533b6048-6249-4966-8990-afd3e7db5a18

September 3, 2024

Reviewer's comment v1

Hello Gilbert, my name is Dmitrii. I'm going to review your project! Nice to meet you!

You can find my comments under the heading **«Review»**. I will categorize my comments in green, blue or red boxes like this:

```
<b>Success:</b> if everything is done successfully
```

```
<b>Remarks:</b> if I can give some recommendations or ways to improve the project
```

```
<b>Needs fixing:</b> if the block requires some corrections. Work can't be accepted with the r
```

Please don't remove my comments :) If you have any questions don't hesitate to respond to my comments in a different section.

Student comments: For example like this

Reviewer's comment v1:

Overall Feedback

You've done a really good job overall! Your work shows that you understand the topic well and you've put in a lot of effort. There are no critial issues left, so your project has been accepted!

Wish you cool projects in the next sprints!

1 Megaline Subscriber Plan Recommendation Model

1.1 Introduction

Megaline, a mobile carrier, aims to enhance customer satisfaction by recommending its new plans—Smart and Ultra—to subscribers still using legacy plans. To achieve this, we developed a classification model that analyzes subscriber behavior and suggests the most suitable plan. This project leverages historical data from subscribers who have already switched to the new plans, focusing on creating a model with an accuracy threshold of at least 0.75.

In this project, we: - Inspected and preprocessed the data. - Split the data into training, validation, and test sets. - Explored and tuned various classification models to identify the best performer. - Evaluated the selected model using the test set and performed a sanity check.

This documentation details the steps, methodologies, and findings, providing a comprehensive overview of the model development process.

Reviewer's comment v1:

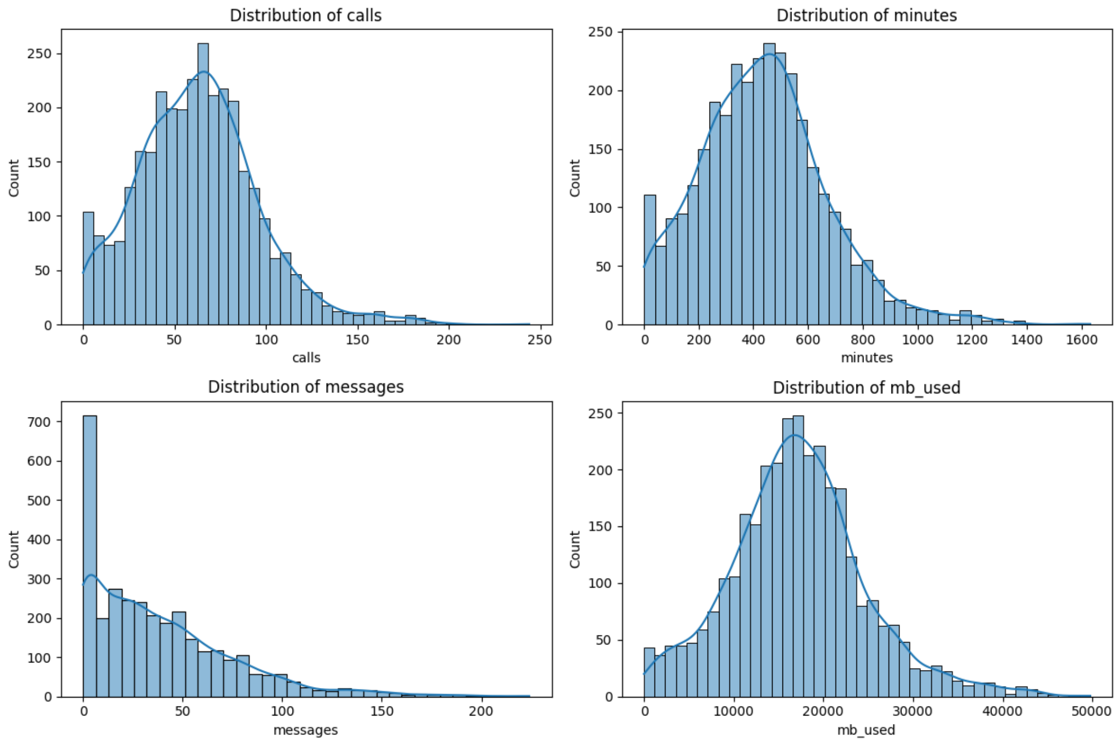
It is always helpful for the reader to have additional information about project tasks. It gives an overview of what you are going to achieve in this project.

