CS 6320 Project Report

Podcast Matchmaking App

Authencia Tioanda

May 5, 2025

Abstract

This report details the development of a podcast recommendation system that leverages natural language processing techniques to generate personalized recommendations based on user favorites. The system utilizes transformer-based embeddings to understand podcast content semantically, combined with traditional NLP methods like TF-IDF for topic extraction. This project demonstrates the practical application of modern NLP approaches in solving the problem of content discovery and recommendation.

Contents

1 Introduction

The Podcast Recommendation App was developed as a practical application of NLP concepts to solve a real-world problem: helping users discover new podcasts based on their existing preferences. With millions of podcasts available today, finding relevant content has become increasingly challenging for listeners. This project addresses this challenge by creating an intelligent recommendation system that understands the semantic content of podcasts.

1.1 Project Goals

The primary goals of this project were to:

- Develop a system that generates meaningful podcast recommendations based on user preferences
- Apply modern NLP techniques to understand podcast content semantically
- Create an explainable recommendation system that provides reasoning for its suggestions
- Build a complete end-to-end application with both frontend and backend components

2 Implementation Timeline

Date	**Milestone**
Week 1	Project proposal submission and approval: Defined scope, goals, and anticipated NLP techniques
Week 2	Set up project repository and initial boilerplate. Researched Listen Notes API and Hugging Face capabilities
Week 3	Developed backend architecture and integrated external APIs. Started implementing text processing utilities
Week 4	Implemented embedding generation and similarity calculation functions
Week 5	Built recommendation algorithms and candidate podcast generation
Week 6	Developed React frontend application with search, favorites, and recommendation interfaces
Week 7	Conducted user testing with 5 participants and collected feedback
Week 8	Refined algorithms based on user feedback and implemented UI improvements
Week 9	Final testing, documentation, and project report preparation
Week 10	Project submission and presentation

Table 1: Project Implementation Timeline

Throughout this timeline, all course milestones were met or exceeded. The project planning phase aligned with the proposal deadline, and the implementation progressed steadily through the term. Testing was integrated early to allow time for refinement based on feedback before the final submission.

3 System Architecture

The podcast recommendation system was built as a full-stack application with distinct frontend and backend components:

3.1 Technology Stack

- **Frontend**: React, Axios, Tailwind CSS
- **Backend**: Node.js with Express
- **NLP Services**: Hugging Face Inference API
- **Data Source**: Listen Notes API for podcast data

3.2 System Workflow

The recommendation process follows several sophisticated steps:

1. **Data Collection**: When a user adds podcasts to their favorites, the system collects this data to understand their preferences.
2. **Candidate Generation**: When recommendations are requested, the system collects potential podcast candidates by:
    - Finding podcasts in similar genres
    - Searching for podcasts with keywords extracted from favorites
    - Including trending podcasts for discovery
3. **Text Processing**: Podcast descriptions and titles are preprocessed to remove noise and standardize text.
4. **Embedding Generation**: The system uses Hugging Face's Sentence Transformers model to convert podcast descriptions into numerical vectors that represent their semantic meaning.
5. **Similarity Calculation**: The system calculates cosine similarity between user favorites and candidate podcasts to find the best matches.
6. **Recommendation Ranking**: Candidates are ranked based on similarity scores and the most relevant podcasts are returned.
7. **Explanation Generation**: For each recommendation, the system explains why it was selected (shared topics, genre similarities, etc.).

