

Group 8 Progress Report: Emoji Prediction 😂

Zifan (Frank) Si, Xianzhao (Paul) Duan
`{siz,duanx15}@mcmaster.ca`

1 Introduction

People use emojis to show tone and meaning in short messages. A single emoji can say “love” , “wow” , “funny” , or “holiday” . But picking the right emoji is slow: you have to search through many icons or accept suggestions that do not match your message. Our goal is to make emoji choice fast and helpful by **predicting** a small, ranked list of emojis that fit the text.

In this project, we build a simple and clear system that works end to end. First, we load a large set of short, social-media style messages with one emoji label each. Next, we clean the text (fix broken characters, lower-case, and optionally remove links, mentions, and hashtags). Then we turn each message into basic text features using TF-IDF on words and short phrases (unigrams and bigrams). Finally, we use two easy-to-understand methods to suggest emojis: (1) a *keyword* method that looks for the top words linked to each emoji, and (2) a *cossine* method that compares a message to an average example (“prototype”) of each emoji.

We measure success in two ways. First, we check **top-1 accuracy** (how often our first guess is correct). Second, we check **top- k accuracy** (whether the correct emoji is in the first k suggestions), which is more useful for a typing assistant. Early results match our expectations: emojis tied to clear topics, like  or , are easier to predict; emojis with similar feelings (for example, different hearts and smiles) are harder, but still appear in the top few suggestions.

This work follows our proposal: keep the system simple, fast, and easy to explain, while establishing strong baselines. Next, we plan to keep hashtag and mention *words* (strip only #/@), grow the TF-IDF vocabulary, and add

a basic trained classifier (such as logistic regression). These steps should improve top-1 accuracy while keeping the model small, transparent, and ready to plug into real chat apps.

2 Related Work

Our approach builds on simple and well-known ideas from text classification. Classic work showed that bag-of-words with TF-IDF weighting is a strong baseline for short texts (Salton and Buckley, 1988). Linear models, such as support vector machines (SVMs), have long been effective on these features (Joachims, 1998), and modern toolkits make them easy to use and compare (Pedregosa et al., 2011).

To get strong baselines with little engineering, fastText introduced lightweight text classifiers that use word/ n -gram features and train very quickly while staying competitive with heavier models (Joulin et al., 2017). Another useful idea is to compare texts and classes in a vector space: word and document vectors help measure similarity via cosine distance (Mikolov et al., 2013). In our work, we apply the same idea to TF-IDF by building a simple “prototype” vector for each emoji and ranking classes by cosine similarity.

Closest to our task is the SemEval 2018 shared task on emoji prediction, which provided datasets and baselines for mapping social media text to emojis (Barbieri et al., 2018). Their findings match our setup: careful tokenization, keeping informative tokens (e.g., hashtags), and using larger vocabularies can noticeably improve results—even for simple linear or prototype-style methods. Based on that, we use unigram and bigram TF-IDF, report top- k accuracy (useful for suggestions), and plan ablations on preprocessing choices (e.g., keeping hashtag/mention *words*).

In short, prior work suggests three practical lessons we follow here: (i) TF-IDF with linear or prototype scoring is a strong, transparent starting point (Salton and Buckley, 1988; Joachims, 1998; Pedregosa et al., 2011); (ii) small modeling choices (subword/ n -grams, vocabulary size) matter (Joulin et al., 2017); and (iii) for emoji prediction specifically, dataset handling and evaluation with ranked outputs are important (Barbieri et al., 2018).

3 Dataset

3.1 Files and labels

We use a 20-class emoji dataset of short, social-media style messages. All files are in data/:

- **Train.csv:** 70,000 rows with TEXT and Label.
- **Test.csv:** 25,958 rows with TEXT (no labels).
- **Mapping.csv:** maps label numbers 0 … 19 to emojis (e.g., 5 → , 7 → , 9 → , 11 → ).
- **OutputFormat.csv:** submission template (id,Label); not used for training.

