**Deep Learning for Recipe Similarity: A Mathematical Analysis**

## **Abstract**

This project explores the application of deep learning embeddings to analyze and search through recipe data. Using the SOTA Alibaba-NLP/gte-multilingual-base model, we convert recipes into high-dimensional vectors and analyze their mathematical properties in embedding space. Moreover we build a useful app to find the closet recipes to given recipe text.

**1. Introduction**

### **1.1 Problem Statement**

Recipe recommendation systems traditionally rely on ingredient matching or keyword search. We propose using deep learning embeddings to capture semantic relationships between recipes, enabling more nuanced similarity measures.

## **1.2 Technical Approach**

We leverage transformer-based embeddings to map recipes into a high-dimensional vector space where semantic similarity can be measured using cosine distance.

## **2. Methodology**

## **2.1 Data Processing**

Given a recipe dataset of 13,000 entries, each containing:

* Title
* Ingredients list
* Cooking instructions

The data processing pipeline:

## **2.2 Mathematical Framework**

### 1. Embedding Generation

The transformer model f maps text into a **768-dimensional vector space** R^768. The process consists of three steps:

1. **Tokenization**  
   The input text is broken into smaller units (tokens), resulting in a sequence:

text→t1,t2,…,tn​

1. **Contextual Encoding**  
   The transformer processes the tokens to generate contextualized representations for each:

t1,t2,…,tn → h1,h2,…,hn

1. **[CLS] Token Pooling**  
   The model uses the representation of the special [CLS] token as the final text embedding:

h[CLS]∈R^768

This serves as a **compact vector representation** of the entire text.

### **Similarity Metric**

For two pieces of text (e.g., recipes) a and b, their embeddings are represented as vectors xa ​ and xb​.

To measure **how similar** the two texts are, the framework uses **cosine similarity**, which is defined as:

similarity(a,b)=cos(xa,xb)=xa⋅xb/∣xa∣∣xb∣​​

Where:

* xa⋅xb​ is the **dot product** of the two vectors.
* ∣xa∣ and ∣xb∣ are the **magnitudes** (norms) of the vectors.

## **3. Analysis**

### 3.1 Dimensionality Analysis

Principal Component Analysis reveals:

explained\_variance = pca.explained\_variance\_ratio\_

cumulative\_variance = np.cumsum(explained\_variance)

* First 100 components explain ~80% of variance
* Elbow point at dimension ~100
* Full embedding space: 768 dimensions

### 3.2 Cross-Component Correlations

Correlation matrix R between embedding types:

