Name: Amruta Nadgouda

MLE Exercise Prudential Financial

BMI Prediction

Introduction

Develop a Machine Learning model to predict BMI of the customers. Based on the predicted BMI, share the insurance quote and reason for the same.

Dataset

For this exercise, we are provided with **Dummy-Data.csv** dataset. The dataset has information about 100 applicants describing 6 different attributes.

Variable	**Description**
AppID	Anonymized Applicant ID
Ins_Age	Applicant Age
Ins_Gender	Applicant Gender
Ht	Height of the Applicant in Ft with Inches, The first digit refers to the ft and the following digits refers to the inches. For example, 507 means 5 ft and 7 inches.
Wt	Weight of the Applicant in pounds.
Issue_Date	Date of Application (Can be Null)

Data Cleaning

- **AppID**: This attribute does not provide useful information towards solving our problem and hence it is dropped.
- **Ht**: This attribute is converted into inches.
- **Issue_Date**: This attribute contains only NULL values and hence it is dropped.

There are no other missing values in the dataset.

Data Preprocessing

- **BMI**: Create BMI attribute using height (in inches) and weight (in pounds) using the formula:

$$BMI = \frac{weight}{height * height} * 703$$

- **Gender**: Encoding the gender (categorical variable) into numerical variable using **OneHotEncoder()**.

Exploratory Data Analysis

After visualizing different type of plots such as kde, lineplot, boxplot, histogram, correlation matrix we can summarize the findings as below:

- BMI decreases as Height increases, if Weight is kept constant. BMI and Height are inversely proportional and hence have negative correlation between them.
- BMI increases as Weight increases, if Height is kept constant. BMI and Weight are directly proportional and hence have positive correlation between them.
- BMI in male customers is higher than female customers.
- Age has no effect on BMI.

The plots can be found in the **main.ipynb** jupyter notebook uploaded to the GitHub reposiroty linked below.

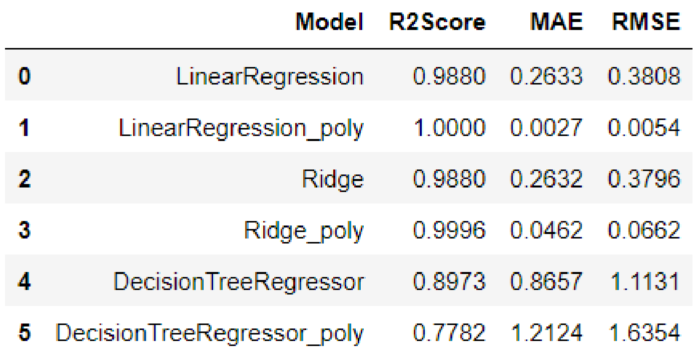
Part 1 - Feature Engineering

After splitting the dataset into Train (X_train, y_train) and Test (X_test, y_test) datasets. I created polynomial features on X_train and X_test and saved them named as X_train_poly and X_test_poly. Therefore, I now have 2 set of datasets one normal and other includes polynomial features.

Create Polynomial Features: Polynomial features are those features created by raising existing features to an exponent. This is done using **scikit-learn's PolynomialFeaturesTransform()**. The process is carried out for each input variable in the dataset, creating a transformed version of each. The "degree" of the polynomial is used to control the number of features added. I have used a degree of 3 which will add two new variables for each input variable.

Part 1 - Model Training, Testing and Results

For this part, I used three Machine Learning models Linear Regression, Ridge Regression and Decision Tree Regressor with default parameters. I trained these 3 models separately on both X_train, X_train_poly and predicted on X_test, X_test_poly respectively. The performance of the models is evaluated using R2_score, MAE, RMSE as key metrics. The table 1 shows the models along with their evaluation metrics.



	Model	R2Score	MAE	RMSE
0	LinearRegression	0.9880	0.2633	0.3808
1	LinearRegression_poly	1.0000	0.0027	0.0054
2	Ridge	0.9880	0.2632	0.3796
3	Ridge_poly	0.9996	0.0462	0.0662
4	DecisionTreeRegressor	0.8973	0.8657	1.1131
5	DecisionTreeRegressor_poly	0.7782	1.2124	1.6354

Figure 1: Part 1 - Model Results

From the table 1, we can conclude that Linear Regression trained with Polynomial Feature Transformed dataset outperforms the other models.

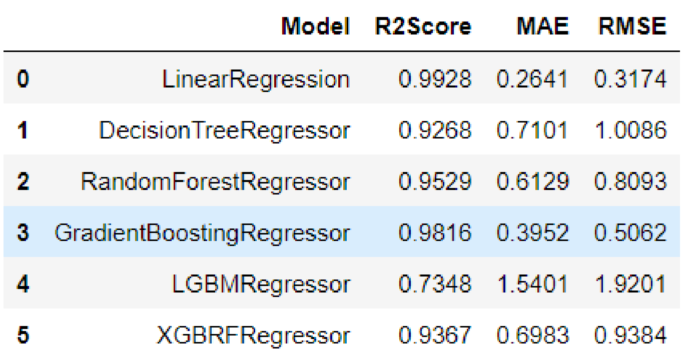
Experiment: I also tried scaling the polynomial features before training the models but result was same as non scaled version of polynomial features hence I removed those metrics from the result.

