Tweet Sentiment Label Quality: A Text Classification Analysis

David Ewing · Student ID: 82171165

1 Problem Statement

This project examines how well a text classifier can reproduce the sentiment labels applied to a sample of tweets. The dataset—Cardiff NLP's tweet_eval sentiment subset—has 45,615 tweets, each tagged as negative, neutral, or positive. Labelling was conducted as part of the SemEval 2017 Task 4A shared task on sentiment analysis in Twitter (Rosenthal et al., 2017), and the labels are widely used as an NLP benchmark.

My central questions:

- *How reliably is a text classifier able to reproduce the human-assigned sentiment labels?*
- *What does a text classifier's performance tell us about the quality and consistency of the labels?*

This matters because classification performance may reflect the difficulty of the task and inconsistency in how the labels were assigned. A classifier that struggles on neutral tweets may be picking up that the boundary between neutral and mildly positive or mildly negative is ambiguous. Different annotators may draw the line differently, e.g, low F1 may be evidence of noisy labels. Both may be true at once.

Four smaller questions guide this analysis:

1. What kinds of problems are present in the labelling of tweets?
2. What types of tweets are systematically hard to classify?
3. What features are most helpful for distinguishing sentiment classes?
4. Does removing the neutral class substantially improve classifier performance?

Annotated datasets are expensive to produce. My assumption — once created it is used repeatedly. Models comparisons will be using the same labels, and not be questioned. Gold-standard labels then contain systematic noise. A model with a higher F1 than a competitor may then be better at reproducing the particular pattern of annotator disagreement.

Macro average F1 is the most appropriate metric for this task. The validation split has 2,000 tweets. The training data has a class-imbalanced—neutral tweets make up about 45%. To address this, random undersampling was applied to reduce all classes to the size of the smallest. Every class ends up with 7,093 tweets—so 21,279 in total. I did not want to oversample. A model that simply memorises or recombines training examples would be a bad fit. The training data already has noisy labels. Learning to replicate those examples would just bake the noise in rather than generalise past it

2 Dataset

The dataset is the sentiment subset of the Cardiff NLP tweet_eval benchmark (Barbieri et al., 2020), collected from Twitter and annotated for the SemEval 2017 Task 4A competition. Labels—negative, neutral, and positive—were assigned by crowdsourced annotators, with the final label determined by majority vote. I find myself questioning whether majority vote is an adequate basis for a gold standard. It is cost-effective, but it introduces noise that is difficult to separate from genuine disagreement.

The labels came from Amazon Mechanical Turk[1]. Workers were paid small amounts to read each tweet and click a sentiment label (Rosenthal et al., 2017). No expertise required. Each tweet receives at least three annotations. The majority label stands. Annotators disagree—sometimes the final label reflects little more than a coin toss—and when I look at the data closely, the so-called "ground truth" starts to feel less certain:

- a mildly sarcastic tweet;
- an expression of resignation that could be read as neutral or mildly negative; and
- the resulting label may not reflect genuine consensus.

If a model gets a tweet "wrong", I can't help but ask if the gold label was ever solid to begin with. A lot of times, it wasn't.

Random undersampling (random_state=82171165) was applied to balance the training classes, producing 21,279 tweets—7,093 per class. The validation split (2,000 tweets) serves as the evaluation set throughout. The test split (12,284 tweets) is held out and not used. Exact per-class counts at each stage are shown in notebook section 2.5.

The validation set is class-imbalanced: 312 negative, 869 neutral, 819 positive. The negative class has fewer than half the examples of positive, and less than 40% as many as neutral. The moment I noticed this, I realised that interpreting negative-class results would require care. Poor performance there could reflect label noise, weak feature representations, or simply insufficient test examples—most likely some combination of all three.

A further complication is that confidence intervals for per-class estimates differ substantially. With only 312 negative tweets, any F1 score for that class carries wide error bars compared to the 869 neutral examples.

[1]Amazon Mechanical Turk (MTurk) is a crowdsourcing platform where requesters post small tasks and workers complete them for small payments, typically a few cents each.

Tweets are short, informal, and noisy: hashtags, @-mentions, URLs, slang, emoji, and abbreviations are common. They are also context-dependent. The same words can carry different sentiments depending on:

- the event being discussed;
- who is being addressed; and
- whether the author is being ironic.

Tweets about a sports loss sometimes use “great” sarcastically—I found several such examples when reviewing the data. Others, directed at celebrities, use “love” sincerely. These properties make Twitter sentiment harder to classify reliably than product reviews or news text, where context is more stable and predictable. I reviewed a random sample of 10 training tweets during the exploratory phase. Many were ambiguous even on careful reading. One: a concert announcement in neutral descriptive language, labelled positive. This is defensible—the author approved of the event—but the text alone gives no obvious signal. Another expressed mild frustration indirectly—no swearing, no exclamation, just a brief complaint—and was labelled neutral. I would have called it negative. These borderline cases are prevalent in Twitter data, and I expect the classifier to struggle with them for the same reasons a human reader might pause.

3 Model

Given the pipeline’s modular feature assembly and the need for interpretable baselines, I chose to run these two classifiers,

- logistic regression, and
- a shallow decision tree,

across six feature setups and two task types. That makes four total runs. Both use the same scikit-learn pipeline, balanced training, and I check performance on the validation split. Figure 1 in the notebook shows the pipeline.

The pipeline operates as follows:

- TextCleaner normalises whitespace;
- SpacyPreprocessor tokenises each tweet using the `en_core_web_sm` spaCy model and caches the results in an SQLite feature store;
- FeatureUnion assembles the feature matrix from the active feature blocks; and
- the classifier receives the matrix and generates predictions.

Swapping feature configurations is straightforward—preprocessing remains constant across all runs.

The classifiers are configured as follows:

- logistic regression: `max_iter = 5000, random_state = 82171165;`
- decision tree: `max_depth = 3, random_state = 82171165.`

That shallow depth means no more than seven leaf nodes,

- it is forced to identify only discriminating features;
- the resulting model is small enough to visualise; and
- small enough to interpret directly.

I made the tree shallow on purpose. I wanted to understand what the model latches onto more than getting the highest score. The `max_depth=3` setting is as much a diagnostic tool as a classifier (see Section 5).

3.1 Feature configurations

Six feature configurations are tested. They are designed to progress from simple token-count representations through to richer, more linguistically informed features, so that each step in complexity can be assessed against the baseline.

Unigrams. A TokensVectorizer extracts count features for the top 200 unigrams, after removing stop words and punctuation, with lowercasing applied. This is the simplest bag-of-words representation—word order, negation, and context are all discarded. I expected this to perform poorly, and it does to some degree. Still, it gives me a floor — anything worse than this is a real problem.

Bigrams. The same vectorizer extracts the top 200 bigrams. Bigrams capture some local context—"not good" differs from "good"—but remain shallow and sensitive to data sparsity. Twitter language is varied. Most bigrams are rare. A bigram like "really love" may appear frequently, but "really enjoy" or "absolutely love" might appear only once. The top-200 constraint captures only the common patterns, which are not always sentiment-informative.