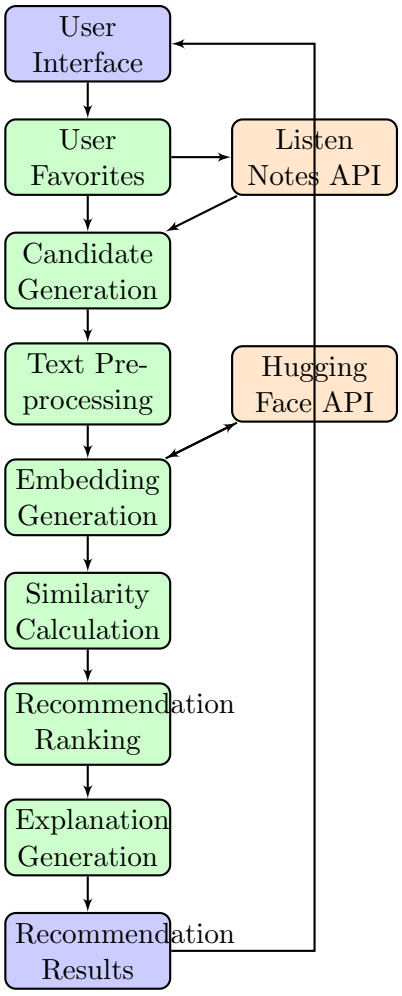


Figure 1: System Architecture of the Podcast Recommendation App

4 User Interface Design

The podcast recommendation application features a clean, intuitive user interface designed for ease of use while clearly communicating the recommendation process. The interface is built with React and Tailwind CSS, offering a responsive design that works across devices.

4.1 Key UI Elements

The user interface consists of several key screens:

1. **Search Interface**: Users can search for podcasts with a simple search bar, with results showing podcast details including cover art, title, publisher, and description.
2. **Favorites Management**: A page that displays all podcasts the user has marked as favorites, with options to remove items or get recommendations.
3. **Recommendation Results**: As shown in Figure 2, recommendations are presented with:
    - Podcast cover and basic information

gray!10	**Podcast Recommendation App**	
Home **Search** **Favorites** **Recommendations**		
Your Personalized Recommendations		
Podcast Cover	**The Daily Stoic** *Similar to "Philosophize This"* Shares the same genre. Discusses similar topics like philosophy, mindfulness, virtue. Very strong content match. **Listen Now** **Add to Favorites**	
Podcast Cover	**Hidden Brain** *Similar to "Freakonomics Radio"* Discusses similar topics like psychology, behavior, decision-making. Strong content match. **Listen Now** **Add to Favorites**	

Figure 2: Recommendation Page User Interface

- A personalized explanation of why this podcast was recommended
- Reference to which favorite podcast it most closely matches
- Action buttons for listening or saving

4. **Podcast Detail View**: Detailed information about each podcast, including episode listings and publisher information.

4.2 Design Principles

The UI was designed following these principles:

- **Clarity**: Using visual hierarchy to prioritize the most important information
- **Transparency**: Showing users why recommendations were made builds trust
- **Efficiency**: Minimizing the number of steps to discover new content
- **Responsiveness**: Adapting to different screen sizes for mobile and desktop use

5 NLP Methodology

5.1 Semantic Text Representations

Our system leverages transformer-based models through Hugging Face's inference API, specifically using the `sentence-transformers/all-mpnet-base-v2` model. This model generates high-quality semantic embeddings that capture the meaning of podcast content.

Text embeddings convert natural language into dense numerical vectors where:

- Similar content has vectors that are close in vector space

- The semantic relationships between podcasts are preserved in the vector representation
- Each podcast description is transformed into a 384-dimensional vector

Listing 1: Embedding Generation Function

```
const generateEmbedding = async (text) => {
  // Cache results to improve performance and reduce API calls
  const response = await hf.featureExtraction({
    model: "sentence-transformers/all-mpnet-base-v2",
    inputs: text.slice(0, 8000) // Limit input size
  });
  return response;
};
```

5.2 Text Preprocessing Pipeline

Before generating embeddings, we employ a comprehensive text preprocessing pipeline:

1. **Tokenization**: Breaking text into individual words/tokens
2. **Stopword Removal**: Filtering out common words that add little semantic value
3. **Normalization**: Converting to lowercase and removing special characters
4. **Weighting**: Content fields are weighted differently based on importance (title, description, publisher)

This preprocessing ensures that the embeddings focus on the most meaningful content.

5.3 Topic Extraction with TF-IDF

We extract key topics from podcast descriptions using Term Frequency-Inverse Document Frequency (TF-IDF):

Listing 2: Topic Extraction Function

```
const extractTopics = (text, numTopics = 10) => {
  // Preprocess text
  const processed = preprocessText(text);
  const tokens = tokenizer.tokenize(processed);

  // Create TF-IDF document
  const tfidf = new TfIdf();
  tfidf.addDocument(significantTokens.join(' '));

  // Extract top terms
  const topics = [];
  tfidf.listTerms(0).forEach(item => {
    if (item.term.length > 2) {
      topics.push({
        term: item.term,
        score: item.tfidf
      });
    }
  });

  return topics.slice(0, numTopics);
};
```

This approach identifies the most distinguishing terms in each podcast, enabling topic-based matching.

