Alban Puech - Artem Golovin - Artem Kovalenko - Nikolay Bezkhodarnov

SD 201 Project: Better Data Encoding For Flight Delay Prediction.

Introduction

Flight delays are inevitable and very frequent, but they have a terrible impact on airline operations and on the passenger experience. Anticipating flight delays may help to better cope with their consequences.

Flight delay prediction is a very popular task on Kaggle. A simple Kaggle search for “Flight Delay” shows more than 623 notebooks and 71 Datasets. Among the available datasets was the one that we decided to use.

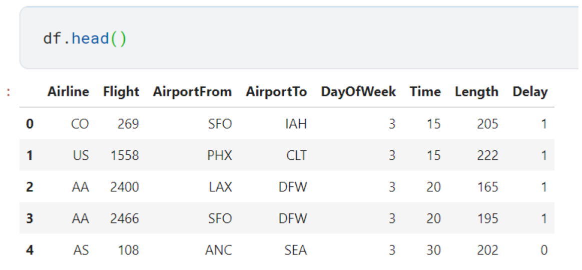
Looking at the 26 notebooks related to our dataset, we noticed that all of them did not achieve better accuracies than 0.65 for the classification task consisting of predicting if a given flight has been delayed or not. All those notebooks benchmark various machine learning models from the simplest to the most complicated ones, but all obtain similar results.

One surprising observation we could make reviewing those pieces of work is that all of them focus on the model selection. However, none of them bothers with encoding the categorical variable correctly.

Based on this information and seeing that we would probably not be able to get better accuracy than the one achieved by the numerous previous attempts, we decided to study how better encoding the (categorical) features that we have at our disposal would improve the performance of a given model.

Dataset Description

Our dataset has 539383 instances and 8 different features:



```
df.head()
```

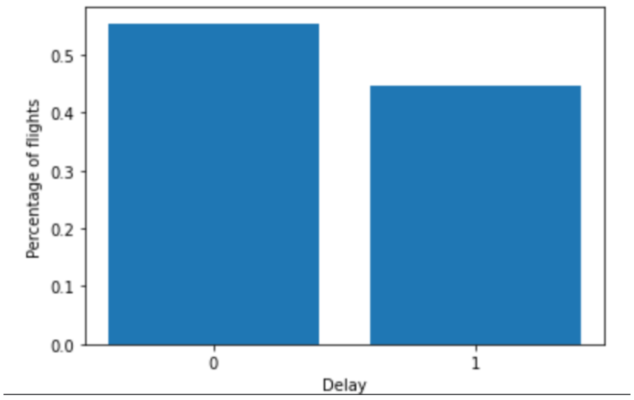
	Airline	Flight	AirportFrom	AirportTo	DayOfWeek	Time	Length	Delay
0	CO	269	SFO	IAH	3	15	205	1
1	US	1558	PHX	CLT	3	15	222	1
2	AA	2400	LAX	DFW	3	20	165	1
3	AA	2466	SFO	DFW	3	20	195	1
4	AS	108	ANC	SEA	3	30	202	0

It includes the following features:
1. “Airline” – Types of commercial airlines
2. “Flight” – Type of Aircraft
3. “Airport From” – Departure Airport
4. “Airport To” – Destination Airport
5. “DayOfWeek” – Day of the week when the departure took place
6. “Time” – Departure time in minutes from 0 to 1439
7. “Length” – Flight duration
8. “Delay” – Whether the flight was delayed or not.

Naive Approach

The approach that will henceforth be referred to as the "naive approach" is the one used by most kaggle users who tried to classify the flights.

We worked with the DecisionTreeClassifier from SciKitLearn and used the following features: "Flight", "DayOfWeek", "Time", and "Length".
We removed the "Delay", "AirportFrom", "AirportTo" and "Airline" features, which can't be directly handled by the tree classifier, as they are categorical data represented by strings.
Using 80 per cent of the dataset for training, we obtain an accuracy of 0.61. However, the data is imbalanced: We have more flights being on time in our dataset than flights being delayed:



It is thus more relevant to look at the recall and the precision scores, rather than the accuracy.

The recall score is the ratio between the number of positive instances that were correctly identified as positive (true positives) and the total number of positive instances. In our case, the recall quantifies the share of delayed flights that were correctly classified as delayed among the total number of delayed flights.

The precision is the ratio between the number of positive instances that were correctly identified as positive (true positives), and the total number of instances that were identified as positive, whether they are indeed part of class 1 or 0.

