School Board Library: Document Classification

Here's how we can represent this in a classification_decision_tree.json file:

JSON

```
{
  "pdf": {
    "primary": "PDF",
    "secondary": {
      "use_cases": [
        "extract_text_from_pdf",
        "detect_images_in_pdf",
        "detect_tables_in_pdf"
      ]
    }
  },
  "docx": {
    "primary": "DOCX",
    "secondary": {
      "use_cases": [
        "analyze_docx_structure"
      ]
    }
  },
  "jpg": {
    "primary": "JPEG",
    "secondary": {
      "use_cases": [
        "analyze_image_content"
      ]
    }
  },
  "jpeg": {
    "primary": "JPEG",
    "secondary": {
      "use_cases": [
```

```
        "analyze_image_content"
      ]
    }
  },
  "png": {
    "primary": "PNG",
    "secondary": {
      "use_cases": [
        "analyze_image_content"
      ]
    }
  },
  "gif": {
    "primary": "GIF",
    "secondary": {
      "use_cases": [
        "analyze_image_content"
      ]
    }
  },
  "bmp": {
    "primary": "BMP",
    "secondary": {
      "use_cases": [
        "analyze_image_content"
      ]
    }
  },
  "tiff": {
    "primary": "TIFF",
    "secondary": {
      "use_cases": [
        "analyze_image_content",
        "analyze_tiff_for_multi_page"
      ]
    }
  },
  "pptx": {
    "primary": "PPTX",
    "secondary": {
      "use_cases": [
        "analyze_pptx_content"
      ]
    }
  },
  "txt": {
    "primary": "TXT",
    "secondary": {
```

```
      "use_cases": [
        "analyze_txt_content"
      ]
    }
  },
  "csv": {
    "primary": "CSV",
    "secondary": {
      "use_cases": [
        "analyze_csv_structure"
      ]
    }
  },
  "xlsx": {
    "primary": "XLSX",
    "secondary": {
      "use_cases": [
        "analyze_xlsx_content"
      ]
    }
  },
  "rtf": {
    "primary": "RTF",
    "secondary": {
      "use_cases": [
        "analyze_rtf_content"
      ]
    }
  },
  "html": {
    "primary": "HTML",
    "secondary": {
      "use_cases": [
        "analyze_html_structure"
      ]
    }
  },
  "xml": {
    "primary": "XML",
    "secondary": {
      "use_cases": [
        "analyze_xml_structure"
      ]
    }
  },
  "zip": {
    "primary": "ZIP",
    "secondary": {
```

```
      "use_cases": [
        "extract_and_classify_zip"
      ]
    }
  },
  "odt": {
    "primary": "ODT",
    "secondary": {
      "use_cases": [
        "analyze_odt_structure"
      ]
    }
  }
}
```

Explanation:

- **Top Level:** The top-level keys are the file extensions (without the dot), matching the primary classification.
- **primary:** This key still holds the primary classification string (e.g., "PDF", "DOCX").
- **secondary:** This key now contains a dictionary with a use_cases key.
- **use_cases:** This key holds a list of use case names (strings) that should be executed in order for the given file extension. These use case names map to the secondary classification functions we defined earlier.

Mapping Use Cases to Functions:

Here's how the use cases map to the functions (this mapping will be used in the code):

Python

```
use_case_to_function = {
    "extract_text_from_pdf": classify_pdf_for_text,
    "detect_images_in_pdf": classify_pdf_for_images,
    "detect_tables_in_pdf": classify_pdf_for_tables,
    "analyze_docx_structure": classify_docx_document,
    "analyze_image_content": classify_image_document,
    "analyze_tiff_for_multi_page": classify_tiff_as_multipage,
    "analyze_pptx_content": classify_pptx_document,
    "analyze_txt_content": classify_text_document,
    "analyze_csv_structure": classify_csv_document,
    "analyze_xlsx_content": classify_excel_document,
    "analyze_rtf_content": classify_rtf_document,
    "analyze_html_structure": classify_html_document,
    "analyze_xml_structure": classify_xml_document,
```

```
    "extract_and_classify_zip": classify_zip_contents,
    "analyze_odt_structure": classify_odt_document,
}
```

Updated classify_document Function (in the notebook):