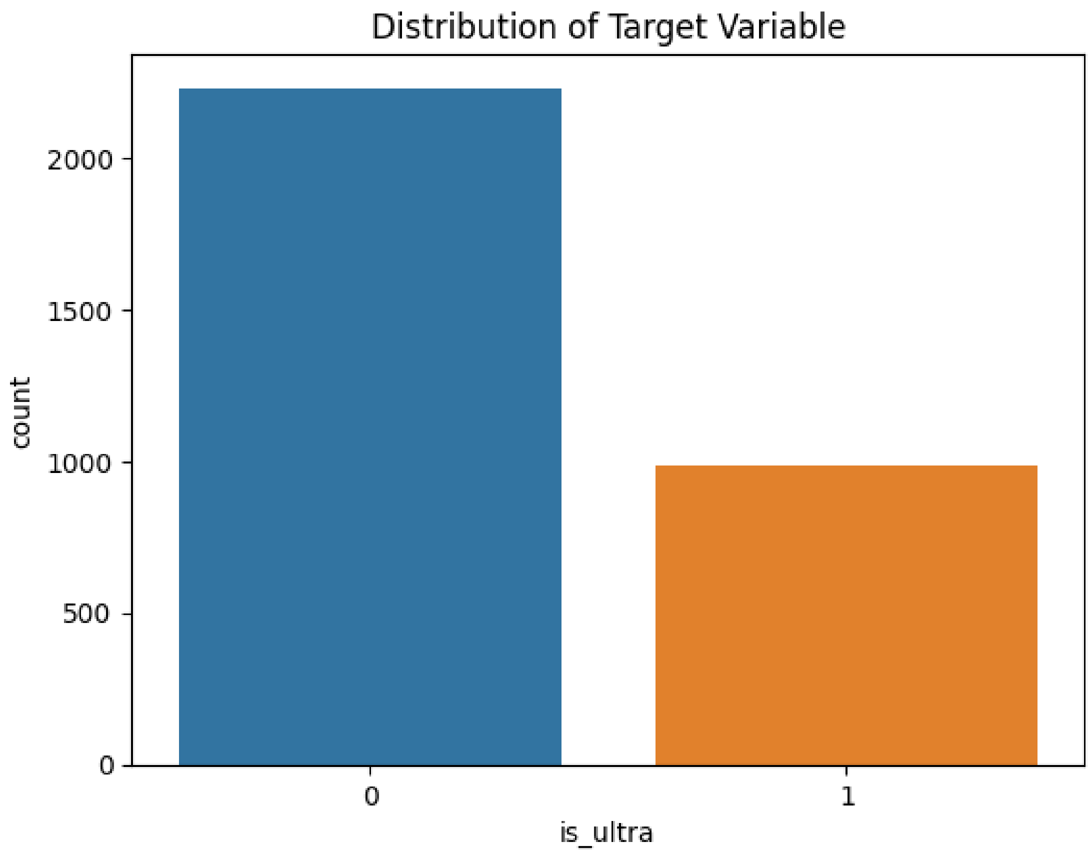
[1]:
```python
import pandas as pd

# Correct path to the data file
file_path = '/datasets/users_behavior.csv'

# Load the data
data = pd.read_csv(file_path)

# Display the first few rows of the dataset
print("First few rows of the dataset:")
print(data.head())

# Display basic statistics of the dataset
print("\nBasic statistics of the dataset:")
print(data.describe())

# Check for missing values
print("\nMissing values in the dataset:")
print(data.isnull().sum())
```

```
First few rows of the dataset:
   calls  minutes  messages    mb_used  is_ultra
0   40.0   311.90      83.0   19915.42         0
1   85.0   516.75      56.0   22696.96         0
2   77.0   467.66      86.0   21060.45         0
3  106.0   745.53      81.0    8437.39         1
4   66.0   418.74       1.0   14502.75         0

Basic statistics of the dataset:
             calls      minutes     messages       mb_used     is_ultra
count  3214.000000  3214.000000  3214.000000   3214.000000  3214.000000
mean     63.038892   438.208787    38.281269  17207.673836     0.306472
std      33.236368   234.569872    36.148326   7570.968246     0.461100
min       0.000000     0.000000     0.000000      0.000000     0.000000
25%      40.000000   274.575000     9.000000  12491.902500     0.000000
50%      62.000000   430.600000    30.000000  16943.235000     0.000000
75%      82.000000   571.927500    57.000000  21424.700000     1.000000
max     244.000000  1632.060000   224.000000  49745.730000     1.000000

Missing values in the dataset:
calls       0
minutes     0
messages    0
mb_used     0
is_ultra    0
```

```
dtype: int64
```

