فاز 2

کیانا کرمانی

بهتر است در فاز 1 برای پر کردن داده‌های ناقص، از روش‌های مختلفی استفاده کنیم:

1. حذف داده‌هایی که مقادیر ناقص دارند: این روش ممکن است اطلاعات مهمی را از دست بدهد و باید با احتیاط استفاده شود.

2. استفاده از میانگین یا مد: برای داده‌های عددی، می‌توانیم میانگین یا مد را به عنوان مقدار جایگزین استفاده کنیم. باید توجه داشت که این روش فقط برای داده‌های عددی قابل استفاده است.

3. استفاده از مد برای داده‌های غیر عددی: در این حالت برای داده‌هایی که نوع عددی ندارند، از مد استفاده می‌کنیم. این نیاز به جدا کردن داده‌های عددی و غیرعددی را دارد.

4. استفاده از روش forward / backward: این روش‌ها بر اساس مقادیر مجاور یا پسین داده‌های ناقص، مقادیر جایگزین را تخمین می‌زنند.

در هر یک از این روش‌ها، مدل‌هایی که برای تولید داده‌های جایگزین استفاده می‌شوند، باید ذخیره شوند تا در صورت نیاز به آنها مراجعه شود.

در نهایت برای مقایسه این که کدام روش بهتر هست یک کد مقایسه کردن نوشتم .

هر خروجی که گرفتیم رو با دیتاست اصلی مقایسه میکنیم: تابع summarize_data را برای محاسبه آمار خلاصه (میانگین، میانه، واریانس) برای یک DataFrame اصلی تعریف می کند.

محاسبه آمار اصلی بر روی دیتاست اصلی : آمار خلاصه (میانگین، میانه، واریانس) مجموعه داده اصلی را محاسبه می کند.

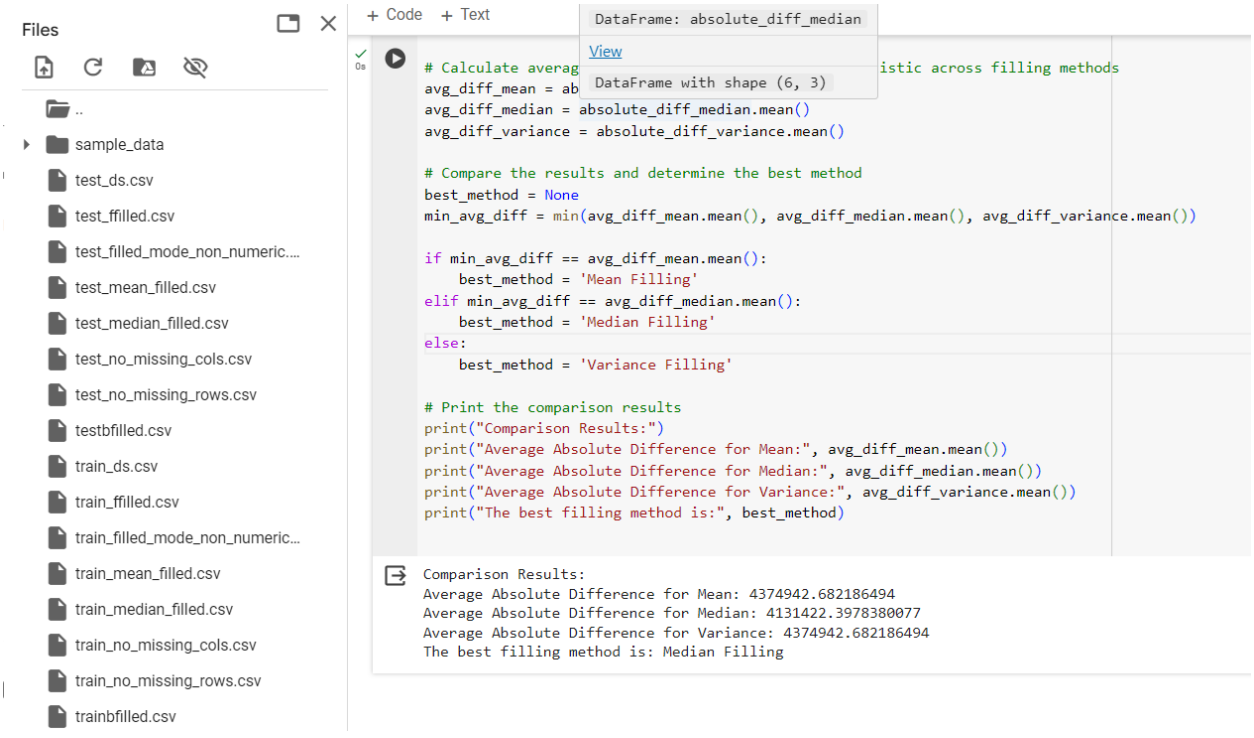
Summarize Filled Datasets: هر مجموعه داده پر شده را با محاسبه آمار خلاصه آن خلاصه می کند.

