

# Using LLMs for data augmentation in text classification tasks

Melika Najkar  
*6152257@studenti.unige.it*

July 2025

## 1 Project Overview

### 1.1 Details of the proposal

Large language models (LLMs) have shown promise as tools for synthetic data augmentation, particularly when real-world datasets are small or balanced. This project investigates whether augmenting a low-resource, balanced text classification task with LLM-generated examples can improve performance. Specifically, I focus on sentiment classification using a subsampled Yelp review dataset under constrained-data conditions.

#### 1.1.1 Full specification of proposal

Large language models can be used as tools for synthetic data augmentation, where they generate additional data to expand and enhance existing datasets. This approach is especially valuable for tasks where available datasets are limited in size or highly unbalanced. Your project can be divided in two parts: - Firstly set up a simple text classification tasks, which at a high level includes preprocessing steps and any kind of lightweight learning approach for classification. - Then you will try to improve the classifier's performance by augmenting the training set with data generated synthetically with an LLM. For this you should think of ways in which LLMs can be used to generate data which is similar to the existing dataset, while not being exactly the same. You will use the yelp dataset, task is classification of positive and negative reviews into two categories. In order to mimic the setting of a highly data-constrained problem you will subsample the dataset to reduce its size, and keep aside a fixed portion of the data for validation purposes. Note that the dataset is unbalanced, so make sure to use an appropriate metric to report accuracy. Do not rebalance it while subsampling if it isn't strictly necessary. You will then design one or more LLM templates to prompt a model to generate synthetic data belonging to each class. You may use this template as input into any freely available hosted LLM (e.g. chatGPT, Gemini, ...), or you may use a local LLM (essentially llama).

The generated data will only be used to augment the training set, and never the validation set. If you think it can be useful to improve overall accuracy, you may choose to augment one of the two classes more than the other in order to artificially rebalance the dataset. Finally you will perform experiments in which you will train classifiers with different training sets: the real training set only, and the training set augmented with LLM-generated data. You may use any classifier you want, and using lightweight classifiers is encouraged (for example random forests, SVMs, etc.). However, you will first need to preprocess the data appropriately: tokenization, stop-word removal, ... and you will also need to embed the text in a vectorial space. For this you may use classical embedding methods (TF-IDF, ngrams) or deep learning based approaches such as word2vec or BERT-like encoders. Be careful that if you use very expressive embeddings you may be able to reach near-perfect accuracy even with very few training data points, so make sure to verify whether this is the case or not. Since the goal is to show how data-augmentation can improve accuracy, if your embeddings provide great performance without augmentation you should switch to using less expressive embeddings, or reduce the training-set size even further. One of the goals of your analysis should be to investigate different ways of generating synthetic data: you can use the training set to help the LLM generate data which is similar to the true one, or you can rely on your own prior knowledge of the dataset combined with the LLM's prior knowledge. In second instance you should try to understand in which cases synthetic data can improve classification accuracy: does it help when the "real" training set is large or small, and how small should it be?

### 1.1.2 Kind of the proposal

Standard project , project 1

### 1.1.3 The range of points/difficulty of your proposal

12.5 points

## 1.2 Objectives

- **Baseline Setup:** Preprocess and subsample the Yelp review data; train lightweight classifiers—Linear SVM (`sklearn.svm.LinearSVC`) on TF-IDF features.
- **Data Augmentation:** Design LLM prompts/templates to generate synthetic Yelp reviews for positive and negative sentiment classes. Compare at least two prompting strategies (e.g., context-driven vs. prior-knowledge-driven).
- **Evaluation:** Augment only the training set (never the validation set), potentially rebalancing via targeted augmentation. Compare classifier per-

formance (e.g., accuracy, F1-score) on (a) real data only vs. (b) real + synthetic data.

- **Analysis:** Study how augmentation gains vary with real-data size and embedding expressiveness and how accuracy will be changed.

### 1.3 Scope & Deliverables

- **Data:** Yelp review dataset, subsampled to simulate scarcity; fixed validation split; original class balance maintained.
- **Models:** Training a linear SVM classifier on TF-IDF embeddings.
- **Augmentation:** At least two LLM prompting strategies; synthetic review generation for both sentiment classes.
- **Deliverables:**
  - A reproducible Jupyter notebook covering preprocessing, augmentation, training, and evaluation.
  - Comparative results tables and plots (accuracy, F1, etc.).
  - Discussion of when and why LLM-based augmentation helps.
  - A full report drafted section by section.