Unigrams with POS tags (uni+pos). A FeatureUnion combines the top 100 unigram counts (scaled) with POS tag sequence features from a POSVectorizer. The motivation is that grammatical structure may carry sentiment signal that vocabulary alone misses—intensifiers, negations, and adjective usage patterns may be partially captured here. **Text statistics (textstats).** A TextstatsTransformer extracts surface features—tweet length, punctuation counts, and similar properties—scaled with StandardScaler. Negative tweets tend to use more punctuation and exclamation marks. Short tweets are often reactive expressions. Longer ones tend toward neutral, more analytical content. I doubted textstats would perform well in isolation but expected it might contribute in combination with other features. I ran it alone mainly to see what surface properties could do on their own.

VADER lexicon (lexicon). A LexiconCountVectorizer counts matches against the VADER sentiment lexicon, scaled with StandardScaler. VADER was designed specifically for social media sentiment (Hutto and Gilbert, 2014):

- it encodes human judgements about which words are positive, negative, and neutral;
- it includes common social media expressions; and
- it includes emoticons and slang.

This is probably the most relevant feature set for this task. I expected it to perform well. The key question is whether the lexicon covers enough of the actual vocabulary in the tweet_eval dataset, which was collected later than the original VADER development set.

Embeddings. A Model2VecEmbedder generates vector representations of each tweet using a pre-trained embedding model. Embeddings capture semantic similarity in ways that sparse count features cannot—two tweets expressing the same sentiment in different words should produce similar vectors. Whether this generalises to short, noisy Twitter text is an open question. Pre-trained embeddings reflect the distributional properties of their training corpus; if that corpus is general-purpose text rather than social media, the embeddings may not reliably represent abbreviated Twitter language. Of everything I tested, the embeddings gave me the most to think about.

3.2 Task variants

The 3-class task (negative, neutral, positive) is the primary task. The binary task (negative vs positive only, with neutral tweets excluded) was more of a diagnostic run. My instinct was to start with the simpler binary task and then ask what happens when neutral is added back. Here's the idea: if performance jumps substantially in the binary setting, then the trouble is concentrated at the neutral boundary. If not, the mess is spread somewhere else. That's directly relevant to sub-question 4.

For the binary task, a separate, independently undersampled training and test split was constructed. The binary training set contains 14,186 tweets (7,093 per class), and the binary test set contains 1,131 tweets.

4 Results

The results are organised into four runs:

- RUN 3: binary $\times$ logistic regression;
- RUN 4: binary $\times$ decision tree;
- RUN 1: 3-class $\times$ logistic regression;

- RUN 2: 3-class × decision tree.

The binary runs are presented first to establish a cleaner baseline: without the neutral class, the classification boundary is simpler and the results are easier to interpret. The 3-class runs follow, where the complexity of neutral reveals the primary sources of difficulty. For each run, classification reports and confusion matrices are shown for all six feature configurations. A summary table and a 3-class vs binary comparison table are provided at the end of this section.

4.1 RUN 3 and RUN 4: binary task

In the binary task, neutral tweets were excluded and the classifier was asked to distinguish negative from positive only. If all the trouble was really concentrated at the neutral boundary, this should make things a lot easier. I expected a significant gain.

The gain table at the end of Section 4 shows how macro F1 changes from 3-class to binary for each feature configuration. A large positive gain for most configurations would confirm that neutral is the primary source of difficulty. A small or inconsistent gain would suggest that the negative–positive boundary is also problematic, possibly because of label inconsistency at both boundaries.

The decision tree acts differently in the binary setting. Without the neutral class as a majority-class anchor, the tree must find a split that distinguishes negative from positive. This could mean a more useful model—maybe recall balances out more. Or maybe it just shifts which class gets over-predicted. Depends on the features. This is one of the results I was most curious to see.

4.2 RUN 1: 3-class classification, logistic regression

The classification reports and confusion matrices for RUN 1 cover all six feature configurations. The unigram baseline achieved a macro F1 of 0.447, which serves as the reference point for all subsequent configurations. I was surprised that raw word counts performed this well—words like “love”, “hate”, “great”, and “terrible” appear in the top 200 and carry clear sentiment signal. The model is doing something real. For a random three-class guesser, macro F1 would be 0.333. We are well above that.

The unigram model cannot handle negation, context, or words that appear across multiple sentiment classes. Terms like “day”, “time”, and “people” appear frequently in both positive and negative tweets and receive near-zero weights. The model relies on a small set of vocabulary items reliably associated with a single class. This is acceptable as a baseline, but it leaves a great deal unexplained.

VADER lexicon was built for this job. It bakes in how people judge sentiment words and grabs social-media slang you won’t find in vocabulary lists. I expected it to outperform the unigram baseline, and the results are worth looking at closely when assessing which feature type captures useful signal. If it only boosts negative precision, maybe it’s just good for spotting strong negative language and not much else.

Embeddings are supposed to be semantically rich. In theory, they grab meaning beyond just counting words. But do they actually do better on noisy, short tweets? That’s a real

question, and the answer from this dataset says a lot about the limits of general-purpose semantic representations. Figure 1 shows the confusion matrix for the best LR result.

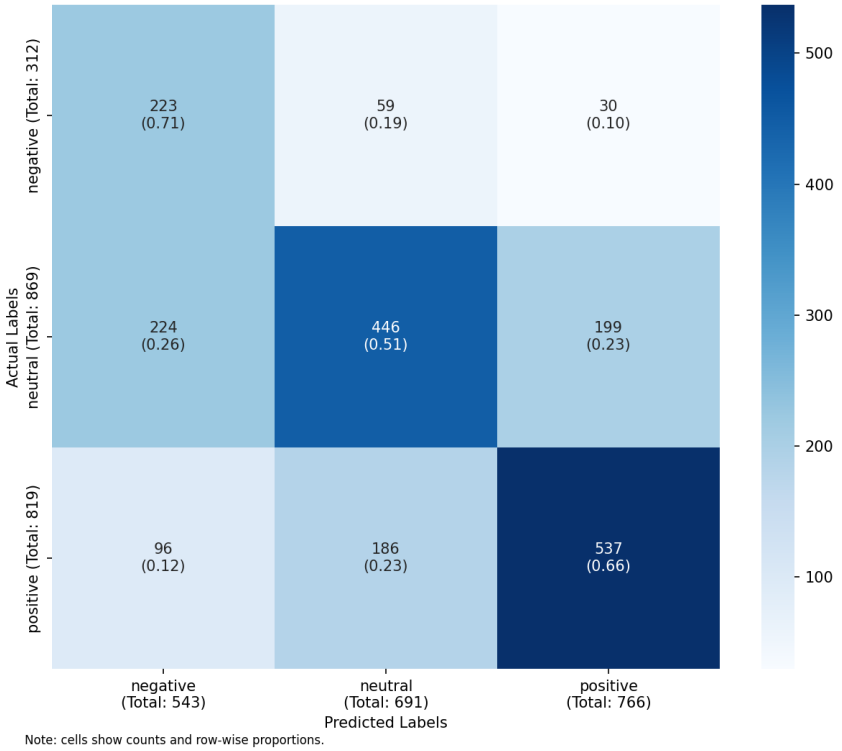

Figure 1: Confusion matrix: 3-class logistic regression with embeddings (macro F1 = 0.590). Errors spread realistically across all three classes — the model is uncertain, not just defaulting to neutral.