"F1" is a great scoring metric for imbalanced data when more attention is needed on the positives. It is computed as the harmonic mean of precision and recall. An F1 score of 1 corresponds to a perfect score. We will thus focus on this metric. With the naive model, we obtain an F1 score of 0.515 and an accuracy of 0.611.

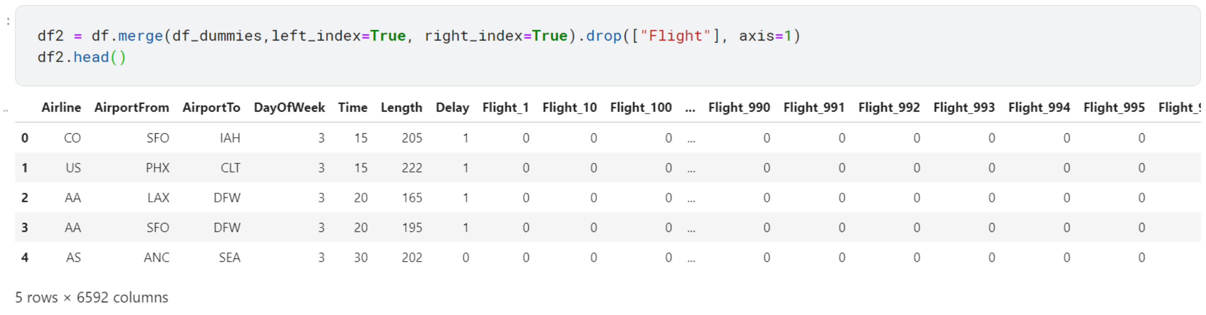
Better encode the Flight Number

one-hot encoding

In the naive approach, the "Flight" column was treated like a numerical variable, while it is in fact a categorical variable. Treating it like a numerical variable suggests that there is a notion of order between the flight numbers, while this is probably not the case.

The most basic way to encode categorical variables is one-hot encoding. Assuming that there are k different flight numbers in our dataset, we replace the flight number of each of our flights with a vector of length k, filled with k-1 zeros, and a one at the position corresponding to its flight number. This is done using the pd.get_dummies function.

This adds a number of columns equal to the number of possible values for the flight number. Adding those columns to the one we already had, we end up with 6592 columns.



```
df2 = df.merge(df_dummies,left_index=True, right_index=True).drop(["Flight"], axis=1)
df2.head()
```

```
   Airline AirportFrom AirportTo DayOfWeek Time Length Delay Flight_1 Flight_10 Flight_100 ... Flight_990 Flight_991 Flight_992 Flight_993 Flight_994 Flight_995 Flight_
0       CO         SFO       IAH         3   15    205     1        0         0          0 ...          0          0          0          0          0          0
1       US         PHX       CLT         3   15    222     1        0         0          0 ...          0          0          0          0          0          0
2       AA         LAX       DFW         3   20    165     1        0         0          0 ...          0          0          0          0          0          0
3       AA         SFO       DFW         3   20    195     1        0         0          0 ...          0          0          0          0          0          0
4       AS         ANC       SEA         3   30    202     0        0         0          0 ...          0          0          0          0          0          0

5 rows × 6592 columns
```

Because the dataset becomes now very large, the classifier takes a long time to train. Moreover, the data is now very sparse. We have too many features and not enough training data: We fall into the curse of dimensionality.

One hot encoding is used when there is no sense of order present, when the column cardinality is low, and when we simply want to convert categorical type to numerical type.

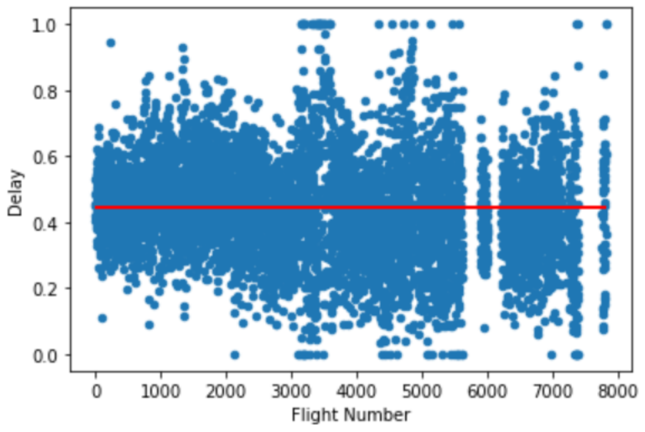
Target Encoding

We saw that OHE was not a good way of encoding the flight number, because it makes the data really sparse. We need to find a better encoding for the flight number: one that won’t add too many dimensions to the dataset, and that won't rely on some arbitrary ordering.

