Machine Learning on CMS Plan Benefit Package (PBP)

Data is like a treasure chest, brimming with hidden gems and a few tricky surprises. With machine learning guiding your journey, you can unlock these treasures, analyzing vast datasets with ease. The right model can spot patterns and detect anomalies in a flash—100 times faster than manually sifting through data. That’s how I used a decision tree model to solve the data puzzle!

# Let’s talk about the data

The Center for Medicare & Medicaid Services (CMS) Plan Benefit Package (PBP) outlines benefits for Medicare Advantage (MA) or Prescription Drug Plan (PDP) plans. Organizations and plan sponsors submit PBPs to CMS for approval before marketing and benefit analysis. CMS reviews, approves, and publishes this data on cms.gov, making it available in a public [plan comparing tool](https://www.medicare.gov/plan-compare/).

## Too many small columns in multiple data files

The data is quite complex, with too many columns to easily summarize each benefit. According to the Q1 2024 PBP data dictionary, there are 33,814 data columns supporting 180 medical benefits across Medicare and Non-Medicare services, and Prescription (RX) benefits. For example, inpatient hospital coverage (service category 1a) is a Medicare service and the service has 612 data columns spread across six files. This complexity makes it challenging to identify business logic relationships and determine the importance of each data field for generating text.

# Goal

I want to identify the data patterns of the medical benefits, categorize the benefits into groups, and implement functions to generate human-readable cost shares for the benefit categories.

# Pack up for your journey

## Web crawler

To match the benefit text functions with the public site, I built a web crawler to extract the benefit text from the public site. (Sorry, CMS!) The crawler reads the HTML, parses it, and generates a dataset for over 3,000 plans. This dataset serves as the target label for training the decision tree model and verifying the output results of the decoding functions.

## Choosing the correct tool

I needed a machine learning model that could identify related data columns among many and determine their weight or priority. Using these results, I aimed to build a text-generating function for each benefit.

### Decision Tree

The [Decision Tree](https://www.geeksforgeeks.org/decision-tree-introduction-example/) model stood out as a promising choice. It uses Gini impurity, or Gini ratio, to measure how mixed a dataset is. This helps quantify the cleanliness or messiness of the data, especially useful for classification tasks. The model also creates a decision tree diagram, which I can use to write functions.

# Step by Step: Navigate your path

For each benefit, I created a separate sample data file targeting that specific benefit for the decision tree model. I then generated the code block based on the model’s execution results. Let me show you how I created a decision tree for Hearing Aids – all types. The service code for this benefit is 18b1(NMC).

## Prepare sample dataset

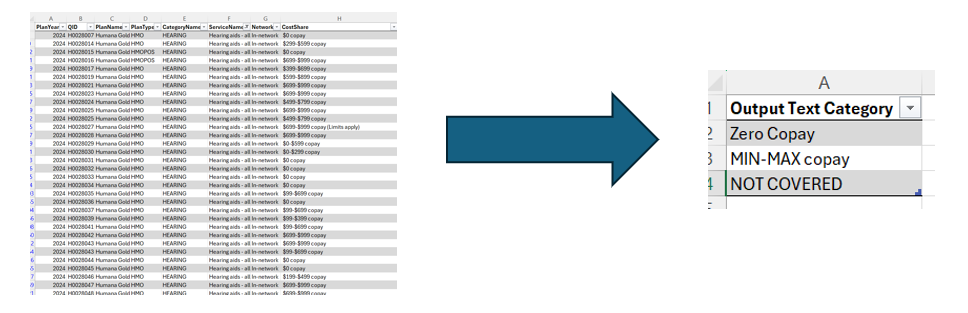
To run the decision tree model, you need sample data. The sample data file should contain features (PBP data columns related to the benefits) and a target label (benefit cost share displayed on the public site).

1. **Query PBP and Crawled Data**

To avoid [noise](https://medium.com/mitb-for-all/what-is-noise-676be449f752), I queried the crawled data for a single benefit and created a dataset for that benefit. Using the category name (HEARING), service name (Hearing aids - all types), and network (In-network), I obtained a dataset with 3,459 rows. I then queried the PBP data columns related to the service code in all data files. Using the data dictionary, I located the data columns for service codes 18b1(NMC) and 18b. Unsure of the relationship between 18b1(NMC) and 18b, I selected data columns for both service codes. This query generated a PBP dataset with 5,728 rows. I joined the Plan Identifier in both datasets into dataset 3,458 rows.