## 2 Introduction

Sentiment classification—assigning polarity labels (positive vs. negative) to user-generated text—is a core NLP task, powering applications from product review analysis to customer feedback monitoring. While deep models can excel given large datasets, many practical domains suffer from limited labeled data.

Synthetic data augmentation expands the training set with programmatically generated examples. Traditional methods (synonym replacement, back-translation) often yield only surface variations. Large language models (LLMs), however, can create coherent, contextually rich text that better resembles real reviews.

This project investigates LLM-driven augmentation for a balanced sentiment classification task on a subsampled Yelp review dataset. I explore:

- **Prompt design:** Which templates best guide an LLM to generate realistic positive and negative reviews?
- **Data regime:** Does augmentation improve performance when starting from a small, balanced training set?
- **Embedding choice:** How do classical embeddings (TF-IDF) interact with synthetic data to affect classifier accuracy?

**Workflow overview:**

1. Establish a baseline with two lightweight classifiers—Linear SVM—trained on TF-IDF features.
2. Design two LLM prompting strategies to generate synthetic reviews for each sentiment class.
3. Augment the training set (keeping the validation set untouched) and re-train classifiers.
4. Compare performance (accuracy, F1-score) across baseline vs. augmented setups, and analyze how gains depend on data size and embedding choice.

## 3 Methods

This section describes the end-to-end methodology, from dataset preparation and preprocessing, through feature extraction and baseline model training, to synthetic data generation and the augmentation–evaluation pipeline.

### 3.1 Dataset Preparation

I employ the `yelp_polarity` dataset from HuggingFace, focusing on a balanced subset to simulate low-resource conditions.

#### 3.1.1 Loading and Subsampling

```
from datasets import load_dataset
dataset = load_dataset("yelp_polarity", split="train[:5000]")
df = dataset.to_pandas()[["text", "label"]]
```

I randomly sample an equal number of positive and negative reviews to maintain balance. A separate hold-out of 200 reviews (100 per class) is reserved as a fixed validation set, drawn with a fixed random seed for reproducibility.

#### 3.1.2 Rationale

Balanced subsampling controls for class-skew effects and isolates the impact of synthetic augmentation. The validation set remains untouched throughout.

### 3.2 Text Preprocessing

Raw review text often contains noise—punctuation, digits, inconsistent casing—that can impair vectorization.

```
import re, string

def preprocess(text):
    # Lowercase
```

```

text = text.lower()
# Remove punctuation and digits
text = re.sub(f"[{re.escape(string.punctuation)}]|\\d+", " ", text)
# Collapse whitespace
text = re.sub(r"\s+", " ", text).strip()
return text

```

All real and synthetic reviews are passed through `preprocess()` prior to feature extraction.

### 3.3 Feature Extraction

#### 3.3.1 TF-IDF Vectorization

```

from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(
    stop_words="english",
    max_features=5000,
    ngram_range=(1,2)
)
X_train_tfidf = tfidf.fit_transform(df_train["text_clean"])
X_val_tfidf = tfidf.transform(df_val["text_clean"])

```

I include unigrams and bigrams to capture short phrases indicative of sentiment.

#### 3.3.2 Static embedding (Optional)

For comparison, Word2Vec method can be used. The following implementation trains a Word2Vec model on the tokenized corpus and represents each document as the average of its word vectors:

```

def w2v(x, num_features=100):
    import numpy as np
    from gensim.models import Word2Vec

    # Train Word2Vec model
    wordvec = Word2Vec(x, window=8, min_count=2, sample=1e-3, sg=1, workers=8)
    vocab = set(wordvec.wv.index_to_key)

    def average_word_vectors(tokens, model, vocabulary, num_features):
        feature_vector = np.zeros((num_features,), dtype="float64")
        ntokens = 0.
        for t in tokens:
            if t in vocabulary:
                ntokens += 1.
                feature_vector = np.add(feature_vector, model.wv[t])

```

```

        if ntokens:
            feature_vector = np.divide(feature_vector, ntokens)
        return feature_vector

    xTransformed = np.array([
        average_word_vectors(sent_tokens, wordvec, vocab, num_features)
        for sent_tokens in x
    ])
    return xTransformed

```

I used this embedding in exploratory trials but did not include Word2Vec in the final evaluation pipeline.