Another way of encoding the flight numbers would be by replacing each flight number x with the proportion of flights having this flight number that are delayed, which is an estimate of the conditional probability of delay given that the flight number is x.
To do this, for each flight number, we simply divide the number of delayed flights having this flight number by the total number of flights having this same flight number.

To avoid target leakage (we consider the distribution of the testing set to be unknown), we learn the target encodings from the training dataset only, by computing the conditional probabilities on the training dataset. We then replace the labels by their encoding on both the training and the testing set.

In the following plot, we can see that many flight numbers have a probability of being delayed very far from the total probability of delay computed on the entire dataset and represented by the red horizontal line:



We can now replace the flight number feature with the conditional probabilities that we computed earlier.

```
X_train.head()
```

	DayOfWeek	Time	Length	Flight
0	7	1210	190	0.548387
1	4	1055	255	0.454545
2	6	360	175	0.389831
3	4	740	367	0.410256
4	5	1205	125	0.559471

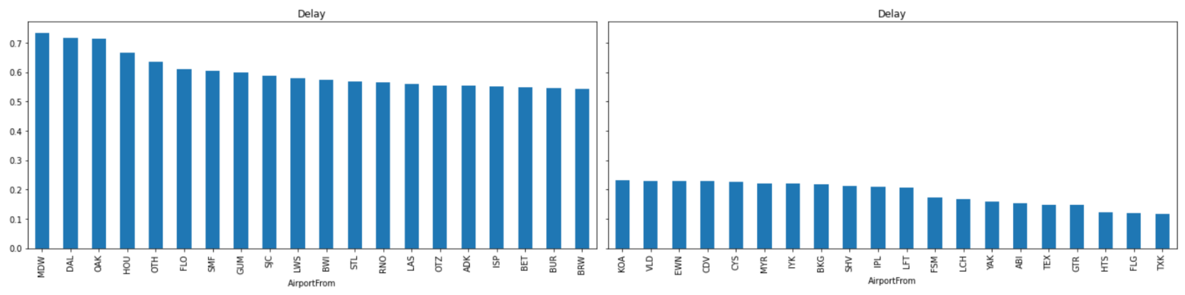
We obtain an F1 score of 0.527. This is slightly better than before.

Arrival and Departure Airport encoding:

While the flight number is a categorical feature represented by a numerical value, AirportTo and AirportFrom are categorical features that are represented by strings. We thus have to encode them if we want to use them with our classifier. We will again use target encoding and compute the conditional probabilities of delay for each departure airport, and store them in “delay_airport_from”.

We then plot the 20 airports having the largest chance of having their flights delayed, and the 20 having the lowest chance:

```
delay_airport_from = df.groupby("AirportFrom")["Delay"].mean()
fig, axes = plt.subplots(nrows=1,ncols=2, figsize=(20,5),sharey=True)
delay_airport_from.sort_values(ascending=False)[:20].plot.bar(ax = axes[0], subplots=True)
delay_airport_from.sort_values(ascending=False)[-20:].plot.bar(ax = axes[1], subplots=True)
plt.tight_layout()
```

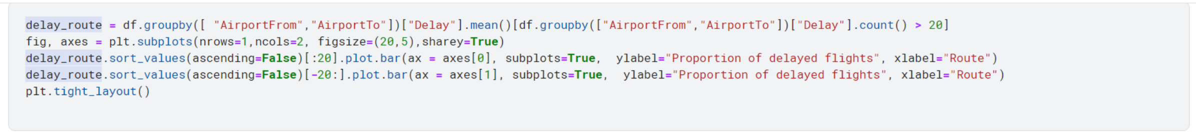


It is interesting to see that the probability of delay is very different across departure airports. For example, MDW (Chicago Airport), has more than 70 per cent of its flights delayed, while TXK (Texarkana Regional Airport) has only 10 per cent of its flights delayed. This is probably because it is a small domestic airport, which makes flights easier to keep on time.
We repeat the same operation for the arrival airport. Using our new labels for the “Flight”, “AirportTo” and “AirportFrom” features with the same classifier as before, we obtain an accuracy 0.640 of and an F1-score of 0.551.