1. **Categorize the target value**

The decision tree model doesn’t perform well with too many leaves. To address this, I apply regular expressions to categorize the target values in the CostShare column. Each CostShare value category is then appended to a new column named Classified\_CostShare. The diagram below shows that the 18b1 benefit’s CostShare values are categorized into three groups: Zero Copay, MIN-MAX Copay, and NOT COVERED.



1. **Remove duplicates**

I removed all unnecessary columns, such as plan names and other plan-specific columns, from the dataset to isolate the feature columns and the Classified\_CostShare (target label). After removing duplicates, the sample dataset was reduced to 120 rows, each with a unique combination of features and target labels. The sample dataset for the benefit is saved for later use.

Now, the data file containing features (PBP data columns related to the benefits) and the target label (categorized benefit cost) is ready to run the Decision Tree.

## Run Decision Tree

The Decision Tree model is trained using the sample dataset. The model training produces a diagram that highlights the pivot columns and conditions in the dataset used to generate the target labels. I use the columns and conditions listed in the diagram to analyze the sample data and implement the pseudo code for the target label output.

## Implement pseudo code

The diagram below shows how 119 samples are distributed and identifies two data fields: pbp\_b18b\_copay\_at\_max\_amt (gini=0.526) and pbp\_b18b\_bendesc\_yn (gini=0.31). These features have the highest Gini indices, resulting in the target labels. The features and conditions can be translated into pseudo code.

A computer screen shot of a computer screen

Description automatically generated

### Don’t copy and paste: *Do your homework*

The decision tree diagram highlights the data columns with the highest Gini index to reach the leaf nodes (classes). It does not accurately represent data definitions or business logic within the features. Some features are omitted because they have static values. Therefore, you should not simply translate the diagram. Instead, follow these steps:

1. Review the data with the columns in the sample file.
2. Rewrite the if-else conditions based on the data dictionary.
3. Reorganize the if-else conditions.
4. Set the default class to return.

The diagram shows two features: pbp\_b18b\_bendesc\_yn and pbp\_b18b\_copay\_at\_max\_amt. These are the findings in sample dataset and data dictionary.

* **pbp\_b18b\_bendesc\_yn**: integer value [1: Yes, 2: No]
  + pbp\_b18b\_bendesc\_yn = 1: Zero Copay OR MIN-MAX copay
  + pbp\_b18b\_bendesc\_yn = 2: NOT COVERED
* **pbp\_b18b\_copay\_at\_max\_amt**: decimal value or None
  + pbp\_b18b\_copay\_at\_max\_amt is None, Zero Copay.
  + pbp\_b18b\_copay\_at\_max\_amt = 0, Zero Copay
  + pbp\_b18b\_copay\_at\_max\_amt > 0: MIN-MAX copay
    - pbp\_b18b\_bendesc\_yn is always 1

Using these findings, I built the following pseudo code for 18b1. As you can see, the if-else conditions and the order of the data columns in the if-else conditions are quite different from the diagram. A screenshot of a computer

Description automatically generated

# Generalize the psuedo code for multiple benefit

The Decision Tree model generates different diagrams and pseudo code for each benefit. Doing this for every individual benefit could be very time-consuming and tiring. I want to reuse the pseudo code to decode multiple benefits with the same data structure.

The diagram below shows the execution result for the Hearing Exam benefit 18a1 (NMC). As you can see, the diagram uses only one feature (pbp\_b18a\_bendesc\_ehc) with a very low Gini index (0.089), instead of the two features (pbp\_b18b\_bendesc\_yn and pbp\_b18b\_copay\_at\_max\_amt) used for 18b1 (NMC).

A close-up of several colorful rectangular objects

Description automatically generated

However, 18a1(NMC) and 18b1(NMC) are Non-Medicare (NMC) benefits with the same benefit cost structure. The datasets have the same number of data columns in the same data files. The column names are slightly different column names by naming convention rules.

To build a pseudo code for 18a1 (NMC) and 18b1 (NMC), I need to create a dataset with unified features and target labels for both benefits. After inspecting the feature columns, I found discrepancies in naming conventions and renamed the feature names to common names.

## Standardizing Column Names

### Service Code Prefix:

### The column names in the pbp data start with the prefix pbp\_b followed by the service code. I removed this prefix to standardize the column names across both datasets. For example, pbp\_b18a\_bendesc\_yn and pbp\_b18b\_bendesc\_yn are renamed to bendesc\_yn. Similarly, pbp\_b18a\_bendesc\_ehc and pbp\_b18b\_bendesc\_ehc are renamed to bendesc\_ehc.

