AI/ML coding challenge

Spaceship Titan

Chirobocea Mihail-Bogdan

2023

Content:

- Kaggle profile & notebook links
- Exploratory Data Analysis
  - Group data
  - Processing data
  - Deal with missing
  - Statistics
- AI/ML model application code
  - Basic models and why they are important
  - Fine-tune based on previous results
  - The best-performing solution
  - Comparison between models
  - Challenges
  - What I learned
  - Feature improvements

Kaggle profile & notebook links

Name & surname: Mihail Chirobocea

Kaggle username: mihailchirobocea

Profile link: https://www.kaggle.com/mihailchirobocea

Leaderboard score: 0.81014

Exploratory Data Analysis

First things first, we can take a look at the data. We have some composed features as well as some missing values and also categorical data. We will have to deal with all this.

● Group features

We found two composed features: PassengerId, Cabin. We can split them like that:

```
PassengerId [gggg_pp] -> Group [gggg], Passenger [pp]
Cabin [deck/num/side] -> Deck [deck], Cabin_num [num], Side [side]
```

In this way, we will be able to focus on each as an individual feature. We can also get rid of the individual features that are not relevant like we will see later, the case of Passenger, but keep the relevant part, Group.

● Processing data

Now we have to deal with categorical data. We can see that we have features with discrete values, so we can put some numerical labels like the following:

```
HomePlanet:      Europa      Earth       Mars
LabelEnocder:    0           1           2
```

Keep in mind that we have to avoid this labeling for NaN values. We will use a mask for that.

● Deal with missing

There are two options to deal with this: filling the missing values or dropping the rows with missing values.

Dropping will leave us with more accurate data, but we will have less data. Along this, we have to keep in mind that our testing data have missing values as well, and we want to predict all of them. So when it comes to testing, dropping is not an option. We will see that filling the data for training will result in better testing performance.

However, if we do both training and validation on dropped data we can make an idea of the performance of the model on a more accurate data that might be achieved by the improvements that will be mentioned further in this paper.

One first approach to fill data might be to use a special value like -1. The idea behind this is that we hope a neural network will be capable through its nonlinearity to process this data in a different way than any other values. It is also worth noting that -1 is a unique value for this dataset, and that is very important to achieve our goal. However, will we have a hope for neural networks to work with that, this will clearly impact negativly models like SVM, KNN, regression etc .This approach might be done before processing data as well.

Taking advantage of the previous processing and that we have numerical data, we can fill NaN values in a more mathematical way. We can fill a missing value based on the other values in that column. We experiment using mean and median for continuous values like age and the most frequent apparition for discrete values like home planet. We found that mean is performing a little bit better than median here.

● Statistics

A good starting point is to look at how many positive samples do we have compared to negative. As they are already balanced, we don't have to worry about this during training.

We are also interested in features distributions, plotting them, we can see that some fall as expected into normal distribution (age), uniform distribution (group) and some of them look quite similar to geometric distribution (passenger).

More interesting than that could be to compare the distribution of each feature for positive samples and negative samples. Doing that, we can see that they are not really linear separable or not even close to. We can still hope that a good kernel (for SVM) will do a great job, but it already looks like this approach will be very limited. We can try to use KNN to overcome this problem, but this approach also depends on the density of points in n-dimensional space. As the data is balanced and have similar distribution across positive and negative samples, this approach might not be good enough when we aim for accuracy.

As a conclusion of this analysis we might hope for decent results from linear models but it's likely that we will get better results using decisionals models and non-linear ones.

Nevertheless, we can see that the range of each feature is very different and this will hurt training as it might give more importance to features from a specific range just because of numerical reasons and not because it is more valuable.

To overcome this issue we will apply standardization on training data to also speed it up, and after that we will use the same transformation (with the same parameters) for validation and testing.

AI/ML model application code

Before doing any training, we have to split the data into train and validation. We used different proportions. The 80% - 20% is a good starting point. However we find that 90% - 10% might work good as well, giving among the best results in the leaderboard.

● Basic models and why they are important

Here we implement a bunch of ML models. As we expected from the EDA, SVM, KNN and logistic regression are not performing very well, this fact confirms that the data is not easily linearly separable even in n-dimensional space, and it's spread it's quite iregular. An alternative for this might be decision tree and random forest. We can see that these two have slightly better performance. To test the limits of this model we chose to use the XGBoost library which makes a combination between decision tree and random forest using learning rate. We also use SGD classifiers with hinge loss (for SVM) and log loss (for Logistic regression) to take advantage of learning rate, but more important, the advantage of using L1 regularization which tends to make feature selection.