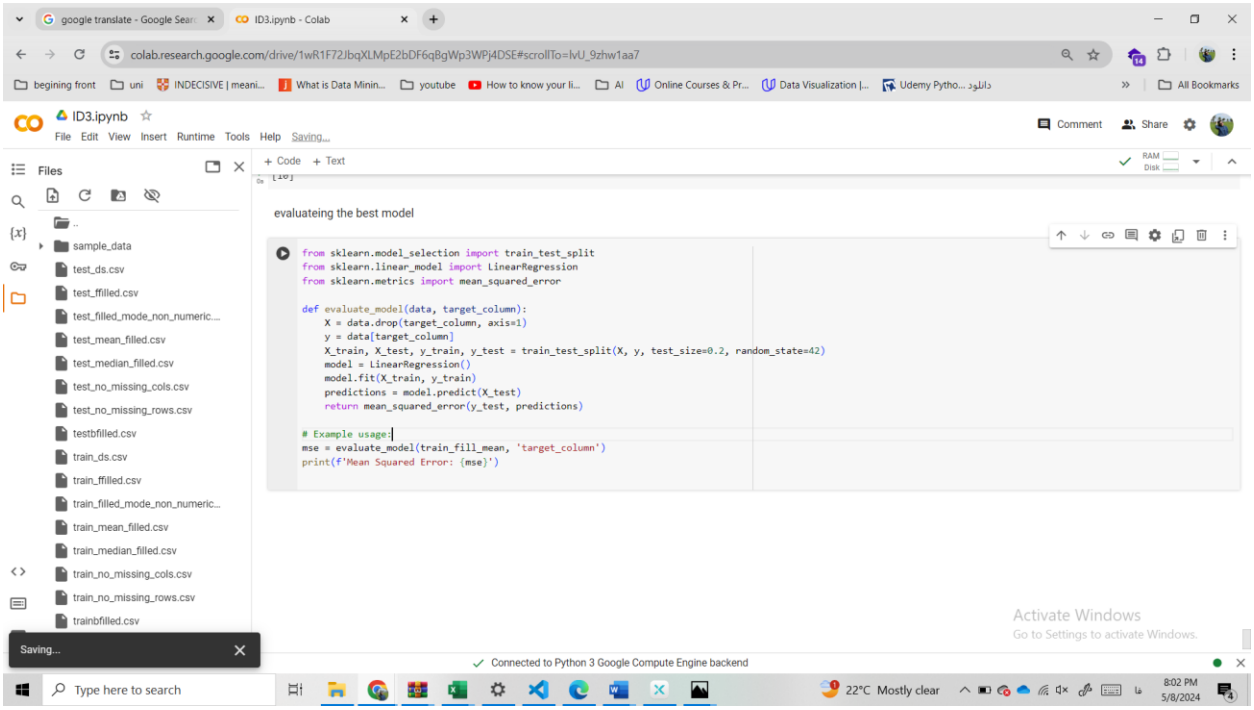
محاسبه تفاوت مطلق: تفاوت مطلق بین آمار خلاصه مجموعه داده اصلی و هر مجموعه داده پر شده را محاسبه می کند.