### Custom Column Names:

Some column names contain extra words describing specific benefits and usage of the data fields. Using the data dictionary, I identified matching fields and standardized their names. For example, pbp\_b18b\_copay\_at\_max\_amt (maximum copay amount for 18b1) and pbp\_b18a\_copay\_amt\_max\_rht (maximum copay amount for 18a1) are renamed to copay\_max.

After renaming features, I merged the both datasets and ran the decision tree model on the new sample data. Interestingly, the generated diagram was similar the diagram of 18b1 (NMC), and the generated code block was identical to the code block of 18b1 (NMC).

A computer program with a blue arrow

Description automatically generated with medium confidence

#### Why?

My expectation was that the diagram would evolve with new conditions involving bendesc\_ehc. After reviewing the data sample of 18a1, I found that the pbp\_b18a\_desc\_ehc condition could be satisfied by pbp\_b18a\_bendesc\_yn and pbp\_b18a\_copay\_amt\_max\_rht.

* Zero Copay
  + pbp\_b18a\_desc\_ehc is 1 or 11
  + pbp\_b18a\_bendesc\_yn is 1 and *pbp\_b18a\_copay\_amt\_max\_rht* is 0 or None.
* Not Covered
  + pbp\_b18a\_desc\_ehc is None
  + pbp\_b18a\_bendesc\_yn is 2 and *pbp\_b18a\_copay\_amt\_max\_rht is None*

The features from 18b1 might be more dominant or have a stronger influence on the decision tree model. If pbp\_b18a\_desc\_ehc conditions are already well-represented by pbp\_b18b\_bendesc\_yn and pbp\_b18b\_copay\_amt\_max\_rht, the model might not need to create new splits based on bendesc\_ehc.

# Test the pseudo code

I converted the pseudocode into a function and implemented the cost share using it. For 18a1 and 18b1, the function only requires three parameters instead of all 74A computer screen shot of a blue arrow

Description automatically generated

The implemented data is compared against the crawled data. If there is no mismatch, we created a function for NMC services 18a1 and 18b1. Outdated data could cause mismatches in a few plans. CMS releases PBP data every quarter, and carriers or plan sponsors can request changes anytime. The data on the public site is updated by these requests, but the released PBP data may not yet reflect these changes. If the mismatch is caused by obsolete data, I removed the mismatching data from the crawled dataset for further testing.

# Add new services

Instead of starting from scratch to build a sample dataset for the service, I chose the appropriate functions and use the function to implement the benefit costs. Since the functions are general enough to be reused for multiple services, it was easy to locate the data fields for the function parameters. The implemented output is compared to the crawled data, and I repeated this process until there is no mismatch.

### Mismatches

The mismatches indicate that the function is missing some business logic or features, or it is not suitable to support the service. I compared the sample dataset of the service to the data samples of other services that use the function and ran the decision tree model. The new decision tree diagram could introduce new features or new if-else conditions on existing parameters.

#### New features

I checked that the new features are available in existing datasets. If the matching features only exist in the dataset for the new service, the function should be branched out for the new service with the additional parameters. If the feature is available for sample datasets for all services, the feature should be added to the function parameters, and the business logic for the parameter should be added to the new function parameter to support both new and existing services.

##### New if-else conditions

I review the sample dataset using if-else conditions condition and find data pattern created by the new if-else condition. If the data pattern works along with other services, I updated the function to support new if-else conditions. If not, I branch out the function and use the new function to support the new service.

# Treasure chest is open

Using the repeated process, I was able to build five functions that generate benefit costs for the following services:

|  |  |
| --- | --- |
| Function name | Service Codes |
| Benefit\_MC\_Tiers | 1a, 2 |
| Benefit\_MC | 4a, 4b, 7a, 7c, 7d, 7i, 7j, 11a, 15\_1\_I |
| Benefit\_MC\_EHC | 7b, 7e1, 7e2, 7f, 7f(NMC), 7h1, 7h2, 8a1, 7a2, 8a1, 8a2, 8b3, 9a1, 10a1, 11b1, 11c1, 15\_2, 15\_3, 17b, 18a |
| Benefit\_NMC | 7b1, 7b2, 13a, 13b, 13c, 14c4, 14c7, 14c10, 14c11, 14c21, 14c22, 16a1, 16a2, 16a3, 16a4, 17a1, 17b1, 17b2, 17b3, 17b4, 17b5, 18a1, 18a2, 18b1, 18b2, 18b3, 18b4 |
| Preventive services | 14a |

This is the result of two weeks with one person. This process used to be a month-long task that required multiple individuals, including data engineers, data analysts, QA, and business knowledge holders who could explain each service to the data engineers. Amazing!