4.3 RUN 2: 3-class classification, decision tree

The decision tree performed substantially worse than logistic regression across most feature configurations. With unigram features and max_depth=3, its macro F1 was 0.277—barely above the 0.333 floor of random chance. It predominantly predicts neutral for most inputs. Recall for neutral approaches 1.0, while recall for negative and positive is near zero. This is consistent with shallow tree behaviour: "predict the most common class" is the least-bad rule when only three splits are available.

But it's not just weaker—it fails in a totally different way. Logistic regression spreads its mistakes around. The tree just gives up and uses a trivial heuristic. Three levels deep isn't enough for anything subtle. Maybe the features just don't offer clean single-variable splits, either. Sentiment isn't a simple threshold on one word count.

Accuracy for the tree? 0.474. Logistic regression? 0.470. Practically the same. But this is exactly why accuracy is the wrong metric for imbalanced multi-class problems. You can overwhelmingly predict the majority class and look fine on paper, but you're useless for the others. Macro F1 catches this; accuracy hides it. Running both models side by side on this dataset taught me more than any textbook example. Figure 2 shows the extent of this collapse.

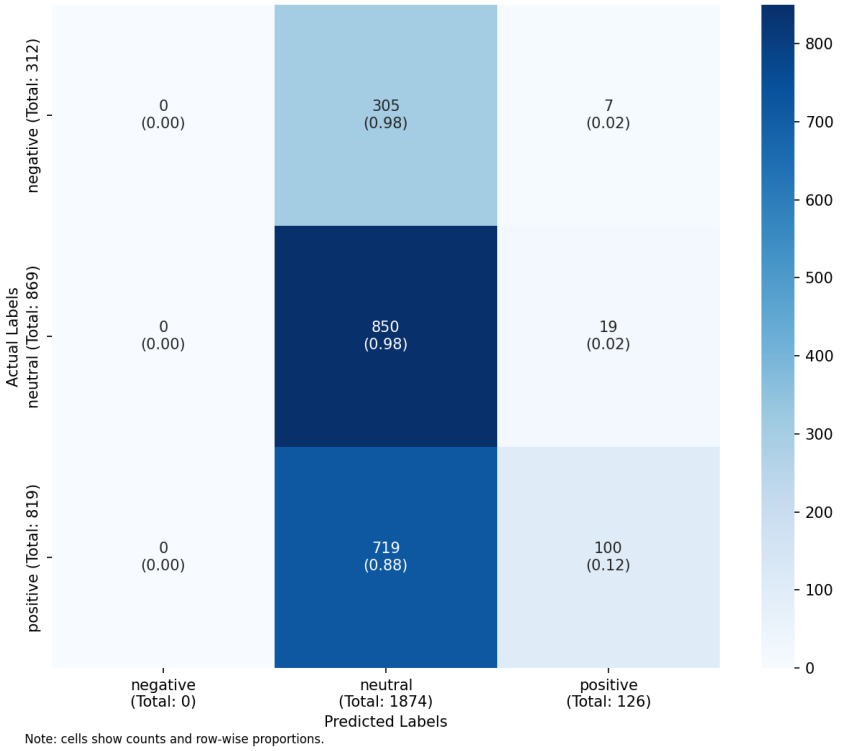

Figure 2: Confusion matrix: 3-class decision tree with unigrams (macro F1 = 0.277). The tree predicts neutral for almost every input — recall for negative and positive is near zero.

4.4 3-class vs binary: gain summary

Table 1: Macro F1 for 3-class and binary tasks, and gain from removing neutral.

	Logistic Regression			Decision Tree		
Feature set	3-class	binary	gain	3-class	binary	gain
unigrams	0.466	0.667	+0.201	0.277	0.521	+0.244
bigrams	0.335	0.525	+0.190	0.232	0.448	+0.217
uni+pos	0.472	0.650	+0.178	0.327	0.561	+0.234
textstats	0.387	0.540	+0.154	0.369	0.553	+0.184
lexicon	0.540	0.639	+0.099	0.539	0.638	+0.098
embeddings	0.590	0.806	+0.216	0.419	0.614	+0.195

5 Discussion

5.1 Overall performance

The logistic regression baseline achieved a macro F1 of 0.447 on the 3-class task. A score below 0.5 is not strong, but it is not meaningless . With the level of annotation noise present and the known difficulty of the neutral class, a macro F1 in the 0.4–0.5

range is a credible baseline. A random classifier on a balanced three-class problem would achieve macro F1 of 0.333. A model that always predicts the majority class gets high recall for one class but near-zero for the rest—macro F1 drops below 0.333. So 0.447 from logistic regression is comfortably above both floors. The real question isn't whether the classifier works at all. It's why it doesn't work better, and what that says about the data.

The decision tree's macro F1 was 0.277—tough to defend. Accuracy? 0.474, basically the same as logistic regression (0.470). That just shows how useless accuracy is here. With 43.5% neutral in the test set, a model can just say "neutral" for every input and score 43.5% accuracy without learning anything. The tree isn't quite that bad, but it's close. Macro F1 exposes this; accuracy conceals it.

5.2 Feature comparison

The six feature configurations produced different results, and a few patterns stood out. I started with unigrams as the baseline, from the lab work, Bigrams seemed like a natural extension, though I suspected Twitter's noisy, varied vocabulary might limit how much local context they actually captured. Adding POS tags felt like it would help by introducing grammatical structure. Text statistics were the most speculative—things like tweet length and punctuation density felt like blunt proxies for emotional intensity, and I wasn't confident they'd contribute much on their own. VADER I felt better about. Embeddings I was less certain about going in.

As I said, I thought the VADER lexicon might do the best. It was designed for social media, encodes human judgements about sentiment, and incorporates slang, emoticons, and expressions that general vocabulary counts miss. Hutto and Gilbert (2014) showed it worked well on Twitter data, and this task felt very close to what they evaluated it on.

If the embeddings came from generic text rather than social media, I expected they'd struggle with the informal shorthand. A tweet like "can't even rn" expresses frustration, but I wasn't confident a general-purpose embedding model would reliably encode it as negative.

I wasn't sure whether text statistics would contribute anything useful. Tweet length, punctuation count, capitalisation patterns—these feel like blunt proxies for emotional intensity, and I wasn't confident there was real signal there. I figured short tweets are often reactive, and heavy punctuation might signal strong emotion—but that's a guess. I ran them as a standalone configuration mainly to see what they'd actually do on their own.

One pattern I noticed when comparing feature configurations: the per-class ordering remains stable. In logistic regression, per-class F1 almost always follows negative $<$ positive $<$ neutral, even as the absolute values vary. I believe this suggests the relative difficulty of each class is a property of the data rather than any particular feature representation. The negative class is harder to classify regardless of the features used—that is a data observation, not a modelling one.

5.3 What the binary task reveals

Sub-question 4 is really asking: was neutral the problem all along, or is the trouble more general?