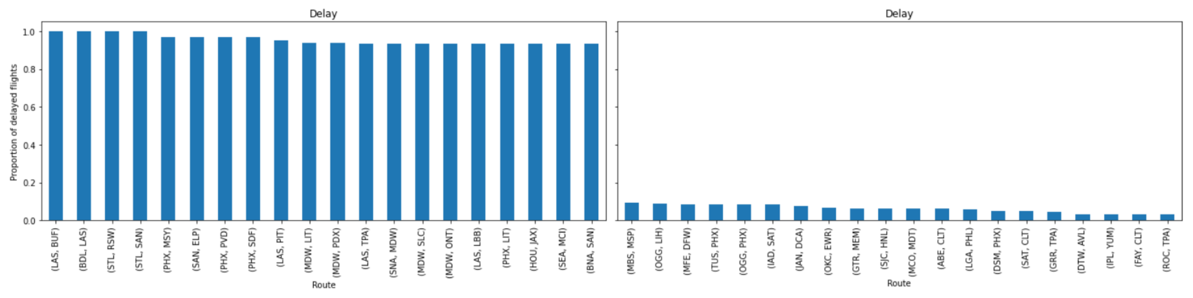
Creating Departure Airport - Arrival Airport pairs

It sounds reasonable to assume that some routes might be more often delayed than others. For example, this could be because the route goes over a zone where the weather often changes.

Grouping the flights by their (AirportTo, AirportFrom) pairs, and filtering to keep only the routes that have at least 20 flights taking them in the dataset, we can plot the probability of delay for the 20 routes that have the largest chance of having their flights delayed, and the 20 that have the lowest chance:



```
delay_route = df.groupby([ "AirportFrom","AirportTo"])["Delay"].mean()[df.groupby(["AirportFrom","AirportTo"])["Delay"].count() > 20]
fig, axes = plt.subplots(nrows=1,ncols=2, figsize=(20,5),sharey=True)
delay_route.sort_values(ascending=False)[:20].plot.bar(ax = axes[0], subplots=True,  ylabel="Proportion of delayed flights", xlabel="Route")
delay_route.sort_values(ascending=False)[-20:].plot.bar(ax = axes[1], subplots=True,  ylabel="Proportion of delayed flights", xlabel="Route")
plt.tight_layout()
```



There are actually many routes having all their flights delayed. This variance across routes suggests that this newly created feature will have a great discriminating power once added to our model.

This is confirmed by the new accuracy and F1-score that we obtain when we add this new feature to our dataset: 0.641 and 0.559 respectively.

Adding the Airline feature:

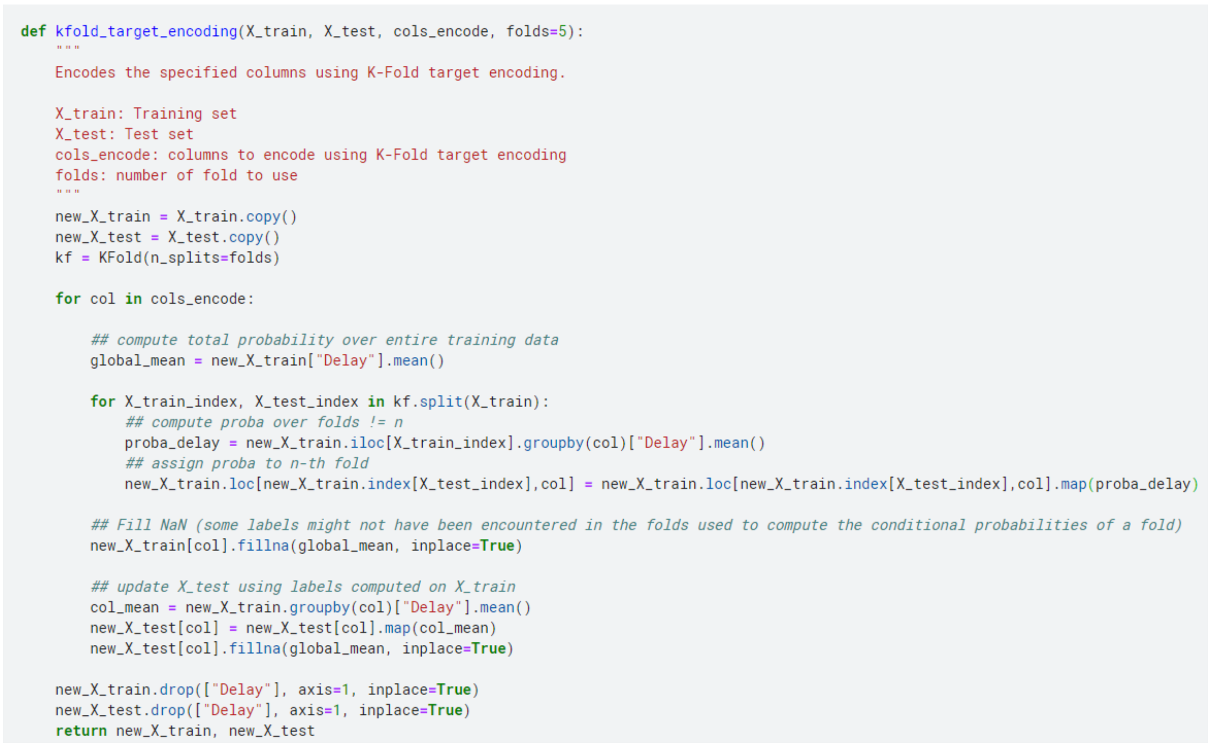
Adding the target encoding of the Airline, we obtain an accuracy of 0.652 and an F1-score of 0.567, which represents again a slight improvement on our previous model.