محاسبه میانگین تفاوت مطلق: میانگین تفاوت مطلق را برای هر آمار (میانگین، میانه، واریانس) در تمام روش های پر کردن محاسبه می کند.

تعیین بهترین روش: بهترین روش پر کردن را بر اساس کمترین میانگین تفاوت مطلق در تمام آمارها تعیین می کند.

نتایج مقایسه چاپ: نتایج مقایسه شامل میانگین تفاوت مطلق میانگین، میانه و واریانس و بهترین روش پر کردن را چاپ می کند.


Files
+ Code + Text
DataFrame: absolute_diff_median
View
DataFrame with shape (6, 3)
# Calculate averag istic across filling methods
avg_diff_mean = ab
avg_diff_median = absolute_diff_median.mean()
avg_diff_variance = absolute_diff_variance.mean()
# Compare the results and determine the best method
best_method = None
min_avg_diff = min(avg_diff_mean.mean(), avg_diff_median.mean(), avg_diff_variance.mean())
if min_avg_diff == avg_diff_mean.mean():
best_method = 'Mean Filling'
elif min_avg_diff == avg_diff_median.mean():
best_method = 'Median Filling'
else:
best_method = 'Variance Filling'
# Print the comparison results
print("Comparison Results:")
print("Average Absolute Difference for Mean:", avg_diff_mean.mean())
print("Average Absolute Difference for Median:", avg_diff_median.mean())
print("Average Absolute Difference for Variance:", avg_diff_variance.mean())
print("The best filling method is:", best_method)
sample_data
test_ds.csv
test_ffilled.csv
test_filled_mode_non_numeric....
test_mean_filled.csv
test_median_filled.csv
test_no_missing_cols.csv
test_no_missing_rows.csv
testbfilled.csv
train_ds.csv
train_ffilled.csv
train_filled_mode_non_numeric...
train_mean_filled.csv
train_median_filled.csv
train_no_missing_cols.csv
train_no_missing_rows.csv
trainbfilled.csv
Comparison Results:
Average Absolute Difference for Mean: 4374942.682186494
Average Absolute Difference for Median: 4131422.3978380077
Average Absolute Difference for Variance: 4374942.682186494
The best filling method is: Median Filling


ID3.ipynb - Colab
colab.research.google.com/drive/1wR1F72JbqXLMpE2bDF6qBgWp3WPj4DSE#scrollTo=lvU_9zhw1aa7
ID3.ipynb
File Edit View Insert Runtime Tools Help Saving...
Comment
Share
Files
+ Code + Text
evaluateing the best model
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
def evaluate_model(data, target_column):
X = data.drop(target_column, axis=1)
y = data[target_column]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
return mean_squared_error(y_test, predictions)
# Example usage:
mse = evaluate_model(train_fill_mean, 'target_column')
print(f'Mean Squared Error: {mse}')
Saving...
Connected to Python 3 Google Compute Engine backend
Activate Windows
Go to Settings to activate Windows.
Type here to search
22°C Mostly clear
8:02 PM
5/8/2024

فاز دوم:

1. تعریف کلاس گره:

- در کلاس "Node" هر گره در درخت تصمیم را نشان می‌دهیم، که ویژگی‌هایی مانند ویژگی برای تقسیم (به عنوان "ویژگی")، مقدار آستانه برای ویژگی‌های پیوسته (به عنوان "آستانه")، کلاس یا مقدار پیش‌بینی شده در یک گره برگ (به عنوان "مقدار")، و اشاره به گره‌های فرزند چپ و راست (به عنوان "چپ" و "راست") را ذخیره می‌کند.

2. توابع آنتروپی و IG:

- تابع "entropy": این تابع آنتروپی یک مجموعه از برچسب‌ها را محاسبه می‌کند که معیاری برای تصادفی بودن یا غیرقابل پیش‌بینی بودن داده‌ها است. آنتروپی کمتر به معنای عدم قطعیت کمتر است.