In code, src/dataset.py loads these files. We convert labels to emojis only when printing examples or figures (via src/utils.py); internally we keep labels as integers.

3.2 Text preprocessing

We apply light, configurable cleaning to keep the text realistic but easier to model:

- **Unicode fixes:** repair broken characters (ftfy).
- **Normalize text:** lowercase, single-line (clean-text).
- **Social markup (optional):** remove URLs, @mentions, and #hashtags using a Twitter preprocessor. These flags can be turned on/off for ablations.
- **Keep punctuation and numbers:** repeated marks (e.g., “!!”) and digits can carry tone or event hints.

Example effects:

- “Merry Christmas!! #christmas” → “merry christmas !!” (if hashtags removed), which still helps detect .

- “Link in bio ” stays readable and useful for .

For speed, we cache cleaned copies to data/cleaned/; later runs load from cache.

3.3 Train/validation split

We split **Train.csv** into 80% train and 20% hold-out with a fixed random seed and stratify to preserve class ratios. We fit the TF-IDF vectorizer on the train split only, then transform the hold-out. This avoids leakage. We report top-1 and top- k accuracy, and macro/weighted F1 on the hold-out.

3.4 Feature input to models

After cleaning, each message is turned into a sparse TF-IDF vector with unigrams and bigrams. These vectors feed both of our simple baselines:

- **Bag-of-Words Weighted Classifier:** sums TF-IDF weights at top tokens per class.
- **Nearest Centroid Classifier:** computes one average vector (“prototype”) per class and ranks by cosine similarity to each class.

Using one shared vectorizer makes results comparable and reproducible.

3.5 Class balance and observations

The dataset is imbalanced (the largest class is about 22% of the split). This makes top-1 accuracy for trivial baselines (majority class) fairly high. It also lowers macro-F1 and can bias simple statistics toward frequent classes. We therefore track both macro and weighted metrics, and we look at top- k accuracy because the user sees a short list of suggested emojis.

3.6 Limitations and care

Texts may contain informal language or cultural references. Removing hashtags/mentions can drop useful signals (e.g., #christmas, #usa), so we plan ablations that *keep the words* but strip only the “#” and “@” characters. We also plan to grow the TF-IDF vocabulary and

Label	Emoji	CLDR name (for \emoji)
0	😊	winking-face-with-tongue
1	📸	camera-with-flash
2	😍	smiling-face-with-heart-eyes
3	😂	face-with-tears-of-joy
4	😉	winking-face
5	🎄	christmas-tree
6	📷	camera
7	🔥	fire
8	😘	face-blowing-a-kiss
9	❤️	red-heart
10	🤗	beaming-face-with-smiling-eyes
11	🇺🇸	flag-united-states
12	☀️	sun
13	✨	sparkles
14	💙	blue-heart
15	💕	two-hearts
16	😎	smiling-face-with-sunglasses
17	😊	smiling-face-with-smiling-eyes
18	💜	purple-heart
19	💯	hundred-points

Table 1: Label \leftrightarrow emoji mapping used in experiments.

adjust frequency thresholds to capture informative phrases without adding too much noise.

3.7 Reproducibility

Paths are project-relative, random seeds are fixed, and preprocessing settings are logged. We avoid fitting on the hold-out, and we save tables/figures to a results folder so the report can be rebuilt without rerunning every step.

4 Features

Goal. Turn each message into a simple numeric vector that models can read, keep it fast and easy to explain, and reuse the same features across different baselines.

Text to vectors (TF-IDF). We use a standard bag-of-words with TF-IDF weighting. Words that are common in a class but not common everywhere get higher weight. We include unigrams and bigrams (single words and two-word phrases) so short phrases like “merry christmas” are captured. Vectors are L_2 normalized so cosine similarity is meaningful. *Examples:* “merry christmas” \rightarrow ; “good morning” \rightarrow ; “link in bio” (often used in promos) \rightarrow .