If dropping neutral produces a large, consistent improvement across classifiers and feature types, that would suggest neutral was the main source of difficulty—a middle category that's hard to separate from mild negative and mild positive expression. That would fit with what I'd read about the SemEval annotation process, where neutral was the default label for tweets that didn't clearly go one way or the other.

But if the gain is small or inconsistent, it suggests the difficulty isn't just at the neutral boundary. Maybe the negative–positive boundary is also messy, or the features just aren't rich enough to resolve it regardless. I found this the more interesting possibility, honestly.

Table 1 shows macro F1 for both setups and the gain from dropping neutral. I found that view more useful than raw F1 figures—it speaks directly to where the difficulty is concentrated rather than just how well the model did overall. Big, consistent gains would suggest neutral is mostly to blame. Small or variable gains would suggest the problem is spread more evenly.

The binary task uses a different, smaller test set—the validation set with all neutral tweets removed. So some of any F1 gain might just reflect the changed composition rather than the task being easier. I don't think it reverses the picture, but it complicates a clean comparison.

5.4 Class-level analysis

The 3-class LR results are not balanced across classes. Positive tweets got the best F1, with high precision but only middling recall—the model doesn't call many tweets positive, but when it does, it's usually right. negative is the worst performer: low precision, moderate recall. The model tags too many tweets as negative, and most of those turn out to be neutral or positive. Neutral sits in the middle.

I didn't expect neutral to beat negative, but it did. The conventional view—supported by the course notes and by Kenyon-Dean et al. (2018)—is that neutral is the hardest class because it lacks distinctive lexical markers. The results don't clearly support this. Neutral F1 exceeds negative F1. Maybe that's because neutral is the largest group in the test set, giving the model more evaluation signal. Or maybe neutral tweets cluster around non-sentiment-loaded vocabulary that shows up well in the top-200 unigrams. negative might just be more varied and context-dependent language-wise.

The decision tree collapse is a separate phenomenon. It's not just weak on negative and positive—it's nearly zero on both. The tree has apparently learned that predicting neutral minimises loss on the training data even after undersampling. That bothers me. It suggests the feature representations don't yield a clean split between neutral and the other classes at any single threshold.

5.5 Label quality evidence

The thing I kept returning to is that a low macro F1 doesn't tell you much on its own. It could be the features, the model, the labels, or all three. All three are probably true to some degree here, and I genuinely cannot tell which dominates.

negative stands out. Low precision means the model generates many false positive predictions for negative sentiment. If the model predicts negative based on reasonable lexical cues but the gold label is neutral, this suggests annotators labelled borderline tweets as neutral rather than negative. The SemEval annotation guidelines used neutral as the default for tweets that did not clearly express positive or negative sentiment. The neutral label may have introduced systematic ambiguity in this dataset (Kenyon-Dean et al., 2018):

- the category was designed to capture genuine neutrality alongside mild negative expression;
- annotators were uncertain about tweets that fell between the two;
- sentiment is a property of the author's relationship to content, not just of the words used.

A phrase like "not bad at all" could be positive or ironic depending on context. Unigram count features, which discard word order, negation scope, and context, are poorly equipped to capture these distinctions. But sometimes, the trouble isn't the model's fault—it's just ambiguous:

- even human annotators would disagree on these cases;
- the low classifier performance may be tracking that annotator disagreement;
- rather than merely reflecting model inadequacy.

I kept coming back to the negative precision. It's low, but not in a random way. The model and the annotators seem to agree that certain vocabulary is negative, but their labels keep diverging:

- the model predicts negative, and in many cases the tweet probably is negative in some sense;
- the annotators, however, labelled it neutral;
- possibly because the expression was not strong enough to trigger the positive or negative label.

If I had access to per-annotator votes rather than only the majority-vote outcome, I could check whether these tweets had high disagreement rates. Not having that is a genuine limitation.

5.6 Misclassification analysis

I ran the misclassification breakdown in notebook section 4.2. Looking at individual cases where the model went wrong felt more useful than just staring at the summary F1.

Some misclassifications are model failures. A tweet containing strongly negative language predicted as neutral is likely a case where the relevant tokens fell outside the top-200 vocabulary, or where negation flipped the meaning of otherwise neutral words. These are feature engineering problems.

Other misclassifications are more interesting. When I checked tweets the model called negative but the label was neutral, a recurring pattern came up:

- tweets expressing mild dissatisfaction, slight frustration, or low-key complaint;
- the kind of expression that sits at the boundary between neutral and negative in everyday language.

I'd expect human annotators to disagree on these. The model isn't clearly wrong. The label isn't clearly right. That ambiguity is the point.

Across all feature configurations, the tweets the model consistently misses are the sarcastic, ironic, or context-dependent ones. "Great job guys" directed at a losing team could mean anything. No unigram or bigram feature set can resolve this. A human with knowledge of the team and the result would get it immediately. The classifier has only the words. That's a hard ceiling for surface-form features, and it's a real constraint.

I don't think that's the model's fault, exactly. That just seems to be what this problem is. Did annotators have the context they needed when labelling? If they saw tweets in isolation—no thread, no event background—they probably defaulted to neutral when unsure. That would explain a lot of the patterns observed here.

5.7 Limitations and future directions

There are things I couldn't do, or wish I'd done differently.

The feature configurations tested here are all applied independently. No multi-feature combinations are evaluated. A combined model may outperform any single configuration:

- for example, unigrams plus VADER lexicon plus text statistics;
- the interaction effects between feature types are not captured here.

Mixing features is the obvious next step, and the pipeline architecture supports it directly through FeatureUnion. The cost is missing those interactions; the benefit is a clean additive comparison of what each feature type contributes.

I set max_depth=3 deliberately — I wanted to see the actual splits, not just get the highest score. A deeper tree would likely do better on F1, but I'd lose the ability to read it. There's a real risk of overfitting too, given the training set size.

The embeddings used here are from a pre-trained general-purpose model. Fine-tuning on Twitter data, or using something like BERTweet (pre-trained on 850 million tweets), would likely produce more useful tweet embeddings. General models just don't always capture the quirks of short text.

The analysis is constrained to the validation split. The held-out test split (12,284 tweets) has not been used and would provide a more reliable estimate of generalisation performance. The validation set has only 312 negative examples, which limits the precision of per-class estimates for that class. Those wide confidence intervals mean negative F1 should be taken as indicative rather than definitive.

The labels themselves are uncertain in ways I can't fix. They're majority-vote outcomes, not ground truth. Where annotators disagreed, the label is uncertain, and classifier performance on those cases is not a reliable measure of model quality. If I had inter-annotator agreement scores, I could separate the hard cases from the clearly labelled ones. Without that, I cannot distinguish "model failure" from "contested annotation." That is probably the most valuable extension of this analysis.

References

Barbieri, F., Camacho-Collados, J., Espinosa-Anke, L., and Noves, L. (2020). TweetEval: Unified benchmark and comparative evaluation for tweet classification. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1644–1650. Association for Computational Linguistics.

Horn, R. E. (1989). *Mapping Hypertext: The Analysis, Organization, and Display of Knowledge for the Next Generation of On-Line Text and Graphics*. Lexington Institute, Lexington, MA.