- تابع "information_gain": این تابع IG حاصل از تقسیم یک مجموعه داده به دو بخش را بر اساس یک ویژگی خاص محاسبه می‌کند. این اندازه‌گیری می‌کند که چقدر عدم اطمینان در برچسب‌ها پس از تقسیم کاهش می‌یابد.

3. عملکرد پیش‌پردازش داده:

- تابع "preprocess_data": این تابع مجموعه داده را برای مدل‌سازی آماده می‌کند. مقادیر از دست رفته در ستون‌های عددی را با میانه آنها و در ستون‌های طبقه‌بندی شده با حالت آنها پر می‌کند. همچنین متغیرهای طبقه‌بندی را با استفاده از "LabelEncoder" در کدهای عددی رمزگذاری می‌کند و آنها را برای الگوریتم درخت تصمیم مناسب می‌سازد.

4. بهترین عملکرد تقسیم‌بندی:

- تابع "find_best_split": این تابع هر ویژگی در مجموعه داده را ارزیابی می‌کند تا مشخص کند کدام ویژگی و آستانه (برای ویژگی‌های پیوسته) حداکثر IG را ارائه می‌دهد. روی تمام مقادیر ممکن (برای ویژگی‌های طبقه‌بندی) تکرار می‌شود یا آستانه‌های بین مقادیر (برای ویژگی‌های پیوسته) را محاسبه می‌کند تا بهترین تقسیم را پیدا کند.

5. عملکرد ساختار درخت تصمیم:

- تابع "build_decision_tree": این یک تابع بازگشتی است که درخت تصمیم را با انتخاب بهترین ویژگی برای تقسیم داده‌ها در هر گره (با استفاده از "find_best_split") می‌سازد. برای هر نقطه تصمیم یک "گره" ایجاد می‌کند و تا زمانی که به حداکثر عمق مشخص‌شده برسد یا دسترسی به اطلاعات بیشتری امکان‌پذیر نباشد، به تقسیم شدن ادامه می‌دهد. برای گره‌هایی که همه نمونه‌ها به یک کلاس یا سایر موارد پایه تعلق دارند، یک گره برگ با رایج‌ترین برچسب کلاس ایجاد می‌کند.

6. بخش اجرا:

- این قسمت مجموعه داده را بارگذاری می کند، آن را با استفاده از «preprocess_data» پیش پردازش می کند، و سپس «build_decision_tree» را فراخوانی می کند تا درخت تصمیم را بر اساس ویژگی «Manufacturer» به عنوان متغیر هدف بسازد. پارامتر "max_depth" عمق درخت را برای جلوگیری از برازش بیش از حد محدود می کند.

جواب سوالات:

ویژگی‌های پیوسته: کد ویژگی‌های پیوسته را با محاسبه آستانه بهینه برای هر نقطه تقسیم بالقوه بر اساس افزایش اطلاعات کنترل می‌کند. این رویکرد به طور موثر داده های پیوسته را بر اساس اینکه آیا آنها زیر آستانه یا بالاتر از آستانه هستند، به دو دسته تقسیم می کند.

- حداکثر مقادیر آنتروپی و سود: آنتروپی زمانی به حداکثر می رسد که داده ها بیشترین ترکیب را داشته باشند (همه کلاس ها به یک اندازه محتمل هستند)، و gain اطلاعات زمانی به حداکثر می رسد که یک تقسیم به طور کامل کلاس ها را به زیر مجموعه های خالص جدا کند.

1. ساختمان درختی بازگشتی (تابع build_decision_tree):

- درخت تصمیم به صورت بازگشتی با انتخاب ویژگی و آستانه ای ساخته می شود که حداکثر اطلاعات را در هر گره به دست می آورد. این فرآیند بازگشتی از ریشه شروع می‌شود و تا زمانی ادامه می‌یابد که همه داده‌های یک گره برچسب کلاس یکسانی داشته باشند یا معیار توقف دیگری مانند رسیدن به حداکثر عمق ('max_depth') برآورده شود.

