Model Development Phase Template

Date	9 Dec 2025
Team ID	RT
Project Title	Restaurant Recommendation System
Maximum Marks	4 Marks

Initial Model Training Code, Model Validation and Evaluation Report

Trained single content-based model via TF-IDF + cosine similarity on 12k cleaned Zomato records.

Initial Model Training Code:


```
def __init__(self, clean_data_path: str = "../DATASET/zomato_clean.csv"):  # PATH
    self.clean_data_path = Path(clean_data_path)
    self.df = None
    self.tfidf = None
    self.feature_matrix = None
    self.similarity_matrix = None
```

1 usage new *

```
def load_data(self):
    """Load cleaned Zomato data."""
    self.df = pd.read_csv(self.clean_data_path)
    print(f" Loaded {self.df.shape[0]:,} restaurants ({self.df.shape[1]} columns)")
    print("Sample:")
    print(self.df[["name", "location", "cuisines", "rate"]].head(3))
```

1 usage new *

```
def create_features(self):
    """TF-IDF vectorization on cuisines + location + name."""
    print("\n🔄 Creating TF-IDF features...")

    # Combine features into "soup" text
    self.df["soup"] = (
            self.df["cuisines"].fillna("") + " " +
            self.df["location"].fillna("") + " " +
            self.df["rest_type"].fillna("") + " " +
            self.df["name"].fillna("")
    )

    # TF-IDF ( core algorithm)
    self.tfidf = TfidfVectorizer(
        max_features=5000,
        stop_words="english",
        ngram_range=(1, 2),
        lowercase=True
    )

    self.feature_matrix = self.tfidf.fit_transform(self.df["soup"])
    print(f" Feature matrix: {self.feature_matrix.shape}")
```

1 usage new *

```python
def compute_similarity(self):
    """Cosine similarity between all restaurants."""
    print("🔄 Computing similarity matrix...")
    self.similarity_matrix = cosine_similarity(self.feature_matrix)
    print(f" Similarity matrix ready ({self.similarity_matrix.shape})")
```

2 usages (1 dynamic) new *

```python
def recommend(self, query: str, priority: str = "name", top_k: int = 10) -> pd.DataFrame:
    query_lower = query.lower().strip()

    if priority == "location":
        matches = self.df[self.df['location'].str.lower().str.contains(query_lower, na=False)]
    elif priority == "cuisines":
        matches = self.df[self.df['cuisines'].str.lower().str.contains(query_lower, na=False)]
    else:  # "name"
        matches = self.df[self.df['name'].str.lower().str.contains(query_lower, na=False)]

    if len(matches) == 0:
        return pd.DataFrame()

    idx = matches.index[0]
    sim_scores = list(enumerate(self.similarity_matrix[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)[1:top_k * 2 + 1]  # MORE candidates

    indices = [i[0] for i in sim_scores]
    recs = self.df.iloc[indices].copy()
    recs['similarity'] = [score[1] for score in sim_scores]

    # FIXED: Softer dedupe—only if > top_k unique chains
    recs['base_name'] = recs['name'].str.split().str[0].str.upper()
    if len(recs['base_name'].unique()) > top_k:
        recs = recs.loc[recs.groupby('base_name')['similarity'].idxmax()]

    # Shuffle for variety
    recs = recs.sample(frac=1, random_state=None).reset_index(drop=True)

    return recs.drop(columns=['base_name']).head(top_k)
```

```python
def save_model(self, model_path: str = "../model/tfidf_model.pkl"):
    """Save trained model for Flask."""
    Path(model_path).parent.mkdir(exist_ok=True)
    with open(model_path, "wb") as f:
        pickle.dump(self, f)
    print(f"Model saved: {model_path}")
```

```python
def evaluate_model(self, test_restaurants: list = None):
    """Model evaluation: Success Rate + Similarity Metrics"""

    if test_restaurants is None:
        test_restaurants = ["McDonald's", "Domino's", "KFC",
                            "Paradise", "Biryani", "Castle Rock"]

    results = []
    for restaurant in test_restaurants:
        recs = self.recommend(restaurant, priority="name", top_k=10)
        found = not recs.empty
        avg_sim = 0.0
        if found and 'similarity' in recs.columns:
            avg_sim = recs['similarity'].mean().round(3)

        top_match = recs['name'].iloc[0] if found else "None"
        results.append({
            'query': restaurant,
            'found': found,
            'avg_similarity': avg_sim,
            'top_match': top_match
        })
    # METRICS CALCULATION
    eval_df = pd.DataFrame(results)
    success_rate = eval_df['found'].mean() * 100
    avg_similarity = eval_df['avg_similarity'].mean().round(3)
    precision_03 = (eval_df['avg_similarity'] >= 0.3).mean() * 100

    print(f"📊 SUCCESS RATE:     {success_rate:.1f}% ({eval_df['found'].sum()}/6)")
    print(f"📊 AVG SIMILARITY:   {avg_similarity}")
    print(f"📊 PRECISION@10:     {precision_03:.1f}% (sim≥0.3)")

    return eval_df
```

Model Validation and Evaluation Report:

Model	Classification Report	Accuracy	Confusion Matrix
TF-IDF Base (5000 feats)	Success Rate: 85.7% (12/14), Avg Similarity: 0.446, Diversity: 6.2 chains/10, Latency: 5.9ms	85.7%	FULL METRICS: Success Rate: 85.7% (12/14) Avg Similarity: 0.446 Avg Diversity: 6.3 unique chains/10 Avg Latency: 5.6ms/query 95th Percentile: 8.0ms Model saved: ../model/tfidf_model.pkl
TF-IDF Tuned	Success Rate: 85.7% (12/14), Avg Similarity: 0.601 (+35%), Diversity: 6.1 chains/10, Latency: 6.0ms	85.7%	FULL METRICS: Success Rate: 85.7% (12/14) Avg Similarity: 0.601 Avg Diversity: 6.1 unique chains/10 Avg Latency: 5.5ms/query 95th Percentile: 7.6ms Model saved: ../model/tfidf_model.pkl