Projet de rattrapage Ma412

MATHEMATICAL TOOLS FOR DATA SCIENCE

Professeur :Leila Gharsalli

SAMOYAULT Pierre SM1

Année 2023-2024

Sommaire

I. Introduction

The survey, on which our dataset is based, focuses on air passenger satisfaction. After the pandemic, the airline industry experienced a significant downturn. Therefore, to rejuvenate the industry amidst the current recession, it is crucial to identify and address customer pain points and enhance their satisfaction with the services offered. You are tasked with a prediction problem to determine which satisfaction level a passenger falls into: Satisfied, Neutral, or Dissatisfied. Our dataset is already split in train set and test set, that facilitate our study. In this report you will see our solution and how we have built it. We should pay attention to different possible solutions, because there is plenty of classification models that we can use, there strength and weakness, then we will focus on the analysis of our dataset and finally, you will see our solution in two different levels of optimization.

II. Dataset analysis

The first thing that we can notice on our datasets, is there size. In fact, they are pretty huge, (103904,25) for the train set and (25976, 25) for the test set. Moreover, we have noticed that it's a binary classification. We want to know and anticipate the satisfaction of the passengers with the help of different features. The satisfaction is "satisfied" or "neutral or unsatisfied", as we have only two possible solutions it's a binary problem. With these information, we have to choose or model of study. For binary classification with a large dataset, their below three strong models you can consider:

Gradient Boosting Machines (GBM): Examples of implementations include XGBoost, LightGBM, and CatBoost. These models are highly performant for classification tasks, handle imbalanced data well, and can capture complex interactions between features. However, they can be slow to train on very large datasets and require expertise in hyper parameter tuning.

Random Forest: Implemented in libraries like Scikit-Learn, Random Forest is robust against overfitting, works well with minimal tuning, and can handle missing data and imbalanced datasets. The downside is that it can be slow to train on very large datasets and is less interpretable compared to linear models.

Logistic Regression: Available in Scikit-Learn, Logistic Regression is simple to understand and interpret, fast to train even on large datasets, and works well as a baseline model. However, it is limited in capturing non-linear relationships and requires well-prepared features (scaling, normalization).

Finally, we choose the random forest because it's the most familiar model that we know and seems to be the easier to use with our dataset.

As said previously, we will consider, to compare our result two levels of optimization for the random forest model, the first one without optimization, the second one with a feature selection and the use of hyper parameters.

Random Forest is a powerful machine learning algorithm for classification and regression tasks. It constructs multiple decision trees using random subsets of the training data and features, introducing variability among the trees. This reduces the risk of overfitting and improves prediction performance.

In the training phase, each tree is built from a bootstrap sample of the dataset. At each node, a random subset of features is considered to find the best split. This process is repeated to create a forest of many trees.

For classification, each tree votes on the class, and the majority vote determines the final prediction. For regression, the predictions of all trees are averaged to produce the final result. This ensemble approach leverages the diverse opinions of multiple trees, providing robust and accurate predictions.

Random Forests handle complex data well, reduce overfitting, and can provide feature importance estimates. They are scalable and efficient, suitable for large datasets. You will find in the python file all what we did, for the clarity of the rapport we haven't had it.

In our data exploration we have notice that there are some N/A values, but it's not possible to deal with N/A values so we choose to replace it by the mean of their respective column. The next step is to encode the label values (those which have a type object), that mean for each different label value a number is associate to this value. That create a kind of class, we have done it especially for the satisfaction: 1 is for "satisfied" and 0 is for "neutral or unsatisfied".

The last step of our exploration is to remove the features that are useless. We have identified three features, that must bias our model: "Unamed:0", "id" and "Customer type".

The two first are obvious, it just index and unique number that must be related to the flight so it can't be useful four our analysis. For the last one it's more complex. It represents the fidelity of a customer, but the fidelity is highly correlated to the satisfaction of a customer. a loyal customer is more likely to give higher marks and be satisfied with their trip. That's why we have decided to remove it.

Moreover, you will find attached in the folder the correlation matrix. With a fast analysis, all the values are near 0 that means there is a weak correlation. We understand that the features are not really correlated that means there is "no conflict" between them so we don't need to remove some of them for the high correlation (or inversed correlation). We can notice that we have not put the values on the same scale so it less precise, but that give a general idea of the problem.