- در هر گره، الگوریتم تمام ویژگی هایی را که هنوز در مسیر ریشه به گره استفاده نشده اند، بررسی می کند. این امر از استفاده مجدد از ویژگی ها در یک مسیر، جلوگیری از چرخه ها و اطمینان از استفاده از ویژگی های متنوع در قسمت های مختلف درخت جلوگیری می کند.

مدیریت ویژگی های پیوسته

2. آستانه بهینه برای ویژگی های پیوسته:

- انتخاب تقسیمات:

- برای هر ویژگی پیوسته، مقادیر منحصر به فرد مرتب شده و آستانه های بالقوه ارزیابی می شوند. این آستانه ها معمولاً به عنوان نقطه میانی بین مقادیر متوالی منحصر به فرد انتخاب می شوند.

- این رویکرد گسسته‌سازی تضمین می‌کند که الگوریتم می‌تواند هر روش ممکن برای تقسیم داده‌ها به دو گروه را ارزیابی کند، که در آن یک گروه دارای مقادیر کمتر یا مساوی با آستانه و گروه دیگر دارای مقادیر بیشتر از آستانه است.

- ارزیابی تقسیمات:

- برای هر آستانه، مجموعه داده به دو زیر مجموعه تقسیم می شود. آنتروپی برای هر زیرمجموعه محاسبه شده و برای محاسبه سود کلی اطلاعات از ایجاد آن تقسیم استفاده می شود.

- آستانه ای که منجر به بالاترین کسب اطلاعات می شود، به عنوان نقطه بهینه برای تقسیم داده های آن ویژگی در آن گره انتخاب می شود.

3. حداکثر مقادیر:

- حداکثر آنتروپی:

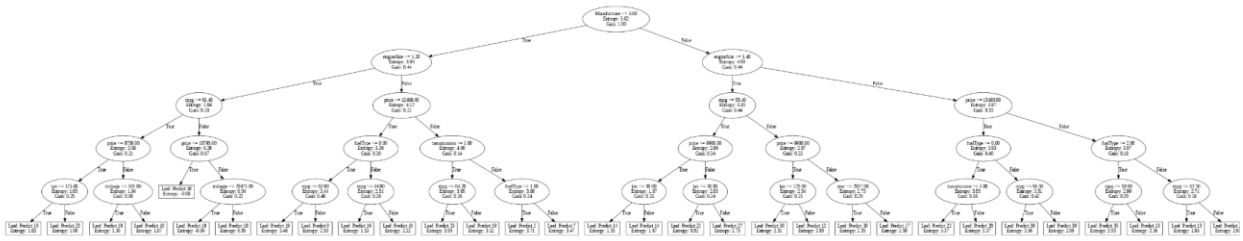
- آنتروپی زمانی به حداکثر مقدار خود می رسد که داده های درون یک گره کاملاً مخلوط شوند، به این معنی که کلاس ها به یک اندازه محتمل هستند. برای طبقه‌بندی باینری، این حداکثر آنتروپی 1 (بر حسب بیت) است که زمانی حاصل می‌شود که 50 درصد داده‌ها به یک کلاس و 50 درصد به کلاس دیگر تعلق داشته باشد.

- حداکثر افزایش اطلاعات:

- IG اطلاعات زمانی به حداکثر می رسد که یک تقسیم منجر به کاهش قابل توجه آنتروپی شود. حداکثر IG زمانی اتفاق می‌افتد که زیر مجموعه‌های ایجاد شده توسط تقسیم کاملاً خالص باشند (یعنی تمام نمونه‌های هر زیر مجموعه به یک کلاس تعلق دارند). در این سناریو، IG اطلاعات برابر با آنتروپی مجموعه اولیه قبل از تقسیم است، زیرا آنتروپی هر زیر گروه پس از تقسیم 0 است.

در کد ، هر فراخوان بازگشتی به «build_decision_tree» همه ویژگی‌های استفاده نشده را در نظر می‌گیرد، همه تقسیم‌های ممکن (آستانه برای ویژگی‌های پیوسته) را ارزیابی می‌کند و IG اطلاعات را برای این تقسیم‌ها محاسبه می‌کند.

