Project

Mariya Zacharneva, Ruslan Melnyk

The Task

The project aims to predict which passenger will travel to another dimension. This task is based on the Kaggle competition: Spaceship Titanic | Kaggle. As a reference point, we are using a starting guide: Spaceship Titanic with TFDF | Kaggle.

Dataset

The main training dataset contains the following info about each passenger:

- PassengerId - A unique Id for each passenger in the form of gggg_pp where gggg indicates a group the passenger is travelling with and pp is their number within the group.
- HomePlanet - The planet the passenger departed from, typically their planet of permanent residence.
- CryoSleep - Indicates whether the passenger elected to be put into suspended animation for the duration of the voyage. Passengers in cryosleep are confined to their cabins.
- Cabin - The cabin number where the passenger is staying. Takes the form deck/num/side, where side can be either P for *Port* or S for *Starboard*.
- Destination - The planet the passenger will be debarking to.
- Age - The age of the passenger.
- VIP - Whether the passenger has paid for special VIP service during the voyage.
- RoomService, FoodCourt, ShoppingMall, Spa, VRDeck - Amount the passenger has billed at each of the *Spaceship Titanic*'s many luxury amenities.
- Name - The first and last names of the passenger.
- Transported - Whether the passenger was transported to another dimension. This is the target, the column you are trying to predict.

The dataset is already split into training and testing substed, the training subset has about 8700 entries, when the testing part is about 4300 of passengers personal records. The main goal is to predict the value of the “Transported” field.

Basic Exploration and Data Preprocessing

We will perform a basic exploration of the dataset:

- First of all, we have to determine which features are numerical, which are categorical, and which entries are missing. Missing entries should be filtered or substituted with some artificial values (like zero values).
- We then have to analyze the features in the dataset and decide which ones are useful. Some features might be redundant.

- We will evaluate data distribution and correlation between different features.
- We will also perform standardization, a common requirement for many learning estimators.

Generally, we will prepare the data to train the estimators.

Algorithms

Since we have to predict the parameter, which takes only two values, we will stick with the models typically used for binary classification, find their hyperparameters, and compare performance. For example:

1. **Logistic Regression:** popular statistical model, often used for binary classification, which we explored during the labs. It uses a logistic function, which indicates the probability of the value being equal to 0 or 1 (in our case, true or false).
2. **K-Nearest Neighbors:** classification algorithm, based on clusterization - the object is assigned to some class based on the classes of its neighbors. The weights of different classes is a variable parameter of the algorithm.
3. **Random Forest:** the machine learning method which uses decision trees for assigning the object to one of many classes.
4. **Support Vector Machine:** method based on an idea of transferring data to a higher dimension space, and attempting to draw a hyperplane to divide them.
5. **Gradient Boosting:** a machine learning technique based on boosting in a functional space; gradient boosting combines weak "learners" into a single strong learner iteratively. We can use **XGBoost** or LightGBM for this.

We can use the `sklearn` or `tensorflow` library, in particularly:

- LogisticRegression — scikit-learn 1.6.1 documentation
- KNeighborsClassifier — scikit-learn 1.6.1 documentation
- RandomForestClassifier — scikit-learn 1.6.1 documentation
- 1.4. Support Vector Machines — scikit-learn 1.6.1 documentation
- GradientBoostingClassifier — scikit-learn 1.6.1 documentation

Tests

A grid search with cross-validation will be utilized to find hyperparameters for each model. Also, the final models will fit the whole dataset and be tested on an unseen one-third of the data, holdout cross-validation.

Visualization

1. We will create a visualization for some special cases, showing the provability of the object to be assigned to any of two classes. Example: https://www.tensorflow.org/tutorials/keras/classification#verify_predictions
2. We will compare the results of different hyperparameters of each model.

3. We will visualize the speed of learning for each selected model by drawing the trend of errors.

Additionally, we can utilize inline plot methods for some of the models (for example, tensorflow library has a model_plotter class).

```
tfdf.model_plotter.plot_model_in_colab(rf, tree_idx=0, max_depth=3)
```

Measures

Submissions are evaluated based on classification accuracy - the percentage of correctly predicted labels.

Additionally, to obtain better insights for each model we can check such measures as:

- Recall, or true positive rate: the proportion of all actual positives that were classified correctly as positives
- Precision: the proportion of all the model's positive classifications that are actually positive
- F-score: the harmonic mean of the precision and recall.