Fitting once, reusing everywhere. We fit the TF-IDF vocabulary and IDF values on the training split only, then transform the hold-out

with the same settings. This avoids leakage and makes results comparable across models.

Feature limits and cleanup. To reduce noise and save memory, we remove English stopwords, drop very rare terms, cap the vocabulary size, and keep punctuation/digits in the raw text because they can carry tone (e.g., “!!”). We will ablate whether to keep hashtag and mention *words* (strip only “#”/“@”) since they are often informative. *Examples:* “#christmas” \rightarrow “christmas” (); “#usa” \rightarrow “usa” (); “@user thanks!!” keeps “thanks!!” (/ cues).

Feature selection for the Bag-of-Words

Weighted Classifier baseline. For interpretability, we build a small list of top tokens per class (e.g., the 10–20 highest-scoring terms within that class). However, at the prediction time, we still use the full TF-IDF vector to compute the weights at those token positions. This keeps the method simple and fast while allowing easy explanation of which words mattered. *Examples:* words like “merry”, “christmas”, “tree” boost ; words like “usa”, “4th”, “july” boost .

Prototype features for Nearest Centroid

Classifier scoring. Using the same TF-IDF space, we compute one average vector (“prototype”) per class. A test message is compared to all prototypes with cosine similarity, which yields a ranked list of labels. This uses all features and is less sensitive to message length. *Examples:* “sunny beach day with family” is close to the prototype; “shooting with my new lens downtown” is close to / ; “this is lit — link in bio” is close to .

What we do not use (yet). We do not learn embeddings and we do not train a neural network in this phase. Keeping TF-IDF makes the system small, fast, and easy to debug. As a next step, we will try a simple trained linear model (e.g., logistic regression) on the same TF-IDF features to lift top-1 accuracy while preserving interpretability.

Why these choices. Short, noisy texts benefit from n-grams and simple weighting. TF-IDF remains a strong, transparent baseline; keyword lists aid explanation; cosine prototypes provide better rankings. Together, they

give a solid feature foundation for both current baselines and future models.

5 Implementation

Overview. Our system is a small pipeline with clear steps. Each step has one job and a simple input/output. This makes the project easy to run, debug, and improve. The four steps are: (1) load and clean data, (2) build text features, (3) score emojis, and (4) evaluate and save results.

Step 1: Data loading and cleaning. We read the CSV files for train, test, and the label–emoji mapping. We clean the text in a light way: fix broken characters, lowercase, remove extra spaces, and (optionally) remove URLs, @mentions, and #hashtags. We cache the cleaned data so repeated runs are fast. Output: a table with TEXT and Label ready for models. *Example:* “merry christmas!! #xmas” ⇒ “merry christmas !!” (🎄).

Step 2: Text features (TF-IDF). We turn each message into a sparse TF-IDF vector using unigrams and bigrams. We fit the TF-IDF vocabulary on the training split only, then apply the same settings to the hold-out. Vectors are averaged and L_2 normalized so cosine similarity is meaningful. Using one shared vectorizer keeps all experiments consistent. *Examples of helpful phrases:* “merry christmas” (🎄), “link in bio” (🔥), “good morning” (☀️).

Step 3: Baseline scoring methods. We use two simple, fast, and explainable scorers that run on the same TF-IDF features.

- **Bag-of-Words Weighted Classifier:** For each emoji class, we keep a short list of top tokens. A message is scored for that class by summing TF-IDF weights of all tokens. The class with the largest sum is the top-1 prediction; all sums give a ranked list for top- k use. *Example:* words like “merry”, “christmas”, “tree” boost (🎄); words like “usa”, “4th”, “july” boost (🇺🇸).
- **Nearest Centroid Classifier:** For each emoji class, we compute one average TF-IDF vector (a “prototype”). We rank classes by cosine similarity between the message vector and all prototypes. This

uses all features and often produces better rankings. *Example:* a beach post with “sun”, “beach”, “summer” is close to the (☀️) prototype even if no single keyword dominates.