Part 2 - Feature Engineering

For this part, I used Kmeans algorithm to group similar customers into three clusters. To select the optimal number of clusters, I used **KElbowVisualizer** from yellowbricks module where I fit the visualizer on all the features except the target variable.
Once I get the optimal number of clusters, I refit the Kmeans algorithm to create the clusters and use the labels as a new feature along with age, gender, height, weight.
Next, I use SelectKBest() to select the best features to train the models.
Now that we have obtained our new dataset, we can split this dataset into X_train, X_test, y_train and y_test.
Next, we perform scaling of the features mainly height and weight as we are training both Linear models and Ensemble models. Otherwise, Ensemble models do not need scaling as they perform well even on non-scaled datasets.

Part 2 - Model Training, Testing and Results

For this part, I used six ML models namely Linear Regression, Decision Tree Regressor, Random Forest Regressor, Gradient Boosting Regressor, Light GBM Regressor, and XGB Regressor using their default parameters. The models are trained on X_train and predicted on X_test. The performance of the models is evaluated using R2_score, MAE, RMSE as key metrics. The table 2 shows the models along with their evaluation metrics.

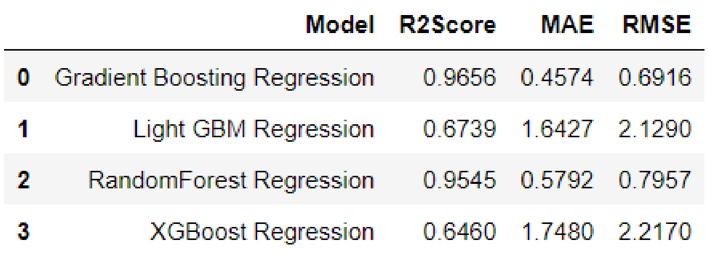


	Model	R2Score	MAE	RMSE
0	LinearRegression	0.9928	0.2641	0.3174
1	DecisionTreeRegressor	0.9268	0.7101	1.0086
2	RandomForestRegressor	0.9529	0.6129	0.8093
3	GradientBoostingRegressor	0.9816	0.3952	0.5062
4	LGBMRegressor	0.7348	1.5401	1.9201
5	XGBRFRegressor	0.9367	0.6983	0.9384

Figure 2: Part 2 - Model Results

From the table 2, we can conclude that apart from Linear Regression model, Gradient Boosting Regressor is the next best performing model.

To observe the impact of parameters, I performed hyperparameter tuning using GridSearchCV for the ensemble models and trained the models using the best parameters and made predictions. The table 3 shows the performance of models after performing GridSearchCV.



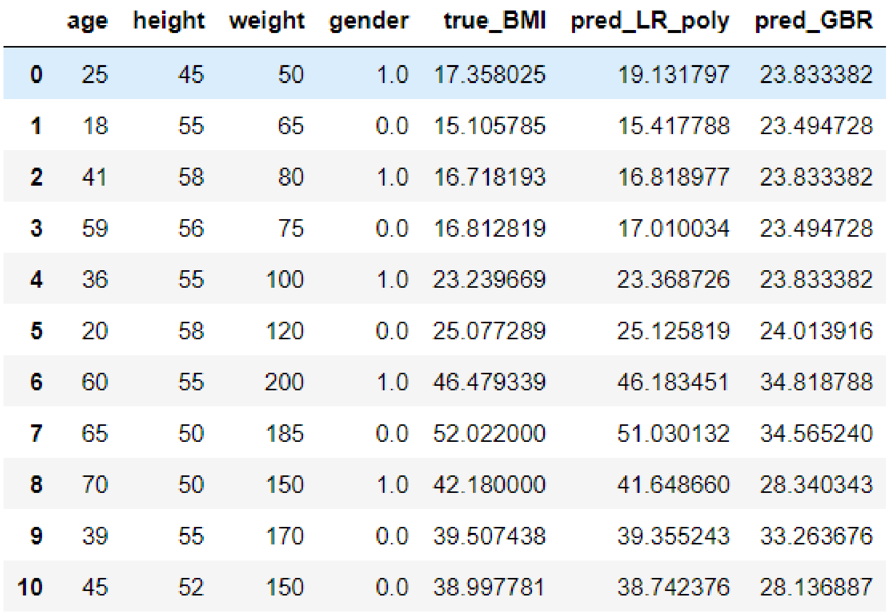
	Model	R2Score	MAE	RMSE
0	Gradient Boosting Regression	0.9656	0.4574	0.6916
1	Light GBM Regression	0.6739	1.6427	2.1290
2	RandomForest Regression	0.9545	0.5792	0.7957
3	XGBoost Regression	0.6460	1.7480	2.2170

Figure 3: Part 2 - Model Results

We can see that the performance rather decreased after hyperparameter tuning. Models with default parameters performed better than hyperparameter tuned models.