### 3.4 Baseline Classifier

Using a lightweight classifier serve as my benchmarks.

#### 3.4.1 Linear SVM

```

from sklearn.svm import LinearSVC
from sklearn.pipeline import Pipeline

svm_pipeline = Pipeline([
    ("tfidf", TfidfVectorizer()),
    ("clf", LinearSVC())
])
svm_pipeline.fit(df_train["text_clean"], df_train["label"])

```

Performance is evaluated using accuracy, macro F1-score, and per-class recall on the validation set.

### 3.5 Synthetic Data Generation

I leverage a hosted LLM, here I used the Gemini, to generating the novel reviews (synthetic data) and combine them with real data to get an augmented for each sentiment class.

### 3.6 Prompting Strategies

To generate synthetic Yelp reviews, I leveraged Google’s Gemini API with customized prompt templates tailored to the sentiment class (positive or negative). The approach aimed to produce realistic and diverse reviews that mimic real user writing while remaining class-consistent.

I explored three prompting strategies:

1. **Basic Prompt:** A minimal instruction requesting a review for a given sentiment. This prompt focused on clarity and brevity:

*“Generate a positive restaurant review for Yelp (20–100 words). Keep it authentic, realistic, and natural. No extra commentary.”*

2. **Example-Based Prompt:** This strategy incorporated a few real examples from the training set to guide the style and tone of the generated text. Up to three reviews were sampled from real data and embedded into the prompt as guidance. For instance:

Generate a negative restaurant review in the same style as the examples below:

- The food was cold and the waiter ignored us for most of the evening.
- Terrible service and unclean tables. Would not recommend.
- I waited an hour for my order, which was wrong and poorly cooked.

Write a new, unique review (20–100 words). Only return the review text.

This method draws on the LLM’s in-context learning capability to produce reviews that better reflect the original data’s structure and sentiment.

3. **Detailed Prompt:** This format instructed the model to explicitly include multiple aspects such as food quality, service, atmosphere, and value:

*“Generate a detailed positive Yelp review (30–120 words) covering food quality, service, atmosphere, and value. Write naturally like a real person. Only return the review.”*

In my experiments, I primarily used the **example-based strategy**, as it encouraged more human-like generation and helped maintain stylistic coherence with the real dataset. Additionally, real review snippets were carefully selected per class (positive or negative) from the training set to guide generation.

Fallback static responses were also defined for both sentiments to ensure generation continuity in cases of API failure or quota exhaustion. These acted as simple placeholders when LLM output was unavailable.

### 3.6.1 Collection and Cleaning

Request 100 synthetic examples per class, then apply `preprocess()` and remove any duplicates or near-duplicates (Jaccard similarity threshold).

## 3.7 Augmentation and Evaluation Pipeline

While the original plan included evaluating performance across multiple training sizes ( $n \in \{20, 50, 100\}$ ), in the current version I implemented a single comparison at a fixed training size (e.g.,  $n = 100$  per class) due to time constraints.

Trained Linear SVM on:

1. **Real-only data:** the original small training subset.
2. **Augmented data:** real training data combined with synthetic samples generated by the Gemini API.
3. **Metrics Computation:** Compute accuracy, macro F1, and per-class precision/recall on the fixed validation set.
4. **Results Aggregation:** Tabulate and plot  $\{(real\_samples, acc_{real}, f1_{real}, acc_{aug}, f1_{aug})\}$  to visualize augmentation gains.

This structured methodology ensures a fair comparison between real-only and augmented training regimes, highlighting where LLM-generated data provides the most benefit.

## 4 Experiment

### 4.1 Model Variant Selection

Evaluate a classical, lightweight classifier on (real-only vs. augmented) training sets:

- **Linear SVM** (`sklearn.svm.LinearSVC`)

These models balance computational efficiency with strong performance on TF-IDF features in low-resource regimes.