- ویژگی و آستانه ای که IG را به حداکثر می رساند برای تقسیم داده ها در گره فعلی انتخاب می شود. اگر هیچ IG ای ممکن نباشد (یعنی حداکثر IG بی نهایت منفی باقی بماند)، گره به برگه ای با رایج ترین مقدار کلاس در میان نقاط داده باقیمانده تبدیل می شود.



فاز 3:

وقوع مشکل :

برازش بیش از حد در درخت‌های تصمیم زمانی اتفاق می‌افتد که مدل بیش از حد پیچیده می‌شود و شروع به گرفتن نویز در داده‌ها می‌کند نه فقط الگوی اصلی واقعی. این معمولا زمانی اتفاق می افتد که درخت اجازه دارد بدون محدودیت رشد کند تا زمانی که تمام نمونه های آموزشی را به طور کامل طبقه بندی کند. در اینجا دلیل این است که این می تواند به ویژه برای درخت های تصمیم مشکل ساز باشد:

1. درختان عمیق: درختانی که در عمق رشد می کنند تمایل به یادگیری الگوهای بسیار نامنظم دارند که تعمیم پذیری مدل را کاهش می دهد.

2. اندازه برگ: اگر هر برگ درخت در نهایت تعداد بسیار کمی از نمونه های آموزشی را نشان دهد، درخت بسیار مختص به داده های آموزشی می شود.

3. تقسیمات پیچیده: داشتن تقسیمات زیاد ممکن است به این معنی باشد که مدل شروع به گرفتن نکات جزئی می کند، که فراتر از مجموعه داده آموزشی تعمیم نمی یابد.

Overfitting

برای پرداختن به بیش از حد برازش در درختان تصمیم، چندین استراتژی را می توان به کار گرفت:

1. هرس:

- پیش هرس (توقف زودهنگام): رشد درخت را قبل از طبقه بندی کامل داده های آموزشی متوقف کنید. این را می توان با تنظیم محدودیت هایی بر روی پارامترهایی مانند حداکثر عمق ('max_depth')، حداقل تعداد نمونه های مورد نیاز در یک گره برگ ('min_samples_leaf') یا حداقل افزایش مورد نیاز برای ایجاد یک تقسیم ('min_impurity_decrease') به دست آورد.

- پس از هرس: به درخت اجازه می دهد تا به عمق کامل خود رشد کند و سپس شاخه هایی را که قدرت کمی در پیش بینی متغیرهای هدف دارند حذف کند. این کار با ارزیابی بهبود خطای پیش‌بینی در زمانی که شاخه‌ها هرس می‌شوند، انجام می‌شود.

2. تنظیم محدودیت برای رشد درخت:

- محدود کردن حداکثر عمق درخت ('max_depth'): یک راه ساده و موثر برای جلوگیری از پیچیده شدن بیش از حد درخت. درخت کم عمق کمتر رسا است و در نتیجه کمتر به نویز تناسب دارد.

- افزایش Minimum Sample Split ('min_samples_split'): حداقل تعداد نمونه هایی را که یک گره باید قبل از تقسیم داشته باشد را مشخص می کند. مقادیر بالاتر از یادگیری الگوهای بسیار ریز در مدل جلوگیری می کند، بنابراین قابل تعمیم تر است.

- افزایش حداقل نمونه ها در گره های برگ ('min_samples_leaf'): این تضمین می کند که هر برگ بیش از تعداد معینی نمونه دارد، که با جلوگیری از تصمیم گیری هر برگ بر اساس نمونه های بسیار کمی، پیش بینی های مدل را هموارتر می کند.