Both methods are non-parametric at test time and produce scores per class. The keyword method is very interpretable (it shows which words mattered), while the cosine method improves ranking quality.

NOTE: Learning objective and optimization. Our two baselines (Bag-of-Words Weighted and Nearest Centroid) are *non-learning* methods, so there is no trained model, loss, or optimizer in this iteration. In the next iteration we will add a trained baseline: **one-vs-rest logistic regression** on the same TF-IDF features with **cross-entropy (log) loss** optimized by **LBFGS** (liblinear as an alternative for small/very sparse problems).

Step 4: Training, evaluation, and reporting. We split the training data into 80% train and 20% hold-out with a fixed seed (and stratify to keep class ratios). We fit TF-IDF on the train split, run both scorers on the hold-out, and compute:

1. **Top-1 accuracy** (first guess correct?),
2. **Top- k accuracy** for $k \in \{3, 5\}$ (correct label in the first k guesses?),
3. **Precision/recall/F1 per class** plus macro and weighted averages,
4. (Optional) a **confusion matrix** for error patterns.

We save tables (CSV) and figures (PNG) to a results folder so they can be inserted into the report without re-running experiments.

Design choices. Each layer has one responsibility: the data layer outputs clean text, the feature layer outputs TF-IDF matrices, the model layer outputs per-class scores, and the evaluation layer turns scores into metrics and plots. We avoid leakage by fitting TF-IDF only on the training split. Random seeds are fixed for repeatability. Settings such as vocabulary size, n -grams, and cleaning options live in one place so ablations are easy. *Example labels we handle:* (❤️), (🔥), (😂), (🎄), (🇺🇸), (📸), (💯).

What works well. The pipeline is reproducible and easy to extend. The keyword method provides human-readable evidence (top words per class). The cosine method gives better ranked suggestions, which matches the product goal of a short, helpful emoji list.

Current limits and next steps. Sum-based keyword selection can favor big classes and generic words. We will switch to mean-based statistics per class, keep hashtag/mention *words* (strip only #/@) to recover useful signals, and grow the TF-IDF vocabulary with tuned frequency thresholds. We will also add a simple trained linear model (e.g., logistic regression) on the same features to improve top-1 accuracy while keeping the system small and easy to explain.

6 Results and Evaluation

Setup. We split the 70k training data into 80% train and 20% hold-out with a fixed random seed (stratified to keep class ratios). We fit TF-IDF on the train split only and apply it to the hold-out. We test two simple baselines on the hold-out: (1) **Bag-of-Words Weighted Classifier** and (2) **Nearest Centroid Classifier**. We report top- k accuracy and a per-class classification report (precision/recall/F1).

Baselines (short recap). **Bag-of-Words Weighted Classifier** uses a list of all tokens' average weights per emoji and sums their TF-IDF weights to score each class. **Nearest Centroid Classifier** builds one average TF-IDF vector (a prototype) for each emoji and ranks classes by cosine similarity to the message vector.

Metric	Keyword	Cosine
Top-1	0.1726	0.1870
Top-3	0.3395	0.3704
Top-5	0.4770	0.5096

Table 2: Main results (top- k accuracy) comparing keyword and cosine methods.

Table 2 shows top- k accuracy for both methods. Cosine outperforms Bag-of-Words Weighted Classifier at every k : With 20 classes, random top-1 is ≈ 0.05 . The largest class is ≈ 0.22 of the data, so an “always majority” baseline would be ~ 0.22 for top-1. Our top-1

Metric	Weights	Cosine
Accuracy	0.1723	0.1864
Macro Avg. Precision	0.1781	0.1917
Macro Avg. Recall	0.1813	0.2012
Macro Avg. F1-score	0.1677	0.1828
Weighted Avg. Precision	0.2465	0.2621
Weighted Avg. Recall	0.1723	0.1864
Weighted Avg. F1-score	0.1847	0.1957

Table 3: Comparison of classification performance between the *Weights* and *Cosine* models.

is below that, but the ranked list is useful: cosine reaches ~ 0.51 top-5, so in about half the cases the correct emoji is within the first five suggestions.