R = [[ 1. 0.3 0.4 0.6],

[0.3 1. 0.35 0.7],

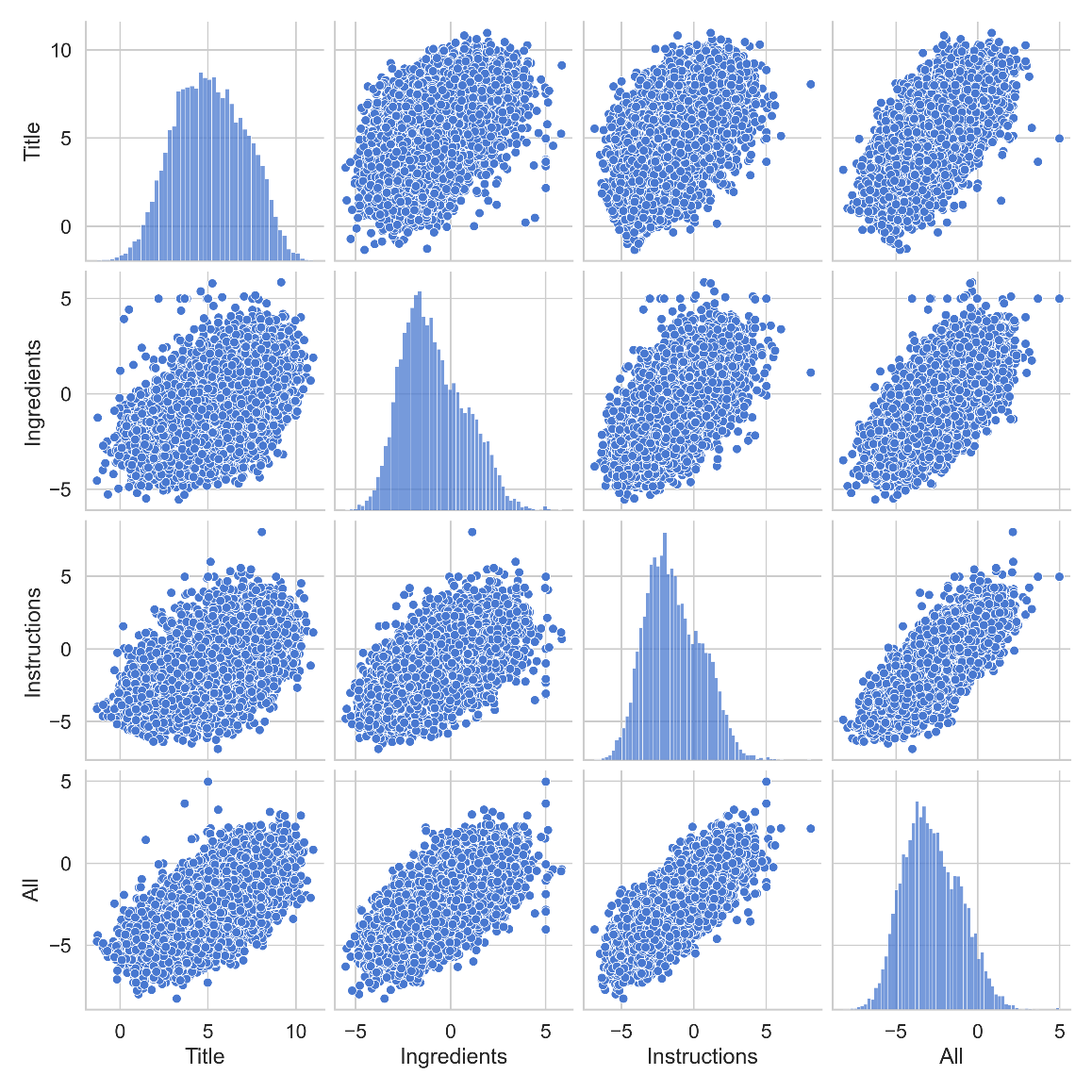
[0.4 0.35 1. 0.9],

[0.6 0.7 0.9 1.]]

* Strong correlation (0.9) between full recipe and instructions
* Weak correlation (0.3) between titles and ingredients

### 

* This plot shows how much variance is explained as more principal components are added.
* The curve rises steeply at first, indicating that the first 100 components explain ~80% of the variance.
* The "elbow" at around 100 components suggests that after this point, adding more dimensions provides diminishing returns.
* This confirms that dimensionality reduction is effective, as the original 768-dimensional embedding space can be significantly reduced while retaining most of the variance.