Hutto, C. J. and Gilbert, E. (2014). VADER: A parsimonious rule-based model for sentiment analysis of social media text. In *Proceedings of the Eighth International Conference on Weblogs and Social Media (ICWSM 2014)*. AAAI Press.

Kenyon-Dean, K., Ahmed, E., Bhosale, S., Brooks, B., Laframboise, C., Roper, F., Singh, S., Cheung, J. C. K., and Precup, D. (2018). Sentiment analysis: It's complicated! In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1886–1895. Association for Computational Linguistics.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63(2):81–97.

Nielsen, J. (1997). How users read on the web. Nielsen Norman Group.

Redish, J. C. (1985). The plain English movement. In Greenbaum, S., editor, *The English Language Today*. Pergamon Press, Oxford.

Rosenthal, S., Farra, N., and Nakov, P. (2017). SemEval-2017 task 4: Sentiment analysis in twitter. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 502–518, Vancouver, Canada. Association for Computational Linguistics.

A How to Run the Submission Package

The notebook can be run manually in the usual way, but I've spent some time automating this. The main reason is that the notebook uses relative paths and creates sibling folders during execution — so where you open it matters. Here is what I'd recommend.

This submission is distributed as a zip archive. When unpacking, keep the top-level folder `FINAL-PROJECT/` intact. Open the notebook from inside `FINAL-PROJECT/python/` — do not move it elsewhere.

For one-line execution after extraction, run `python run_submission.py` from the package root. This launcher executes the notebook via Jupyter `nbconvert`.

This is important because the notebook uses relative paths and expects enough headroom under `FINAL-PROJECT/` to create and use sibling folders such as `data/`, `figs/`, `results/`, and `report/`. If the folder structure is preserved, no extra setup instructions should be needed by the user. After the first run has created these working folders, the notebook can also be run manually in the usual way, provided it remains in the same `FINAL-PROJECT/python/` location.

```
python -m zipfile -e FINAL-PROJECT-SUBMISSION.zip .
python run_submission.py
```

B All Confusion Matrices

The figures below show confusion matrices for every feature configuration across all four runs. RUN 1 and RUN 2 are 3-class (negative, neutral, positive); RUN 3 and RUN 4 are binary (negative vs positive). Each matrix is normalised by true label so the diagonal shows recall per class.

RUN 1 — 3-class, Logistic Regression

RUN 2 — 3-class, Decision Tree

RUN 3 — binary, Logistic Regression

RUN 4 — binary, Decision Tree

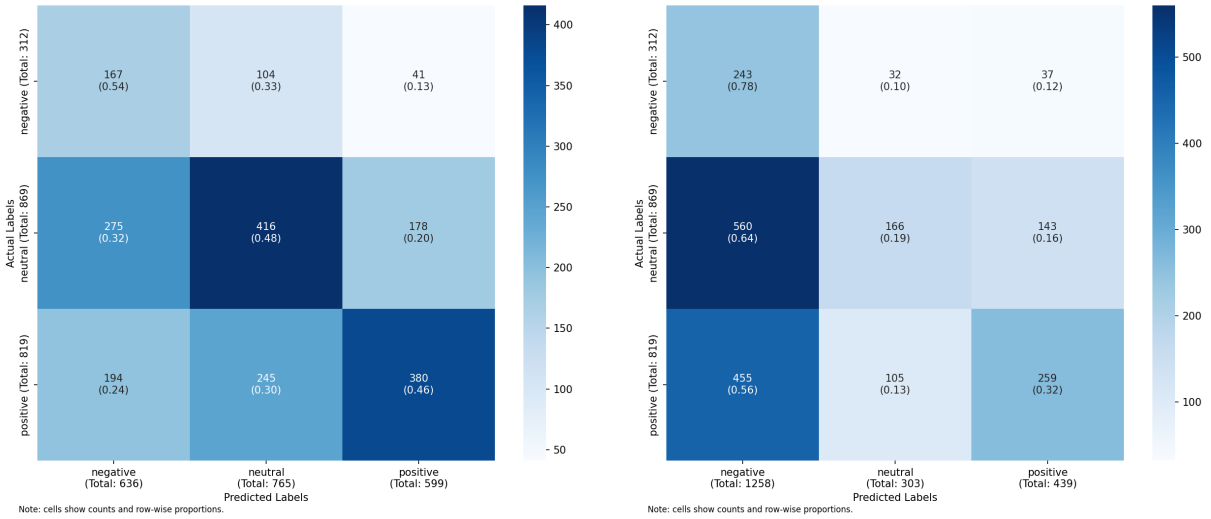

Figure 3: RUN 1 LR: unigrams (left), bigrams (right)

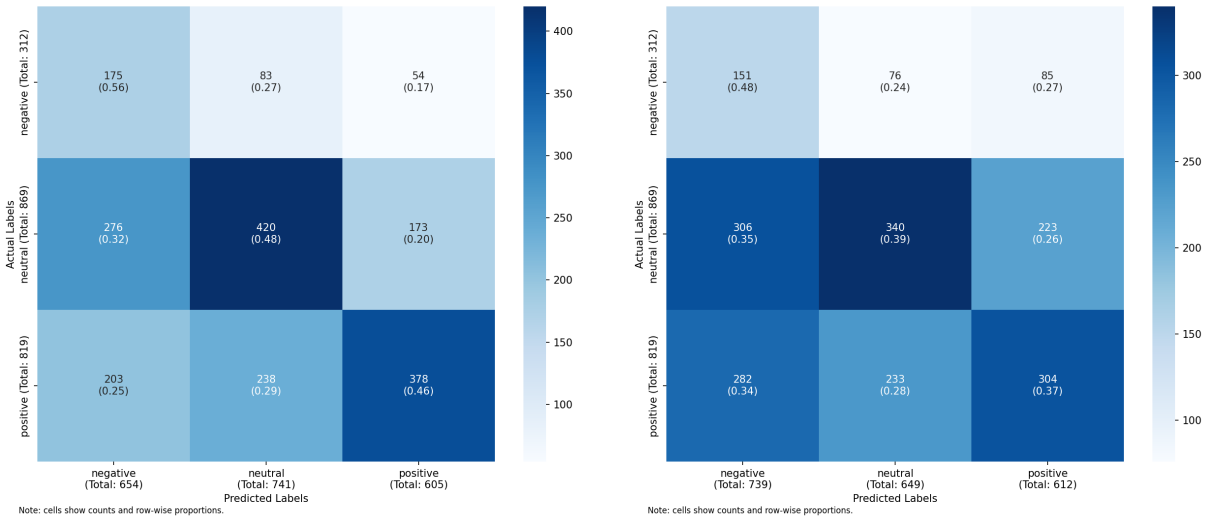

Figure 4: RUN 1 LR: unigrams+POS (left), text statistics (right)

