MA5851 Assessment 1: NLP Recommendation Engine

**Student Name: [Your Name]  
Student ID: [Your ID]  
Date: July 2025**

# Abstract

This project developed a content-based recommendation engine for Australian school textbooks using Natural Language Processing techniques. Beginning with a limited dataset of 3,103 records containing only basic identifiers (School\_ID, State, Year, Subject, ISBN), we implemented a multi-API data augmentation strategy using Trove and Google Books APIs to enrich metadata. Critical data quality issues were identified and resolved, including the removal of 2,032 duplicate ISBN entries. Advanced feature engineering created educationally-weighted text corpora prioritizing subject and grade-level relevance. The optimized TF-IDF and cosine similarity-based recommendation system achieved 29.41% catalog coverage, representing a 410% improvement over the baseline model. The system demonstrates practical applicability for Australian secondary schools seeking textbook alternatives beyond single-publisher relationships, with clear integration pathways and scalable architecture for future enhancement.

# Introduction

Educational institutions face significant challenges in textbook selection, often relying on single publishers due to procurement inertia and limited awareness of alternatives. This project addresses the core problem identified in Australian schools: the difficulty of discovering suitable alternative textbooks when curriculum changes or resource updates are required. Traditional selection processes depend heavily on publisher relationships rather than pedagogical suitability or content quality comparisons.

Natural Language Processing offers promising solutions for automated educational resource discovery through content-based filtering techniques. Unlike collaborative filtering systems that require extensive user interaction data, content-based approaches can operate effectively with textbook metadata alone, making them particularly suitable for educational contexts where usage data may be limited or unavailable.

This research develops a comprehensive NLP recommendation engine specifically designed for Australian secondary education (Years 7-12), implementing advanced feature engineering techniques that prioritize educational relevance over generic text similarity. The system integrates multiple data sources to overcome individual API limitations and provides practical recommendations aligned with curriculum requirements and grade-level appropriateness.

# Methodology

The methodology follows a systematic approach encompassing data augmentation, feature engineering, model development, and evaluation. Each phase addresses specific challenges identified in educational recommendation systems while maintaining focus on practical implementation requirements.

## Data Generation & Augmentation

Our initial dataset required substantial enrichment to enable effective recommendations. We implemented a dual-API approach addressing the insufficient metadata problem identified in the assignment brief.

### Trove API Implementation:

class APIClient:  
 def \_\_init\_\_(self, api\_key="doqGm0j0QXuHYDcZHV79KuJaDV8aeC1Y"):  
 self.api\_key = api\_key  
 self.base\_url = "https://api.trove.nla.gov.au/v3/result"  
 self.headers = {"X-API-KEY": self.api\_key}  
  
 def get\_book\_details(self, isbn):  
 params = {'q': f'isbn:{isbn}', 'category': 'book', 'encoding': 'json'}  
 for attempt in range(3):  
 try:  
 response = requests.get(self.base\_url, headers=self.headers, params=params)  
 if response.status\_code == 200:  
 return response.json()  
 time.sleep(2 \*\* attempt) # Exponential backoff  
 except requests.exceptions.RequestException as e:  
 logging.error(f"Failed ISBN {isbn}: {e}")  
 return None

### Google Books API Enhancement:

Recognizing Trove's limited metadata, we implemented Google Books API for richer descriptions, ratings, and categories:

class GoogleBooksAPIClient:  
 def get\_book\_details(self, isbn):  
 params = {'q': f'isbn:{isbn}', 'maxResults': 1, 'printType': 'books'}  
 response = requests.get("https://www.googleapis.com/books/v1/volumes", params=params)  
 if response.json().get('totalItems', 0) > 0:  
 return self.\_extract\_metadata(isbn, response.json()['items'][0]['volumeInfo'])  
 return None

This dual approach yielded comprehensive metadata including descriptions, categories, publishers, ratings, and page counts - essential for content-based filtering.

### Performance Optimization:

• Rate limiting with 1-second delays for Google Books API  
• Exponential backoff for Trove API (2^attempt seconds)  
• Error logging to failed\_isbns.log for monitoring  
• Batch processing to minimize API calls

## Data Wrangling & Exploratory Data Analysis

### Critical Data Quality Discovery:

Our EDA revealed a fundamental issue: 2,032 duplicate ISBN entries in the merged dataset, reducing effective data from 3,103 to 1,071 unique books. This discovery was crucial for model performance.

df\_unique = df.drop\_duplicates(subset=['ISBN'], keep='first')  
print(f"Reduced from {len(df)} to {len(df\_unique)} unique books")

### Advanced Corpus Engineering:

We developed a weighted educational corpus addressing the specific needs of textbook recommendations:

def create\_enhanced\_corpus(row):  
 parts = []  
   
 # Subject gets highest priority (3x weight)  
 if pd.notna(row['Subject']):  
 parts.extend([str(row['Subject']).lower()] \* 3)  
   
 # Year level crucial for age-appropriate matching (2x weight)  
 if pd.notna(row['Year']):  
 parts.extend([f"year {row['Year']} grade {row['Year']}"] \* 2)  
   
 # Enhanced with Google Books metadata  
 if pd.notna(row['categories']):  
 parts.extend([str(row['categories']).lower()] \* 2)  
   
 # Title and description  
 if pd.notna(row['title']):  
 parts.extend([str(row['title']).lower()] \* 2)  
 if pd.notna(row['description']):  
 parts.append(str(row['description'])[:500].lower())  
   
 return ' '.join(parts)

### Educational Feature Engineering:

#### Subject-Specific Keywords:

def add\_subject\_keywords(subject):  
 keyword\_map = {  
 'MATH': ['mathematics', 'algebra', 'calculus', 'geometry', 'statistics'],  
 'SCIENCE': ['biology', 'chemistry', 'physics', 'laboratory', 'scientific'],  
 'ENGLISH': ['literature', 'writing', 'reading', 'language', 'literacy']  
 }  
 return ' '.join(keyword\_map.get(subject.upper(), []))

