Developing a Robust Hybrid NLP and Machine Learning Pipeline for Academic Syllabus Verification

Ross Lauterbach

February 2025

1 Introduction

This report outlines the creation of an automated system designed to solve a common "big data" problem in education: distinguishing official course syllabi from the thousands of miscellaneous documents found on university servers. Tthis system filters noise with near-perfect accuracy as shown below. We detail the transition from training on synthetic and scraped data to validating on a 500-sample real-world dataset.

2 The Challenge: High-Volume Document Noise

In academia, researchers often use "web scrapers": automated programs that crawl university websites to collect syllabi. While powerful, these scrapers are "blind"; they often download lecture notes, lab manuals, campus maps, and assignment prompts simply because they are saved as PDFs.

To a human, the difference between a syllabus and a lab manual is obvious. To a computer, they are both just streams of text. Manually sorting 10,000 PDFs is impossible for a human team. Our goal was to build a model that can read these files and decide with 99%+ accuracy if a document is a syllabus or "noise."

3 Data Strategy: Training vs. Validation

A critical part of this project was ensuring the model didn't just "memorize" specific files, but actually learned to understand what makes a syllabus a syllabus. To achieve this, we used two entirely separate pools of data.

3.1 Development and Training Dataset

To build the "brain" of the model, we used a high-volume mix of:

- **Web-Scraped Samples:** Thousands of documents pulled from public university directories.
- **Synthetic Data (GPT-Generated):** We used Large Language Models (LLMs) to generate "fake" syllabi and "fake" non-syllabi. This allowed us to create very tricky examples that forced the model to learn subtle differences in tone and structure.

3.2 The 500-Sample Validation Set

To prove the model actually worked in the real world, we tested it on a **completely new** set of 500 documents that the model had never seen before.

- **100 Personal Samples:** Syllabi sourced directly from friends and family across various majors.
- **400 Internet Samples:** Raw, uncleaned PDFs scraped from the web to simulate a "live" deployment environment.

4 Phase 1: Turning PDFs into Clean Text

Computers cannot "see" a PDF; they only see raw bytes. The first step is extracting the text and cleaning it. We use the `pdfminer` library for extraction and `nltk` for linguistic cleaning.

```
def process_pdf(pdf_path):
    """ Extracts text and applies initial cleaning. """
    try:
        raw_text = extract_text(pdf_path)
        if not raw_text.strip():
            return ""
        # Remove excess whitespace and newlines for uniformity
        clean_text = " ".join(raw_text.split())
        return preprocess_text(clean_text)
    except Exception as e:
        print(f"Error processing {pdf_path}: {e}")
        return ""

def process_folder(folder_path):
    """ Iterates through a directory to collect all PDF data. """
    preprocessed_texts = []
    for filename in sorted(os.listdir(folder_path)):
        if filename.lower().endswith('.pdf'):
            path = os.path.join(folder_path, filename)
            text = process_pdf(path)
            if text:
                preprocessed_texts.append(text)
    return preprocessed_texts
```

Listing 1: Data Extraction and Folder Processing

5 Phase 2: NLP and Feature Engineering

5.1 The "Stemming" Concept

For a non-technical user, imagine the words "Teaching," "Teacher," and "Taught." A simple computer sees these as three different words. Our pipeline uses a **Porter Stemmer** to chop these words down to their root: "Teach." This helps the model realize that a document about "teaching" and a document about "teachers" are likely talking about the same subject.

5.2 TF-IDF: The "Important Word" Finder

We use a mathematical formula called **TF-IDF** (Term Frequency-Inverse Document Frequency).