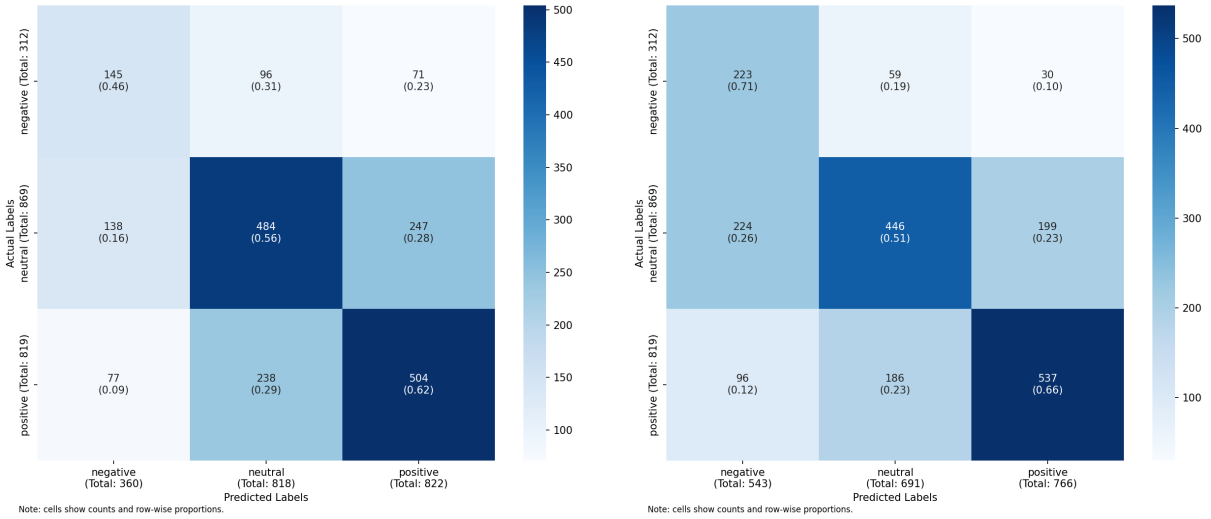

Figure 5: RUN 1 LR: lexicon (left), embeddings (right)

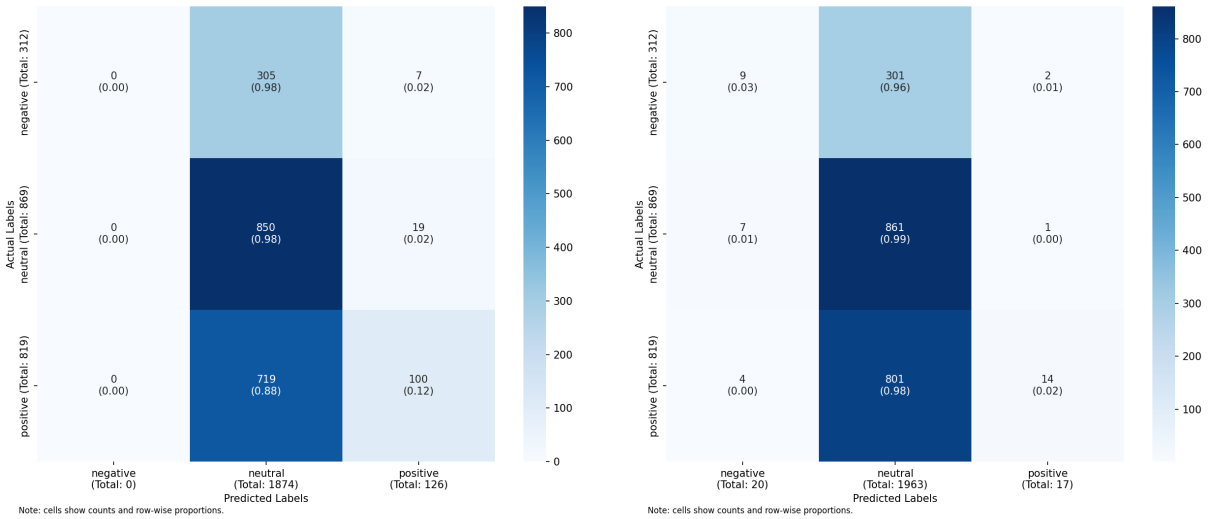

Figure 6: RUN 2 DT: unigrams (left), bigrams (right)

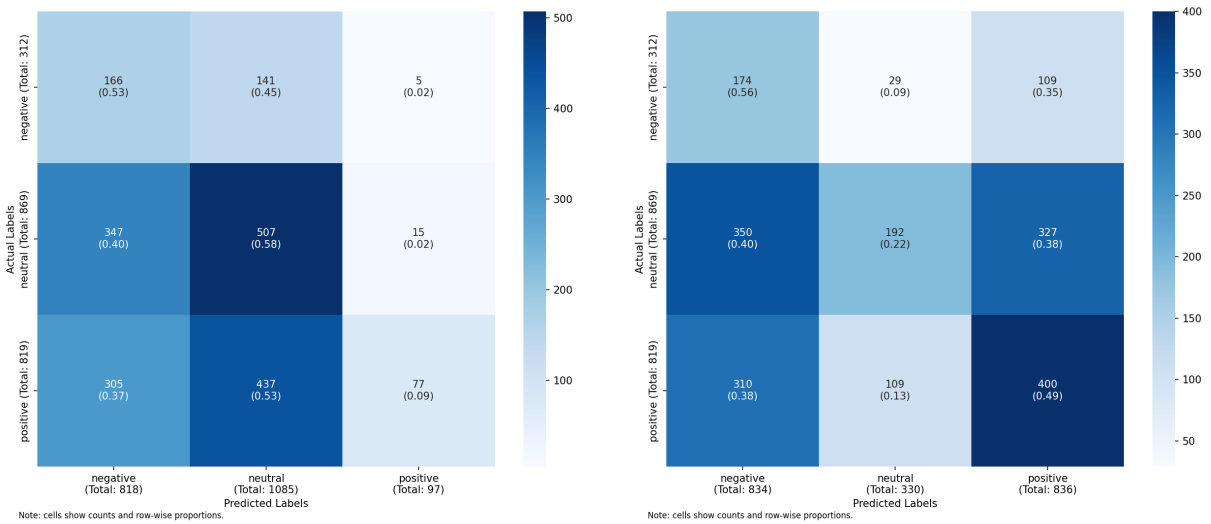

Figure 7: RUN 2 DT: unigrams+POS (left), text statistics (right)