Table 3 shows detailed classification metrics. Cosine again outperforms Bag-of-Words Weighted Classifier on all metrics: accuracy, macro and weighted precision/recall/F1.

Table 4 (in Appendix) shows the full per-class precision/recall/F1/support for both methods.

Per-class summary (keyword baseline). Overall accuracy is 0.1866, macro-F1 is 0.1825, and weighted-F1 is 0.1960, which reflects class imbalance and overlap in wording. Some emojis tied to clear topics do better, for example:

- Class 5 (): F1 = 0.55 (holiday words).
- Class 11 () F1 ≈ 0.30 .
- Class 7 () F1 ≈ 0.28 .

Emojis with similar feelings (hearts/smiles) are often confused. Class 9 () has high precision (0.4351) but low recall (0.1276): when predicted it is usually right, but many true cases are missed.

Takeaways. Cosine beats Bag-of-Words Weighted Classifier at every k because it uses all features and normalizes lengths. The gap between top-1 and top-5 shows that a short list of suggestions is more practical than a single forced choice for this task.

Next steps. To improve top-1 while keeping the system simple and explainable, we will: (1) keep hashtag and mention *words* (strip only #/@) to recover signal, (2) increase the TF-IDF vocabulary and tune frequency limits, (3) switch keyword selection from *sum* to *mean*

within each class to reduce bias, and (4) add a trained linear model (e.g., logistic regression) on the same TF-IDF features.

Qualitative Examples (Top-5, simple list)

- **Text:** “tattoo number 2!... gave me strength...”

True: ❤️

Cosine: [😊, ❤️, 😄, 😊, 💜]

Keyword: [😊, ❤️, 😄, 💜, 😍]

- **Text:** “again with the weak form but cool slowmos evolution”

True: 😎

Cosine: [😎, 📸, 😄, 😊, 😃]

Keyword: [😎, 📸, 😄, 😊, 😃]

- **Text:** “my beautiful oldest daughter... alabama”

True: ❤️

Cosine: [😊, 💜, ✨, 📸, ❤️]

Keyword: [😊, 💜, ✨, 📸, ❤️]

- **Text:** “nyc memories thank you... teachers... york”

True: 📸

Cosine: [📸, 😊, ❤️, ✨, 📸]

Keyword: [📸, 😊, ✨, 🇺🇸, ❤️]

- **Text:** “dbar... going to miss you!! seasons hotel toronto”

True: 😢

Cosine: [❤️, 💜, 😢, ❤️, 😢]

Keyword: [❤️, 💜, 😢, 😢, ❤️]

- **Text:** “time off ...”

True: 😎

Cosine: [😊, 😊, 😎, 😊, ✨]

Keyword: [😊, 😊, 😎, 😊, ✨]

- **Text:** “the more ability... more responsibility...”

True: 💯

Cosine: [😎, 💯, 😊, ❤️, 😊]

Keyword: [😎, 💯, 😊, ❤️, 😊]

- **Text:** “enjoying our last day together... sun”

True: ❤️

Cosine: [☀️, 😊, 😄, 😎, 😊]

Keyword: [☀️, 😊, 😄, 😎, 😊]

- **Text:** “love spending days with my girl... fall festival”

True: ❤️

Cosine: [❤️, ❤️, 💜, 😳, 😍]

Keyword: [❤️, ❤️, 😳, 💜, 😍]

What these examples show. Cosine and Keyword often return a very similar *set* of emojis but in a different *order*, which explains why top-5 is much higher than top-1. Cosine tends to surface topical/contextual cues (e.g., ☀️ for a sunny day) because it uses the whole TF-IDF vector; Keyword highlights directly matched tokens (e.g., camera words 📸/📸). Hearts and affection emojis naturally co-occur (❤️, ❤️, 💜, 😳), so they often appear together in the top-5 even when top-1 is wrong.