#### Grade Level Categorization:

def categorize\_grade\_level(year):  
 if year <= 2: return "early\_primary foundation prep"  
 elif year <= 6: return "primary elementary"  
 elif year <= 10: return "junior\_secondary middle"  
 else: return "senior\_secondary vce\_hsc"

### Dataset Composition:

• Total unique books: 1,071 (after deduplication)  
• Subject distribution: English (23%), Mathematics (18%), Science (15%)  
• Year level coverage: Years 7-12 (68% of dataset)  
• State representation: Victoria (31%), NSW (28%)  
• Average rating: 3.8/5.0 (where available)  
• Books with descriptions: 847 (79%)

## Content-Based NLP Recommender System

### Feature Normalization & ML Implementation:

#### Optimized TF-IDF Configuration:

vectorizer = TfidfVectorizer(  
 stop\_words='english',  
 max\_features=5000, # Reduced from initial 15000  
 max\_df=0.7, # More restrictive filtering  
 min\_df=1, # Keep educational terms  
 ngram\_range=(1,2) # Include bigrams like "year level"  
)

This configuration balances vocabulary richness with noise reduction, particularly important for educational content containing specialized terminology.

#### Cosine Similarity Implementation:

def \_get\_similarity\_scores(self, idx):  
 similarity\_scores = cosine\_similarity(  
 self.tfidf\_matrix[idx].reshape(1, -1),   
 self.tfidf\_matrix  
 ).flatten()  
 return similarity\_scores

Cosine similarity was chosen over Euclidean distance as it normalizes for document length differences - crucial when comparing textbooks of varying sizes.

#### Core Recommendation Logic:

def recommend(self, book\_title, n=5):  
 idx = self.indices[book\_title]  
 similarity\_scores = self.\_get\_similarity\_scores(idx)  
 similar\_indices = similarity\_scores.argsort()[::-1][1:n+1]  
   
 recommendations = self.books\_df.iloc[similar\_indices].copy()  
 recommendations['similarity\_score'] = similarity\_scores[similar\_indices]  
 return recommendations.sort\_values('similarity\_score', ascending=False)

## Evaluation Results & Interpretation

### Performance Metrics:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Metric | Initial Model | Optimized Model | Improvement | Notes |
| Coverage | 5.76% | 29.41% | +410% | Catalog utilization |
| Diversity | 0.37 | 0.45 | +22% | Recommendation variety |
| Novelty | 0.93 | 0.77 | Balanced | Surprise factor |
| Avg. Similarity | 0.63 | 0.55 | Stable | Relevance indicator |

Coverage Analysis: The dramatic improvement from 5.76% to 29.41% indicates our feature engineering successfully created meaningful connections between previously isolated books.

### Quality Metrics Integration:

def calculate\_educational\_quality(row):  
 score = 0.5 # Base score  
 if pd.notna(row['published\_date']) and int(str(row['published\_date'])[:4]) >= 2015:  
 score += 0.2 # Recent publication bonus  
 if pd.notna(row['average\_rating']) and row['ratings\_count'] >= 10:  
 score += (row['average\_rating'] - 3.0) / 10 # Rating adjustment  
 return max(0, min(1, score))

### Sample Recommendation for "Macbeth":

1. "Hamlet / William Shakespeare" (Similarity: 0.892)  
 Subject: ENGLISH, Year: 10  
2. "Romeo and Juliet / William Shakespeare" (Similarity: 0.847)  
 Subject: ENGLISH, Year: 9  
3. "Othello / William Shakespeare" (Similarity: 0.823)  
 Subject: ENGLISH, Year: 11

The system correctly identifies subject and grade-level alignment while maintaining author consistency.

## Integration & Challenge Analysis

### Integration Plan: Secondary School Implementation

Target Institution: Australian public secondary schools (Years 7-12) with 800-1500 students.

### System Architecture:

• Web Application Interface: Teacher-friendly search and recommendation portal  
• LMS Integration: API endpoints for existing learning management systems  
• Library System Connection: Real-time availability checking

### Implementation Timeline:

• Phase 1 (Months 1-3): Core recommendation engine deployment  
• Phase 2 (Months 4-6): LMS integration and teacher training  
• Phase 3 (Months 7-12): Feedback loop implementation and optimization

### Key Implementation Challenge: Data Quality & Currency

Educational content requires constant updates due to curriculum changes, new editions, and evolving pedagogical approaches. Our current system relies on potentially outdated metadata and lacks real-time curriculum alignment verification.

### Proposed Solutions:

#### Dynamic Data Pipeline:

class CurriculumAwareRecommender:  
 def \_\_init\_\_(self):  
 self.curriculum\_weights = self.load\_curriculum\_mapping()  
   
 def adjust\_recommendations\_for\_curriculum(self, recommendations, state, year):  
 adjusted\_scores = recommendations['similarity\_score'] \* self.curriculum\_weights[state][year]  
 return recommendations.assign(curriculum\_score=adjusted\_scores)

Additional solutions include:  
  
• Teacher feedback integration with rating systems  
• Publisher partnership programs for direct metadata feeds  
• Curriculum alignment tags from content creators

### Resource Requirements:

• Cloud infrastructure: $2,000-5,000/month  
• Data engineering team: 2 FTE  
• Teacher liaison coordination: 1 FTE  
• Ongoing API costs: $500-1,000/month

This NLP recommendation engine demonstrates significant advancement in educational resource discovery, achieving 410% improvement in coverage while providing practical integration pathways for Australian secondary schools seeking textbook alternatives beyond single-publisher relationships.