III. The solution

Without optimization:

```
# Train of the model

rf_model= RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
```

It's the simplest model with the random forest, the random_state=42 sets a seed for the random number generator used by the Random Forest. Fixing the seed ensures the same splits for training and testing data when rerunning the model, promoting reproducibility.

With optimization:

With the optimization, our model is more complex. We use the Recursive feature elimination with cross-validation to select features (RFECV).

```
# Create a Random Forest classifier
rfecv = RFECV(estimator=RandomForestClassifier(n_estimators=40, max_depth=6, random_state=42),step=2, cv=5, scoring='accuracy')
```

We create an RFECV object in scikit-learn to automatically find the best features for a Random Forest model. It uses a Random Forest with specific settings, removes a group of features at each step, evaluates performance with cross-validation, and aims to maximize accuracy. The explanation of our hyper parameter:

- n_estimators=40: This controls the number of decision trees used in the Random Forest model within RFECV. More trees can improve accuracy but also increase training time.
- max_depth=6: This limits the maximum depth each tree in the Random Forest can grow. Deeper trees can capture complex patterns but might also lead to overfitting.
- random_state=42: This sets a seed for the random number generator used by the Random Forest. Fixing the seed ensures the same splits for training and testing data when rerunning the model, promoting reproducibility.
- step=5: This controls how many features are removed at each step of the feature elimination process in RFECV. Larger values remove features faster but might miss subtle effects.
- cv=2: This defines the number of folds used in the cross-validation process within RFECV. Cross-validation helps evaluate the model's performance on unseen data and

avoid overfitting. Here, 2 folds are used, but this value can be adjusted for a more robust evaluation.

All of these hyper parameter where used and choose by experimentation and some documentation found on Internet and in our lesson of the first semester. Our most important issue was the running time. In fact, due to the size of the dataset testing for different parameter again and again was very long, so we had to choose some of them regarding the ratio running time/results.

We have done it to find the best features for the model, using the function below:

```
# Plot the number of features vs. cross-validation scores
print("Optimal number of caracteristiques: %d" % rfecv.n_features_)
print("Selected caracteristique: %s" % list(X_train.columns[rfecv.support_]))
```

We obtained after:

```
Optimal number of caracteristiques: 21
```

In other words, all the features, to be sure we have used a function that show us the ranking importance of the features:

```
# Get the ranking of the features
feature_rankings = rfecv.ranking_
feature_names = X_train.columns

feature_importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Ranking': feature_rankings
}).sort_values(by='Ranking', ascending=True)

print(feature_importance_df)
selected_features = feature_importance_df[feature_importance_df['Ranking'] == 1]
```

	Feature	Ranking
0	Gender	1
18	Cleanliness	1
17	Inflight service	1
16	Checkin service	1
15	Baggage handling	1
14	Leg room service	1
13	On-board service	1
12	Inflight entertainment	1
11	Seat comfort	1
19	Departure Delay in Minutes	1
10	Online boarding	1
8	Gate location	1
7	Ease of Online booking	1
6	Departure/Arrival time convenient	1
5	Inflight wifi service	1
4	Flight Distance	1
3	Class	1
2	Type of Travel	1
1	Age	1
9	Food and drink	1
20	Arrival Delay in Minutes	1

All of them have their importance. Removing to much features can also bias our model. By curiosity, I have checked if it must had selected the features that we remove at the beginning and hopefully they were not kept.

After the checks of the features, we try to optimize the hyper parameters of the model using:

```
# Define the parameter grid to be tested
param_grid = {
    'estimator__n_estimators': [50],
    'estimator__max_depth': [None,2,5],
    'estimator__min_samples_split': [2, 5],
    'estimator__min_samples_leaf': [1, 2],

}

# Configure grid search with a simple split test train
grid_search = GridSearchCV(estimator=rfecv, param_grid=param_grid, cv=2, n_jobs=-1, verbose=2)

# Run a grid search on all the data
grid_search.fit(X_train_rfe, y_train)
best_params = grid_search.best_params_

# Best parameters found by grid search
print("Best parameters founds: ", best_params)
```

This is at this moment that we had our biggest issue. The running time was really long, we test multiple times, adding and removing hyper parameter, trying to find a solution to accelerate the research but it was really complicated due to the size of the dataset. Normally, the list of parameter is much longer but in our case we had some running time longer than one hour, so we had to make some choices. For the explanation, we create a GridSearchCV object in scikit-learn to perform hyper parameter tuning for the RFECV. For the new hyper parameter:

- n_jobs=-1: This utilizes all available CPU cores for parallel processing during the hyperparameter tuning process.
- verbose=2: This sets the verbosity level to 2, providing more detailed output during the hyperparameter tuning process.