7 Feedback and Plans

TA feedback (Monday tutorial).

1. **Label space is skewed.** Current labels are mostly positive emojis; there are few or no negative/neutral ones (e.g., 😞, 😔).
2. **Hashtags may carry signal.** Content after a # is not always noise; many hashtags are strong topic cues.
3. **Broader baselines.** We should test more models and add clearer benchmarks.

What we learned. Our pipeline works end to end and is easy to reproduce, but top-1 accuracy is low and the dataset is imbalanced. The keyword scorer is interpretable, yet it struggles when classes share generic words; the cosine scorer ranks better but still misses many *heart/smile* cases. Removing hashtags and mentions likely threw away useful signals.

Planned improvements (next iteration).

- **Keep hashtag/mention words.** Strip only the symbols (#/@), keep the words (e.g., “christmas”, “usa”, team names). Re-run the full evaluation and compare top-1/top-*k* and macro-F1.
- **Better keyword selection.** Change per-class keyword ranking from *sum* TF-IDF to *mean* TF-IDF to reduce bias toward large classes.
- **Richer *n*-grams and vocab.** Increase max_features, tune min_df/max_df, and keep informative bigrams (e.g., “merry christmas” 🎄, “good morning” ☀️).

Model benchmarks to add.

- **Logistic regression (one-vs-rest)** on TF-IDF (strong linear baseline; fast and interpretable).
- **Linear SVM** on TF-IDF (robust for sparse text).
- **Multinomial Naive Bayes** (very fast, good reference).
- **fastText-style** bag-of-tricks (if time permits).
- (Stretch) **Tiny transformer** (e.g., DistilBERT) for a small neural reference using the same train/hold-out split.

Evaluation and analysis. Keep reporting top-1 and top- k accuracy, plus macro/weighted F1. Add a confusion matrix and a short error analysis: show a few typical mistakes (e.g., ❤️ vs. 😊) and note which features/hashtags might fix them. Track per-class support to make imbalance effects clear.

Data and labels. If we cannot add new negative/neutral emojis now, document the label skew clearly and, if feasible, try simple rebalancing: class-weighted loss for linear models or downsampling the largest class for an ablation. (If additional data becomes available, include a few negative labels such as 😞, 😢 for a richer task.)

Engineering hygiene. Save all artifacts (vectorizer, metrics CSVs, plots) to a results folder; add a simple CLI script to reproduce the notebook run; pin the environment in requirements.txt; keep the random seed fixed.

Timeline.

- **Week N:** Hashtag/mention tweak, mean-based keywords, larger TF-IDF; re-run baselines; update tables/plots.
- **Week N+1:** Add logistic regression, linear SVM, Naive Bayes; full comparison; confusion matrix + error analysis.
- **Week N+3 (buffer):** fastText or tiny transformer (if time), ablation summary, finalize report and figures.

These steps address the TA’s feedback directly: we keep informative hashtags, broaden baselines, and make the limits of the label space explicit while steadily improving accuracy and usefulness.

Team Contributions

Team structure. Two authors; contributions are equal (50/50). We work via PRs with mutual code review; both authors sign off on experiments and writing.

Zifan (Frank) Si.

- Set up repo, data loading/cleaning (src/dataset.py), cache flow.
- Built TF-IDF feature pipeline and nearest-centroid (cosine/prototype) scorer.
- Wrote evaluation code (split, metrics, top- k) and notebooks/02_model_experiments.ipynb.
- Generated figures/tables and maintained the Overleaf draft.
- Co-wrote: Introduction, Dataset, Results.

Xianzhao (Paul) Duan.

- Implemented keyword baseline (src/model/KeywordBaseline.py).
- Added label↔emoji utilities (src/utils.py) and qualitative Top-5 examples.
- Co-designed preprocessing choices (hashtags/mentions), organized results exports.
- Co-wrote: Related Work, Features, Implementation, Feedback & Plans.