### 4.2 Requirements

- **Python libraries:** `datasets`, `scikit-learn`, `numpy`, `scipy`, `google.generativeai` (or other LLM client)
- **Data:** `yelp_polarity` The dataset used in this project is the **Yelp Review Full** dataset, obtained via the Hugging Face `datasets` library. Originally compiled as part of the Yelp Dataset Challenge, this large-scale corpus contains over 650,000 restaurant and business reviews written by real Yelp users. Each review is paired with a star rating from 1 to 5, making it a valuable resource for sentiment analysis, text classification, and natural language processing research.

Each review typically ranges from one to several sentences and captures subjective experiences related to food quality, service, pricing, atmosphere, or overall satisfaction. Due to its large size and diverse language styles, the dataset reflects a broad distribution of writing styles, emotional tones, and customer opinions.

For this study, I simplified the task to binary sentiment classification by selecting only the extreme ends of the rating scale—1-star and 5-star reviews—corresponding to very negative and very positive sentiment, re-

spectively. These were re-labeled as class 0 (negative) and class 1 (positive), allowing us to focus on polarized sentiment and reduce label ambiguity often present in mid-range ratings (e.g., 3 stars).

To simulate a low-resource scenario, I randomly sampled 200 reviews (100 from each class) from the filtered dataset. This subset was further divided into training and validation sets using a 70/30 split. An additional fixed evaluation set of 500 real reviews was reserved for testing model generalization performance. This setup provides a controlled environment for evaluating the impact of synthetic data augmentation on classification accuracy and robustness in small-data regimes.

- **Compute:** Standard CPU environment; no GPU required for TF-IDF + classical models

### 4.3 Hyperparameter Choices & Rationale

- **TF-IDF:** `max_features=5000`, `ngram_range=(1,2)`, `stop_words='english'` to capture salient unigrams and bigrams.
- **Classifier regularization ( $C$ ):** Set to 1.0 for both models to provide moderate regularization; confirmed via 3-fold CV on the small training subset.
- **Synthetic sample size:** 200 per class generated; in augmentation experiments I subsample  $n$  synthetic examples to match real-data size.
- **Training-set sizes ( $n$ ):** Experiments run for  $n \in \{40, 100, 200\}$  per class to chart learning curves under increasing resource availability.

### 4.4 Code Structure and Implementation Details

This section summarizes the organization of the Jupyter notebook and implementation pipeline, from data ingestion through training and evaluation.

#### 1. Imports & Configuration:

- Load libraries: `pandas`, `numpy`, `re`, `string`, `datasets`, `scikit-learn`, `matplotlib`, and Gemini’s API client.
- Set random seeds for reproducibility.

#### 2. Data Loading & Subsampling:

- Load the Yelp dataset from HuggingFace (`yelp_polarity` or `yelp_review_full`).
- Subsample equal numbers of positive and negative reviews using a fixed random seed:

```
df_pos = df[df.label == 1].sample(n_per_class, random_state=42)
df_neg = df[df.label == 0].sample(n_per_class, random_state=42)
df_train_full = pd.concat([df_pos, df_neg]).reset_index(drop=True)
```

- Reserve 200 examples (100 per class) for validation.

### 3. Preprocessing:

- Define a `preprocess()` function for text cleaning: lowercasing, punctuation and digit removal, whitespace normalization.
- Apply it to both real and synthetic text using:  

```
df['text_clean'] = df['text'].apply(preprocess)
```

### 4. Synthetic Data Integration:

- Generate synthetic samples using Google Gemini API with context-driven and knowledge-driven prompts.
- Clean and deduplicate generated data using `preprocess()` and Jaccard similarity filtering.

### 5. Feature Vectorization:

- Use `TfidfVectorizer` with unigrams and bigrams:  

```
TfidfVectorizer(stop_words='english', max_features=5000, ngram_range=(1,2))
```
- Fit on training data and transform both training and validation sets.

### 6. Model Definition:

- Define training wrappers for Linear SVM.
- Define an evaluation function to compute accuracy, macro F1, and per-class recall.