K-Fold target encoding:

An issue with greedy target encoding is overfitting. Indeed, the label corresponds to conditional probability estimates which are computed on the training set only. Overfitting will thus be important If the target (the feature to encode) distributions of the training and the testing set are really different. Moreover, there are even more risks of overfitting if the number of values used to compute the probabilities of each class is low. In this case, our target encoding does not generalize well.

A possible way of avoiding overfitting is K-Fold target encoding. When we use K-Fold target encoding, we replace the labels in the n-th fold of the training set with the probability computed out of the data that is in all the other folds of the training set.

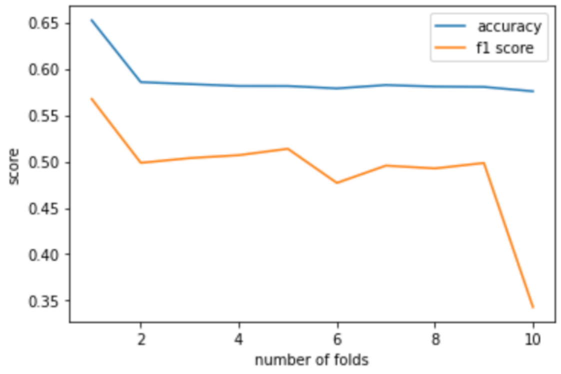
We then replace the labels in the testing set with the labels obtained from the training set, as we did before in the case of greedy target encoding. Some labels can be present in the testing set but not in the training set, in which case we replace them with the total probability of delay measured on the entire training set. This is again similar to what we did previously.
Below is the function that we wrote to encode our features using K-Fold target encoding:



```python
def kfold_target_encoding(X_train, X_test, cols_encode, folds=5):
    """
    Encodes the specified columns using K-Fold target encoding.

    X_train: Training set
    X_test: Test set
    cols_encode: columns to encode using K-Fold target encoding
    folds: number of fold to use
    """
    new_X_train = X_train.copy()
    new_X_test = X_test.copy()
    kf = KFold(n_splits=folds)

    for col in cols_encode:

        ## compute total probability over entire training data
        global_mean = new_X_train["Delay"].mean()

        for X_train_index, X_test_index in kf.split(X_train):
            ## compute proba over folds != n
            proba_delay = new_X_train.iloc[X_train_index].groupby(col)["Delay"].mean()
            ## assign proba to n-th fold
            new_X_train.loc[new_X_train.index[X_test_index],col] = new_X_train.loc[new_X_train.index[X_test_index],col].map(proba_delay)

        ## Fill NaN (some labels might not have been encountered in the folds used to compute the conditional probabilities of a fold)
        new_X_train[col].fillna(global_mean, inplace=True)

        ## update X_test using labels computed on X_train
        col_mean = new_X_train.groupby(col)["Delay"].mean()
        new_X_test[col] = new_X_test[col].map(col_mean)
        new_X_test[col].fillna(global_mean, inplace=True)

    new_X_train.drop(["Delay"], axis=1, inplace=True)
    new_X_test.drop(["Delay"], axis=1, inplace=True)
    return new_X_train, new_X_test
```