Python

```
def classify_document(doc_id, doc_data, decision_tree,
use_case_to_function):
    \"\"\"Classifies a document based on the decision tree and use
cases.\"\"\"
    filepath = doc_data["filepath"]
    extension = get_file_extension(filepath)

    if extension not in decision_tree:
        logging.warning(f"No classification rule found for extension
'{extension}' (document ID: {doc_id})")
        return

    primary_classification = decision_tree[extension]["primary"]
    index[doc_id]["document_type"] = primary_classification
    index[doc_id]["metadata"]["document_type"] =
primary_classification

    secondary_use_cases =
decision_tree[extension]["secondary"]["use_cases"]

    # Execute use cases in order
    for use_case in secondary_use_cases:
        if use_case in use_case_to_function:
            classification_function = use_case_to_function[use_case]
            result = classification_function(filepath)
            if result:
                if isinstance(result, list):
                    for item in result:
                        index[doc_id]["metadata"][use_case] = item
                else:
                    index[doc_id]["metadata"][use_case] = result

                logging.info(f"  Use case '{use_case}' returned:
{result}")

    document_type = determine_document_type(doc_id, index,
```

```
primary_classification)
    index[doc_id]["document_type"] = document_type
    logging.info(f"Document {doc_id} classified as: {document_type}")

def determine_document_type(doc_id, index, primary_classification):
    \"\"\"Determines the final document type based on primary and
secondary classifications.\"\"\"
    metadata = index[doc_id]["metadata"]

    # Handling for PDF documents
    if primary_classification == "PDF":
        if metadata.get("classify_pdf_for_text"):
            if metadata.get("classify_pdf_for_images"):
                return "Text-with-Images" if not
metadata.get("classify_pdf_for_tables") else
"Text-with-Images-and-Tables"
            elif metadata.get("classify_pdf_for_tables"):
                return "Text-with-Tables"
            else:
                return "Text-Only"
        elif metadata.get("classify_pdf_for_images"):
            return "Image-Only"
        else:
            return "PDF-Unknown"

    # Handling for DOCX documents
    elif primary_classification == "DOCX":
        if metadata.get("analyze_docx_structure") ==
"DOCX-Text-with-Images-and-Tables":
            return "Text-with-Images-and-Tables"
        elif metadata.get("analyze_docx_structure") ==
"DOCX-Text-with-Images":
            return "Text-with-Images"
        elif metadata.get("analyze_docx_structure") ==
"DOCX-Text-with-Tables":
            return "Text-with-Tables"
        elif metadata.get("analyze_docx_structure") ==
"DOCX-Text-Only":
            return "Text-Only"
        else:
            return "DOCX-Unknown"

    # Handling for image documents (JPEG, PNG, GIF, BMP)
    elif primary_classification in ["JPEG", "PNG", "GIF", "BMP"]:
        return "Image-Only"

    # Handling for TIFF documents
    elif primary_classification == "TIFF":
```

```
        if metadata.get("classify_tiff_as_multipage"):
            return "Image-Only-Multi-Page"
        else:
            return "Image-Only-Single-Page"

    # Handling for PPTX documents
    elif primary_classification == "PPTX":
        if metadata.get("analyze_pptx_content") ==
"PPTX-Text-with-Images":
            return "Text-with-Images"
        elif metadata.get("analyze_pptx_content") == "PPTX-Text-Only":
            return "Text-Only"
        elif metadata.get("analyze_pptx_content") ==
"PPTX-Image-Only":
            return "Image-Only"
        else:
            return "PPTX-Unknown"

    # Handling for TXT, CSV, XLSX, RTF, HTML, XML, ZIP, ODT documents
    # These are simplified for brevity, assuming basic content
analysis is sufficient
    elif primary_classification in ["TXT", "CSV", "XLSX", "RTF",
"HTML", "XML", "ZIP", "ODT"]:
        return "Text-Only"  # You can refine this based on specific
analysis results

    else:
        return primary_classification  # Default to primary
classification if no specific logic is defined
```

Advantages of this Structure:

- **Clearer Decision Flow:** The decision tree structure is more explicit, making it easier to understand the classification process.
- **Easier Modification:** Adding or changing classification rules is simpler. You just need to update the classification_decision_tree.json file.
- **More Flexible:** You can easily add more complex logic (e.g., nested conditions) within the secondary section if needed.

Next Steps:

1. **Update Notebook:** Replace the old decision_tree loading and the classify_document function in your notebook with the code provided above. Also, add the use_case_to_function mapping to your notebook.
2. **Implement Secondary Classification Functions:** Implement the placeholder secondary classification functions based on the tools and examples I provided in the previous responses.
3. **Test:** Thoroughly test the updated classification logic with various document types to

ensure it's working as expected.

This more structured approach will make your document classifier more robust, maintainable, and easier to extend in the future!