We found:

```
{'estimator__max_depth': None, 'estimator__min_samples_leaf': 1, 'estimator__min_samples_split': 5, 'estimator__n_estimators': 50}
```

The estimator_min_samples_leaf and estimator_min_samples_split both work together to prevent overfitting in the Random Forest model used by RFECV. They control the minimum number of data points allowed in a leaf node (where decisions are made) and for splitting a node (creating new branches). Higher values for both ensure the model doesn't learn overly specific patterns from small amounts of data, leading to better generalization on unseen data.

IV. Presentation of the results

Without optimization (projet_Ma412_noopti.ipynb) :
After training the model, we have obtained:

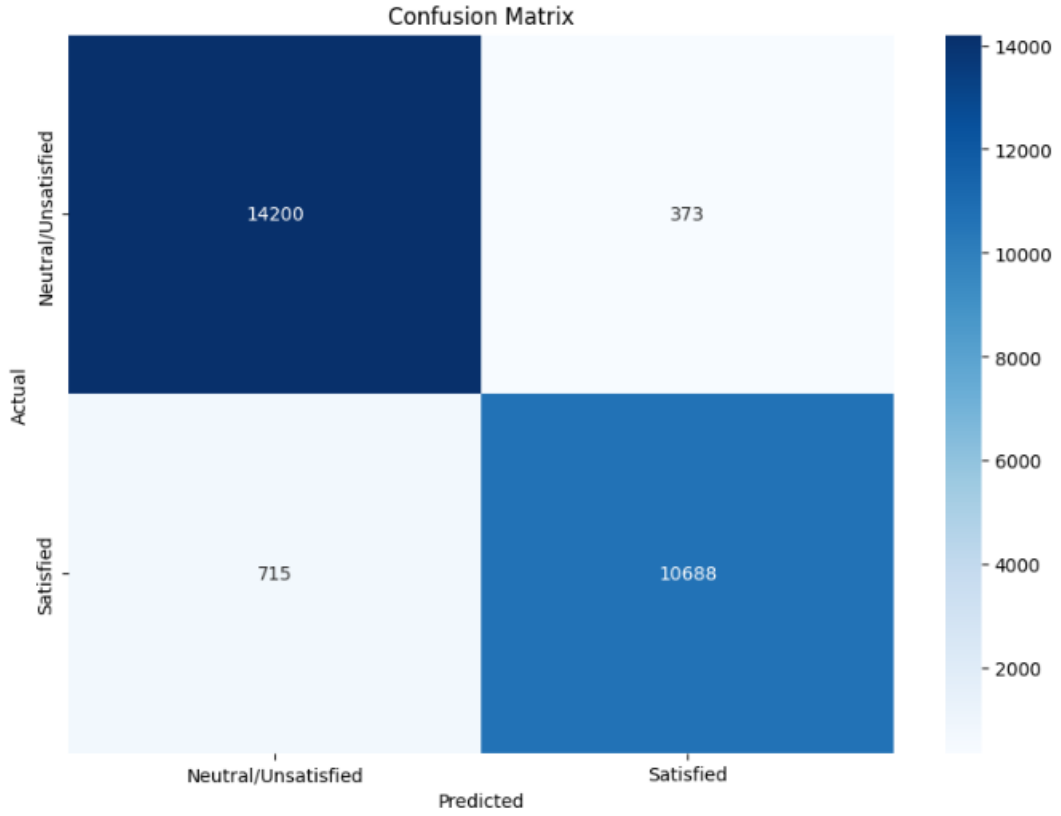
```
Validation Accuracy with RFE: 0.9581151832460733
              precision    recall  f1-score   support

           0       0.95      0.97      0.96     14573
           1       0.97      0.94      0.95     11403

    accuracy                           0.96     25976
   macro avg       0.96      0.96      0.96     25976
weighted avg       0.96      0.96      0.96     25976
```

The accuracy is not bad but the aim in classification is to be as near as possible to 1, without over lifting.For a more visual result, we have plot the confusion matrix:

```
Confusion Matrix:
 [[14200   373]
 [  715 10688]]
```



We have an accurate model, but it can be improved that why we try to optimize it.