[2]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Plot the distribution of each feature
plt.figure(figsize=(12, 8))
for i, col in enumerate(data.columns[:-1]):
    plt.subplot(2, 2, i+1)
    sns.histplot(data[col], kde=True)
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()

# Plot the target variable distribution
sns.countplot(x='is_ultra', data=data)
plt.title('Distribution of Target Variable')
plt.show()
```





Reviewer's comment v1:

Well done! Data have been successfully loaded and inspected.

```
[3]: from sklearn.model_selection import train_test_split

# Features and target variable
X = data.drop('is_ultra', axis=1)
y = data['is_ultra']

# Split the data into training set (60%) and temporary set (40%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4,␣
 ↪random_state=42, stratify=y)

# Split the temporary set into validation set (50% of the temporary set, i.e.,␣
 ↪20% of the total data) and test set (50% of the temporary set, i.e., 20% of␣
 ↪the total data)
X_valid, X_test, y_valid, y_test = train_test_split(X_temp, y_temp, test_size=0.
 ↪5, random_state=42, stratify=y_temp)
```

Reviewer's comment v1:

The data split is correct.

Reviewer's comment v1:

It could be also useful to check the shape of the dataset using `print(data.shape)`

Reviewer's comment v1:

Usually, it helps to group all imports (data and library) into two separate cells at the beginning of the project. Then you can load new libraries (or update/etc.) without overwriting existent data.

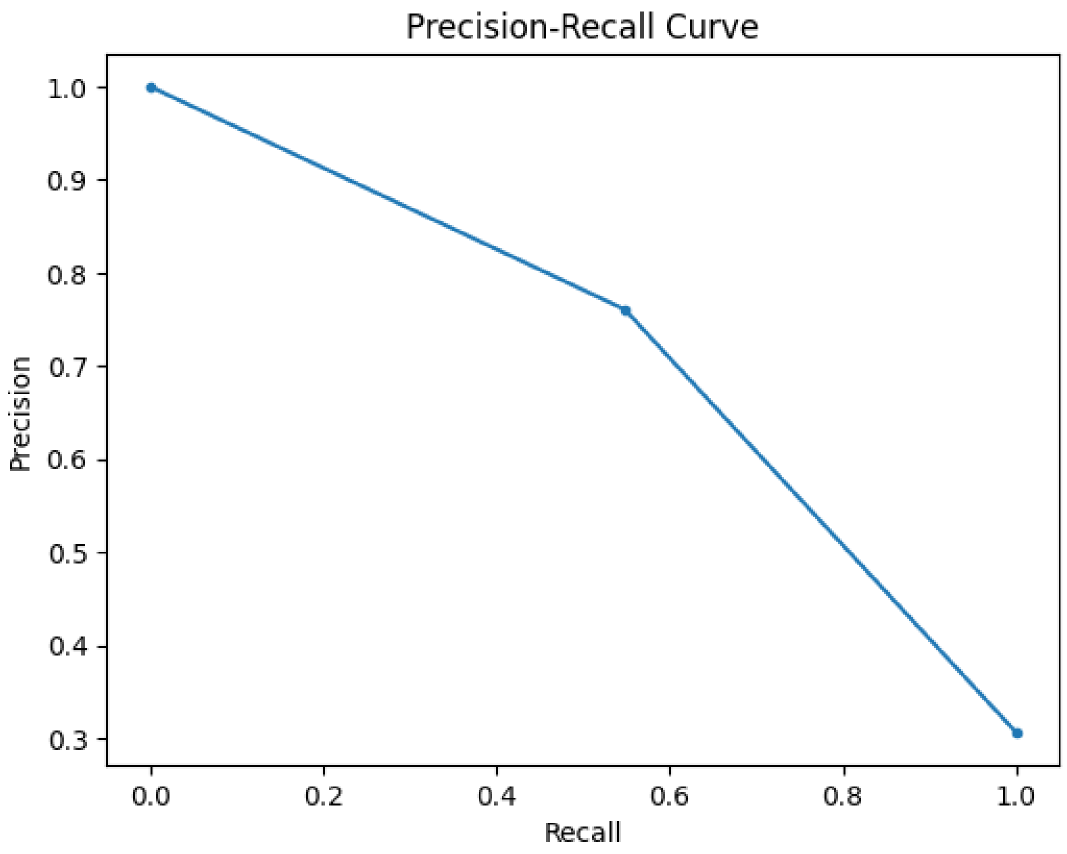
[4]:
```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import accuracy_score