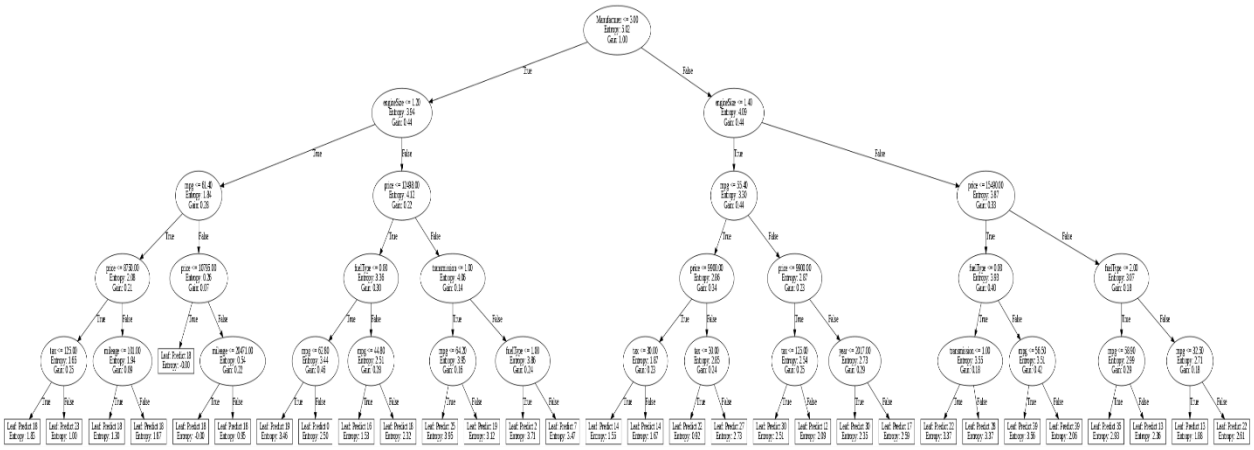
3. روش های گروهی:

- Bagging: ساخت چندین درخت به صورت موازی از نمونه های مختلف مجموعه داده آموزشی (نمونه های بوت استرپ) و میانگین گیری پیش بینی های آنها (مانند جنگل های تصادفی). این واریانس را کاهش می دهد و به جلوگیری از برازش بیش از حد کمک می کند.

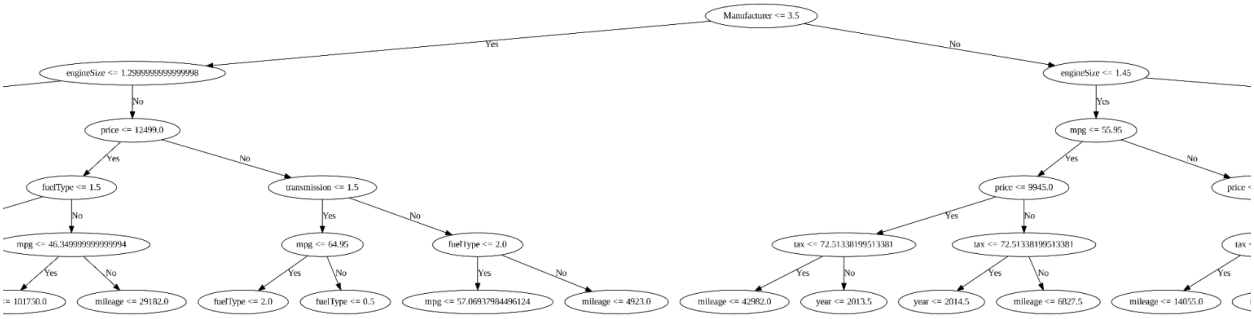
- تقویت: ساختن متوالی درختان، هرکدام بر پیش بینی صحیح مواردی که موارد قبلی بیشترین اشتباه را داشتند تمرکز می کنند. بنابراین، هر درخت به اصلاح پیشینیان خود کمک می کند، که به طور کلی منجر به مدل قوی تری می شود که بیشتر بر موارد سخت تر در مجموعه داده تمرکز می کند.

در نهایت در فاز 3 ، هم با دیتاست تمیز شده برای trainو هم از دیتاست testبرای ساخت درخت استفاده کردم.

در فاز 4 با استفاده از کتابخانه گرافیکس خروجی ها را به صورت عکس ذخیره کردم.



برای دیتاست test



تکه از درخت تولید شده توسط روش mean