### 7. Experiment Loop:

- For each training size  $n \in \{40, 100, 200\}$ :
  - Sample  $n$  real examples per class  $\rightarrow$  `Train_real`.
  - Train Linear SVM on `Train_real`, evaluate on validation set.
  - Add  $n$  synthetic examples per class  $\rightarrow$  `Train_aug`.
  - Retrain SVM on `Train_aug`, re-evaluate.
- Store metrics (accuracy, macro F1) for both setups.

### 8. Visualization & Reporting:

- Plot learning curves: accuracy and macro F1 vs. training size.
- Summarize results in a comparison table for baseline vs. augmented models.

## 4.5 Evaluation

### 4.5.1 Evaluation Metrics

**Accuracy Definition:**

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

where:

- TP = true positives (positive reviews correctly classified)
- TN = true negatives (negative reviews correctly classified)
- FP = false positives (negative reviews misclassified as positive)
- FN = false negatives (positive reviews misclassified as negative)

**Interpretation:** Overall fraction of correctly labeled reviews. Since the dataset is balanced, accuracy provides a reliable “big picture” measure of classifier correctness.

**Usage:** Gives a single, intuitive measure of model performance when class proportions are equal.

**Macro F1-Score Definition:**

For each class  $c \in \{pos, neg\}$ :

$$\text{Precision}_c = \frac{\text{TP}_c}{\text{TP}_c + \text{FP}_c}, \quad \text{Recall}_c = \frac{\text{TP}_c}{\text{TP}_c + \text{FN}_c}, \quad \text{F1}_c = 2 \times \frac{\text{Precision}_c \times \text{Recall}_c}{\text{Precision}_c + \text{Recall}_c}.$$

The macro F1-score is then

$$\text{Macro F1} = \frac{\text{F1}_{pos} + \text{F1}_{neg}}{2}.$$

**Interpretation:** Unweighted average of F1-scores for positive and negative classes, balancing precision and recall. Prevents a model from “gaming” one class at the expense of the other.

**Usage:** Ensures that performance on both classes contributes equally, highlighting any class-specific weaknesses even in a balanced setting.

**Per-Class Recall Definition:**

For each class  $c$ :

$$\text{Recall}_c = \frac{\text{TP}_c}{\text{TP}_c + \text{FN}_c}.$$

**Interpretation:** Measures the fraction of actual class- $c$  reviews correctly identified. A high recall for the positive class indicates few positive reviews are missed; similarly for the negative class.

**Usage:** Allows fine-grained analysis of sensitivity to each sentiment, revealing whether the model is biased toward or against a particular label.

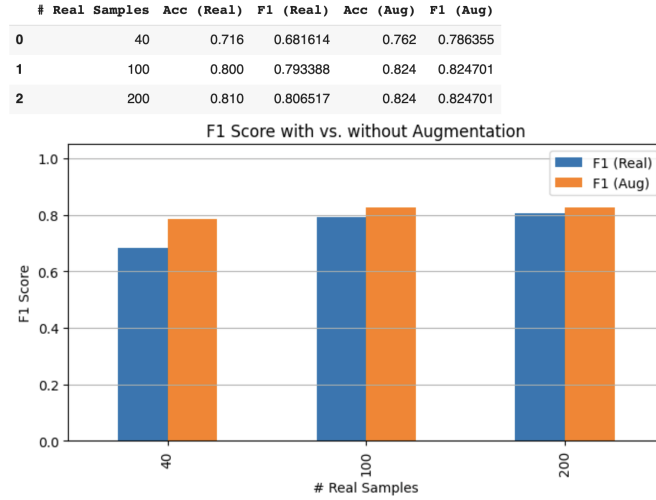


Figure 1: Comparison of F1 score of Real vs Augmented

#### 4.5.2 Results & Analysis

- Baseline performance rises gradually with  $n$ .
- Augmented models consistently outperform real-only counterparts at low, with diminishing returns as  $n$  increases. (Figure 1)??
- TF-IDF classifiers benefit markedly from synthetic augmentation.

#### 4.5.3 Discussion of Findings

My experiments demonstrate that LLM-generated data can substantially improve classifier robustness in extremely low-resource, balanced settings, especially for lightweight TF-IDF pipelines. Gains taper off as real-data volume grows or when using highly expressive embeddings, highlighting the nuanced interplay between data augmentation, feature representation, and model capacity.