Use Cases and Recommended Classifiers

Use Case	Description	Best Classifier Function
1. Extract Text from PDF	Extract text content from PDF files to determine if the PDF is text-based, and potentially identify its purpose (e.g., meeting minutes, report, etc.).	classify_pdf_for_text
2. Detect Images in PDF	Determine if a PDF contains images, suggesting it might be a scanned document, a presentation, or a document with embedded graphics.	classify_pdf_for_images
3. Detect Tables in PDF	Identify the presence of tables in a PDF, which often indicates structured data or tabular information.	classify_pdf_for_tables
4. Analyze DOCX Structure	Analyze the structure of a DOCX file to identify text, images, and tables, and potentially determine the document's purpose (e.g., report, letter, form).	classify_docx_document
5. Analyze Image	(Placeholder for future)	classify_image_document

Use Case	Description	Best Classifier Function
Content	Analyze the content of image files (JPEG, PNG, GIF, BMP) to determine if they are photographs, diagrams, or scanned documents.	
6. Analyze TIFF for Multi-Page	Determine if a TIFF image is multi-page, which might indicate a scanned multi-page document.	classify_tiff_as_multipage
7. Analyze PPTX Content	Extract text and images from PPTX slides to classify the presentation as Text-Only, Text-with-Images, or Image-Only.	classify_pptx_document
8. Analyze TXT Content	Analyze the content of a plain text file to potentially identify keywords or patterns that suggest the document's purpose (e.g., meeting minutes, agenda, policy document).	classify_text_document
9. Analyze CSV Structure	Analyze the structure of a CSV file, looking at headers or data patterns to determine the type of data it contains (e.g., financial data, student records, etc.).	classify_csv_document
10. Analyze XLSX Content	Analyze the content and structure of an XLSX file (e.g., cell values, formulas, formatting) to understand if it's primarily tabular data or contains other elements.	classify_excel_document

Use Case	Description	Best Classifier Function
11. Analyze RTF Content	Analyze the content of an RTF file to extract text and potentially identify basic structural elements.	classify_rtf_document
12. Analyze HTML Structure	Parse and analyze the structure of an HTML file to identify elements like tables, headings, or specific content that might indicate the document's purpose.	classify_html_document
13. Analyze XML Structure	Parse and analyze the structure of an XML file to identify elements and data relevant to the school board domain.	classify_xml_document
14. Extract and Classify ZIP	Extract the contents of a ZIP archive and then classify each individual file within the archive using the appropriate classification functions.	classify_zip_contents
15. Analyze ODT Structure	Analyze the structure of an ODT file to identify text, images, and tables, similar to DOCX analysis.	classify_odt_document

Document Extension to Use Case Mapping

Here's how document extensions map to the use cases and their corresponding classification functions:

File Extension	Primary Classification	Use Cases	Classifier Functions
.pdf	PDF	Extract Text from PDF, Detect Images in PDF, Detect Tables in PDF	classify_pdf_for_text, classify_pdf_for_images, classify_pdf_for_tables
.docx	DOCX	Analyze DOCX Structure	classify_docx_document
.jpg, .jpeg	JPEG	Analyze Image Content	classify_image_document
.png	PNG	Analyze Image Content	classify_image_document
.gif	GIF	Analyze Image Content	classify_image_document
.bmp	BMP	Analyze Image Content	classify_image_document
.tiff	TIFF	Analyze Image Content, Analyze TIFF for Multi-Page	classify_image_document, classify_tiff_as_multipage
.pptx	PPTX	Analyze PPTX Content	classify_pptx_document
.txt	TXT	Analyze TXT Content	classify_text_document
.csv	CSV	Analyze CSV Structure	classify_csv_document
.xlsx	XLSX	Analyze XLSX Content	classify_excel_document

File Extension	Primary Classification	Use Cases	Classifier Functions
.rtf	RTF	Analyze RTF Content	classify_rtf_document
.html	HTML	Analyze HTML Structure	classify_html_document
.xml	XML	Analyze XML Structure	classify_xml_document
.zip	ZIP	Extract and Classify ZIP	classify_zip_contents
.odt	ODT	Analyze ODT Structure	classify_odt_document

Implementation Notes:

- The table above provides a clear roadmap for implementing the secondary classification functions.
- You can prioritize the implementation based on the file types most frequently encountered in your school board data.
- Remember that the classify_image_document function is a placeholder for now. You can add more sophisticated image analysis logic later if needed.

Next Steps:

1. **Confirm:** Please review this use case mapping and the recommended classifier functions. Let me know if you have any questions or want to make adjustments.
2. **Code Review:** If you have started implementing any of the classification functions, feel free to share the code, and I can provide feedback and suggestions.
3. **Implementation:** We can continue implementing the remaining classification functions based on your priorities.

I'm ready to proceed with the implementation or any further refinements as soon as you give the go-ahead!