● Fine-tune based on previous results

Almost all of this previous training can give us a feature importance for the model (except for KNN where it is not easy to compute). We can use these metrics to decide if we have irrelevant features and remove them. As a result we removed Passenger and VIP features. While SVM, KNN and logistic regression already had very small feature importance for these two and they get similar results, the decision tree and random forest get improved by this feature selection.

We test a range of values for XGBoost and end up with an accuracy around 80-81% on validation (and about 80% in leaderboard), being the best performance as far. We also noticed that this model doesn't get improved by using dropped data instead of filed one, which means that this limit could be a final one.

● The best-performing solution

We can get use of the previous information and build a neural network to overcome the issues. We are going to use Binary Cross-Entropy loss and sigmoid final activation. Along with this each layer, except the last one, will have batch normalization for more robust results and dropout to prevent overfitting. As we use batch normalization we can get rid of bias as it would get canceled by the mean anyway.

We experiment with different numbers of layers and end up using five layers. As activation functions we found out that tanh and GELU works the best, especially the last one. Also it is worth to notice that tanh might have problems with vanishing gradients, as the derivative of it is very close to 0.

Using a width (in this case, the number of neurons in each layer) equal to the input is a good starting point. The model seemed not to be able to make connections between the features so we chose to raise the width at the power of two. This shows big improvements, increasing accuracy on validation with about 1-2%.

When it comes to optimizers, we started with SGD with momentum of 0.9. We saw that his approach got stacked in local minima, having poor accuracy even on training. To overcome this issue we move to ADAM optimizer which is doing way better on training and slightly better on validation.

Furthermore we had to try and test different values for learning rate and batch size.

Final results.

After fine-tuning the hyperparameters we still had two options. We could either use the filled data or the dropped data. Because we need to fill the testing data anyway we saw that it is better to also fill the training and validation to get a slightly better score in leaderboard (81.014%) while the dropped training lies just behind (80.827%) .

However, using both filled training and validation data resulted in a highest score of 82.8511%, while using both dropped training and validation data resulted in a highest score of 84.3441%. So it is clear that the filling process might be improved and as such, it would result in better performances for this model.

- ## Comparison between models

We saw that SVM, KNN and logistic regression suffer a lot by the distribution of the data and in this case they are simply limited. While the decisional models seem to work better, the fact that the performance on dropped data does not improve shows us that this could be a final performance. Neural networks overcome these problems, being more sensitive to data and surpassing the bad distribution by non-linearities.

- ## Challenges

    - One of the first challenges was to deal with missing values. Our first attempts to drop data or fill it with special values didn't perform very well on the leaderboard so we had to come up with statistical ideas to fill the data. Finally, using mean for continuous values and most frequent for discret ones gave the best performance on the leaderboard from our submissions.
    - Another problem was to decide which features are irrelevant, as in this case it is hard to simply guess which one helps and which does not. To overcome this issue we took advantage of basic ML models to analyze the feature importance and their results.
    - While we did all this, the XGBoost seems limited to low accuracy after fine-tuning, and the main problem is that it does not improve even if we use more qualitative data, like the dropped one. Our solution was to build a neural network.
    - Using a conventional width of each layer the same as the input gave poor results. Changing the depth of the network still does not improve performance. The reason why a neural network performed worse than SVM on such a task was kind of a mystery. We thought that the reason could be that the network doesn't use at full capacity the power of neuron connections, so we chose to make more connections by increasing the width. This solved the problem giving competitive results to the previous models.
    - Even when we achieved these results, the network had poor accuracy even on training (about 81%) so that might be a problem. To overcome this issue we used ADAM optimizer instead of SGD with momentum. This allowed the model to achieve way better performance on training (about 87-92%) while the dropout did his job to prevent overfitting. As a result we ended with better performance on validation too.

- ## What I learned

During this project I developed my ability to handle missing data on both data processing and training fields. Along with that I future improved my skills for feature extraction and opened my perspective to different methods to do that. I found out about models that combine other basic ones, like XGBoost does, and I learned how it works and

how to take advantage of this. Furthermore I improved my skills of fine-tuning neural networks, especially when it comes to linear layers rather than convolutionals.

- Future improvements
    - We saw that the model is performing better when it is trained and tested on dropped data, so we want to achieve better filling techniques that overcome the problems raised by mean, median and mode. A better approach is to train a model for each feature that has missing values to predict that value. We can take all the samples that don't have any missing value and train on this data to predict for each feature. However, while this is a good approach, this might take a lot of time as it needs to train and fine-tune a lot of models. By doing this we expect an improvement of about 2% in accuracy.
    - While label encoding is a good starting point and gives decent results for such a case, it might be helpful to take a little bit more care about spatiality and use one-hot encoding instead of label encoding for categorical data.
    - Furthermore, an attempt to integrate the idea of multi-head attention into the model might see big improvements.