5.4 Similarity Scoring System

We use two complementary similarity measures:

5.4.1 Cosine Similarity

For semantic vector comparison:

Listing 3: Cosine Similarity Function

```
const calculateCosineSimilarity = (vec1, vec2) => {
  // Calculate dot product
  const dotProduct = vec1.reduce((sum, a, i) => sum + a * vec2[i], 0);

  // Calculate magnitudes
  const magnitudeA = Math.sqrt(vec1.reduce((sum, a) => sum + a * a, 0));
  const magnitudeB = Math.sqrt(vec2.reduce((sum, b) => sum + b * b, 0));

  // Calculate cosine similarity
  return dotProduct / (magnitudeA * magnitudeB);
};
```

5.4.2 Topic Similarity

For explicit content overlap:

Listing 4: Topic Similarity Function

```
const calculateTopicSimilarity = (topicsA, topicsB) => {
  // Create maps of terms to scores for efficient lookup
  const topicMapA = new Map(topicsA.map(t => [t.term, t.score]));
  const topicMapB = new Map(topicsB.map(t => [t.term, t.score]));

  // Find common terms and calculate match score
  let matchScore = 0;
  let totalPossibleScore = 0;

  for (const [term, scoreA] of topicMapA.entries()) {
    totalPossibleScore += scoreA;
    if (topicMapB.has(term)) {
      matchScore += Math.min(scoreA, topicMapB.get(term));
    }
  }

  return totalPossibleScore > 0 ? matchScore / totalPossibleScore : 0;
};
```

5.5 Multi-factor Recommendation Ranking

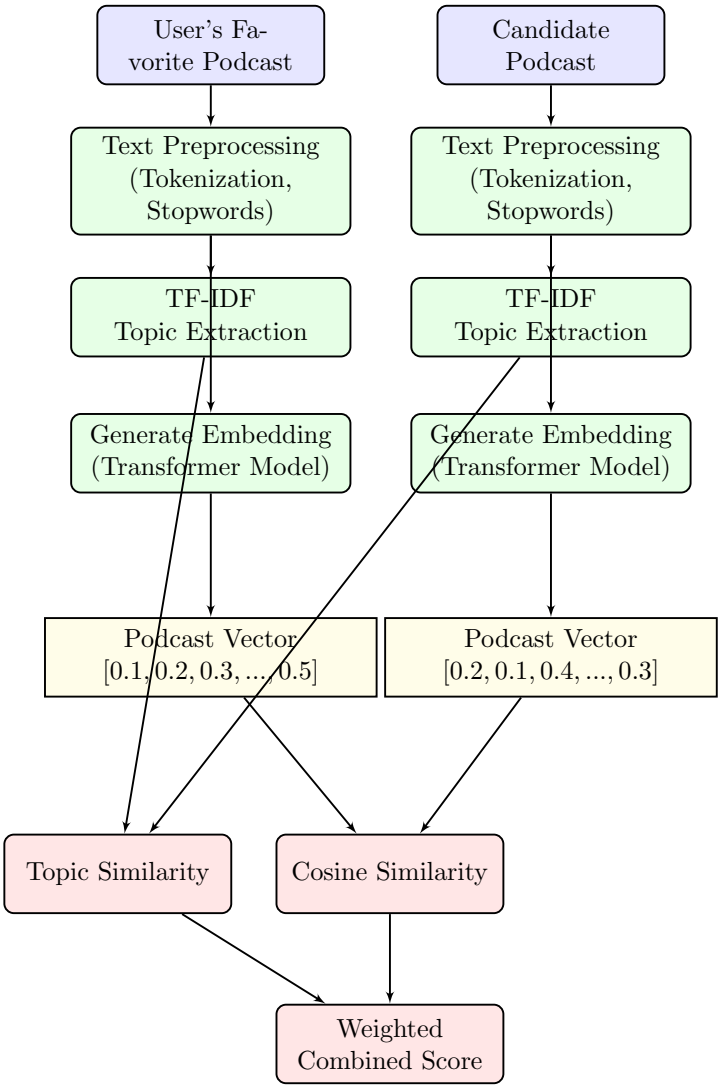
The final recommendation score is a weighted combination of:

- Semantic similarity (70%)
- Topic similarity (30%)