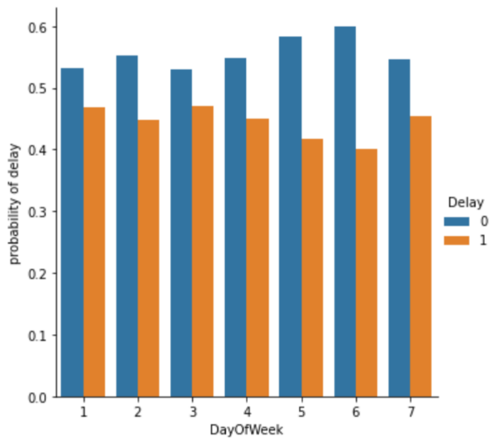
We can then fit and test our Classifier on the data with the "Airline", "AirportFrom", "AirportTo", and "Flight" features encoded with K-Fold target encoding.

K-Fold target encoding doesn't give us a better accuracy nor F1 score. This is probably because our dataset is too small making us use too few values to compute the mean delay for each label. The fact that the dataset is small and that we use more folds also increases the chance of ending up with NaN (later replaced by the total probability) in the testing set in situations where the label hasn't been encountered in the training set.



Encoding the departure time and day

In the original dataset, the day is encoded using an integer between 1 and 7. This is not an optimal way of encoding data as it suggests that, for example, Sunday (7) is closer to Saturday (6) than to Monday (1), while in reality, the probability of a flight being delayed on Sundays is closer to the one observed on Mondays:



The cardinality of the DayOfWeel feature being low, we could use One-Hot encoding (OHE). However, OHE doesn't conserve the relationship between the days (e.g the fact that Tuesday comes after Monday for example). This is also the case with Target encoding.

Angular encoding

We thus want to encode the fact that a Tuesday comes after a Monday, but we also want to represent the data in a way that shows that "Sunday comes after a Monday". A week is a cycle that repeats itself and we want to keep this idea in our representation. A possible way is to use Angular distances.

We assign to each day of the week an angle multiple of 360/7~=51° to cover the entire circle. However, we can't directly use them because these angle values and the absolute distances between them are not unique. For example, the distance between Monday (51 deg) and Sunday(360deg) can be computed as |360-51| = 309deg, and the distance between Saturday (360/7 * 6) and Sunday can be computed as |309-360| = 51deg, while we should have |Monday-Sunday| = |Sunday-Saturday| since only 24 hours separates both Saturday and Monday from Sunday. We thus use sin and cos to get a unique representation of this angle, so that:

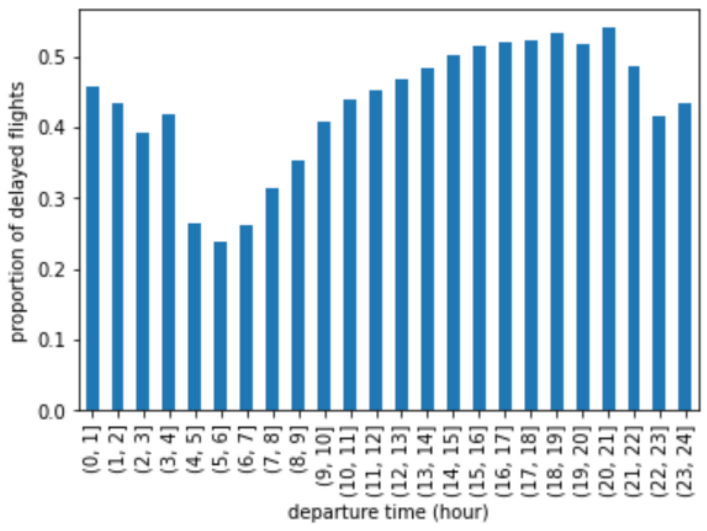
$$\begin{aligned}\text{Saturday} &= (\ \cos(360/7*6)\ ,\ \sin(360/7*6\ )\ )\\ \text{Sunday} &= (\ \cos(360/7*7)\ ,\ \sin(360/7*7)\ )\\ \text{Monday} &= (\ \cos(360/7*1)\ ,\ \sin(360/7*1)\ )\end{aligned}$$

and we get:

$$\begin{aligned}|\text{Monday}-\text{Sunday}|^2 &= (\cos(51)-\cos(360))^2+(\sin(51)-\sin(360))^2\\ &= (\cos(309)-\cos(360))^2+(\sin(309)-\sin(360))^2\\ &= |\text{Saturday}-\text{Sunday}|\end{aligned}$$

This can also be done with the “Time” feature, which represents the departure time.

