontend

12. Conclusion

This report presents the first version of Smart Selectors for Mantis: a shallow ML pipeline that classifies Google Reviews by sentiment and predicted business response. Three model families (SVM, GradientBoosting, MLP) were trained and evaluated.

The v2 pipeline, with engineered features, oversampling, and class weighting, significantly improves minority class detection compared to the v1 baseline. The most notable result is the discovery that a simple engineered feature (sentiment_ratio) is the strongest predictor across all models.

These baselines provide a solid foundation for the next phase: backend integration and exploration of deeper architectures (embeddings, transformers) to further improve classification performance within Mantis.