Instead of just counting how many times a word appears, TF-IDF asks: *"How unique is this word to this document?"*

- A word like **"University"** appears in almost every PDF. TF-IDF gives it a **low score** because it's not helpful for sorting.
- A word like **"Prerequisite"** or **"Learning Objectives"** appears often in syllabi but rarely in campus flyers. TF-IDF gives these **high scores**.

```
def extract_tfidf_features(preprocessed_texts):
    # The vectorizer converts words into a giant grid of numbers (a
    Matrix)
    # where each number represents the 'importance' (TF-IDF score) of a
    word.
    vectorizer = TfidfVectorizer(max_features=5000) # Only keep the
    5000 most useful words
    X = vectorizer.fit_transform(preprocessed_texts)
    return X, vectorizer
```

Listing 2: Vectorizing Text into Numbers

6 Phase 3: The Ensemble "Committee" Strategy

Rather than trusting one single algorithm, we use a "Committee of Experts" (an Ensemble). For a document to be labeled a syllabus, at least two out of these three models must agree.

1. **Logistic Regression:** Acts like a weighted checklist. "If it has 'Grade' and 'Schedule,' it's probably a syllabus."
2. **Support Vector Machine (SVM):** Looks for the "gap" between syllabi and non-syllabi in a 5,000-dimensional space.
3. **Naïve Bayes:** Uses probability. "Based on the 100 syllabi I've seen before, what are the odds this new one is also a syllabus?"

```
# We initialize the three models with 'balanced' weights
# to ensure the model doesn't favor one class over the other.
models = {
    'LogReg': LogisticRegression(class_weight='balanced'),
    'SVM': SVC(probability=True, class_weight='balanced'),
    'NB': MultinomialNB()
}

# Training (Fitting) the models on our Training Data
for name, model in models.items():
    model.fit(X_train, y_train)
```

Listing 3: Training the Ensemble Committee

7 Phase 4: The Heuristic "Common Sense" Layer

Machine Learning can be "tricked" by high-quality fakes. To prevent this, we added a final layer of human-like logic. Even if the models say "Yes, this is a syllabus," the code runs two final checks:

- **The Profanity Filter:** Real syllabi almost never contain profanity. If the `better_profanity` library detects "bad words," the document is instantly rejected, regardless of what the ML models think.
- **The Length Check:** Syllabi are typically between 2 and 10 pages. If a document is 100 pages (a textbook) or 1 paragraph (an email), it fails the "Length Consistency" test.

```python
def final_decision_logic(preds, text, avg_len, threshold=50):
    # Logic Gate 1: Profanity (Integrity Check)
    if profanity.contains_profanity(text):
        return 0

    # Logic Gate 2: The ML 'Majority Vote'
    num_yes_votes = np.sum(preds == 1)
    if num_yes_votes >= 2:
        # Logic Gate 3: Does the length make sense?
        doc_len = len(text.split())
        if abs(doc_len - avg_len) <= (threshold / 100) * avg_len:
            return 1 # Fully verified syllabus
    return 0 # Rejected
```

Listing 4: The Final Logic Gate

8 Results and Performance Analysis

When we finally ran our finished "Digital Gatekeeper" on the **500-sample real-world validation set**, the results were outstanding.

Category	**Value**	**Percentage**
Total Real-World Samples	500	100%
Accurate Identifications	497	99.4%
Missed Documents (False Negatives)	3	0.6%
Incorrectly Accepted (False Positives)	0	0.0%

Table 1: Performance on Final Validation Set

8.1 Why did it fail 3 times?

The 3 failures were all **Art Studio Syllabi**. These documents are outliers in the academic world. While a History syllabus talks about "Exams" and "Textbooks," an Art syllabus talks about "Studio Time," "Charcoal Mediums," and "Gallery Critiques." Because the model was trained on more traditional subjects, it didn't recognize the "Art vocabulary" as being academic in nature.

9 Conclusion

By combining the statistical power of Machine Learning with a common-sense Heuristic layer, we created a system that is 99.4% accurate and, perhaps more importantly, has a **0% False Positive rate** (it never let a "bad" document through). This tool allows academic researchers to process massive amounts of data with total confidence.