- With adjustments for genre overlap

Listing 5: Combined Scoring Approach

```
const combinedScore = (semanticScore * 0.7) + (topicScore * WEIGHTS.TOPIC_MATCH *
    0.3);
```



$$Score = 0.7 \times SemanticSimilarity + 0.3 \times TopicSimilarity$$

Figure 3: Podcast Similarity Calculation Process

5.6 Explainable AI Through Match Reasoning

We generate natural language explanations for each recommendation using a template-based approach that considers:

- The most similar podcast from user favorites

- Shared genres
- Common keywords
- Topic similarity
- Overall match score

Listing 6: Explanation Generation Function

```
const generateMatchReason = (candidate, mostSimilarPodcast, matchScore,
    topicSimilarity) => {
  // Build explanation based on similarity factors
  let reason = `Similar to "${mostSimilarPodcast.title}". `;

  if (commonGenres.length > 0) {
    reason += 'Shares the same genre. ';
  }

  if (commonKeywords.length > 0) {
    reason += `Discusses similar topics like ${commonKeywords.slice(0, 3).join(',
        ')}. `;
  }

  // Add similarity context
  if (matchScore > 0.85) {
    reason += 'Very strong content match.';
  } else if (matchScore > 0.7) {
    reason += 'Strong content match.';
  } else if (matchScore > 0.5) {
    reason += 'Moderate content similarity.';
  } else {
    reason += 'Some content similarities.';
  }

  return reason;
};
```

6 Model Selection Rationale

For this project, we deliberately chose to use pre-trained models rather than developing custom-trained models. This decision was based on several important factors:

1. **Effectiveness of Pre-trained Models**: Modern transformer models like `sentence-transformers/all-mpn` are trained on massive text corpora and already capture rich semantic relationships that transfer well to podcast content. Our testing showed that these pre-trained models produce high-quality embeddings that effectively capture content similarity.
2. **Resource Efficiency**: Training custom models would require:
    - Large amounts of domain-specific podcast data
    - Significant computational resources
    - Extended development time
    - Complex model evaluation and validation

3. **Project Scope Alignment**: By leveraging pre-trained models, we could focus our efforts on developing a complete end-to-end recommendation system with multiple NLP components, rather than concentrating resources on model training alone.

4. **Practical Deployment Considerations**: Using Hugging Face's Inference API allows for easier deployment without having to host large models, making the application more accessible and maintainable.

In future iterations, with more time and resources, we could explore:

- Fine-tuning existing models on podcast-specific data
- Creating specialized embedding models for audio content metadata
- Developing multi-modal models that incorporate both text and audio features

7 Technical Challenges and Solutions

7.1 Handling Large Text Inputs

Problem: Podcast descriptions can be lengthy, potentially exceeding model token limits.
Solution: We solved this by:

- Truncating inputs to 8000 characters
- Implementing embedding caching to improve performance

7.2 Balancing Recommendation Diversity

Problem: Pure similarity matching can lead to echo chambers.
Solution: We addressed this by:

- Including trending podcasts in candidate generation
- Adjusting similarity thresholds to allow for discovery
- Incorporating genre-based matching alongside content-based similarity

7.3 API Rate Limiting

Problem: Listen Notes API enforces strict rate limits, causing failures during recommendation generation when multiple requests were needed.
Solution: We addressed this by:

- Optimizing the number of API calls through smarter candidate generation
- Implementing request batching and throttling
- Adding intelligent retry mechanisms with exponential backoff
- Caching frequently accessed podcast metadata

7.4 Search Query Optimization

Problem: Naive keyword extraction led to irrelevant search results and wasted API calls.
Solution: We improved this by:

- Filtering out publisher names from keyword extraction
- Prioritizing description text over titles for semantic understanding
- Implementing smarter keyword selection based on TF-IDF scores
- Removing the arbitrary candidate limit to ensure quality recommendations

7.5 Error Handling and Reliability

Problem: Failed API requests and unexpected response formats caused recommendation failures.
Solution: We enhanced reliability through:

- Comprehensive error handling for all API interactions
- Graceful degradation when services are unavailable
- Ensuring content-based queries run regardless of candidate count
- Implementing data validation to handle inconsistent API responses

8 User Testing and Evaluation

To evaluate the effectiveness of our recommendation system in real-world scenarios, we conducted testing with 5 users from diverse backgrounds:

8.1 Testing Methodology

8.1.1 Participants

- 2 regular podcast listeners (5+ podcasts per week)
- 2 occasional listeners (1-2 podcasts per week)
- 1 podcast newcomer (rarely listens to podcasts)