Testing

Besides the provided dataset, I created my own dataset with various combinations of height, weight, age, gender to test the finalized models.
I am comparing the predictions from Linear Regression with Polynomial Feature Transformation and Gradient Boosting Regressor which are our best performing models.
Since, Linear Regression with Polynomial Feature Transformation is one of the final model, I created polynomial features on the new test dataset. Also, for Gradient Boosting Regressor, I created new feature which is the cluster label and scaled the features (height and weight) in the new test dataset.
The table 4 shows the dataset with True BMI as well as the predictions from both the models.



	age	height	weight	gender	true_BMI	pred_LR_poly	pred_GBR
0	25	45	50	1.0	17.358025	19.131797	23.833382
1	18	55	65	0.0	15.105785	15.417788	23.494728
2	41	58	80	1.0	16.718193	16.818977	23.833382
3	59	56	75	0.0	16.812819	17.010034	23.494728
4	36	55	100	1.0	23.239669	23.368726	23.833382
5	20	58	120	0.0	25.077289	25.125819	24.013916
6	60	55	200	1.0	46.479339	46.183451	34.818788
7	65	50	185	0.0	52.022000	51.030132	34.565240
8	70	50	150	1.0	42.180000	41.648660	28.340343
9	39	55	170	0.0	39.507438	39.355243	33.263676
10	45	52	150	0.0	38.997781	38.742376	28.136887

Figure 4: Final Predictions

We can see that the predictions from Linear Regression with Polynomial Feature Transformation are very close to True BMI values and the predictions from Gradient Boosting Regressor are all in the same range.
The model should predict BMI accurately so that we can further share accurate insurance quote with the customers.

Save the Model and Create PredictBMI Application

I saved the feature engineering processes which includes One hot encoding and Polynomial feature transformation as well as the final linear regression model into three separate pickle files.

Now that I have a ML model that predicts the BMI, I created a Flask application to get the inputs from the users such as age, height, weight, gender and predict the BMI along with Insurance Quote and Reason. I deployed this application on the Heroku platform.
The application can be accessed using the link: https://predictbmi.herokuapp.com/

The application still needs improvements in terms of validating the inputs and defining the range for each of the inputs.

GitHub Link

The code for this project can be found at https://github.com/Amruta-Nadgouda/MLE-Exercise.

Conclusion

In this exercise, we are dealing with a very small dataset with only four useful attributes and hence ensemble models do not perform well on this small dataset and rather overfits. If we add a ”reason” column depending on the age and predicted BMI to the table 4 then Gradient Boosting Regressor predictions will have ”BMI is in right range” reason for all the rows which is incorrect.

To overcome the problems of insufficient data, we can augment the dataset by generating synthetic data using SMOTE but it is widely used for classification problems. Since this a regression problem, we need SMOTE variation for regression problems whose implementation is not readily available and is still in the unit testing phase. Because of these reasons, we use Polynomial Feature Transformation that generates new features to the Linear Regression model to learn the patterns.
In conclusion, we explored different Feature Engineering techniques and various Machine Learning models and finally came up with the model that predicts the BMI close to the value of true BMI. Then, created an

application which takes users input and predicts the BMI and displays correct insurance quote based on age and predicted BMI.

Steps to Operationalize the model

1. Identify and Build:

   Identify the data that we are trying to collect and then work to build an analytic pipeline through predictive models. An ML-model should have the minimum possible number of features but with good evaluation measures and should also be,

   - **Portable**: A portable model decreases the response time of code and the amount of code to be rewritten when the goals change will be minimal.
   - **Scalable**: A scalable model doesn't need to be redesigned to maintain effective performance.

2. Manage:

   Management strategies need to be easy-to-follow and effective to keep track of necessities. It is necessary to track things like development, training, versioning, and deployment of models.

   Key features could be Data gathering and analysis, Data transformation, Model development and training, Model monitoring and re-training. This involves experiments, along with different parameters, hyperparameters, optimizers, and even loss functions.

3. Deploy/Integrate:

   For deployment, the pipeline built is taken from its original development environment and expressed in a form that can be executed independently and integrated into business applications and software. This step requires rigorous testing and reformulating to ensure everything runs smoothly, and that it's going to integrate well with day-to-day operations. Ex: Using CI/CD tools like Jenkins.

4. Monitor:

   Monitoring helps us stay up-to-date with prediction performance and service updates, which are important to any model. After a model has been deployed, it should be monitored for the accuracy of its predictions and impact on the business. The model needs to remain accurate even as the underlying data changes.