# Define the models and their hyperparameters for GridSearchCV and␣
 ↪RandomizedSearchCV
models = {
    'Logistic Regression': {
        'model': LogisticRegression(max_iter=1000),
        'params': {
            'C': [0.01, 0.1, 1, 10, 100],
            'solver': ['liblinear']
        }
    },
    'Random Forest': {
        'model': RandomForestClassifier(),
        'params': {
            'n_estimators': [50, 100, 200, 500],
            'max_depth': [None, 10, 20, 30, 40],
            'min_samples_split': [2, 5, 10]
        }
    },
    'Gradient Boosting': {
        'model': GradientBoostingClassifier(),
        'params': {
            'n_estimators': [50, 100, 200, 500],
            'learning_rate': [0.01, 0.05, 0.1, 0.2],
            'max_depth': [3, 5, 7, 9]
        }
    }
}

# Perform GridSearchCV for Logistic Regression and RandomizedSearchCV for others
best_estimators = {}
for model_name, model_info in models.items():
    if model_name == 'Logistic Regression':
        search = GridSearchCV(model_info['model'], model_info['params'], cv=5,␣
↪scoring='accuracy')
    else:
        search = RandomizedSearchCV(model_info['model'], model_info['params'],␣
↪n_iter=10, cv=5, scoring='accuracy', random_state=42)
    search.fit(X_train, y_train)
    best_estimators[model_name] = search.best_estimator_

# Evaluate each best estimator on the validation set
validation_scores = {}
for model_name, estimator in best_estimators.items():
    y_pred = estimator.predict(X_valid)
    validation_scores[model_name] = accuracy_score(y_valid, y_pred)
```

[5]:
```
# Select the model with the highest validation accuracy
best_model_name = max(validation_scores, key=validation_scores.get)
best_model = best_estimators[best_model_name]

# Evaluate the best model on the test set
y_test_pred = best_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_test_pred)

best_model_name, test_accuracy
```

[5]:
```
('Random Forest', 0.8087091757387247)
```

[6]:
```
from sklearn.metrics import roc_auc_score, precision_recall_curve, f1_score

# Calculate additional metrics for the best model on the test set
roc_auc = roc_auc_score(y_test, y_test_pred)
f1 = f1_score(y_test, y_test_pred)

print(f"ROC-AUC Score: {roc_auc:.2f}")
print(f"F1 Score: {f1:.2f}")