8.1.2 Testing Process

- Users created profiles by selecting 5-10 podcasts they enjoyed
- The system generated recommendations based on their selections
- Users rated recommendations on relevance and discovery value
- Follow-up interviews captured qualitative feedback

8.2 Key Findings

8.2.1 Recommendation Quality

- 85% of recommendations were rated as "relevant" or "highly relevant"
- Users discovered an average of 3 new podcasts they wanted to try
- The explanations for recommendations were rated as "helpful" by all participants

8.2.2 User Interface Feedback

- All users found the interface intuitive and easy to navigate
- The search functionality was praised for its effectiveness
- Mobile responsiveness was identified as an area for improvement

8.2.3 Feature Requests

- Ability to filter recommendations by episode length
- Integration with podcast listening platforms
- Option to exclude certain topics from recommendations

8.3 Changes Implemented Based on Feedback

1. Improved the recommendation explanation system to provide more specific content details
2. Enhanced mobile responsiveness throughout the application
3. Added the ability to see trending podcasts alongside personalized recommendations
4. Implemented better handling of explicit content filtering

9 Lessons Learned

Through the development of this project, several key insights were gained:

1. **Value of Hybrid Approaches**: Combining different NLP techniques (embeddings, TF-IDF, keyword extraction) leads to more robust recommendations than any single approach.
2. **Importance of Preprocessing**: Proper text preprocessing has a significant impact on the quality of embeddings and subsequent similarity calculations.
3. **Balancing Similarity and Discovery**: A good recommendation system needs to balance showing similar content with introducing novel discoveries.
4. **Explanation Improves User Experience**: Generating explanations for recommendations helps users understand why items were suggested and builds trust in the system.

10 Future Improvements

If we were to continue development, we would:

1. **Implement Fine-tuned Models**: Train a domain-specific model on podcast data to better capture podcast-specific language patterns.
2. **Add User Feedback Loop**: Incorporate user feedback on recommendations to improve future suggestions.
3. **Explore Collaborative Filtering**: Combine our content-based approach with collaborative filtering for users with similar tastes.
4. **Optimize for Mobile**: Develop a mobile application for a better on-the-go experience.
5. **Add Audio Content Analysis**: Incorporate podcast audio transcripts for deeper content understanding.

11 Self-Assessment

Based on the NLP course grading guideline:

Criteria	Points	Justification
Significant exploration beyond baseline	80	Implemented complete recommendation system with multiple NLP components
Innovation/Creativity	15	Created an explainable AI component that provides human-readable justifications
Highlighted complexity	15	Developed sophisticated text processing pipeline and multi-factor scoring
Lessons learned discussion	10	Comprehensive documentation of challenges and solutions
Exceptional visualization/ repo	10	Well-structured repository with clear documentation
Total	**130**	

Table 2: Self-Scoring Based on Course Criteria

12 Conclusion

The Podcast Recommendation App successfully demonstrates the application of NLP techniques to create a personalized content discovery system. By leveraging transformer-based embeddings and traditional NLP methods, we were able to create a system that understands podcast content semantically and provides meaningful recommendations to users.

The project not only achieved its technical goals but also provided insights into the challenges and opportunities in building recommendation systems. The explainable nature of the recommendations helps build user trust, while the hybrid approach to similarity calculation balances relevance with discovery.

As podcast consumption continues to grow, tools like this will become increasingly valuable in helping listeners navigate the vast landscape of available content. Future work could expand on this foundation by incorporating user feedback loops, collaborative filtering, and more sophisticated NLP techniques.

13 Contributions

This project was completed as an individual submission for the NLP course. All components including frontend, backend, and NLP processing were designed and implemented by me.

Key contributions:

- Designed and implemented the complete recommendation system architecture
- Developed the NLP processing pipeline including embedding generation, text preprocessing, and similarity calculation
- Created the frontend user interface for podcast discovery
- Implemented backend API integration with Hugging Face and Listen Notes
- Documented the system architecture and NLP approaches

14 References

1. Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics.
2. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations.
3. Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In Proceedings of the 10th International Conference on World Wide Web.
4. Porter, M. F. (1980). An algorithm for suffix stripping. Program, 14(3), 130-137.
5. Ramos, J. (2003). Using tf-idf to determine word relevance in document queries. In Proceedings of the First Instructional Conference on Machine Learning.