It is interesting to see that the probability increases during the day, which might be explained by the fact that a delayed plane will also be late for its following flights, and that the delay accumulates throughout the day.
It is also interesting to see that the flights leaving between 10 pm and 2 am have a similar probability of being delayed. The distance between the time of these flights would have been very large with the previous encoding (ex: a flight leaving at 00:01 and one leaving at 23:59 were represented by a time of 1439 and 1 respectively). This is not the case anymore thanks to angular encoding.



Replacing the “Time” and “DayOfWeek” features by their angular encoding, and re-using the target encoded features we defined in the previous sections, we get an accuracy of 0.65- and an F1-score of 0.566. This is not an improvement compared to our previous model. This is probably because, as we will discuss it later, decision trees are good at handling label-encoded data, and the fact that

CatBoost way of encoding categorical data

Some classification algorithms handle categorical features using other encoding methods. It is the case of cat boost. According to the documentation:

“Before each split is selected in the tree (see Choosing the tree structure), categorical features are transformed to numerical. This is done using various statistics on combinations of categorical features and combinations of categorical and numerical features.”

The creator of Catboost also explains how categorical features are handled by Catboost in this conference talk at EuroPython. We also used these slides to better understand the algorithm.

When using catboost, we can declare what features are categorical, and let the model handle them. Catboost also automatically combines features, as we did when we created the “Route” feature.

Declaring all our features categorical features, we obtain an accuracy of 0.678
and an F1-score of 0.597.

If we used the numerically encoded categorical features (DayOfWeel, flight number, time) and removed the string-encoded categorical features (Airline, AirportTo, AirportFrom), as it was the case in the naive approach, we obtain an accuracy of 0.628 and an F1-score of 0.529. This shows the relevance of the (Airline, AirportTo, AirportFrom) features to the model.

Label Encoding

Some pieces of work on flight prediction using this dataset use the LabelEncoder function from scikitlearn to encode the categorical features. This function encodes target labels with values between 0 and n_classes-1. As we explained in the case of the DayOfWeek feature, this is not the right way of encoding categorical features, and even the documentation mentions that “This transformer should be used to encode target values, i.e. y, and not the input X.”.

Nonetheless, we obtained a good accuracy (0.649) and F1-score (0.563) with this encoding, which is probably why many people use this method on kaggle despite it being wrong. One explanation for this result is that decision trees are more tolerant with data encoding if we don’t limit the number of splits that the model can create for each categorical predictor.

Conclusion

In this project, we showed the importance of correctly encoding the categorical features of our dataset when performing a classification task.

Starting from a baseline model corresponding to the approach taken by most kaggle users working with this dataset, we have gradually added categorical features, encoding them using different strategies, and showing the importance of using the one suitable for each feature.

Assessing the performance of our model at each step, we advocated for the use of target encoding instead of one-hot encoding for high cardinality features.
We have also decided to use the F1 score as an evaluation metric instead of the accuracy, as it is more suited to imbalanced classification tasks. At each of these steps, we also analysed the distribution of delayed flights according to each feature that we were looking at.

We have also re-encoded categorical features that were already numerically encoded, using angular encoding, to better represent the circular nature of weeks and days.

Moreover, we implemented K-Fold target encoding, which did not perform well on our dataset, and explained the possible reasons.

Finally, we tried the catboost library, which natively supports categorical features by dynamically encoding them during the training process, outperforming all of our previous models.

Although we showed that a correct encoding improves the accuracy of our model, the difference in performance with a model ran on label-encoded features (An approach that we explained to be inappropriate) is not significant. This is because the accuracy of our models is low, and because the Decision trees are naturally more robust to data encoding than other methods. This indicates that the dataset and the model that we used were probably not the best suited for our particular purpose.

Another observation is that, even though we have outperformed the models presented by other kaggle users, the accuracy and the F1 score that we obtain are very low. This shows us that the dataset is more important than the model selection.

Repository:

The notebook, as well as the dataset, are available here:
https://github.com/albanpuech/Better-Data-Encoding-For-Flight-Delay-Prediction.