Joint work.

- Selected metrics (top-1/top- k , macro/weighted F1), ran experiments, debugged data issues.
- Curated qualitative examples; finalized the report; ensured reproducibility (fixed seeds, saved artifacts).

Ongoing work aligned with TA feedback (Sections 3, 4, 5, 7).

- **Keep hashtag/@mention words** (strip only #/@) and re-run evaluation — *Owner: Paul*.

- **Keyword stats:** sum → mean TF-IDF to reduce class-size bias — *Owner: Paul.*
- **Expand TF-IDF vocab/n-grams** (tune min_df/max_df) — *Owner: Frank.*
- **Add trained baselines** (logistic regression OvR with cross-entropy + LBFGS/liblinear; linear SVM; Multinomial NB) with macro/weighted F1 and top- k reporting — *Owner: Frank* (LR/SVM) and *Paul* (NB); joint review.
- **Error analysis & confusion matrix;** update tables/figures and write-ups — *Joint.*

Both authors will implement the TA feedback in Sections 3–5 & 7 (hashtags kept as words, mean TF-IDF keywords, expanded TF-IDF vocabulary, and trained baselines with macro/weighted F1 and top- k).

References

- Francesco Barbieri, Miguel Ballesteros, Horacio Saglion, and 1 others. 2018. Semeval-2018 task 2: Multilingual emoji prediction. In *SemEval*, pages 24–33.
- Thorsten Joachims. 1998. Text categorization with support vector machines: Learning with many relevant features. In *ECML*, pages 137–142.
- Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of tricks for efficient text classification. In *EACL*, pages 427–431.
- Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *ICLR (Workshop)*.
- Fabian Pedregosa and 1 others. 2011. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830.
- Gerard Salton and Chris Buckley. 1988. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523.

A Full Classification Report

Class	Weights				Cosine			
	Prec.	Rec.	F1	Support	Prec.	Rec.	F1	Support
0	0.0313	0.1277	0.0503	282	0.0380	0.1206	0.0578	282
1	0.1050	0.1394	0.1197	531	0.1030	0.1544	0.1236	531
2	0.2486	0.1257	0.1670	1408	0.2321	0.1286	0.1654	1408
3	0.3441	0.2464	0.2872	1384	0.3722	0.2811	0.3203	1384
4	0.0431	0.0887	0.0580	372	0.0544	0.1102	0.0728	372
5	0.5824	0.5297	0.5548	387	0.6031	0.5969	0.6000	387
6	0.0686	0.0951	0.0797	431	0.0820	0.1183	0.0969	431
7	0.2900	0.2777	0.2837	875	0.3025	0.3143	0.3083	875
8	0.0914	0.1830	0.1219	377	0.0991	0.1963	0.1317	377
9	0.4360	0.1273	0.1970	3049	0.4638	0.1177	0.1878	3049
10	0.0495	0.1239	0.0708	355	0.0562	0.1408	0.0803	355
11	0.3124	0.2927	0.3022	509	0.4018	0.3576	0.3784	509
12	0.2184	0.4108	0.2852	370	0.2152	0.4757	0.2963	370
13	0.1545	0.2065	0.1767	644	0.1768	0.1988	0.1871	644
14	0.0972	0.1180	0.1066	466	0.1180	0.1438	0.1296	466
15	0.1367	0.1126	0.1235	728	0.1357	0.1195	0.1271	728
16	0.1280	0.0903	0.1059	587	0.1614	0.1124	0.1325	587
17	0.0816	0.1111	0.0941	531	0.0700	0.0979	0.0816	531
18	0.0580	0.0559	0.0569	358	0.0562	0.0531	0.0546	358
19	0.0861	0.1629	0.1126	356	0.0928	0.1854	0.1237	356

Table 4: Per-class precision, recall, F1-score, support for *Weights* vs. *Cosine*.