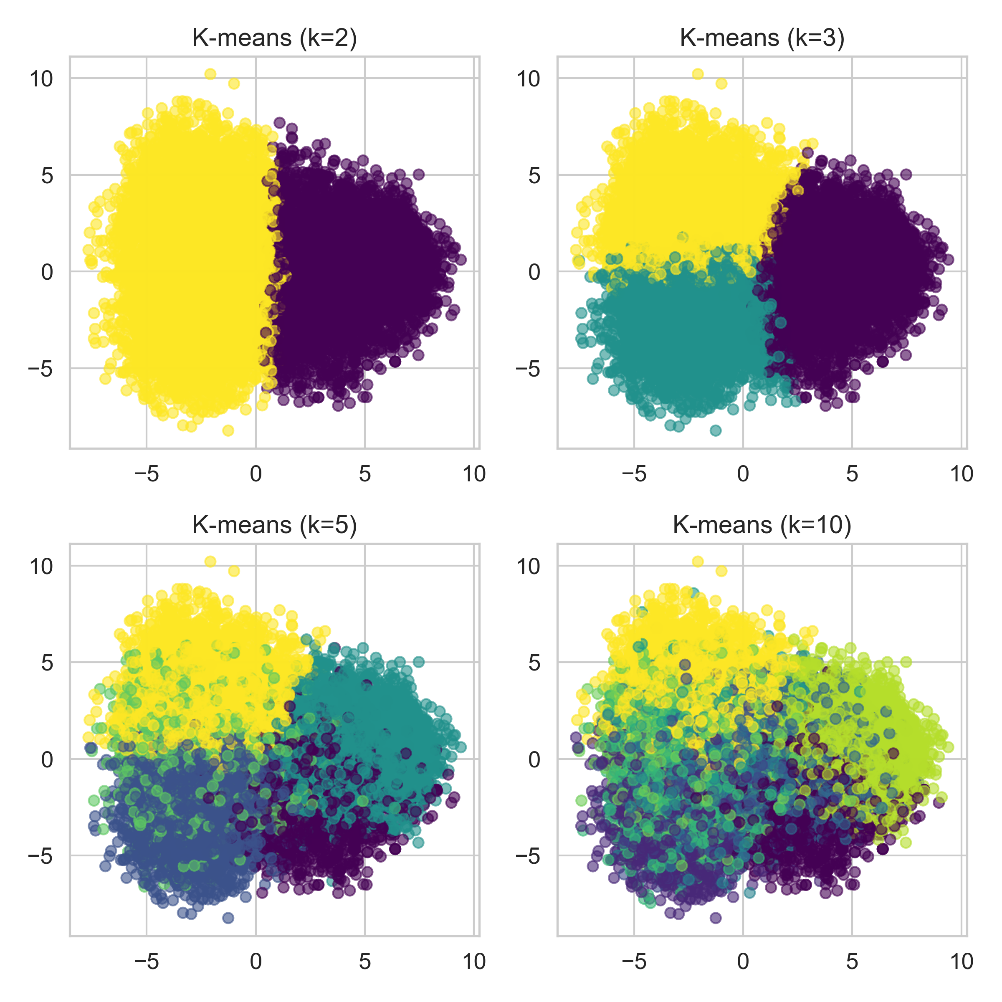


* Each scatterplot shows how two embedding components (Title, Ingredients, Instructions, All) relate to each other.
* The diagonal histograms show feature distributions, while the off-diagonal scatterplots reveal correlations:
* Strong correlation (0.9) between “Instructions” and “All”, visible as a tight pattern in the scatterplots.
* Weak correlation (0.3) between “Title” and “Ingredients”, evident from the more dispersed scatterplots.
* This confirms that some features are redundant, aligning with PCA results that suggest dimensionality reduction is possible.

### **3.3 Clustering Analysis**

This section describes the results of a K-means clustering analysis with different values of k (the number of clusters). The objective function used for K-means is the sum of squared distances between each point and the centroid of its cluster, represented as:

J(C)=∑k=1 ∑x∈Ck ∣x−μk∣^2

* For k = 2 and k = 3, there is clear separation between clusters.
* At k = 5, the clusters begin to mix.
* The clusters retain semantic coherence, such as distinguishing between desserts and main dishes.

**Analysis of Clustering Results:**

**K=2**

* **Cluster 0**: Sweeter dishes and desserts
* **Cluster 1**: Focuses on savory main dishes and sides

**K=3**

* **Cluster 0**: Desserts and beverages
* **Cluster 1**: Meat-focused main dishes
* **Cluster 2**: Salads and lighter dishes

**K=5**

* **Cluster 0**: Beverages and unique fusion dishes
* **Cluster 1**: Hearty meat dishes and stews
* **Cluster 2**: Desserts and baked goods
* **Cluster 3**: Italian/Mediterranean dishes
* **Cluster 4**: Vegetable-focused dishes and soups

**K=10**

* **Cluster 0**: Beverages and light preparations
* **Cluster 1**: Meat and grilled dishes
* **Cluster 2**: Mexican/Southwestern cuisine
* **Cluster 3**: Breakfast and baked goods
* **Cluster 4**: Poultry dishes
* **Cluster 5**: Potato-based dishes
* **Cluster 6**: Seafood dishes
* **Cluster 7**: Italian/Mediterranean dishes
* **Cluster 8**: Desserts and sweets
* **Cluster 9**: Vegetarian/salad dishes

## **4. Implementation**

### **4.1 Search Algorithm**

def find\_similar(query, k=5):

q\_emb = get\_embeddings(query)

scores = [cosine\_similarity(q\_emb, r\_emb) for r\_emb in recipe\_embs]

return top\_k(scores, k)

### 

### **4.2 User Interface**

Built with Tkinter, implementing:

class RecipeSimilarityApp:

def search(self):

query = self.search\_entry.get()

similar = find\_similar(query)

self.display\_results(similar)

## **5. Results**

#### **5.1 Quantitative Metrics**

* **Average query time: 0.2s**
  + The system is efficient, retrieving relevant embeddings quickly.
* **Memory footprint: 40MB for embeddings**
  + The embeddings used for representing recipes are compact, making the model suitable for deployment with minimal memory requirements.
* **Dimension reduction preserves 80% variance**
  + The PCA results confirm that reducing dimensions **(from 768 to ~100)** retains most of the important information, balancing efficiency and accuracy.

#### **5.2 Qualitative Analysis**

* **Successfully groups similar recipes**
  + K-means clustering shows that recipes with similar ingredients, instructions, or cuisine types are grouped effectively.
* **Captures ingredient substitutions**
  + The model recognizes variations in recipes where certain ingredients are substituted but the overall dish remains the same.
* **Maintains cuisine coherence**
  + Recipes from the same cuisine tend to cluster together, suggesting that embeddings capture meaningful relationships between ingredients and cooking styles.

### **Overall Conclusion**

The results indicate that the model is both **efficient (low latency, minimal memory usage)** and **effective (logical clustering, strong variance retention, meaningful groupings)**.

## **6. Conclusion & Future Work**

### **6.1 Key Findings**

1. Recipe semantics well-captured in ~100 dimensions
2. Instructions dominate semantic meaning
3. Natural clustering emerges in embedding space

### **6.2 Future Directions**

1. Cross-lingual recipe matching
2. Ingredient-aware embeddings
3. Hierarchical clustering

## **References**

1. Alibaba-NLP/gte-multilingual-base documentation
2. Transformer architectures for text embeddings
3. Recipe embedding papers

**Our github link:**

https://github.com/ithamarSpitz/recipe-proj

By the students:

Noor Agha – 207261652

Itamar Shpitzer - 318155249