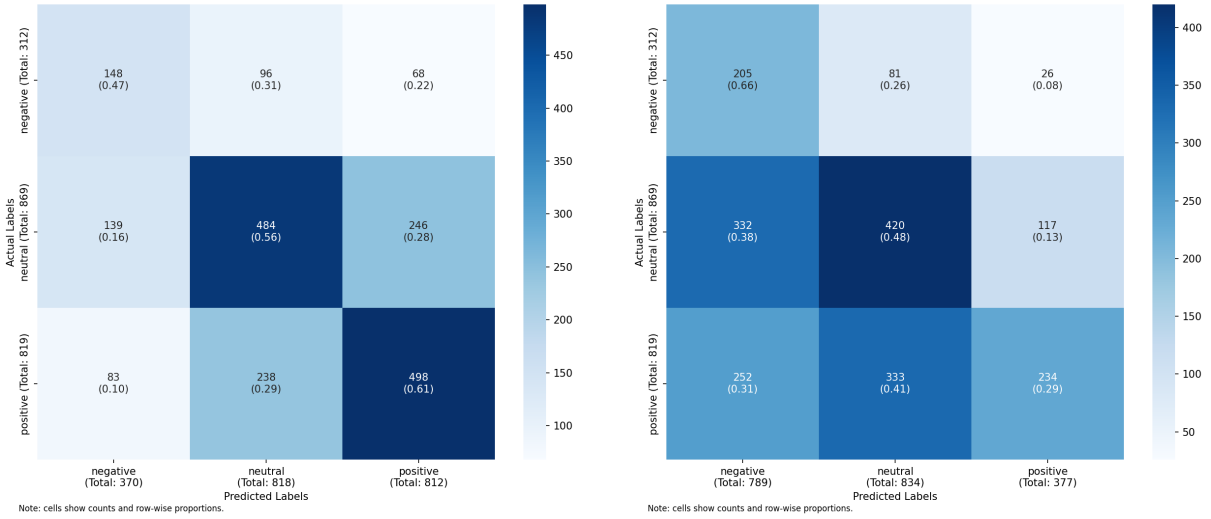

Figure 8: RUN 2 DT: lexicon (left), embeddings (right)

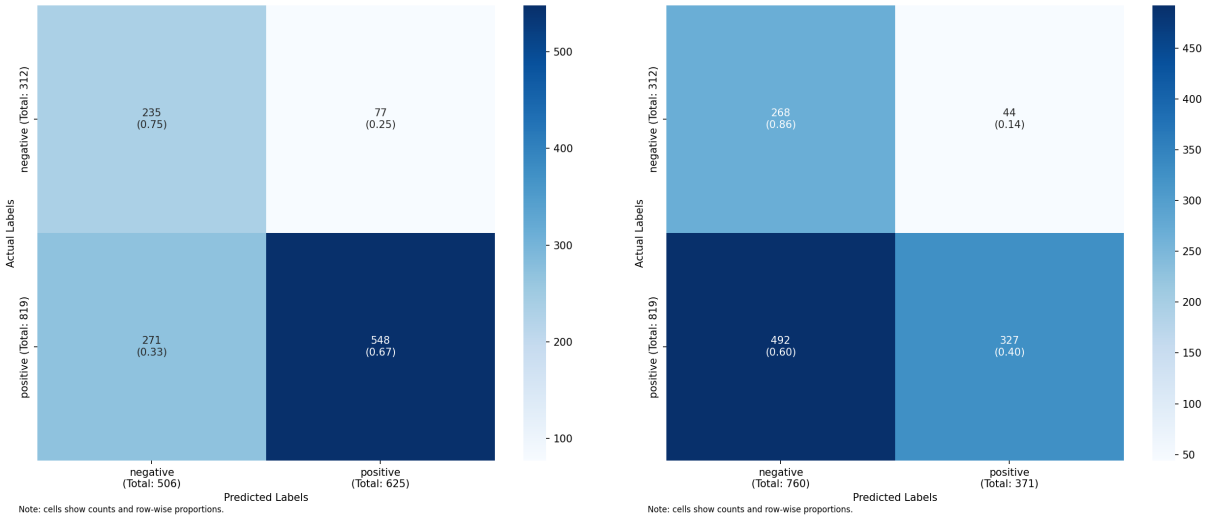

Figure 9: RUN 3 LR: unigrams (left), bigrams (right)

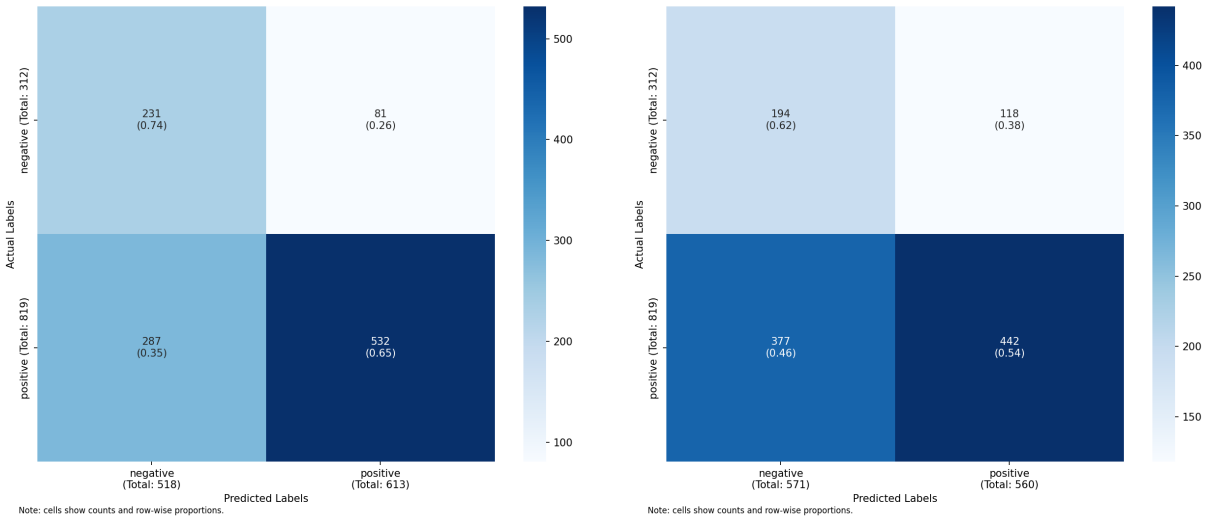

Figure 10: RUN 3 LR: unigrams+POS (left), text statistics (right)

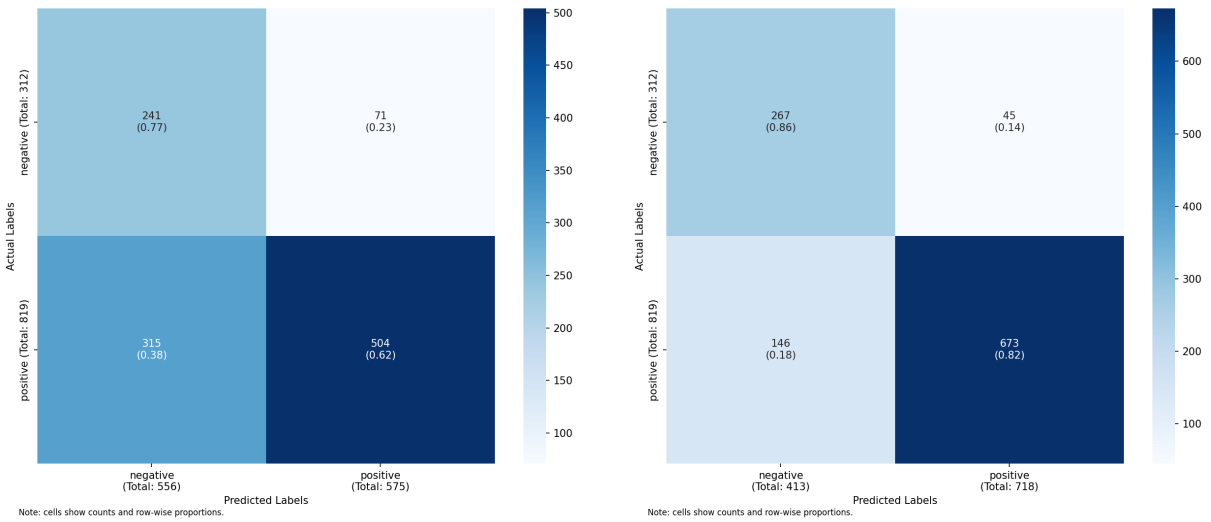

Figure 11: RUN 3 LR: lexicon (left), embeddings (right)