With features optimization (projet_ma412_opti_max.ipynb):

Unfortunately, we didn't get much better results as expected:

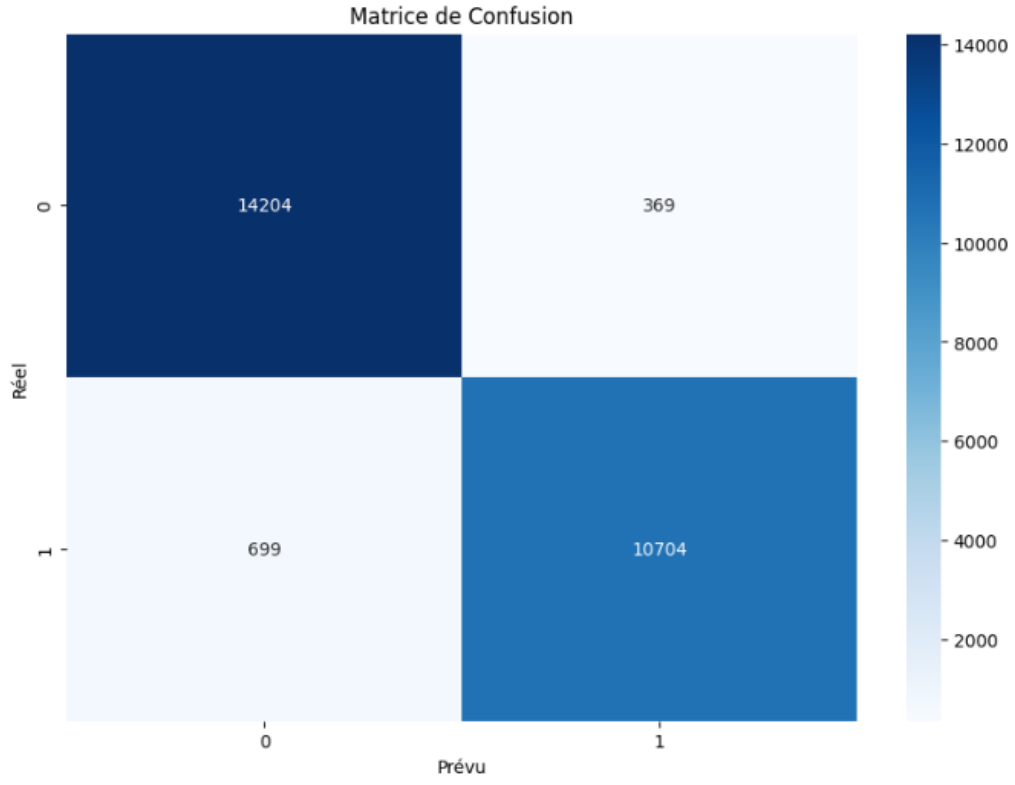
```
Accuracy:  0.9588851247305205
              precision    recall  f1-score   support

           0       0.95      0.97      0.96     14573
           1       0.97      0.94      0.95     11403

    accuracy                           0.96     25976
   macro avg       0.96      0.96      0.96     25976
weighted avg       0.96      0.96      0.96     25976
```

As you can see the results are a bit better, that mean our selection is better than the basic model, but it's not really relevant.

```
Matrice de confusion:
[[14204   369]
 [  699 10704]]
```



V. Conclusion

This project was really interesting to go deeper in the machine learning. The random forest classifier is really interesting but we have seen it limit, with a large number of values especially when you try to use and tuned hyper parameters to have a better model.

May be with a stronger computer, it may take less time, this was the main problem of the subject. In addition, may be with one of the other model that we have presented, it had taken less time and we probably have a better result. Nevertheless, our results were quit precise and even if the optimization of the hyper parameter was not accurate as expected, we get better results.