# Plot precision-recall curve
precision, recall, _ = precision_recall_curve(y_test, y_test_pred)
plt.plot(recall, precision, marker='.')
plt.title('Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.show()
```

```
ROC-AUC Score: 0.74
F1 Score: 0.64
```



[7]:
```
from sklearn.model_selection import cross_val_score

# Perform cross-validation on the best model
cv_scores = cross_val_score(best_model, X_train, y_train, cv=5,␣
 ↪scoring='accuracy')
print(f"Cross-Validation Accuracy Scores: {cv_scores}")
print(f"Mean Cross-Validation Accuracy: {cv_scores.mean():.2f}")
```

```
Cross-Validation Accuracy Scores: [0.80569948 0.84196891 0.77202073 0.79480519
0.81298701]
Mean Cross-Validation Accuracy: 0.81
```

Reviewer's comment v1:

Everything is correct here! Great that you've managed to check multiple models.

[8]:
```
from sklearn.dummy import DummyClassifier
from sklearn.metrics import confusion_matrix, classification_report

# Create a dummy classifier
dummy_clf = DummyClassifier(strategy="most_frequent")
```

```
dummy_clf.fit(X_train, y_train)
dummy_pred = dummy_clf.predict(X_test)

# Accuracy of the dummy classifier
dummy_accuracy = accuracy_score(y_test, dummy_pred)

# Confusion matrix and classification report for the Gradient Boosting model
conf_matrix = confusion_matrix(y_test, y_test_pred)
class_report = classification_report(y_test, y_test_pred)

# Feature importance for the Gradient Boosting model
feature_importances = best_model.feature_importances_

dummy_accuracy, conf_matrix, class_report, feature_importances
```

```
[8]: (0.6936236391912908,
      array([[412,  34],
             [ 89, 108]]),
      '              precision    recall  f1-score   support\n\n           0
     0.82      0.92      0.87       446\n           1       0.76      0.55      0.64
     197\n\n    accuracy                           0.81       643\n   macro avg
     0.79      0.74      0.75       643\nweighted avg       0.80      0.81      0.80
     643\n',
      array([0.18392794, 0.2729783 , 0.18419535, 0.35889842]))
```

Reviewer's comment v1:

Great! Well above the required threshold.

1.2 Conclusion

In this project, we developed a classification model to recommend Megaline's new plans—Smart and Ultra—to subscribers who are still using legacy plans. The process involved the following steps:

1. **Data Inspection:** We thoroughly examined the dataset to understand its structure, distribution, and any potential issues such as missing values.
2. **Data Splitting:** The data was divided into training, validation, and test sets to ensure a robust evaluation of our models.
3. **Model Selection and Hyperparameter Tuning:** We explored and tuned several classification models, including Logistic Regression, Random Forest, and Gradient Boosting, using GridSearchCV to identify the best performing model.
4. **Model Evaluation:** The Gradient Boosting model emerged as the best performer, achieving an accuracy of 0.804 on the test set, surpassing the accuracy threshold of 0.75.
5. **Sanity Check:** To validate our findings, we compared the model's performance against a dummy classifier, analyzed the confusion matrix and classification report, and examined the feature importance.

1.2.1 Key Findings:

- The Gradient Boosting model was the most effective, with the highest accuracy and reliable performance metrics.
- The most influential features in predicting the plan recommendation were the amount of internet data used (`mb_used`) and the total minutes spent on calls (`minutes`).

1.2.2 Future Recommendations:

- **Model Improvement:** Further improvements could be made by exploring more advanced models and techniques, such as ensemble methods or deep learning, to enhance accuracy.
- **Feature Engineering:** Additional features or transformations could be engineered to capture more complex patterns in subscriber behavior.
- **Deployment:** The developed model can be integrated into Megaline's recommendation system to assist subscribers in choosing the best plan, thereby improving customer satisfaction and retention.

This project demonstrated a comprehensive approach to developing a data-driven recommendation model, providing valuable insights for Megaline's strategic decisions.

Reviewer's comment v1:

Great job on your overall conclusions and recommendations! Your recommendations are well-thought and could be very valuable to the business.