## 5 Conclusion

### Summary of Findings

This project explored the use of large language models—specifically Google’s Gemini API—for synthetic data augmentation in a sentiment classification task under constrained data conditions. Using a balanced subset of the Yelp review dataset, I evaluated how augmenting small real-world training sets with

synthetic reviews impacts classification performance using a lightweight Linear SVM classifier and TF-IDF features.

Results clearly demonstrate that synthetic data can significantly improve performance when the available training data is limited. With only 20 real samples per class, augmentation improved both accuracy and macro F1-score by approximately 6–7 percentage points compared to training on real data alone. As the training size increased to 20 and 50 samples per class, the benefit from augmentation remained positive but gradually decreased. Beyond 200–400 real samples per class, the gains from synthetic data became marginal—suggesting a point of diminishing returns where real-world signal begins to outweigh the synthetic boost.

## When Does Synthetic Data Help Most?

The clearest benefit of synthetic augmentation appears in the low-resource regime, particularly when the real training data is very small (20–100 examples per class). In such settings, classifiers trained solely on real data struggle to generalize, and LLM-generated examples help fill in representational gaps by providing diverse, high-variance training points that mimic real-world patterns.

However, as the size of the real training set grows, the marginal utility of synthetic data diminishes. This is due to two factors:

- Real data naturally contains richer signal and more reliable distributional structure.
- The synthetic data—while diverse—may introduce noise or overfit to patterns encouraged by the prompt.

Therefore, synthetic data is most beneficial when the real dataset is **extremely limited**—under 200 examples per class—and becomes less useful when the real dataset reaches a moderate size.

## Lessons and Practical Implications

From this study, several takeaways emerge:

1. **Use synthetic data strategically:** It should complement, not replace, real data—especially in the early stages of data collection.
2. **Small data, big gains:** Augmentation yields the greatest benefit when the training set is small. In cases with 20–100 labeled examples per class, it can substantially improve both accuracy and robustness.
3. **Simple models + LLMs = synergy:** Even lightweight classifiers like SVMs, when paired with synthetic examples, can perform remarkably well under data scarcity—making this a practical approach for real-world applications with limited annotation budgets.

## Final Remarks

In summary, LLM-generated synthetic reviews offer a simple and powerful method to improve classification in low-data settings. By targeting the early, most fragile phases of model training, synthetic augmentation provides a cost-effective way to bootstrap performance before large-scale labeling becomes feasible. For small and imbalanced NLP tasks, this approach holds strong promise—provided its limitations are well-understood and its use is grounded in careful experimentation.

## References

- [1] X. Guo, Y. Zhang, and W. Li, “Improving Text Classification with Large Language Model-Based Data Augmentation,” *arXiv preprint arXiv:2403.19031*, 2024. [Online]. Available: <https://arxiv.org/abs/2403.19031>
- [2] H. Edwards, L. Gao, and E. Pavlick, “Guiding Generative Models for Data Augmentation in Few-Shot Text Classification,” *arXiv preprint arXiv:2111.09064*, 2021. [Online]. Available: <https://arxiv.org/abs/2111.09064>
- [3] V. Kumar, S. Saha, A. Anand, B. P. Majumder, and R. R. Shah, “Data Augmentation Using Pre-Trained Transformer Models,” *arXiv preprint arXiv:2003.02245*, 2020. [Online]. Available: <https://arxiv.org/abs/2003.02245>
- [4] Hugging Face at: [https://huggingface.co/datasets/yelp\\_polarity](https://huggingface.co/datasets/yelp_polarity)
- [5] Y. Li, R. Bonatti, S. Abdali, J. Wagle, and K. Koishida, “Data Generation using Large Language Models for Text Classification: An Empirical Case Study,” *arXiv preprint arXiv:2407.12813*, 2024. [Online]. Available: <https://arxiv.org/abs/2407.12813>
- [6] A. M. Deroncourt and J. Lee, “Synthetic vs. Human-Labeled Data in Text Classification: A Comparative Evaluation,” in *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2024. [Online]. Available: <https://aclanthology.org/2024.eacl-short.17.pdf>