AI and ML Fundamentals

Student: Laman Asgarova
Group: 21.2
Lab: 2
Report Submitted: 30.10.2024

I. Introduction

The purpose of this laboratory work is to process a dataset on hotel bookings, prepare it for machine learning classification, and evaluate the performance of different classifiers. By using techniques such as data exploration, train_test_splitting, and PCA, we aim to improve model performance and generalizability. Additionally, hyperparameter tuning was applied to optimize each model for improved accuracy and reduced risk of overfitting.

II. Analyze The Data

After loading the dataset, we should analyze it to understand its overall structure and gain insights.

By using **df.describe()**, we can see statistical information about the numeric features of the dataset. This includes the **count** (number of records in each column), as well as **mean** and **std** (standard deviation) of features, **min**, **max**, **25%** (Q1), **50%** (Q2 - median), **75%** (Q3) values of each column.

With the help of **df.info()** we can see if there is any missing value in any column, shape of the dataset, their datatypes and so on. The shape of the dataset is (36275, 19), so for each column we have 36275 non-null values, it means we don't have null value in any column. Dtypes are correct (14 - numeric, 5 - object type).

When examining the categorical columns, I see that we do not have enough information about their ordinality. Therefore, I use OneHotEncoding to convert categorical columns into numeric format, as models cannot be trained directly with categorical values. After encoding, I apply StandardScaler to scale the features, ensuring they have a mean of 0 and a standard deviation of 1 so that all features contribute equally to the model. These preprocessing steps are performed after train_test_split to avoid data leakage. Additionally, the dataset is imbalanced, with an unequal distribution of classes: Class 0 has 24,390 instances, which is almost twice as many as Class 1 with 11,885 instances. We can observe this imbalance by using **df['booking_status'].value_counts()**.

III. Train, Test Split Tricks

Before performing PCA and preprocessing, the dataset was split into training and testing sets. Splitting first helps prevent data leakage by ensuring that transformations (such as PCA and other preprocessing techniques) are only fitted on the training data, preserving the independence of the test set. This approach provides an unbiased assessment of the model's performance on unseen data.

IV. Test Decision Tree

I tried decision tree classifier and obtained the following results: **Accuracy**: 0.835, **F1 Score**: 0.754 **F1 score for train set**:0.991. The same model without PCA: **Accuracy**: 0.871, **F1 Score**: 0.807, **F1 score for train set**: 0.991. Given the imbalanced nature of our dataset, we prioritize the F1 score as it better reflects the model's performance in handling both classes effectively. In this case we have overfitting in both case. For avoiding overfitting we should use hyperparameter tuning.

V. Test Other classifiers by using Grid Search Algorithm

To find the best classifier, Grid Search was applied to tune hyperparameters for multiple models, including Support Vector Machine (SVM), Random Forest, Decision Tree, and Logistic Regression. Grid Search systematically explored different hyperparameter combinations for each model, enhancing accuracy and generalization.

Various performance metrics were considered, including F1 score, accuracy, precision, recall, and ROC AUC. However, I printed only F1 score and accuracy due to the long runtime.

After hyperparameter tuning I got these best params and results:

SVM - Best Params: {'C': 10, 'kernel': 'rbf'}, Accuracy: 0.839, F1: 0.738

Decision Tree - Best Params: {'criterion': 'gini', 'max_depth': 10}, Accuracy: 0.817, F1: 0.710

Random Forest - Best Params: {'criterion': 'entropy', 'max_depth': 15, 'n_estimators': 300}, Accuracy: 0.875, F1: 0.796

Logistic Regression - Best Params: {'C': 1}, Accuracy: 0.783, F1: 0.625

Then I built each model with their best parameters and got results:

SVM - Accuracy: 0.843, F1: 0.746, F1_train=0.763

Decision Tree - Accuracy: 0.823, F1: 0.721, F1_train=0.769

Random Forest - Accuracy: 0.884, F1: 0.815, F1_train=0.943

Logistic Regression - Accuracy: 0.787, F1: 0.638, F1_train=0.628

As we see there is not much overfitting in any model after hyperparameter tuning.

VI. Effect of PCA

In the previous models I used PCA with k=20,selecting this value based on the cumulative explained variance graph, where around 20 components explained nearly 90-95% of the variance.

Then I built the same models without PCA, and got results:

SVM - Best Params: {'C': 10, 'kernel': 'rbf'}, Accuracy: 0.848, F1: 0.755

Decision Tree - Best Params: {'criterion': 'gini', 'max_depth': 10}, Accuracy: 0.873, F1: 0.799

Random Forest - Best Params: {'criterion': 'gini', 'max_depth': 15, 'n_estimators': 200}, Accuracy: 0.889, F1: 0.82

Logistic Regression - Best Params: {'C': 1}, Accuracy: 0.804, F1: 0.675

Again I built models for corresponding parameters and got results:

SVM - Accuracy: 0.852, F1: 0.762, F1_train=0.783

Decision Tree - Accuracy: 0.870, F1: 0.796, F1_train=0.817

Random Forest - Accuracy: 0.890, F1: 0.827, F1_train=0.862

Logistic Regression - Accuracy: 0.806, F1: 0.684, F1_train=0.677

As we see there is no overfitting in any model after hyperparameter tuning.

Most models showed improved accuracy and F1 scores without PCA, indicating that reducing dimensionality may have led to some loss of information. The Random Forest classifier performed best both with and without PCA, achieving the highest accuracy and F1 score in both scenarios. Without PCA, models generally took longer to train due to the increased dimensionality, highlighting PCA's role in improving computational efficiency.

Best Model: Random Forest with tuned hyperparameters (without PCA) provided the best results, achieving 89% accuracy and F1 score of almost 83%.

VII. 1 model with overfitting Hyperparameters

In this task, I experimented with various models and hyperparameters to identify signs of overfitting. The results for the relevant classifiers are as follows:

Decision Tree - Train F1: 0.991, Test F1: 0.749

Random Forest - Train F1: 0.991, Test F1: 0.822

Decision Tree - max_depth=None, criterion='log_loss'

- `max_depth=None`: This allows the decision tree to grow to its maximum possible depth without restriction, which enables it to capture complex patterns. However, this also increases the risk of overfitting, as deeper trees tend to fit closely to the training data.

Random Forest - n_estimators=1000, max_depth=None

- `n_estimators=1000`: This specifies the number of trees in the forest. Using 1000 estimators increases the ensemble's capacity, which can improve accuracy but also risks overfitting when trees are deep.
- `max_depth=None`: Each tree in the forest is allowed to grow without any depth limit, resulting in potentially very deep trees that can learn complex patterns. While this enhances model performance on the training data, it may lead to overfitting if the trees capture noise rather than general patterns.