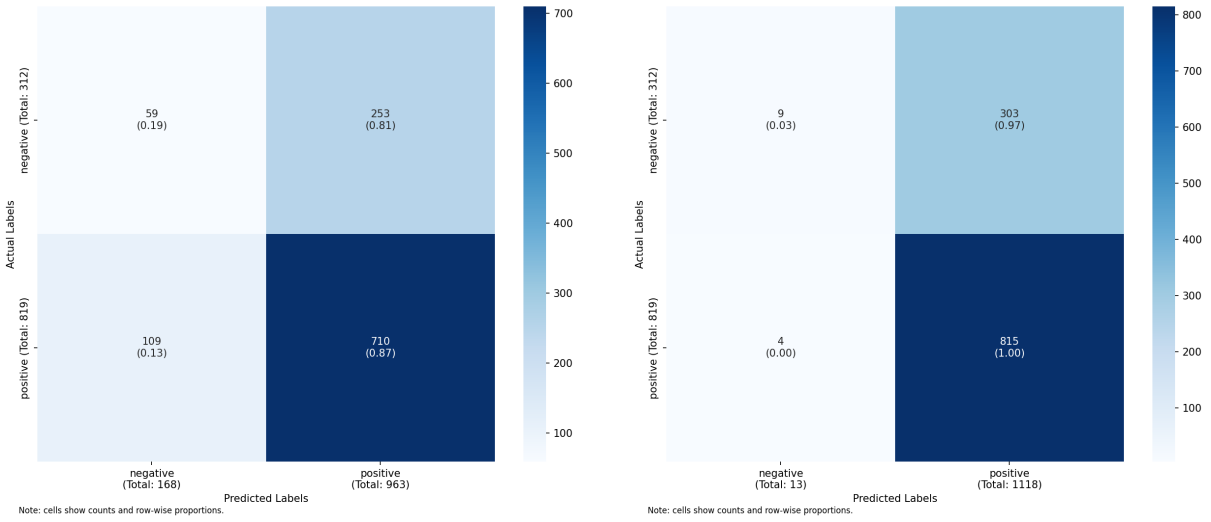

Figure 12: RUN 4 DT: unigrams (left), bigrams (right)

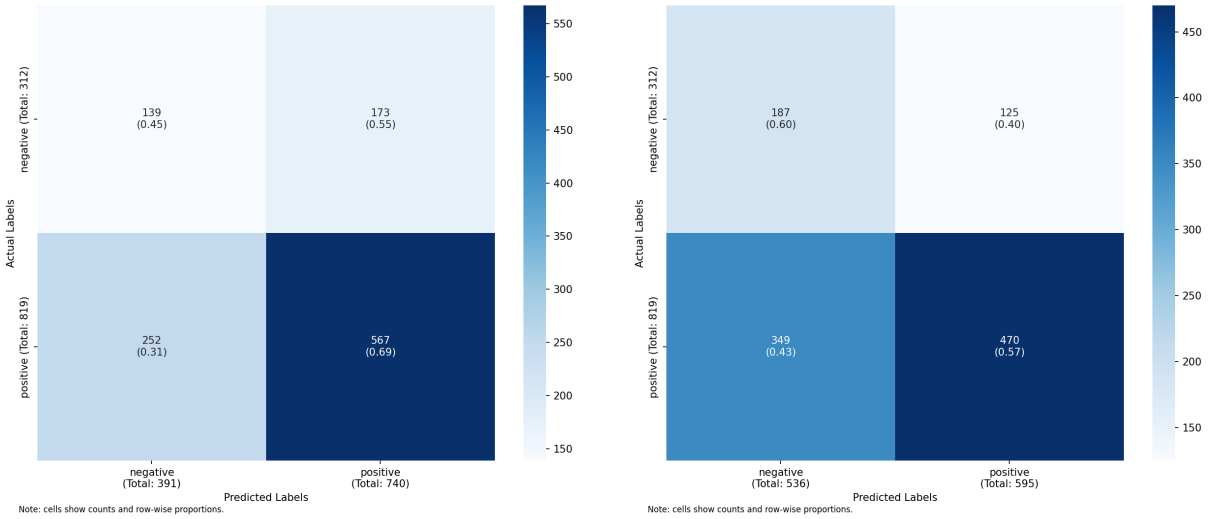

Figure 13: RUN 4 DT: unigrams+POS (left), text statistics (right)

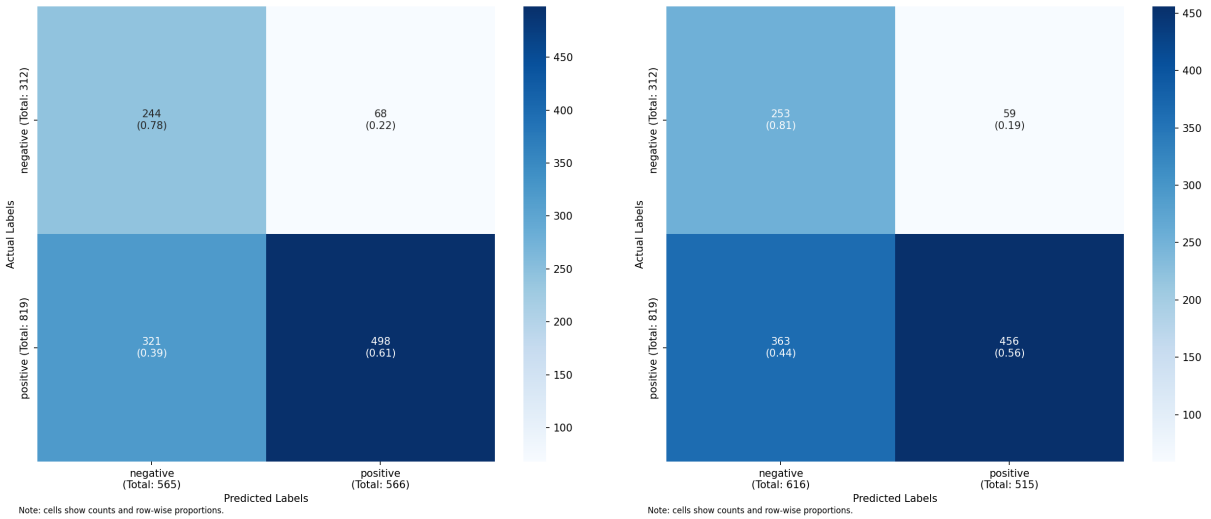

Figure 14: RUN 4 DT: lexicon (left), embeddings (right)